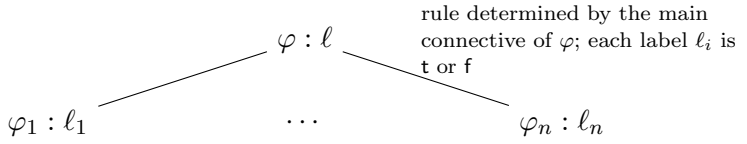
TOWARDS AN OPERATIONAL PROOF-THEORETIC SEMANTICS

ALEXANDER V. GHEORGHIU

Abstract. Model-theoretic semantics comes with a computational counterpart: to check validity is to search for a countermodel, and tableaux, resolution, and satisfiability solvers make the semantics something one can run. Proof-theoretic semantics — the younger paradigm in which meaning is given by inferential role rather than by denotation — has no established counterpart. This paper proposes one: logic programming. We show that Sandqvist's base-extension semantics for intuitionistic propositional logic is executable in a precise sense. Bases are logic programs, and to each formula φ we assign a goal $[\![\varphi]\!]$ of a second-order hereditary Harrop language such that φ is supported in a base $\mathcal{B}$ exactly when $[\![\varphi]\!]$ is operationally derivable from $\mathcal{B}$ read as a program. The encoding is the Russell–Prawitz translation of the connectives with quantification restricted to atoms, and this restriction is not a limitation but the point: it is what the support clauses themselves demand. The correspondence is exact base by base, not merely at the level of validity, so that intuitionistic validity becomes the success of a single query against the empty program. Logic programming stands to proof-theoretic semantics as tableaux and satisfiability solvers stand to model-theoretic semantics.

1. Introduction

Tableaux and satisfiability (SAT) solvers are the basis of many of the tools with which reasoning is automated. Their common shape is a search for a countermodel. To check that φ is valid, suppose it false and decompose: the main connective of each formula determines a rule, branching where the connective demands a choice, closing a branch when it becomes contradictory.



Every node carries a label ℓ, ℓ_1, …, ℓ_n dictating its status in the model under construction.

2020 *Mathematics Subject Classification.* 03B20, 68N17, 03F50.

Key words and phrases. proof-theoretic semantics, base-extension semantics, intuitionistic logic, logic programming, hereditary Harrop formulae, uniform proofs, tableaux.

Behind these tools stands *model-theoretic semantics* (M-tS): fix a class of structures, define satisfaction by recursion on the connectives, and take validity to be truth in every structure. The solvers are how we often use the semantics; a tableau explores precisely the space of structures the semantics quantifies over. They are not accessories to the semantics but operationalize it. They are what makes it useful for representing and automating reasoning — for *good old-fashioned AI*.

An alternative semantic paradigm has recently achieved mathematical maturity: *proof-theoretic semantics* (P-tS) [20]. Philosophically, this approach differs from M-tS by being based on the *meaning-as-use* paradigm (as opposed to meaning-as-denotation) imported into logic, with use read as inferential role. Thus the word "proof" in the name should not mislead: P-tS is not proof theory, but takes inferences (rather than denotations) as primitive. The philosophical literature here is rich, but it is not our concern in this paper.

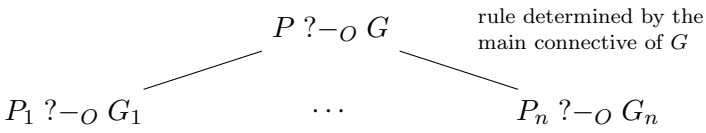
We shall concern ourselves with a particularly successful branch of P-tS known as *base-extension semantics* (B-eS). Here meaning is defined relative to a *base*: a set of inferential commitments governing atomic sentences. The meaning of a formula φ relative to a base $\mathcal{B}$ is given by a judgement $\Vdash_{\mathcal{B}} \varphi$, called *support* ($\Vdash$), whose clauses fix, for each connective, what it takes to support a formula built with it. A base may be thought of as the P-tS counterpart of a model in M-tS, and support as the counterpart of satisfaction.

The success of B-eS in recent years is its expressiveness. It has now been given for classical propositional logic [18] and for intuitionistic propositional logic (IPL) [19]; the relationship between the two is more delicate than one might expect, and we take it up in Section 3. The paradigm has since been extended to first-order logic [5], intuitionistic multiplicative linear logic [7], the logic of bunched implications [8], and modal logics [1]. Given that it has been developed for so many standard logics, the semantics should no longer be thought exotic. How does one use it though? It remains to operationalize the semantics.

So what stands to B-eS as tableaux and SAT solvers stand to M-tS? Transposing the model-search apparatus directly to B-eS would miss exactly the difference that makes the semantics interesting. The underlying idea of B-eS is behaviour, not structure. Therefore, it seems reasonable to suggest that the operational counterpart should be a search through inferential commitments (as opposed to models). Fortunately, there is already a mature technology for such search, itself a piece of good old-fashioned AI: *logic programming* (LP).

Our claim is not that LP may be used to implement B-eS, in the weak sense in which any mathematics can be implemented. Rather it is that LP operationalizes the unfolding of the semantics as tableaux operationalizes the unfolding of M-tS. When LP is represented as goal-directed proof-search, it decomposes a *goal* analogously to how tableaux decomposes a formula:

the main connective of the goal determines which rule applies, disjunctive goals branch, and a branch closes successfully when the goal G has been reduced to something the program P supplies outright:



We write $P \vdash_O G$ for when the search $P \mathrel{?\!-_O} G$ succeeds.

Since B-eS exists for many logics, the claim that it *is* LP has content only once the right LP language has been matched to the right logic. Here we lean on the analysis of Miller et al. [15]: a logic programming language is given by a class of *definite* formulae, which serve as program clauses, together with a class of *goal* formulae, which are queried against them, such that provability is captured by *uniform* proofs — goal-directed search in which the logical structure of the goal drives the computation and the program is consulted only at atoms. Horn clauses meet the requirement, and are the foundation of Prolog; so do hereditary Harrop formulae, the foundation of λProlog [14].

In this short paper, we show that the B-eS for IPL is captured by a second-order fragment of hereditary Harrop logic programming, in the family of λProlog but with quantification over atoms only. That is,

$$\Vdash_{\mathcal{B}} \varphi \qquad \text{iff} \qquad \mathcal{B} \vdash_O [\![\varphi]\!],$$

where $\vdash_O$ is operational derivability and $[\![\varphi]\!]$ is the encoding of φ as a goal formula. A formula's semantic status can be established by running it.

The correspondence pays in both directions. Read from right to left, it says that goal-directed search is not merely a proof-search discipline that happens to be complete for a well-chosen fragment: it is a semantics. A hereditary Harrop program is a meaning-conferring object rather than a database of facts, and the uniform-proof restriction is a semantic condition rather than an operational convenience.

Roadmap. Section 2 presents the three LP languages we require — Horn clauses, hereditary Harrop formulae, and their second-order extensions — with their operational semantics and a worked example. Section 3 defines support, explains its clauses, and recalls the classical/intuitionistic correspondence. Section 4 develops the encoding and proves the theorem above. Section 5 draws out the consequences and the programme they suggest. This is a short paper: apart from the cited theorems of Section 3, the definitions and results of Sections 2–4 are those of the extended version [6], where full proofs are given; we indicate the shape of the main argument as we go.

2. Logic Programming

We present LP before B-eS, since some of the machinery can then be recycled. Throughout we use a proof-theoretic presentation rather than the resolution-based approach of Kowalski [12]. The reason is flexibility: we shall

consider three languages in order of increasing expressiveness. A language $\mathcal{L}$ comprises a class of *definite formulae* (or *clauses*) D and a class of *goal formulae* G.

Definition 2.1 (Program)**.** Let $\mathcal{L}$ be a logic programming language. An *$\mathcal{L}$-program* P is a finite set of definite formulae in the sense of $\mathcal{L}$.

In the proof-theoretic presentation, *resolution* is replaced by *backchaining*: to establish an atom, one looks in the program for a clause whose head is that atom. This is a change of presentation only, not of substance. A Prolog clause $H :- B_1, ..., B_n$ is thus captured by a definite clause $H \subset B_1, ..., B_n$. Here $\subset$ is Kowalski's reverse implication, read "H if $B_1, \ldots, B_n$"; in the grammars below we write clauses head-last, so the same clause appears as $(B_1 \cap \cdots \cap B_n) \supset H$.

Definition 2.2 (Horn clauses)**.** The language of *propositional Horn clauses* (H) consists of the *definite formulae* D and *goal formulae* G generated by

$$D \ ::= \ A \mid G \supset A \mid D \cap D \qquad\qquad G \ ::= \ A \mid G \cap G \mid G \cup G.$$

Observe that an implication may occur only in definite formulae, never in goals. The language can therefore express conditionals but not hypotheticals — and hypotheticals are precisely what intuitionistic logic requires. This motivates an extension in which hypothetical goals are admitted. Crucially, the extension must be such that proof-search remains goal-directed, since goal-directedness is the point of the LP formalism. The following extension is due to Miller [13] and Harland [10].

Definition 2.3 (Hereditary Harrop formulae)**.** The language of *propositional hereditary Harrop formulae* (hH) is generated by

$$D \ ::= \ A \mid G \supset A \mid D \cap D \qquad\qquad G \ ::= \ A \mid D \supset G \mid G \cap G \mid G \cup G.$$

Clause heads are atomic, as in the extended version [6]: the definite formulae are those of H, and the addition is in the goals. It is the hypothetical goal $D \supset G$, whose procedural reading is: *to establish G under the assumption D, add D to the program and carry on.* The program is thus no longer fixed for the duration of a computation; it grows as the search descends into implicational goals. This is the LP counterpart of discharging an assumption in natural deduction, and it is the device that will give implication its intuitionistic content.

For our systems to be genuinely operational, programs must be computable in some sense. Of course, we could simply require that they have computable descriptions — that they be recursively enumerable. However, with one small addition we can constrain the programs used throughout the paper to be *finite*, and hence certainly tractable for implementation. The device, following Miller et al. [15], is to observe that every program we require has a regular structure that can be captured by second-order quantification. A quantified formula packs a schematic family — one instance for each atom — into a single finite formula.

Definition 2.4 (Second-order hereditary Harrop formulae)**.** The language $\mathcal{L}^2$ extends $\mathcal{L} \in \{\mathsf{H}, \mathsf{hH}\}$ with a *second-order quantifier* ranging over atoms:

$$D \ ::= \ \ldots \mid \forall X\, D \qquad\qquad G \ ::= \ \ldots \mid \forall X\, G.$$

Here X ranges over a set of propositional variables disjoint from the atoms. The *atomic formulae* are the atoms together with the variables, and A ranges over them in the grammars above, so that, for instance, $G \supset X$ is a definite formula. Programs and goals are *closed*: every variable occurring in them is bound. We write $G[A/X]$ for the result of replacing the free occurrences of X in G by the atom A, and say that an atom is *fresh* for a formula or program if it does not occur in it.

We have glossed the procedure as backchaining, which requires looking inside the program for clauses. Since programs in the above languages have internal structure, this inspection itself needs a procedural account. The following provides one.

Definition 2.5 (Elaboration)**.** The elaboration $[P]$ of a program P is defined recursively as follows:

- $P \subseteq [P]$;
- if $D_1 \cap D_2 \in [P]$, then $D_1 \in [P]$ and $D_2 \in [P]$;
- if $\forall X\, D \in [P]$, then $D[B/X] \in [P]$ for every atom B.

Elaboration acts on clauses one at a time, so $[P]$ is the union of the elaborations of the members of P, and in particular $[P \cup Q] = [P] \cup [Q]$; we use this freely. Note also that $[P]$ may be infinite, since the quantifier condition instantiates a quantified clause at every atom. This is harmless: the operational semantics inspects only finitely much of it in any one derivation. The important feature is that every clause eventually elaborates to atoms and to clauses $G \supset A$ with atomic head.

We may now say what it is to run a query. The relation $\vdash_O$ is defined uniformly for all three languages, and we give the procedural reading of each clause alongside it.

Definition 2.6 (Operational semantics)**.** The relation $P \vdash_O G$ is defined by:

- Fact: $P \vdash_O A$ if $A \in [P]$.
- Backchain: $P \vdash_O A$ if $G \supset A \in [P]$ and $P \vdash_O G$.
- And: $P \vdash_O G_1 \cap G_2$ if $P \vdash_O G_1$ and $P \vdash_O G_2$.
- Or: $P \vdash_O G_1 \cup G_2$ if $P \vdash_O G_1$ or $P \vdash_O G_2$.
- Augment: $P \vdash_O D \supset G$ if $P \cup \{D\} \vdash_O G$.
- Generic: $P \vdash_O \forall X\, G$ if $P \vdash_O G[A/X]$ for some atom A not occurring in P or G.

Read as instructions: to establish an atom, find it in the program, or find a clause with that head and establish the clause's body instead; to establish a conjunction, establish both parts; to establish a disjunction, choose a part and establish it; to establish an implication, assume the antecedent and continue; to establish a schematic claim, establish one instance at an atom

about which nothing is known. Only the first two rules look at the program at all. The rest are determined by the goal.

Example 2.7. Let $P = \{p \supset q\}$. The following query succeeds; each edge of the search tree is labelled with the rule applied.

$$
\begin{array}{c}
P \mathrel{?\!-_O} \forall X\,((q \supset X) \supset (p \supset X)) \\
\downarrow \ \textsc{Generic}, A \text{ fresh} \\
P \mathrel{?\!-_O} (q \supset A) \supset (p \supset A) \\
\downarrow \ \textsc{Augment} \\
P, q \supset A \mathrel{?\!-_O} p \supset A \\
\downarrow \ \textsc{Augment} \\
P, q \supset A, p \mathrel{?\!-_O} A \\
\downarrow \ \textsc{Backchain} \text{ on } q \supset A \\
P, q \supset A, p \mathrel{?\!-_O} q \\
\downarrow \ \textsc{Backchain} \text{ on } p \supset q \\
P, q \supset A, p \mathrel{?\!-_O} p \\
\downarrow \ \textsc{Fact} \\
\checkmark
\end{array}
$$

The goal expresses, schematically, that whatever follows from q follows from p. The fresh atom A stands proxy for an arbitrary conclusion, and the two applications of Augment show the program growing as the search descends: what began as a single clause is, by the fourth line, a program of three.

The freshness condition in Generic is important. A schematic claim would seem to call for infinitely many checks, and the rule performs only one. It does not require that $G[A/X]$ hold for every atom; it requires it for some atom occurring in neither the program nor the goal. Because nothing whatever is known about such an atom, a derivation using it can appeal to no property it does not share with every other atom, and so the single witness establishes the schema. This is the eigenvariable discipline of proof theory, and it is what will make the universal quantifiers of the support semantics finitary — a point on which the main theorem depends.

It remains to record three structural properties of $\vdash_O$, each of which says something one would want to be true of a search procedure. We write $Q \sqsupseteq P$ to mean $Q \supseteq P$, borrowing the notation for base extension deliberately: Section 4 will identify the two relations. All three properties are proved by routine induction on derivations, and all three preserve the height of the derivation — which matters, since the induction in Section 4 is on height.

Lemma 2.8 (Monotonicity)**.** *If $P \vdash_O G$ and $Q \sqsupseteq P$, then $Q \vdash_O G$.*

Lemma 2.9 (Renaming)**.** *Let σ be any function from atoms to atoms, extended homomorphically to formulae and programs. If $P \vdash_O G$, then $P\sigma \vdash_O G\sigma$.*

Lemma 2.10 (Inversion)**.** *If $P \vdash_O D \supset G$, then $P \cup \{D\} \vdash_O G$. If $P \vdash_O G_1 \cap G_2$, then $P \vdash_O G_i$ for each i. If $P \vdash_O \forall X\, G$, then $P \vdash_O G[A/X]$ for every atom A.*

Monotonicity says that adding clauses never destroys a solution. Renaming says that the names of atoms carry no information, and requires care only at Generic, where the eigenvariable must be moved out of the way of σ before the induction hypothesis applies; Monotonicity uses it at the same point, since the eigenvariable of a search over P may occur in the larger program. Inversion is the converse of the goal-directedness we have been emphasising: because the main connective of a goal determines the last rule of its derivation — Fact and Backchain concluding only atomic goals — one can read the premisses back off the conclusion; for the quantifier, Renaming then moves the eigenvariable of Generic to any atom whatever. It is the rule-by-rule character of $\vdash_O$ that makes this possible, and it is what will let the comparison with support in Section 4 run in both directions.

3. Base-extension Semantics

We have already given a brief introduction to B-eS above, and we elide further motivation in favour of the technical details. Our logical syntax is given by the grammar

$$\varphi, \psi \ ::= \ A \ \mid \ \bot \ \mid \ \varphi \wedge \psi \ \mid \ \varphi \vee \psi \ \mid \ \varphi \to \psi,$$

with $\neg\varphi \coloneqq \varphi \to \bot$. Throughout, let $\mathcal{L}$ be one of the LP languages of Section 2.

Definition 3.1 (Base)**.** An *$\mathcal{L}$-base* $\mathcal{B}$ is an $\mathcal{L}$-program. We write $\mathcal{C} \sqsupseteq \mathcal{B}$ for extension of bases, meaning $\mathcal{C} \supseteq \mathcal{B}$ with $\mathcal{C}$ an $\mathcal{L}$-base.

It may seem redundant to define bases and programs separately; the point is to smooth the correspondence between the two literatures. Nothing in the literature on B-eS requires bases to be finite — indeed, it is crucial that they may be infinite. The restriction to finite bases is available here only because of the second-order quantification: readers within the P-tS literature will recognise that this recovers the expressive strength required for the results cited below.

The support relation is defined relative to a fixed choice of $\mathcal{L}$; the base extensions quantified over in its clauses range over $\mathcal{L}$-bases. Where the choice needs emphasis we speak of $\mathcal{L}$-support.

Definition 3.2 (Support)**.** The relations $\Vdash_{\mathcal{B}} \varphi$ and $\Gamma \Vdash_{\mathcal{B}} \varphi$ are defined by the clauses of Figure 1, where Γ is a finite set of formulae and we write $\varphi \Vdash_{\mathcal{B}} \psi$ for $\{\varphi\} \Vdash_{\mathcal{B}} \psi$.

The clauses $(\vee)$ and $(\bot)$ give the meaning of a connective as a consequence condition: they state what one may *conclude* from the formula, an atom A standing proxy for an arbitrary conclusion. The formula $\varphi \vee \psi$ licenses every conclusion that each disjunct separately licenses, and the formula $\bot$ licenses

$\Vdash_{\mathcal{B}} A$	iff $\mathcal{B} \vdash_O A$	(At)
$\Vdash_{\mathcal{B}} \varphi \to \psi$	iff $\varphi \Vdash_{\mathcal{B}} \psi$	$(\to)$
$\Vdash_{\mathcal{B}} \varphi \wedge \psi$	iff $\Vdash_{\mathcal{B}} \varphi$ and $\Vdash_{\mathcal{B}} \psi$	$(\wedge)$
$\Vdash_{\mathcal{B}} \varphi \vee \psi$	iff for every $\mathcal{C} \sqsupseteq \mathcal{B}$ and $A \in \mathrm{At}$, if $\varphi \Vdash_{\mathcal{C}} A$ and $\psi \Vdash_{\mathcal{C}} A$, then $\Vdash_{\mathcal{C}} A$	$(\vee)$
$\Vdash_{\mathcal{B}} \bot$	iff $\Vdash_{\mathcal{B}} A$ for every $A \in \mathrm{At}$	$(\bot)$
$\Gamma \Vdash_{\mathcal{B}} \varphi$	iff for every $\mathcal{C} \sqsupseteq \mathcal{B}$, if $\Vdash_{\mathcal{C}} \psi$ for all $\psi \in \Gamma$, then $\Vdash_{\mathcal{C}} \varphi$	(Inf)

FIGURE 1. Support in a base.

every conclusion. Each is, visibly, the elimination rule for its connective read as a definition of meaning. The clause $(\to)$ fits the same pattern — $\varphi \to \psi$ licenses ψ from φ — expressed through the context-extension clause (Inf). The clause $(\wedge)$ is direct, but it too has an elimination form (Remark 3.3). The clause (At) is the interface with the base: atomic support is provability in the base, and here the reading of bases as programs does its first work, since $\vdash_O$ is proof-search.

Remark 3.3. The clauses of Figure 1 are Sandqvist's [19]. The direct clause for conjunction is equivalent to an elimination-style clause of the same shape as $(\vee)$ [7]:

$$\Vdash_{\mathcal{B}} \varphi \wedge \psi \quad \text{iff} \quad \text{for every } \mathcal{C} \sqsupseteq \mathcal{B} \text{ and } A \in \mathrm{At}, \text{ if } \varphi, \psi \Vdash_{\mathcal{C}} A \text{ then } \Vdash_{\mathcal{C}} A,$$

and implication likewise admits an elimination form: $\Vdash_{\mathcal{B}} \varphi \to \psi$ iff for every $\mathcal{C} \sqsupseteq \mathcal{B}$ and $A \in \mathrm{At}$, if $\Vdash_{\mathcal{C}} \varphi$ and $\psi \Vdash_{\mathcal{C}} A$ then $\Vdash_{\mathcal{C}} A$ [6, Corollary 3.7]. Every connective is thus characterised by what it licenses one to conclude.

Example 3.4. Fix $\mathcal{L} = \mathsf{H}$. Let $\mathcal{B} = \{p\}$ and consider $\Vdash_{\mathcal{B}} p \vee q$. Does it hold? It does. Given $\mathcal{C} \sqsupseteq \mathcal{B}$ and an atom A with $p \Vdash_{\mathcal{C}} A$ and $q \Vdash_{\mathcal{C}} A$, the fact $p \in \mathcal{C}$ yields $\Vdash_{\mathcal{C}} p$ by (At), and the first hypothesis then yields $\Vdash_{\mathcal{C}} A$. Note that no disjunction occurs in the base: the disjunction is supported through what its disjuncts license, not through any disjunctive commitment.

A formula φ is *$\mathcal{L}$-valid* iff $\Vdash_{\mathcal{B}} \varphi$ holds for every $\mathcal{L}$-base $\mathcal{B}$. Equivalently, φ is *$\mathcal{L}$-valid* iff $\Vdash_{\varnothing} \varphi$. What is curious about B-eS is that the same clauses define two different logics depending on the notion of base:

Theorem 3.5 (Sandqvist [18])**.** *A formula φ is H^2-valid if and only if φ is provable in classical propositional logic.*

Theorem 3.6 (Sandqvist [19])**.** *A formula φ is hH^2-valid if and only if φ is provable in intuitionistic propositional logic.*

Sandqvist states Theorem 3.6 for his class of bases, arbitrary sets of atomic rules of level at most two; the proof transfers unchanged to hH^2-bases [6, Section 3].

There is currently no consensus on how to understand this result. The contravariance — hH extends H, while classical logic extends intuitionistic logic — makes some sense: in hH the support clauses quantify over a larger space of bases than in H, so validity is harder to attain and fewer formulae achieve it. But this heuristic does not account for why one should expect exactly classical logic in the one case and exactly intuitionistic logic in the other. The key differentiating feature appears to be the hypothetical reasoning available in hH but not in H.

Example 3.7 (Peirce's formula)**.** Peirce's formula $((p \to q) \to p) \to p$ is classically valid, hence H^2-valid by Theorem 3.5; but it is not provable in IPL, hence not hH^2-valid by Theorem 3.6. So there is a hereditary Harrop base — we shall see in Example 4.12 that the empty base suffices — at which the support clauses refuse it. The refusal is possible only because hH-extensions can install the hypothesis $(p \to q) \to p$ as a genuinely hypothetical commitment rather than a material one.

This completes our introduction to B-eS. The remainder of the paper establishes that LP, as defined above, operationalises it.

4. Soundness and Completeness

We have defined both LP and B-eS; it remains to explain their correspondence. The idea is that support judgements correspond to certain queries. Given the complexity of the clauses in Figure 1, we restrict attention to hH^2, which is expressive enough to encapsulate all the requisite structure. By Theorem 3.6, this means that in this section we are concerned only with IPL; classical logic is treated in Section 5.

Every LP language we consider has two syntactic classes of formula: *goals*, which are queried, and *definite clauses*, which make up the program. Roughly, these play the roles of the query and the clause database in a resolution setting, but with one crucial difference: the rule Augment, on encountering a goal $D \supset G$, installs D into the program as a hypothesis and continues with G. Consequently an encoded formula may appear in either class — as a goal when it is queried, and as a program clause when Augment installs it. Since IPL has only one class of formulae, we work with the fragment of hH^2 that makes no such distinction.

Definition 4.1 (Neutral formulae)**.** A formula is *neutral* if it is generated by the grammar

$$N \ ::= \ A \ \mid \ N \supset A \ \mid \ N \cap N \ \mid \ \forall X\, N,$$

where A ranges over atomic formulae.

Proposition 4.2. *Every neutral formula is both a definite formula and a goal formula of* hH^2.

The proof is a routine induction on the grammar: an implication $N \supset A$ is definite read as $G \supset A$ and a goal read as $D \supset G$, and the remaining cases are immediate. Note that the consequent of an implication in a neutral formula is atomic: a $\cup$-free implication with a compound consequent is a goal but not a definite formula.

We now encode IPL formulae as neutral formulae.

Definition 4.3 (Encoding)**.** To each IPL formula φ, assign a formula $[\![\varphi]\!]$ by:

$$\begin{aligned} [\![A]\!] &= A \\ [\![\bot]\!] &= \forall X\, X \\ [\![\varphi \to \psi]\!] &= \forall X\, \big(([\![\varphi]\!] \cap ([\![\psi]\!] \supset X)) \supset X\big) \\ [\![\varphi \wedge \psi]\!] &= [\![\varphi]\!] \cap [\![\psi]\!] \\ [\![\varphi \vee \psi]\!] &= \forall X\, \big(((([\![\varphi]\!] \supset X) \cap ([\![\psi]\!] \supset X)) \supset X\big) \end{aligned}$$

where X is in each case a variable not occurring in $[\![\varphi]\!]$ or $[\![\psi]\!]$.

By induction on φ, each $[\![\varphi]\!]$ is neutral, so a base extended by an encoded formula is again a base. The encoding transcribes the support clauses. The goal $[\![\varphi \vee \psi]\!]$ says: for each atom X taken as an arbitrary conclusion, if $[\![\varphi]\!]$ yields X and $[\![\psi]\!]$ yields X, then X — this is clause $(\vee)$, with the universal over atoms become $\forall X$ and the quantifier over base extensions become program augmentation. The goal $[\![\bot]\!] = \forall X\, X$ says that every conclusion follows. Conjunction is encoded directly, following clause $(\wedge)$. Implication is encoded in elimination form: the goal $[\![\varphi \to \psi]\!]$ says that, for an arbitrary conclusion X, $[\![\varphi]\!]$ together with the continuation $[\![\psi]\!] \supset X$ yields X — the elimination form of $(\to)$ recorded in Remark 3.3. The direct encoding $[\![\varphi]\!] \supset [\![\psi]\!]$ is the natural first thought, but its consequent is in general compound, so it is not a definite formula and cannot be installed in a base by AUGMENT; the elimination form keeps every clause head atomic [6]. The image of the encoding uses only atoms, $\supset$, $\cap$, and $\forall$: the goal-only connective $\cup$ never occurs.

Remark 4.4 (Nomenclature)*.* Definitions of the connectives from $\supset$ and a propositional quantifier go back to Russell, and their proof-theoretic study to Prawitz; the encoding is accordingly the *Russell–Prawitz translation* of the connectives [17, 16], here with the quantifier restricted to atomic instantiation as in the atomic polymorphism literature [2, 3] (see also Remark 4.6). Functional programmers will recognise the shape of a continuation-passing transform — the atom X is a continuation, an arbitrary conclusion to be delivered — and we retain that reading informally, but we do not call the encoding CPS: the term has a precise, and different, meaning as a transformation on source programs.

Remark 4.5. The encoding $[\![\cdot]\!]$ is not one clever translation among many. It is determined, connective by connective, by the support semantics: conjunction is transcribed from its clause directly, and $\bot$, $\vee$, and $\to$ from their elimination behaviour, by replacing the quantifier over atoms with $\forall$ and the quantifier over base extensions with $\supset$. This canonicity suggests an explanation of the coincidence observed in [4] between Mints' clausal translation of IPL and Sandqvist's completeness construction: both are arrived at by writing the elimination rules as clauses over atomic vocabularies — instances of the same Russell–Prawitz transcription, reached from different directions.

Before proving the encoding faithful, it is worth asking how neutral formulae sit within LP. Atoms are the paradigm of a formula that is simultaneously a goal and a definite clause, and neutral formulae extend that class; so one might expect them to behave like atoms in every respect — in particular, to be substitutable for them. They are not, and the reason is instructive.

In full second-order propositional logic a universal quantifier may be instantiated at an arbitrary formula. The quantifier of hH^2 is not of this kind: operationally it is *atomic*. Generic discharges a universal goal by introducing a single fresh atom, and elaboration instantiates a universal clause only at atoms. This restriction is what keeps proof search effective, and it is exactly what separates neutral formulae from atoms.

Remark 4.6 (Failure of compound instantiation). It is tempting to posit the following substitution principle: if $P \vdash_O G$ and the atom p does not occur in P, then $P \vdash_O G[F/p]$ for every neutral F. It fails. Take

$$P = \{\, \forall Y\, (Y \supset q) \,\}, \qquad G = p \supset q,$$

where p, q, and a are distinct atoms.

The atomic instance succeeds. Augment turns $P \vdash_O p \supset q$ into the query $P, p \mathrel{?\!-}_O q$. As p is an atom, $p \supset q$ is an instance of the clause of P and so lies in the elaboration $[P]$; Backchain on it leaves $P, p \mathrel{?\!-}_O p$, which Fact closes.

The compound instance fails. Substituting $F = a \supset a$ for p yields the goal $(a \supset a) \supset q$, and after Augment the query $P, a \supset a \mathrel{?\!-}_O q$. Since the clause of P elaborates at atoms only, the clauses of $[P \cup \{a \supset a\}]$ with head q are precisely those of the form $B \supset q$ with B atomic, so Backchain demands that some atom be derivable. None is: the only heads available are q, which itself requires an atom, and a, which requires a. Hence $P \nvdash_O (a \supset a) \supset q$.

Atomic polymorphism. By Generic, the successful half says precisely that $P \vdash_O \forall X\, (X \supset q)$. So the universal goal is derivable while one of its compound instances is not: the quantifier of hH^2 is genuinely atomic. This is the operational face of the predicativity of atomic polymorphism [2], and it carries a methodological consequence. Theorem 4.9 cannot be proved by a substitution lemma. The arguments below use instead renaming (Lemma 2.9), which replaces atoms by atoms, together with identity and cut for neutral formulae.

Nonetheless, some of the behaviour of atoms does survive, and it is exactly the behaviour proof search relies on. The rule Fact, which closes an atomic goal sitting in the elaborated program, generalises to neutral formulae; so does the admissibility of cut.

Lemma 4.7 (Identity)**.** *Let F be neutral. If $F \in [P]$, then $P \vdash_O F$. In particular, $P, F \vdash_O F$ for every program P.*

Lemma 4.8 (Cut)**.** *Let F be neutral. If $P \vdash_O F$ and $Q, F \vdash_O G$, then $P, Q \vdash_O G$.*

Note that neither statement would even typecheck without Proposition 4.2: in each, F occurs once as a goal and once as a program clause. This is where membership in both syntactic classes earns its keep.

Identity is proved by induction on the size of F, where $F[A/X]$ has the same size as F. Cut is proved by induction on the size of F and, within it, on the height of the derivation of $Q, F \vdash_O G$. Its key step is that every member of $[F]$ is neutral, of size at most that of F, and derivable from P whenever F is, by inversion (Lemma 2.10); renaming (Lemma 2.9) handles the eigenvariable introduced by Generic. The arguments are those of cut-admissibility for uniform proofs [15], adapted to the atomic quantifier. Details are in the extended version [6].

We can now return to the picture from the introduction. A tableau decomposes a formula according to its main connective; goal-directed search decomposes a goal in the same way. The encoding of Definition 4.3 makes the two decompositions line up with a third: the clauses of Figure 1. Each connective of IPL is sent to a neutral formula whose outermost structure determines which rule of hH^2 fires, and that rule re-enacts the support clause for that connective. Conjunction is sent to $\cap$, where And splits the goal. Falsum is sent to $\forall X\, X$, where Generic introduces a fresh atom to be derived. Implication and disjunction are sent to universal formulae, where Generic introduces a fresh atom, Augment installs the continuation clauses just as the support clauses quantify over base extensions, and Backchain consumes them.

Theorem 4.9 (Soundness and Completeness)**.** *For every base $\mathcal{B}$ and IPL formula φ:*

$$\Vdash_{\mathcal{B}} \varphi \qquad \text{if and only if} \qquad \mathcal{B} \vdash_O [\![\varphi]\!].$$

Proof sketch. By simultaneous induction on φ, both directions together. The case of atoms is clause (At). For $\bot$, Generic at an atom fresh for $\mathcal{B}$ gives one direction and inversion the other. For $\wedge$, clause $(\wedge)$, the induction hypothesis, And, and inversion suffice. For $\vee$ and $\to$, left to right: Generic and Augment reduce the encoded goal to proving a fresh atom A from $\mathcal{C} = \mathcal{B} \cup \{D\}$, where D is the continuation clause — $([\![\varphi_1]\!] \supset A) \cap ([\![\varphi_2]\!] \supset A)$ for $\vee$, and $[\![\varphi_1]\!] \cap ([\![\varphi_2]\!] \supset A)$ for $\to$. For $\vee$, the induction hypothesis and Backchain on $[\![\varphi_i]\!] \supset A$ give $\varphi_i \Vdash_{\mathcal{C}} A$ for each i, and clause $(\vee)$ delivers $\Vdash_{\mathcal{C}} A$. For $\to$, identity and the induction hypothesis give $\Vdash_{\mathcal{C}} \varphi_1$, clauses $(\to)$ and

(Inf) give $\Vdash_{\mathcal{C}} \varphi_2$, and the induction hypothesis and BACKCHAIN on $[\![\varphi_2]\!] \supset A$ give $\mathcal{C} \vdash_O A$. Right to left: given the extension $\mathcal{C}$ and atom A supplied by clause $(\vee)$, or by the elimination form of $(\to)$ (Remark 3.3), monotonicity and inversion unwind GENERIC and AUGMENT to give $\mathcal{C}, D \vdash_O A$; identity, the induction hypothesis, AUGMENT, and AND give $\mathcal{C} \vdash_O D$; and cut gives $\mathcal{C} \vdash_O A$. Since the induction hypothesis is used in both directions, the two are proved together. Compound instantiation is never used, in keeping with Remark 4.6: the fresh atom of GENERIC is moved to the atom supplied by the support clause by inversion, which is to say by renaming. The full proof is [6, Theorem 4.2]. □

Corollary 4.10. *φ is valid in IPL if and only if $\varnothing \vdash_O [\![\varphi]\!]$.*

Example 4.11. The formula $(p \wedge q) \to p$ encodes as $\forall X\,(((p \cap q) \cap (p \supset X)) \supset X)$, since $[\![p \wedge q]\!] = p \cap q$. The query succeeds from the empty program:

$$
\begin{array}{l}
\varnothing \mathrel{?\!-_O} [\![(p \wedge q) \to p]\!] \\
\qquad \downarrow \ \textsc{Generic}, A \text{ fresh} \\
\varnothing \mathrel{?\!-_O} ((p \cap q) \cap (p \supset A)) \supset A \\
\qquad \downarrow \ \textsc{Augment}, D_0 = (p \cap q) \cap (p \supset A) \\
D_0 \mathrel{?\!-_O} A \\
\qquad \downarrow \ \textsc{Backchain} \text{ on } p \supset A \in [D_0] \\
D_0 \mathrel{?\!-_O} p \\
\qquad \downarrow \ \textsc{Fact}, p \in [D_0] \\
\qquad \checkmark
\end{array}
$$

The fresh atom A is the conclusion to be delivered. Backchaining passes control to the continuation $p \supset A$, and the conjunct p is then found in the elaborated program: the elimination rule for conjunction is executed as a projection.

Example 4.12 (Peirce's formula, executed)**.** Write $a = [\![p \to q]\!] = \forall Y\,((p \cap (q \supset Y)) \supset Y)$ and $h = [\![(p \to q) \to p]\!] = \forall Z\,((a \cap (p \supset Z)) \supset Z)$, so that $[\![((p \to q) \to p) \to p]\!] = \forall X\,((h \cap (p \supset X)) \supset X)$. The search from the empty program runs as follows:

$$
\begin{array}{l}
\varnothing \mathrel{?\!-_O} [\![((p \to q) \to p) \to p]\!] \\
\qquad\qquad \big\downarrow \ \textsc{Generic}, \textsc{Augment}, D = h \cap (p \supset A) \\
\qquad\quad D \mathrel{?\!-_O} A \\
\qquad\qquad \big\downarrow \ \textsc{Backchain} \text{ on } (a \cap (p \supset A)) \supset A \in [D], \textsc{And} \\
\qquad\quad D \mathrel{?\!-_O} a \\
\qquad\qquad \big\downarrow \ \textsc{Generic}, \textsc{Augment} \\
\quad D, p \cap (q \supset B) \mathrel{?\!-_O} B \\
\qquad\qquad \big\downarrow \ \textsc{Backchain} \text{ on } q \supset B \\
\quad D, p \cap (q \supset B) \mathrel{?\!-_O} q \\
\qquad\qquad \big\downarrow \ \textsc{Backchain} \text{ on } (a \cap (p \supset q)) \supset q \in [D], \textsc{And} \\
\quad D, p \cap (q \supset B) \mathrel{?\!-_O} a \\
\\
\qquad\qquad \vdots
\end{array}
$$

The alternatives lead to the same place. At the second line, Backchain on $p \supset A$ demands p, and the only clauses with head p are instances of h, which demand a; at the fourth, Backchain on the instance of h at B demands a directly. So every branch returns to the goal $a = [\![p \to q]\!]$, and establishing a requires q, whose only clauses are instances of h demanding a again. Any derivation would therefore contain a strictly shorter one of the same kind, so there is none: $\varnothing \nvdash_O [\![((p \to q) \to p) \to p]\!]$, which by Corollary 4.10 witnesses the intuitionistic invalidity of Peirce's formula — and, by Theorem 4.9, the failure of support at the empty base promised in Example 3.7. The formula is classically valid; what the failed search records is that the hypothesis $(p \to q) \to p$, installed as a genuinely hypothetical commitment, yields p only given $p \to q$, and nothing yields q.

This operationalizes the B-eS for IPL. We motivated it as the analogue of tableaux for P-tS, and before closing the section that motivation merits some final remarks. The analogy breaks in three places, but each break is the M-tS/P-tS difference showing through.

First, polarity. Every node of a tableau carries a label t or f, recording the status the formula is to have in the model under construction; the search is a refutation, begun by supposing φ false. A query $P \mathrel{?\!-_O} G$ carries no such label. There is nothing to suppose false, because support is not truth-in-a-structure and its negation is not a constraint one can propagate. The search is direct: it looks for the commitments that would confer the meaning, not for a structure witnessing that they fail.

Second, and consequently, the two searches read their branches dually. A tableau succeeds by closing every branch and refutes by leaving one open, the open branch being the countermodel it has built. Goal-directed search succeeds on a *single* branch and fails only when no branch can close; and when it fails it hands back nothing. Example 4.12 produces no object witnessing

the failure of support — only the observation that every branch returns to the same unmet demand. In M-tS a refutation must exhibit a structure, since validity is truth in all of them. Here the base is given in advance, and the quantification over extensions has been absorbed into the support clauses themselves. There is no countermodel to build because there is nothing left to construct.

Third, the program grows. AUGMENT installs a hypothesis and the search continues against a larger program, so the object consulted at the leaves is not fixed in advance. A tableau branch also grows, but it grows by decomposition: what appears is always a subformula of what was there. This is what Example 4.12 turns on. Peirce's formula fails not because some branch stays open, but because the hypothesis $(p \to q) \to p$, once installed as a genuine commitment, licenses nothing that can be discharged.

None of this is a defect of the analogy; it is the analogy doing its work. We should not expect the operationalization of a use-based semantics to look like the operationalization of a denotational one, and the respects in which the two differ are exactly the respects in which B-eS differs from M-tS. What survives is the structural claim with which we began: a rule determined by the main connective, branching where the connective demands a choice, and closure when the decomposition reaches something the ambient object supplies outright. That claim survives intact, and Corollary 4.10 shows it survives with something to spare. Validity in M-tS is a quantification over structures, which a tableau discharges by constructing the one candidate that matters. Validity here is a single query against the empty program. The quantifier that a tableau must simulate has already been paid for by the semantics.

5. Extensions and classical logic

We have established the operationalization only for the B-eS of IPL, but such semantics now exist for a range of logics: first-order logic [5], linear logic [7], the logic of bunched implications [8], and modal logics [1]. In each case the support clauses have the shape that drove the present construction — a quantification over base extensions, discharged at atoms. The programme this paper suggests is accordingly: for each such logic, find the LP language whose operational steps internalise those quantifiers. The expected pairings are informative — first-order B-eS with first-order hereditary Harrop programs [15, 10], IMLL with the linear hereditary Harrop programs of Lolli [11] — and the first-order case is the natural first test, both sides of the correspondence being already in place.

In every case the challenge is one of expressiveness, and classical logic shows what is at stake. Theorem 3.5 identifies B-eS over H^2 with classical logic, so the target language is fixed for us; but the encoding of Definition 4.3 cannot live there. Its work is done by $\forall X\,(\cdots \supset X)$ — a continuation abstracted over, then supplied — and while H^2 has the quantifier, it has no

hypothetical goals, so neither the goal $\forall X\,(\cdots \supset X)$ nor its body is available there. The pairing that Theorem 3.5 hands us is therefore the one pairing for which we have no encoding, and the machinery that made IPL work is exactly what H^2 withholds.

Classical *validity*, on the other hand, is already within reach. By Glivenko's theorem, φ is classically valid if and only if $\neg\neg\varphi$ is intuitionistically valid, so by Corollary 4.10 it suffices to run

$$\varnothing \mathrel{?\!-_O} [\![\neg\neg\varphi]\!] \qquad \text{where} \qquad [\![\neg\varphi]\!] = \forall Y\,\big(([\![\varphi]\!] \cap ((\forall X\, X) \supset Y)) \supset Y\big).$$

But this is a reduction, not an operationalization. It decides classical validity by running an intuitionistic query, and so tells us nothing about how a classical base confers meaning; the double negation is a device of the translation rather than anything the search enacts. It also does not travel: Glivenko's theorem is propositional, so the route closes precisely where the programme above says to go next.

A native treatment would have to make the classical structure operational rather than translate it away, and the continuation reading suggests where to look. In the typed λ-calculus, continuations at classical type are control operators [9], which points to an operational semantics with a control facility — a restart, or a backtracking-to-continuation rule — layered over Horn programs. This would keep the program language that Theorem 3.5 demands while recovering operationally the expressive power the encoding obtained from $\forall$. Whether the resulting notion of success matches support over a Horn base is the obvious question, and we leave it open.

References

[1] Timo Eckhardt and David J Pym. Base-extension Semantics for Modal Logic. *Logic Journal of the IGPL*, 33, 2024. doi:10.1093/jigpal/jzae004.

[2] Fernando Ferreira. Comments on Predicative Logic. *Journal of Philosophical Logic*, 35(1):1–8, 2006.

[3] Fernando Ferreira and Gilda Ferreira. Atomic Polymorphism. *The Journal of Symbolic Logic*, 78:260–274, 2013.

[4] Alexander V. Gheorghiu. On an Inferential Semantics for Intuitionistic Sentential Logic. *Journal of Logic and Computation*, 35(4), 2025.

[5] Alexander V. Gheorghiu. Proof-theoretic Semantics for First-order Logic. *Logic Journal of the IGPL*, 33(5), 2025.

[6] Alexander V. Gheorghiu. Support is search. arXiv:2603.13018, 2026.

[7] Alexander V. Gheorghiu, Tao Gu, and David J. Pym. Proof-theoretic Semantics for Intuitionistic Multiplicative Linear Logic. In Revantha Ramanayake and Josef Urban, editors, *Automated Reasoning with Analytic Tableaux and Related Methods — TABLEAUX*, pages 367–385. Springer, 2023.

[8] Alexander V. Gheorghiu, Tao Gu, and David J. Pym. Proof-theoretic Semantics for the Logic of Bunched Implications. *Studia Logica*, 2025.

[9] Timothy G. Griffin. A formulae-as-types notion of control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '90)*, pages 47–58. ACM, 1990.

[10] James Harland. *On Hereditary Harrop Formulae as a Basis for Logic Programming*. PhD thesis, University of Edinburgh, 1991.

[11] Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994.
[12] Robert Kowalski. *Logic for Problem Solving*. Number 7 in Artificial Intelligence Series. North-Holland, New York, 1979.
[13] Dale Miller. A Logical Analysis of Modules in Logic Programming. *The Journal of Logic Programming*, 6(1-2):79–108, 1989.
[14] Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, Cambridge, 2012.
[15] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied logic*, 51(1-2):125–157, 1991.
[16] Dag Prawitz. *Natural Deduction: A Proof-theoretical Study*. Dover Publications, 2006 [1965].
[17] Bertrand Russell. *The Principles of Mathematics*. Cambridge University Press, Cambridge, 1903.
[18] Tor Sandqvist. Classical Logic without Bivalence. *Analysis*, 69(2):211–218, 2009.
[19] Tor Sandqvist. Base-extension Semantics for Intuitionistic Sentential Logic. *Logic Journal of the IGPL*, 23(5):719–731, 2015.
[20] Peter Schroeder-Heister. Proof-Theoretic Semantics. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2018 edition, 2018.

University College London
Email address: a.v.gheorghiu@ucl.ac.uk