

Positive Intuitionistic Propositional Logic (PIPL) and its Algebraic Semantics

Carl Pollard
Ohio State University

Linguistics 681
(Algebraic Linguistics)
Tuesday, January 13, 2009

(1) Why PIPL?

- PIPL is a nice middle-of-the-road logic, in the sense that
 - the most standard (intuitionistic and classical) propositional logics are easily obtained from it by adding disjunction and negation (Chapter 11), while
 - the linguistically useful *substructural* logics that have proliferated in recent decades are easily obtained from it by removing inference rules (Chapter 14).
- PIPL also provides the type logic that underlies the all-important *typed lambda calculus* (Chapters 12 and 13).

(2) PIPL as a formal language

- We start with a finite set **Let** of (*propositional*) *letters*, the *nullary connective* T (read ‘true’), the two *binary connectives* \wedge ‘and’ and \rightarrow ‘implies’, and the two *auxiliary symbols* (‘left paren’ and) ‘right paren’.
- We use upper-case italic letters from the end of the Roman alphabet as metavariables over propositional letters.
- The PIPL language is a certain set of strings over the alphabet $\text{Let} \cup \{T, \wedge, \rightarrow, (,)\}$.
- The strings in the PIPL language are called (PIPL) **formulas**.
- We will use certain lower-case Greek letters (especially ϕ , ψ , ξ , and ζ) as metavariables over formulas.

(3) **Definition of the set of PIPL formulas**

- Pedantic version:
 - a. Each (length-one string consisting of just a single) propositional letter is a formula.
 - b. The (length-one string consisting of just) T is a formula.
 - c. If ϕ and ψ are formulas and c is a binary connective, then $(\phi c \psi)$, i.e. the string consisting of (followed by the string ϕ followed by c followed by the string ψ followed by), is a formula.
- Less pedantic, more colloquial version:
 - a. $X \in \Phi$ (for $X \in \text{Let}$)
 - b. $T \in \Phi$
 - c. $(\phi c \psi) \in \Phi$ (for $\phi, \psi \in \Phi$ and $c \in \{\wedge, \rightarrow\}$).

(4) **Formulas are idealizations of declarative sentences**

- \wedge and \rightarrow correspond to the sentence conjunctions ‘and’ and ‘if ... then’ (or ‘implies’).
- Formulas (called **atomic**) containing no binary connectives correspond to ‘simple’ English declarative sentences (ones with no sentential conjunctions).
- T corresponds to a simple sentence which is *necessarily true* (true no matter how the world is).
- Other atomic formulas correspond to simple sentences which are *contingent* (true or false depending how the world is).
- Internal structure of simple sentences is disregarded.
- The possibility that there might be more than one necessarily true simple sentence is ignored.
- The distinction between a sentence and an utterance of that sentence is ignored.

(5) **Semantic Interpretation**

- a. It is standard to assume that semantic interpretation is a function that maps (utterances of) linguistic expressions to the meanings they express.
- b. Different kinds of linguistic expressions express different kinds of meanings.
- c. The meanings expressed by declarative sentences are usually called **propositions**.
- d. The meanings of the binary sentential connectives, such as *and* and *if ... then* (or *implies*), are assumed to be binary operations on propositions.

(6) **Propositions**

- a. Intuitively, propositions are things that are either true or false.
- b. Propositions are called:
 - i. **necessary truths** if they are true no matter how things are;
 - ii. **necessary falsehoods** if they are false no matter how things are; and
 - iii. **contingent** otherwise (i.e. their truth value depends on how things are).
- c. There is usually assumed to be a binary relation on propositions called **entailment**. Intuitively, p entails q means that, no matter how things are, if p is true with things that way, then so is q .
- d. From the preceding, it is easy to see that entailment is a preorder.
- e. For two declarative sentences S and S' , we say S' **follows from** S iff the proposition expressed by S entails the one expressed by S' .

(7) Modelling Propositions

- a. We use a preordered set to model the set of propositions preordered by entailment.
- b. Clearly any proposition entails any necessary truth, so we model necessary truths as tops.
- c. Below we will show, based on how *and* and *if... then* work in NL argumentation, that the binary relations that model their meanings should be, respectively, a meet operation and an rpc operation.
- d. In short, we model propositions using a heyting presemi-lattice (hereafter, HPS).
- e. Later (Chapter 11), we will add more structure to model the meanings of *or* and *it is not the case that*, obtaining a kind of preordered algebra called a *boolean prelattice*.

(8) Modelling Semantic Interpretation

Putting the pieces together:

- a. We model NL declarative sentences by PIPL formulas.
- b. We model propositions by an HPS.
- c. We model semantic interpretation by an **HPS interpretation of PIPL**, which is defined to be a an HPS $\langle P, \sqsubseteq, \top, \sqcap, \rightarrow \rangle$. together with a function **sem** from Φ to P that satisfies the following conditions, for all formulas ϕ and ψ :
 - i. $\text{sem}(\top) = \top$
 - ii. $\text{sem}(\phi \wedge \psi) = \text{sem}(\phi) \sqcap \text{sem}(\psi)$
 - iii. $\text{sem}(\phi \rightarrow \psi) = \text{sem}(\phi) \rightarrow \text{sem}(\psi)$

(9) Why *and* is interpreted as a meet operation

For any English sentences S , S' , and S'' , according to the intuitions of native English speakers:

- a. S' follows from the conjoined sentence S' and S'' .
- b. S'' follows from the conjoined sentence S' and S'' .
- c. If S' and S'' both follow from S , then so does the conjoined sentence S' and S'' .

(10) **Why *if* ... *then* is interpreted as an rpc operation**

For any English sentences S, S', and S'', according to the intuitions of native English speakers:

- a. S' follows from the conjoined sentence: if S then S', and S.
- b. If S' follows from the conjoined sentence S'' and S, then the conditional sentence if S then S' follows from S''.