DEMONIC DYNAMIC LOGIC PROGRAMMING

FLOPS 2026
TSUKUBA
MAY 26–28, 2026

Rose BOHRER

Permanent Researcher (研究員)

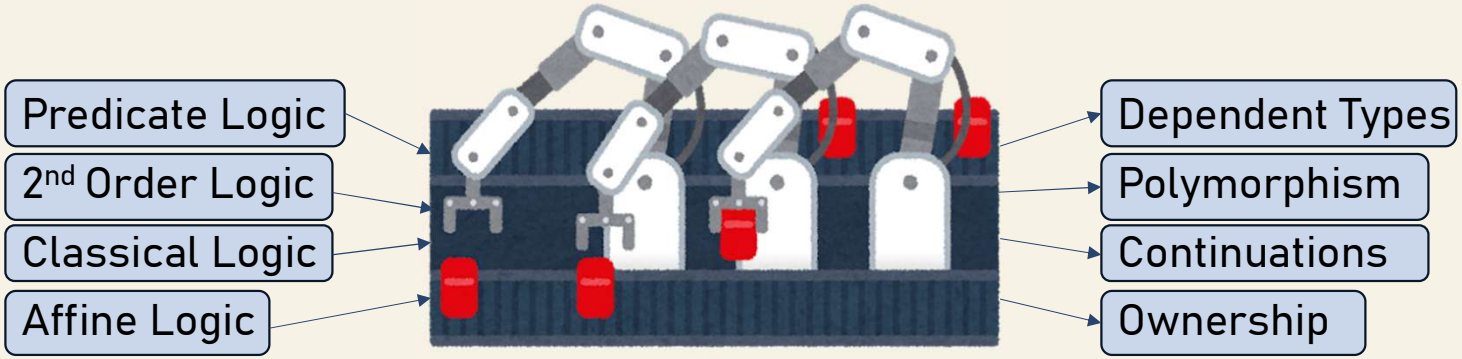
National Institute of Advanced Industrial Science and Technology (AIST)

Koto Ward, Tokyo, JP

INTRODUCTION

GOOD LOGIC IDEAS ARE GOOD LANGUAGE IDEAS

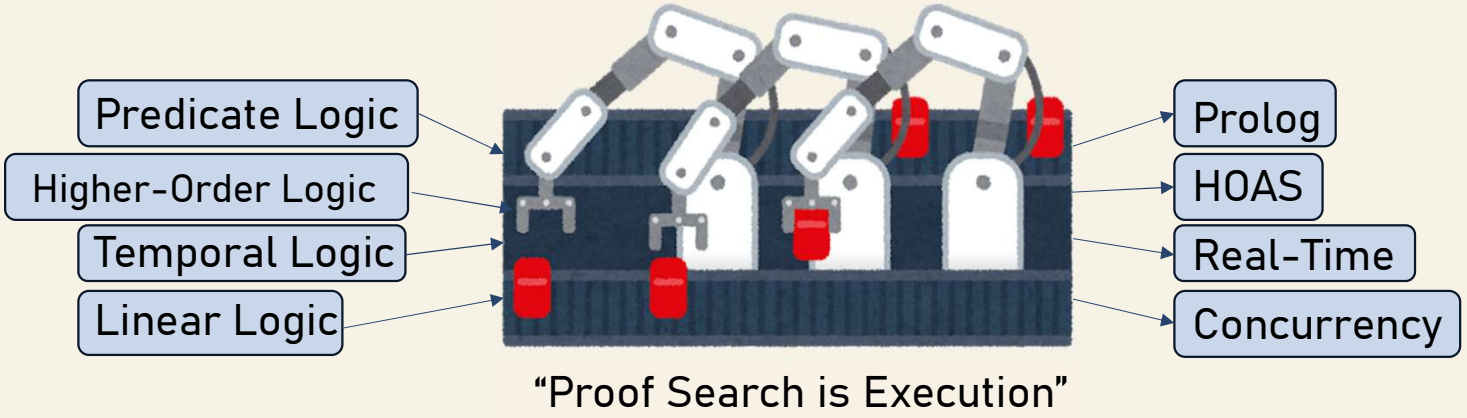
The Curry-Howard Correspondence is like an assembly line that takes in new ideas in logic and creates new type systems



"Propositions Are Types"

GOOD LOGIC IDEAS ARE GOOD LOGIC LANGUAGE IDEAS

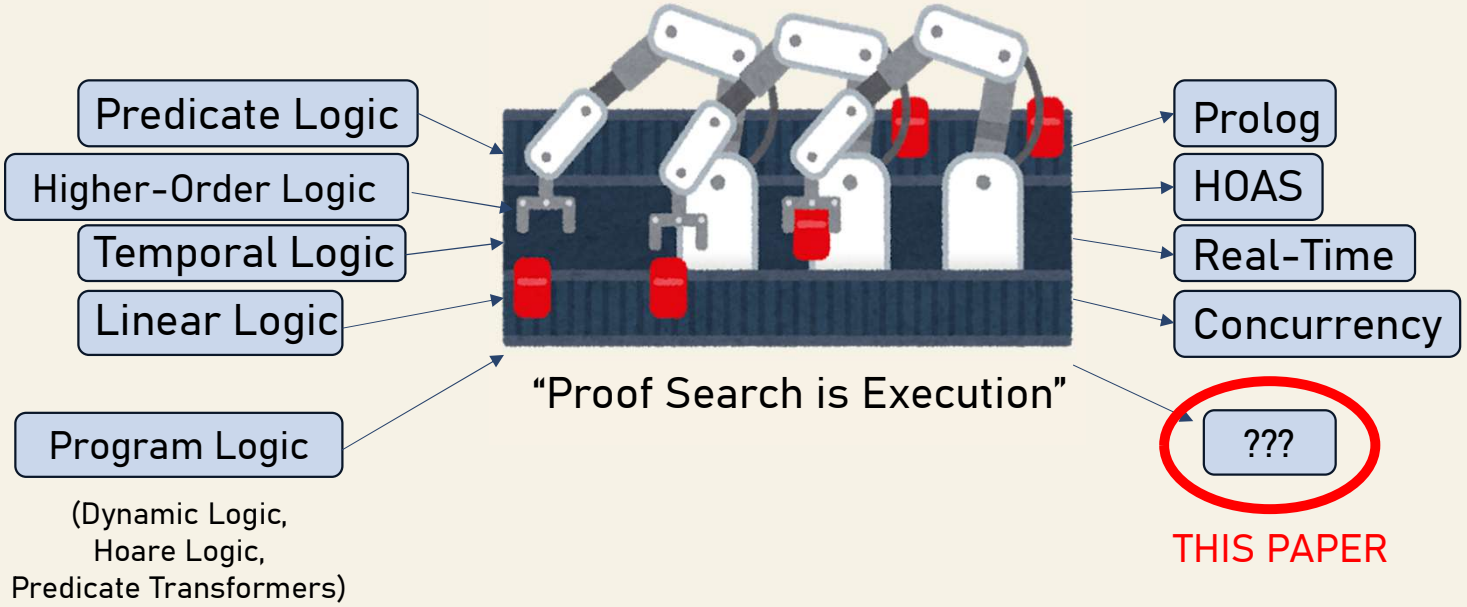
The Programming-as-Proof-Search Correspondence is like an assembly line that takes in new ideas in logic and creates new logic programming languages



"Proof Search is Execution"

PROGRAM LOGIC SHOULD BE A GOOD LOGIC LANGUAGE IDEA

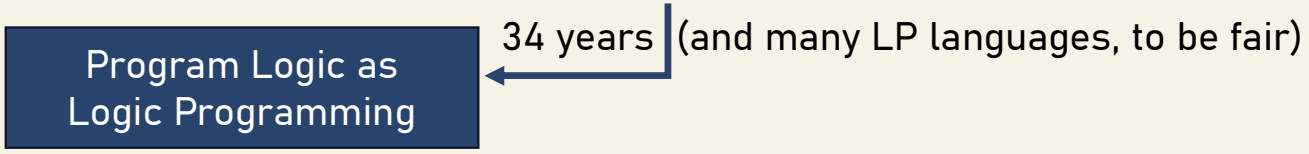
The Programming-as-Proof-Search Correspondence is like an assembly line that takes in new ideas in logic and creates new logic programming languages



HOW IS THIS PROBLEM STILL OPEN?

Timeline of Logic Programming Languages

- Prolog 1972
- λProlog (HOAS) 1987
- Tokio (Temporal) 1985
- Lolli (Linear) 1992



(and many LP languages, to be fair)

Program Logic as Logic Programming Can Only Succeed When

- Computational Content of Proofs is Clear (Bohrer 2020)
- Compelling Applications are Identified (here, reactive synthesis, largely >=2010s)
- Proof Search is Automated, But Sufficiently Expressive (**This Paper**, 2026)

PROGRAM LOGIC AS LOGIC PROGRAMMING PROMISES SCALABLE REACTIVE SYNTHESIS

Proof Content: Proofs Resolve Nondeterminism Present in Programs
Resolving Existential Nondeterminism is Synthesis

Program Logic as Logic Programming in Three Parts

1. **Demonic Dynamic Logic Programming (This Paper)**
   $[\alpha]\phi$ for all behaviors of program α, postcondition ϕ has proof
2. Propositional Dynamic Logic Programming
   $\langle\alpha\rangle\phi$ select behaviors of program α, such that postcondition ϕ has proof
3. Game Logic Programming
   $[\alpha; \beta^d]\phi$ for all behaviors of α, select response behavior of β that has proof

Applied Impact: Provide a Language-Based Framework for Reactive Synthesis,
Improving Synthesis Modularity and Thus Scalability

SYNTAX AND BACKGROUND

BACKGROUND: DYNAMIC LOGIC (DL)

$$\alpha, \beta ::= c \mid \alpha \cup \beta \mid \alpha; \beta \mid ?\phi \mid \alpha^* \qquad \phi, \psi ::= [\alpha]\phi \mid \langle\alpha\rangle\phi \mid \cdots$$

Const. · Choice · Sequence · Assert · Repeat · All Runs · Some Run

DL Subsumes Hoare Logic, Predicate Transformers

$$\{\phi\}\alpha\{\psi\} \Leftrightarrow (\phi \to wlp(\alpha,\psi)) \Leftrightarrow (\phi \to [\alpha]\psi)$$
$$[\phi]\alpha[\psi] \Leftrightarrow (\phi \to wp(\alpha,\psi)) \Leftrightarrow (\phi \to \langle\alpha\rangle\psi)$$

Hoare · Pred. Trans. · DL

Program Semantics

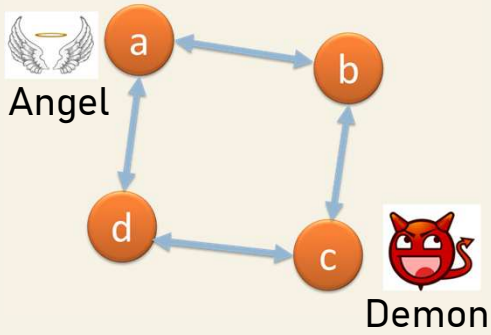
$$[|\alpha \cup \beta|] = [|\alpha|] \cup [|\beta|]$$
$$[|\alpha;\beta|] = [|\alpha|] \circ [|\beta|]$$
$$[|?\phi|] = \{(\omega,\omega) \mid \omega \in [|\phi|]\}$$
$$[|\alpha^*|] = \bigcup_{i\in\mathbb{N}} [|\alpha|]^i$$

Formula Semantics

$$[|[\alpha]\phi|] = \{\omega | \nu \in [|\phi|], all(\omega,\nu) \in [|\alpha|]\}$$
$$[|\langle\alpha\rangle\phi|] = \{\omega | \nu \in [|\phi|], some\ (\omega,\nu) \in [|\alpha|]\}$$
$$[|\phi \vee \psi|] = [|\phi|] \cup [|\psi|]$$
$$[|\phi \wedge \psi|] = [|\phi|] \cap [|\psi|]$$
$$[|\neg\phi|] = [|\phi|]^{\mathbb{C}}$$

EX: PURSUIT-EVASION ON GRAPH (PEG)

Angel wins if Demon + Angel at same vertex
Formula xy = Angel at x, Demon at y



Define Demon's left, right move state-by-state

```
ac -> [l]ab. bd -> [l]bc. ca -> [l]cd. db -> [l]da.
ac -> [r]ad. bd -> [r]ba. ca -> [r]cb. db -> [r]dc.
demon-turn ::= l U r.
```

Demon Turn: go left or right

Angel's moves are similar

```
ab -> [la]db. bc -> [la]ac. cd -> [la]bd. da -> [la]ca.
ad -> [ra]bd. ba -> [ra]ca. cb -> [ra]db. dc -> [ra]ac.
angel-turn ::= (?ab;la U ?bc;la U ?cd;la U ?da;la)
               U (?ad;ra U ?ba;ra U ?cb;ra U ?dc;ra).
```

Angel Turn: Test Demon Position, React to It

```
?- ac -> [(demon-turn;angel-turn)*@inv(ac|bd|ca|db)] (ac|bd|ca|db).
```

Query: | For all Demon behaviors, all turns | Invariant Holds | Postcondition follows

DDLP implementation of PEG on 4-vertex cycle graph.

DDLP SYNTAX NEEDS RESTRICTIONS

$$\text{LP} \rightarrow \text{decl}^{+}\ \text{query} \quad \text{decl} \rightarrow (\text{c} ::= \alpha) \mid (\text{G} \rightarrow \text{p}) \mid (\text{G} \rightarrow [\text{c}]\text{p}) \quad \text{query} \rightarrow (?- \text{G})$$

CLAUSE — Formula, Program

$$D ::= true \mid p \mid G \rightarrow p \mid \text{D}_1 \wedge \text{D}_2 \mid [\delta]p$$
$$\delta ::= c \mid ?\,\text{G} \mid \delta_1 \cup \delta_2$$

No Disjunction Or Repetition

GOAL — Formula, Program

$$G ::= true \mid p \mid \text{G}_1 \wedge \text{G}_2 \mid \text{G}_1 \vee \text{G}_2 \mid \text{D} \rightarrow \text{G} \mid [\gamma]\text{G}$$
$$\gamma ::= \text{c} \mid ?\,\text{D} \mid \gamma_1 \cup \gamma_2 \mid \gamma^{*}@\text{inv(J)} \mid \gamma_1 ; \gamma_2$$

Disjunction, Repetition OK

INVARIANT — Formula, Program

$$J ::= true \mid p \mid J \rightarrow p \mid \text{J}_1 \wedge \text{J}_2 \mid [\iota]p$$
$$\iota ::= \text{c} \mid ?\,\text{J} \mid \iota_1 \cup \iota_2 \mid \iota^{*}@\text{inv(J)}$$

Both Clause and Goal

SEMANTICS

PROOF RULES - PROPOSITIONAL

$$\textsc{Init}\ \frac{\phi \in \Gamma \cup \Sigma}{\Gamma \vdash_\Sigma \phi} \qquad \textsc{Expl}\ \frac{c ::= \alpha \in \Sigma \qquad \Gamma(\alpha) \vdash \phi(\alpha)}{\Gamma(c) \vdash \phi(c)} \qquad \top\ \frac{*}{\Gamma \vdash \mathsf{true}} \qquad \wedge R\ \frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi}$$

$$\wedge L\ \frac{\Gamma, \psi_1, \psi_2 \vdash \phi}{\Gamma, \psi_1 \wedge \psi_2 \vdash \phi} \qquad \vee R1\ \frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad \vee R2\ \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi} \qquad \to L\ \frac{\Gamma \vdash \psi_1 \qquad \Gamma, \psi_2 \vdash \phi}{\Gamma, \psi_1 \to \psi_2 \vdash \phi} \qquad \to R\ \frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}$$

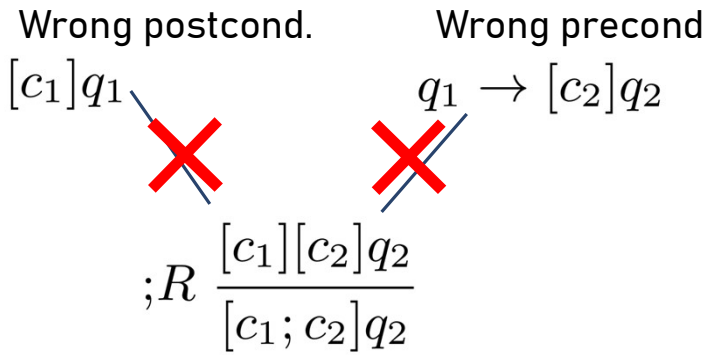
PROOF RULES - PROGRAMS

$$;R\ \frac{\Gamma \vdash [\alpha][\beta]\phi}{\Gamma \vdash [\alpha;\beta]\phi} \qquad \cup R\ \frac{\Gamma \vdash [\alpha]\phi \wedge [\beta]\phi}{\Gamma \vdash [\alpha \cup \beta]\phi} \qquad ?R\ \frac{\Gamma \vdash \psi \rightarrow \phi}{\Gamma \vdash [?\psi]\phi}$$

$$*R\ \frac{\Gamma \vdash \bigvee_i \psi_i \qquad \psi_1 \vdash [\alpha]\bigvee_j \psi_j \cdots \qquad \psi_n \vdash [\alpha]\bigvee_j \psi_j \qquad \psi_1 \vdash \phi \cdots \qquad \psi_n \vdash \phi}{\Gamma \vdash [\alpha^*@inv(\bigvee_i \psi_i)]\phi}$$

NAÏVE RULES GET STUCK

- The given rules do not support nested predicate calls
- Example:
  - Given clauses $[c_1]q_1,\ \ q_1 \rightarrow [c_2]q_2$
  - Query $[c_1\,;c_2]q_2$ should succeed
- Monotonicity rules enable nested calls



MONOTONICITY FIXES PROOF

- The given rules do not support nested predicate calls
- Example:
  - Given clauses $[c_1]q_1,\ \ q_1 \rightarrow [c_2]q_2$
  - Query $[c_1; c_2]q_2$ should succeed
- Monotonicity rules enable nested calls

$$\mathrm{M}\ \frac{[c_1]q_1 \qquad q_1 \rightarrow [c_2]q_2}{[c_1][c_2]q_2} \qquad ;R\ \frac{[c_1][c_2]q_2}{[c_1; c_2]q_2}$$

$$\mathrm{M}\ \frac{\phi \vdash \psi \qquad \Gamma \vdash [\alpha]\phi}{\Gamma \vdash [\alpha]\psi} \qquad \mathrm{K}\ \frac{\Gamma \vdash [\alpha]\phi \qquad \Gamma \vdash [\alpha](\phi \rightarrow \psi)}{\Gamma \vdash [\alpha]\psi} \qquad \mathrm{G}\ \frac{\phi}{[\alpha]\phi}$$

OPERATIONAL SEMANTICS (EX.)

$$\textsc{OpHyp}\ \frac{G \in A \cup \Sigma}{A \to G \Longrightarrow_{\Sigma} \mathsf{true}}$$

$$\textsc{OpPred}\ \frac{(\boldsymbol{A} \to c) \in \Sigma}{A \to M(c) \Longrightarrow_{\Sigma} \wedge_i (A \to M(\boldsymbol{A}_i))}$$

THEORY

SOUNDNESS RESULTS

Theorem 1 (Soundness w.r.t. Proof-Theoretic Semantics). *Let G be a goal formula. If judgement $G \Longrightarrow_\Sigma \mathsf{true}$ holds in the operational semantics then $\cdot \vdash_\Sigma G$ holds in the proof-theoretic dynamics. More generally, let G, G' be goal formulas such that $G \Longrightarrow_\Sigma G'$. If premise $\cdot \vdash_\Sigma G'$ is derivable in proof-theoretic semantics, the conclusion $\cdot \vdash_\Sigma G$ is derivable in proof-theoretic semantics.*

Proof. By induction on the derivation of $G \Rightarrow_\Sigma G'$.

Theorem 2 (Soundness w.r.t. PDL). *The proof-theoretic semantics is sound w.r.t. PDL semantics, i.e., every sequent provable in DDLP is true in PDL.*

Proof. By induction on the derivation.

SOUNDNESS RESULTS

Theorem 1 (Soundness w.r.t. Proof-Theoretic S
goal formula. If judgement $G \Longrightarrow_\Sigma \mathsf{true}$ holds in the op
$\cdot \vdash_\Sigma G$ holds in the proof-theoretic dynamics. More ge
formulas such that $G \Longrightarrow_\Sigma G'$. If premise $\cdot \vdash_\Sigma G'$ is de
semantics, the conclusion $\cdot \vdash_\Sigma G$ is derivable in proof-t

Proof. By induction on the derivation of $G \Rightarrow_\Sigma G$

Theorem 2 (Soundness w.r.t. PDL). *The proof-the*
w.r.t. PDL semantics, i.e., every sequent provable in D

Proof. By induction on the derivation.

Lemma 1 (Invariant formulas and programs). *We prove these claims:*

- *Every invariant formula J is clause (D).*
- *Every invariant formula J is goal (G).*
- *Every invariant program ι is clause (δ).*
- *Every invariant program ι is goal (γ).*

Proof. By simultaneous induction on the structures of J and ι.

Lemma 2 (Admissible Rules). *Monotonicity, regularity, their iterated forms, and iterated rule K are all admissible in the proof-theoretic semantics. Rule G is admissible under the added assumption that its conclusion is a goal formula with definitions c expanded.*

UNIFORMITY

Theorem 3 (Uniformity). *Let Γ be clause, G be goal, and Σ a signature. Assume without loss of generality that explicitly-defined symbols are expanded.*

If $\Gamma \vdash_\Sigma G$ has a proof using only the core fragment of the proof-theoretic semantics Figure 5, then $\Gamma \vdash_\Sigma G$ has a uniform proof.

1. Either T is an instance of Init or:
2. If G is $G_1 \vee G_2$ then T_1 is a proof of $A \to G_1$ or $A \to G_2$
3. If G is $G_1 \wedge G_2$ then T_1 is a proof of $A \to G_1$ and T_2 is a proof of $A \to G_2$.
4. If G is $\forall c\ G'$ then T_1 is a proof of $A \to [c/c']G$ for some fresh parameter c'.
5. If G is $A' \to G'$ then T_1 is a proof of $A \wedge A' \to G'$
6. If G is $[\alpha;\beta]G$ then T_1 is a proof of $[\alpha][\beta]G$
7. If G is $[?D]G$ then T_1 is a proof of $D \to G$
8. If G is $[\alpha \cup \beta]G$ then T_1 is a proof of $[\alpha]G \wedge [\beta]G$
9. If G is $[\alpha^*@inv(\wedge_i J_i)]G$, let there be n invariant branches J_i, then there are $2n+1$ premises. Specifically T_1 proves $\bigvee_i J_i$, for each $i \in [1,n]$, have T_{1+i} proves $J_i \vdash [\alpha]\bigvee_j J_i$, and for each $i \in [1,n]$, have T_{1+n+i} proves $J_i \vdash P$.

FUTURE STEPS

Implementation (not claimed as a contribution in paper) :

- Interpreter implemented in ~2600 lines of Rust
- Solves example problem in <1 second
- Libraries provided for PEG verification and FSM verification

Future Work:

- Support all of PDL, Game Logic
- Expand language implementation into synthesis tool
- Prove correctness of code synthesis
- Apply to game synthesis and verification of state machines
  - Explore educational applications of state machine verification.

SOURCES

https://www.irasutoya.com/2015/09/blog-post_504.html

OPERATIONAL SEMANTICS (2)

$$\textsc{Op}?\ \frac{*}{A \to M([?A']G) \Longrightarrow A \to M(A' \to G)} \qquad \textsc{Op};\ \frac{*}{A \to M([\alpha;\beta]G) \Longrightarrow A \to M([\alpha][\beta]G)}$$

$$\textsc{Op}\cup\ \frac{*}{A \to M([\alpha \cup \beta]G) \Longrightarrow A \to M([\alpha]G \wedge [\beta]G)}$$

$$\textsc{Op}*\ \frac{*}{A \to [\alpha^* @ inv(\bigvee_i J_i)]G \Longrightarrow (A \to \bigvee_i J_i) \wedge \bigwedge_i (J_i \to [\alpha] \bigvee_j J_j) \wedge \bigwedge_i (J_i \to G)}$$

OPERATIONAL SEMANTICS (3)

$$\textsc{Op}?L \quad \frac{*}{A \wedge [?G_1]p \to G \Longrightarrow A \wedge (G_1 \to p) \to G}$$

$$\textsc{Op}\to L \quad \frac{*}{A \wedge (G_1 \to p) \to G \Longrightarrow (A \to G_1) \wedge (A \wedge p \to G)}$$

$$\textsc{Op}\wedge L \quad \frac{*}{A \wedge (A_1 \wedge A_2) \to G \Longrightarrow A \wedge A_1 \wedge A_2 \to G}$$

$$\textsc{Op}\cup L \quad \frac{*}{A \wedge [\alpha \cup \beta]p \to G \Longrightarrow A \wedge [\alpha]p \wedge [\beta]p \to G}$$