The Logical Basis of Evaluation Order & Pattern-Matching

Noam Zeilberger
Thesis Defense
April 17, 2009

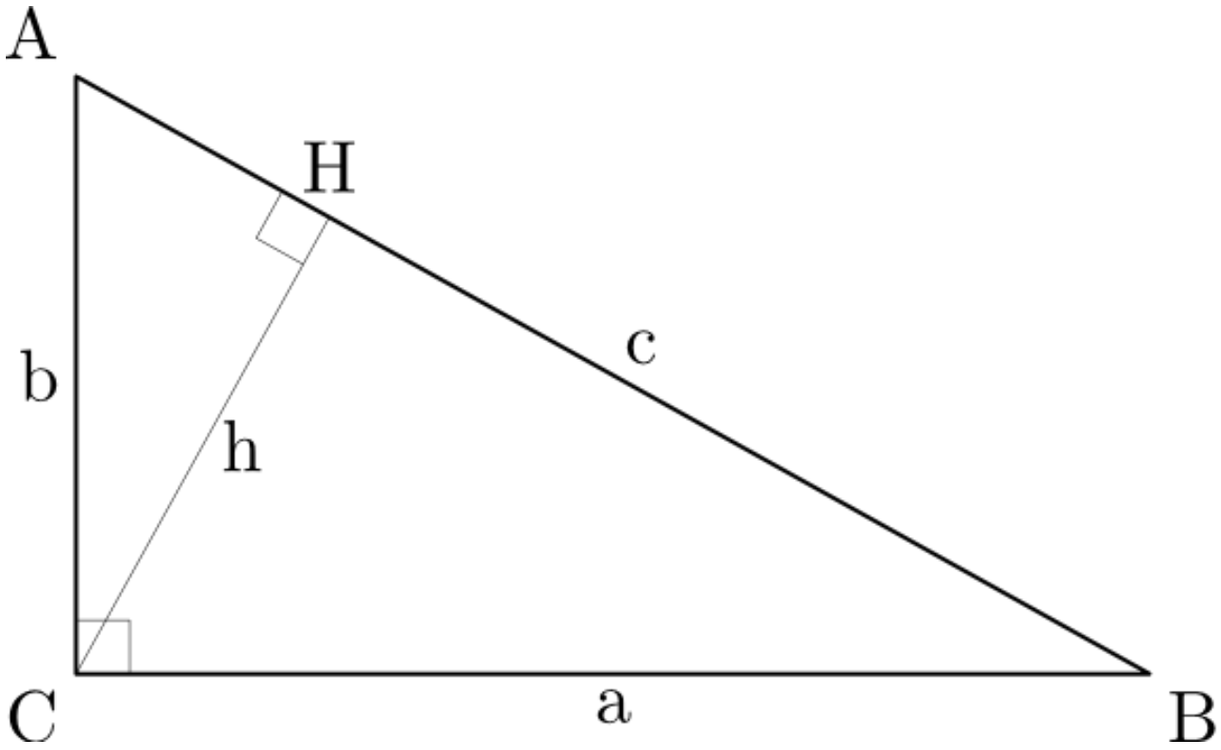
Frank Pfenning (chair)
Peter Lee
Robert Harper
Paul-André Melliès (Paris VII)

A remarkable analogy

Proving is like programming

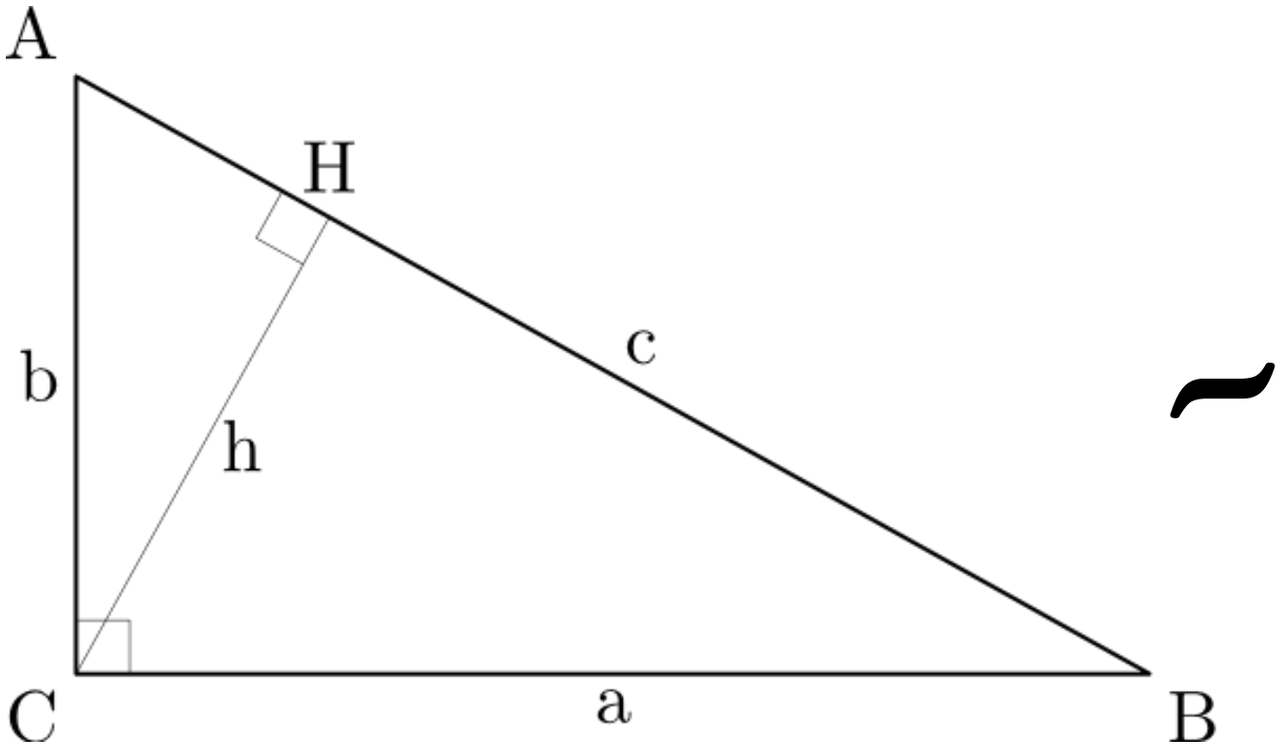
A remarkable analogy

Proving is like programming



A remarkable analogy

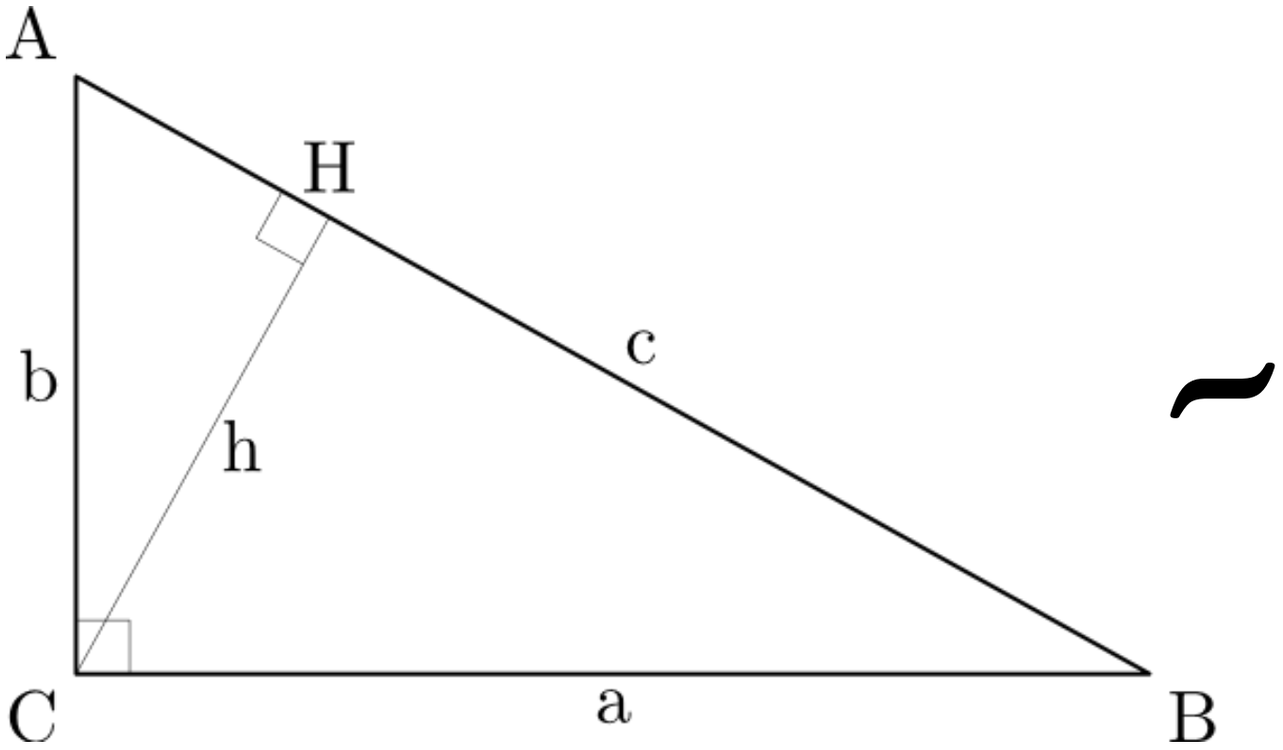
Proving is like programming



~

A remarkable analogy

Proving is like programming



~

```
'ExplodeGorilla:
'  Causes gorilla explosion when a c
'Parameters:
'  X#, Y# - shot location
FUNCTION ExplodeGorilla (x#, y#)
  YAdj = Scl(12)
  XAdj = Scl(5)
  SclX# = ScrWidth / 320
  SclY# = ScrHeight / 200
  IF x# < ScrWidth / 2 THEN PlayerHi
  PLAY "MBO0L16EFGEFDC"

  FOR i = 1 TO 8 * SclX#
    CIRCLE (GorillaX(PlayerHit) + 3
SclY# + YAdj), i, ExplosionColor, ,
    LINE (GorillaX(PlayerHit) + 7 *
(GorillaX(PlayerHit), GorillaY(Playe
  NEXT i
  FOR i = 1 TO 16 * SclX#
```

A remarkable ~~analogy~~ isomorphism

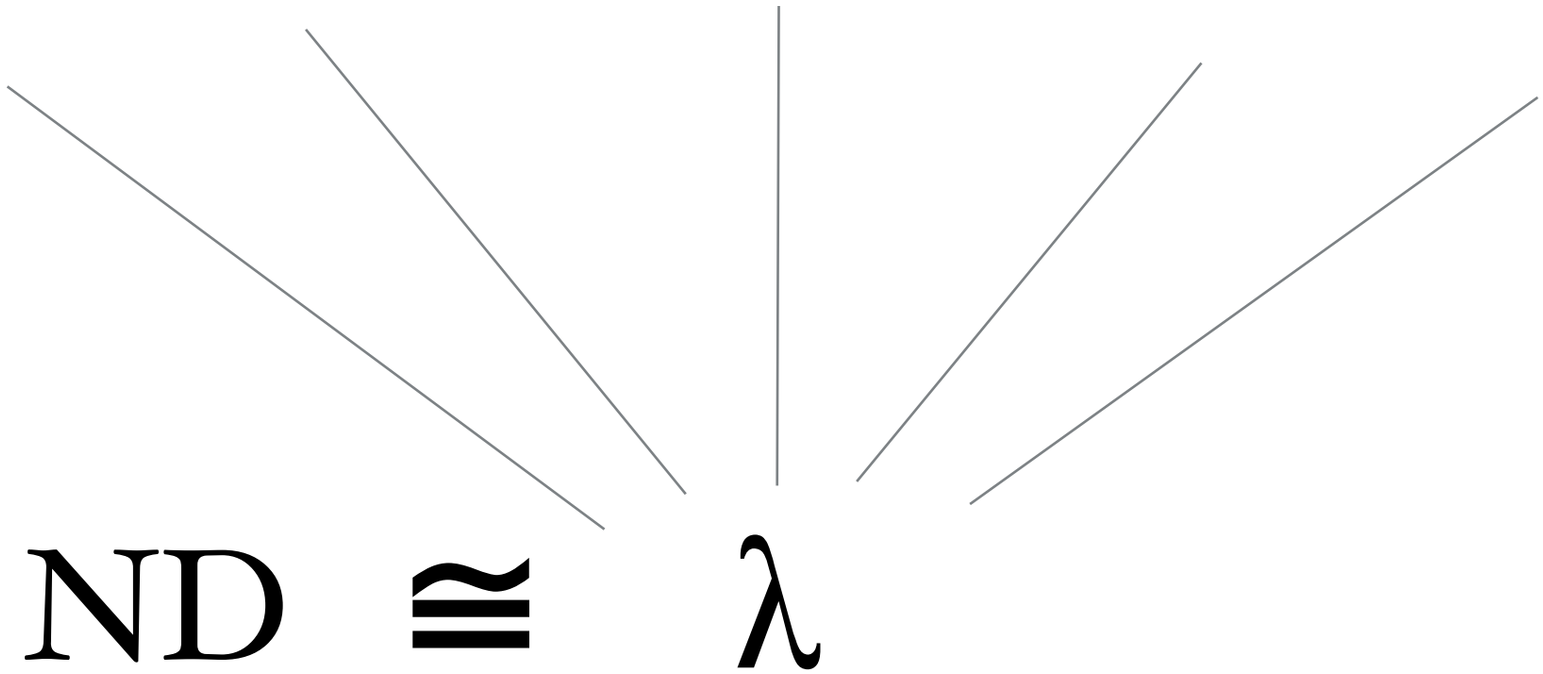
Natural Deduction

$\approx$

Lambda Calculus

The foundation of FP

... Lisp ML Haskell ...



A shaky foundation?

> Purely applicative languages are **often said to be based on a logical system called the lambda calculus,** or even to be "syntactically sugared" versions of the lambda calculus.... **However,** as we will see, although an unsugared applicative language is syntactically equivalent to the lambda calculus, **there is a subtle semantic difference.** Essentially, the "real" lambda calculus implies a different "order of application"...than most applicative programming languages.
>
> —John Reynolds (1972)

Evaluation order

$$(2 + 3) * (5 - 7)$$

Evaluation order

(2 + 3) * (5 – 7)

Evaluation order

5 * (5 – 7)

Evaluation order

$$5 * (5 - 7)$$

Evaluation order

5 * -2

Evaluation order

5 * -2

Evaluation order

-10

Evaluation order

$$(2 + 3) * (5 - 7)$$

Evaluation order

$$(2 + 3) * (5 - 7)$$

Evaluation order

$$(2 + 3) * -2$$

Evaluation order

$$(2 + 3) * -2$$

Evaluation order

5 * -2

Evaluation order

5 * -2

Evaluation order

-10

Evaluation order

(print “hello”; 2 + 3) * (print “ world!”; 5 - 7)

Evaluation order

(print “hello”; 2 + 3) * (print “ world!”; 5 - 7)

Evaluation order

(2 + 3) * (print “ world!”; 5 - 7)

hello

Evaluation order

5 * (print “ world!”; 5 - 7)

hello

Evaluation order

5 * (print “ world!”; 5 - 7)

hello

Evaluation order

5 * (5 - 7)

hello world!

Evaluation order

-10

hello world!

Evaluation order

(print “hello”; 2 + 3) * (print “ world!”; 5 - 7)

Evaluation order

(print “hello”; 2 + 3) * (print “ world!”; 5 - 7)

Evaluation order

(print “hello”; 2 + 3) * (5 - 7)

world!

Evaluation order

(print “hello”; 2 + 3) * -2

world!

Evaluation order

```
(print “hello”; 2 + 3) * -2
```

world!

Evaluation order

(2 + 3) * -2

world!hello

Evaluation order

```
-10
```

world!hello

Evaluation order

Needed to make sense of *effects* (+ non-termination)

(I/O, mutable state, exceptions, callcc, ...)

Not determined by λ-calculus

⇒ Functional langs use many different strategies...

Connection lost with logic & natural deduction?

But wait, there’s more...

Pattern-matching

```
data Nat : Set where
  zero : Nat
  suc  : Nat -> Nat

plus : Nat -> Nat -> Nat
plus  zero    m = m
plus (suc n) m = suc (plus n m)
```

Pattern-matching

Theorem 3.8. *For any λ-geometry, $\overline{r}_\lambda \leq \sqrt{3}/2$.*

Proof. Suppose $a = b = c = 1$. Using condition on angles of $\triangle ABC$ [14], we could compute the length of SMT. For this purpose, we consider the following three cases.

Case 1. $\lambda = 3m$. In this case we have $\overline{L}_s = 2$, $\overline{L}_m = \sqrt{3}$, $\overline{L}_s/\overline{L}_m = \sqrt{3}/2$.

Case 2. $\lambda = 3m + 1$. In this case we have

$$\overline{L}_s = \frac{1}{\sin m\pi/\lambda} + \frac{\sqrt{3}}{2} - \frac{1}{2}\frac{\sin(m+1)\pi/2\lambda}{\sin m\pi/\lambda} \text{ and } \overline{L}_m = 1 + \frac{\sin \pi/3\lambda + \sin 2\pi/3\lambda}{\sin \pi/\lambda}.$$

Note that

$$\frac{\overline{L}_s}{\overline{L}_m} \leq \frac{\sqrt{3}}{2} \text{ if and only if } \frac{\overline{L}_s - \sqrt{3}}{\overline{L}_m - 2} \leq \frac{\sqrt{3}}{2},$$

and

$$\frac{\overline{L}_s - \sqrt{3}}{\overline{L}_m - 2} = \frac{\sin\frac{\pi}{6\lambda}\cos\frac{\pi}{2\lambda}}{\sin\frac{m\pi}{\lambda}\sin\frac{\pi}{3\lambda}} \leq \frac{\cos\frac{\pi}{2\lambda}}{2\sin\frac{\pi}{4}\cos\frac{\pi}{6\lambda}} \leq \frac{1}{2\sin\frac{\pi}{4}} < \frac{\sqrt{3}}{2}.$$

Thus we have $\overline{L}_s/\overline{L}_m \leq \sqrt{3}/2$.

Case 3. $\lambda = 3m + 2$. In this case we have

$$\overline{L}_s = \frac{1}{\sin\frac{(m+1)\pi}{\lambda}} + \frac{\sqrt{3}}{2} - \frac{1}{2}\frac{\sin\frac{m\pi}{2\lambda}}{\sin\frac{(m+1)\pi}{\lambda}} \text{ and } \overline{L}_m = 1 + \frac{\sin\frac{\pi}{3\lambda} + \sin\frac{2\pi}{3\lambda}}{\sin\frac{\pi}{\lambda}}.$$

Pattern-matching

Nice notation for defining fns, and proofs-by-cases

- Enables equational reasoning
- Borrowed directly from mathematical practice...

But not available in natural deduction

Hmm...maybe natural deduction is not so natural?...

Thesis Statement

Focusing proofs give a logical account of evaluation order and pattern-matching

Focusing proofs

Andreoli ’91: search strategy for linear logic sequent calculus

Exploits **polarity** of connectives ([⊗,⊕,1,0,!] vs [⅋,&,⊤,⊥,?])

Extends to classical + intuitionistic logic by *polarization*

My work: new presentation of focusing proofs

- canonical forms of proof and refutation, in terms of **patterns**
- combines best features of sequent calculus and ND

Thesis Statement

Focusing proofs give a logical account of evaluation order and pattern-matching

Thesis Statement, sub I

Evaluation order is encoded in polarity

⇒ one language can [should] mix different evaluation strategies, by reflecting them at the level of types

Thesis Statement, sub II

Pattern-matching is justified by polarity

⇒ pattern-matching is not just “syntactic sugar”: it can [should] be dealt with directly in type theory

Thesis Statement, punchline

Evaluation order and pattern-matching are two sides of the same coin.

This Talk

Some highlights from dissertation

- Polarity and patterns (and the meanings of the connectives)
- Proofs and programs (and pattern-generic reasoning)
- Polarization ↔ double-negation translation ↔ CPS
- Refinement types and subtyping

Related work and future work

Polarity and Patterns

What is a proposition [type]?

Built out of constants/connectives

$A \vee \neg A$ $A \supset (B \wedge C)$ $\neg A \supset A$

nat × nat string → bool

But what do these connectives *mean?*

The approach of **proof-theoretic semantics**...

Pick your side

An introduction rule gives, so to say, a definition of the constant in question...

—Gentzen (1935)

To explain the meaning of an implication A⊃B, we must explain what is the purpose...of a canonical proof... This purpose is to be applied to a canonical proof of...A, thereby yielding a canonical proof of...B. In no way is it correct to say that the meaning of A⊃B is determined by the introduction rule...

—Martin-Löf (1976)

Dummett’s analysis

1976 William James Lectures

- Defined by intros = “verificationist”
- Defined by elims = “pragmatist”
- Described some difficulties for particular connectives...
- ...but required “harmony” between the two aspects

Embracing polarization

Dual approaches to understanding connectives really define *different connectives*

(Instead of requiring harmony, accept diversity!)

"$A \otimes B$" vs. "$A \,\&\, B$", etc.

Subject of my thesis: **polarized logic** (every prop has polarity)

Syntax of polarized propositions [types]

$$A^+ ::= A^+ \otimes B^+ \mid A^+ \oplus B^+ \mid 1 \mid 0$$

$$\mid \neg A^+ \mid A^{-\perp} \mid \downarrow A^-$$

$$\mid \text{nat} \mid \ldots$$

Intuition: + **eager**, – **lazy**

$$A^- ::= A^- \,\&\, B^- \mid A^- \parr B^- \mid \top \mid \perp$$

$$\mid \neg A^- \mid A^{+\perp} \mid \uparrow A^+$$

$$\mid A^+ \to B^- \mid \ldots$$

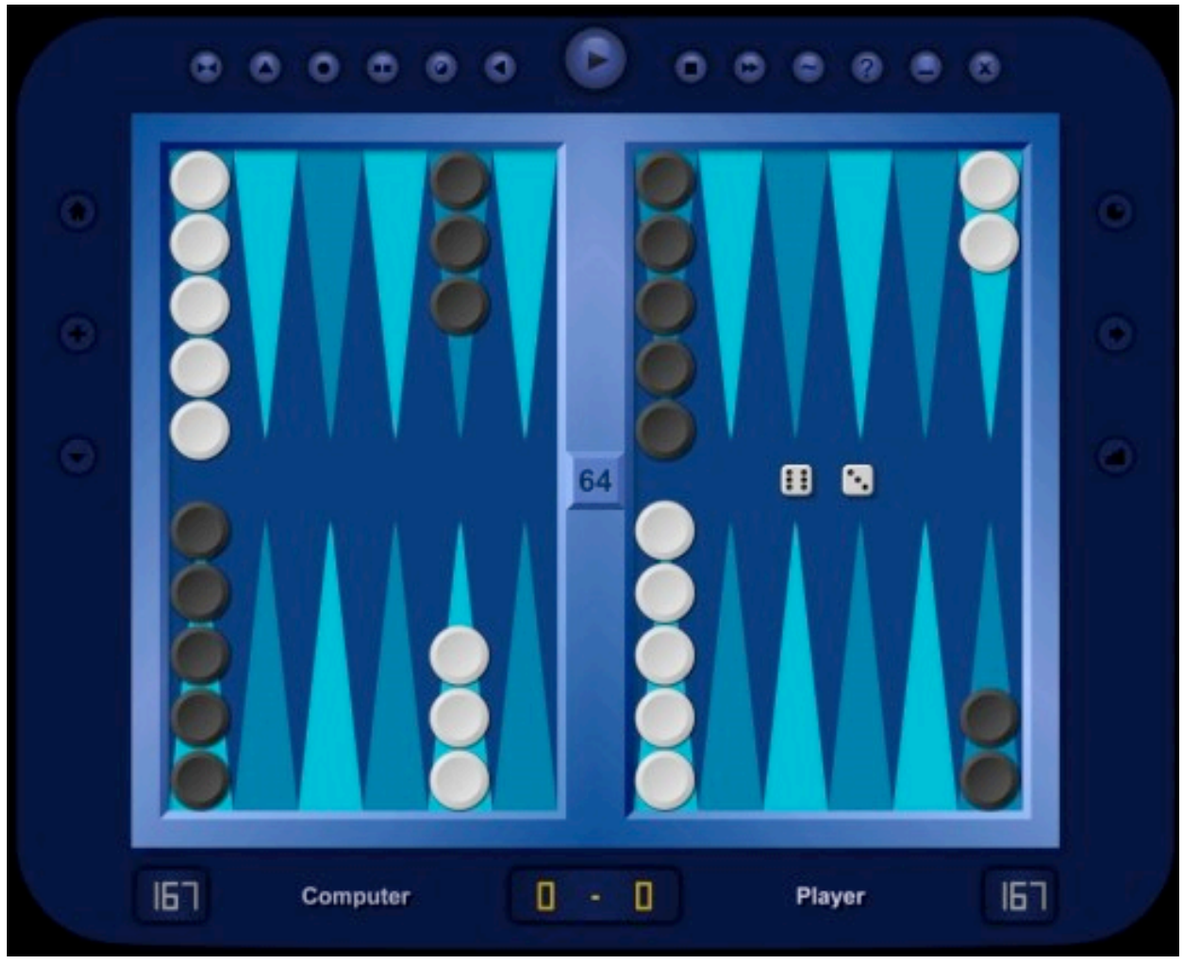
Polarity in game-theoretic semantics

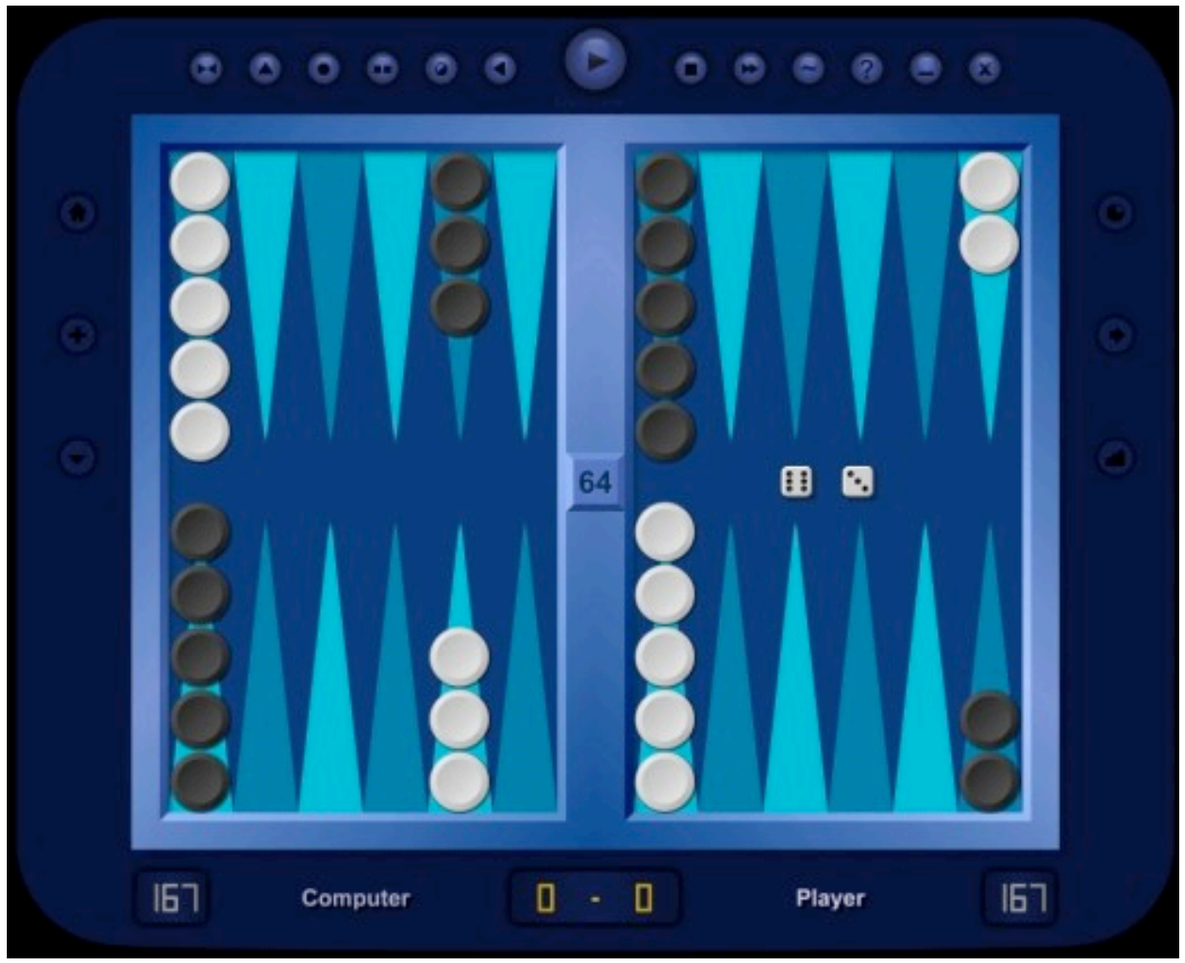
Prop *A* describes a game between Verifier and Refuter

- Lorenzen60, Henkin61, Blass92, ...

- *A* true if winning strategy for Verifier

- Polarity = **who moves first**


64
167
Computer
0 - 0
Player
167


64
167
Computer
0 - 0
Player
167

Polarity from patterns

Idea: *define* connectives by shapes of canonical inference

- + connectives defined by **proof patterns**
- – connectives defined by **refutation patterns**

Patterns are “derivations with holes”...

- **Refutation holes** for + propositions
- **Proof holes** for – propositions

Patterns, formally

Frame (list of holes):

$$\Delta ::= \cdot \mid (\Delta_1 , \Delta_2) \mid A^+ \mathit{false} \mid A^- \mathit{true}$$

A **proof pattern** is a derivation of $\Delta \Vdash A^+ \mathit{true}$

A **refutation pattern** is a derivation $\Delta \Vdash A^- \mathit{false}$

$\Vdash$ is like linear entailment (but connectives nonlinear!)

Rules for $\Vdash$ define the connectives...

Syntax of polarized propositions [types]

$$A^+ ::= A^+ \otimes B^+ \mid A^+ \oplus B^+ \mid 1 \mid 0$$

$$\mid \neg A^+ \mid A^{-\perp} \mid \downarrow A^-$$

$$\mid \text{nat} \mid \ldots$$

$$A^- ::= A^- \,\&\, B^- \mid A^- \parr B^- \mid \top \mid \perp$$

$$\mid \neg A^- \mid A^{+\perp} \mid \uparrow A^+$$

$$\mid A^+ \to B^- \mid \ldots$$

Some positive connectives

$$\frac{\Delta_1 \Vdash A^+ \; true \quad \Delta_2 \Vdash B^+ \; true}{\Delta_1\Delta_2 \Vdash A^+ \otimes B^+ \; true}$$

$$\frac{}{A^+ \; false \Vdash \neg A^+ \; true}$$

$$\frac{\Delta \Vdash A^+ \; true}{\Delta \Vdash A^+ \oplus B^+ \; true} \qquad \frac{\Delta \Vdash B^+ \; true}{\Delta \Vdash A^+ \oplus B^+ \; true}$$

Some negative connectives

$$\frac{\Delta_1 \Vdash A^- \; \mathit{false} \quad \Delta_2 \Vdash B^- \; \mathit{false}}{\Delta_1\Delta_2 \Vdash A^- \otimes B^- \; \mathit{false}}$$

$$\frac{}{A^- \; \mathit{true} \Vdash \neg A^- \; \mathit{false}}$$

$$\frac{\Delta \Vdash A^- \; \mathit{false}}{\Delta \Vdash A^- \& B^- \; \mathit{false}} \qquad \frac{\Delta \Vdash B^- \; \mathit{false}}{\Delta \Vdash A^- \& B^- \; \mathit{false}}$$

Some polarity mixing connectives

$$\frac{\Delta_1 \Vdash A^+ \; true \quad \Delta_2 \Vdash B^- \; false}{\Delta_1\Delta_2 \Vdash A^+ \to B^- \; false}$$

$$\frac{}{A^- \; true \Vdash \downarrow A^- \; true} \qquad \frac{}{A^+ \; false \Vdash \uparrow A^+ \; false}$$

$$\frac{\Delta \Vdash A^- \; false}{\Delta \Vdash A^{-\perp} \; true} \qquad \frac{\Delta \Vdash A^+ \; true}{\Delta \Vdash A^{+\perp} \; false}$$

The basic analogy

proof = **value**
refutation = **continuation**

⟹ proof patterns = value patterns

- like in FP
- *linear* (like in FP)

refutation patterns = continuation patterns

- like in FP, if you squint

A type-free notation...

Connective	π	Patterns
$A^+ \otimes B^+$	+	$(\, p_1, p_2 \,)$
$A^+ \oplus B^+$	+	inl p inr p
1	+	( )
0	+	**none**
$\neg A^+$	+	k

Connective	π	Patterns
$A^- \,\&\, B^-$	–	fst;d snd;d
$A^- \parr B^-$	–	$[d_1, d_2]$
$\top$	–	**none**
$\bot$	–	[ ]
$\neg A^-$	–	x

A type-free notation...

Connective	π	Patterns
$\downarrow A^-$	+	x
$\uparrow A^+$	–	k
$A^+ \to B^-$	–	p@d
nat	+	z s p
⋮		

Proofs and Programs

Once we have understood how to discover individual patterns which are alive, we may then make a language for ourselves, for any building task we face.

—Christopher Alexander

From patterns to programs

Rule of (positive) proof:

$$\frac{\Delta \Vdash A^{+}\ \mathit{true} \qquad \Gamma \vdash \Delta}{\Gamma \vdash A^{+}\ \mathit{true}}$$

From patterns to programs

Rule of (positive) values:

$$\frac{\Delta \Vdash A^+ \; \mathit{true} \qquad \Gamma \vdash \Delta}{\Gamma \vdash A^+ \; \mathit{true}}$$

pattern *substitution*

$$V = p[\sigma]$$

(a few obvious rules for building σs)

From patterns to programs

Rule of (positive) values:

$$\frac{\Delta \Vdash A^+ \; \mathit{true} \qquad \Gamma \vdash \Delta}{\Gamma \vdash A^+ \; \mathit{true}}$$

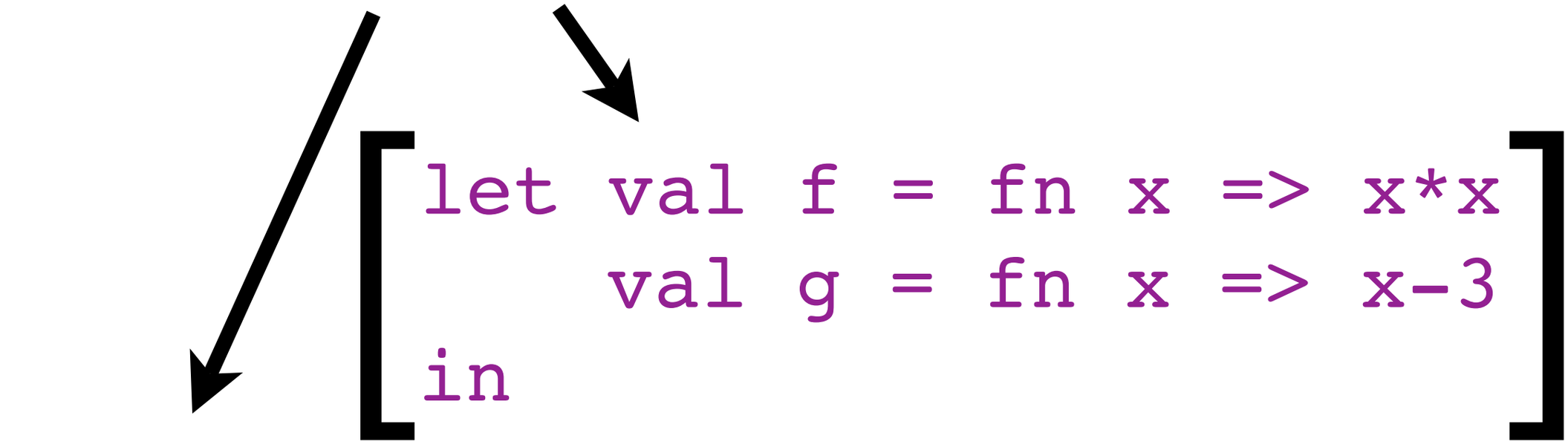
pattern *substitution*

$V = p[\sigma]$

slogan: **a value (of + type) is a pattern under a substitution**

An illustration from ML...

```
(fn x => x*x, fn x => x-3)
```




```
(f,g)
```

From patterns to programs

Rule of (positive) refutation:

$$\frac{\Delta \Vdash A^+\ \mathit{true} \longrightarrow \Gamma, \Delta \vdash \#}{\Gamma \vdash A^+\ \mathit{false}}$$

(notation from IID)

From patterns to programs

Rule of (positive) continuations:

pattern *expression*

$$\frac{\Delta \Vdash A^+ \; true \longrightarrow \Gamma, \Delta \vdash \#}{\Gamma \vdash A^+ \; false}$$

$K = p \rightarrow E$ (second-order definition of syntax!)

(a few simple rules for building *E*s)

From patterns to programs

Rule of (positive) continuations:

$$\frac{\Delta \Vdash A^+ \; \mathit{true} \longrightarrow \Gamma, \Delta \vdash \#}{\Gamma \vdash A^+ \; \mathit{false}}$$

pattern *expression*

$K = p \rightarrow E$ (second-order definition of syntax!)

slogan: **a continuation (of + type) is a map from patterns to expressions**

Another illustration from ML...

```
(* pattern-matching function *)
```

Exercise in duality

$$\frac{\Delta \Vdash A^+\ \mathit{true} \qquad \Gamma \vdash \Delta}{\Gamma \vdash A^+\ \mathit{true}} \qquad \frac{\Delta \Vdash A^+\ \mathit{true} \longrightarrow \Gamma, \Delta \vdash \#}{\Gamma \vdash A^+\ \mathit{false}}$$

Exercise in duality

$$\frac{\Delta \Vdash A^+\ \mathit{true} \qquad \Gamma \vdash \Delta}{\Gamma \vdash A^+\ \mathit{true}} \qquad \frac{\Delta \Vdash A^+\ \mathit{true} \longrightarrow \Gamma, \Delta \vdash \#}{\Gamma \vdash A^+\ \mathit{false}}$$

So much time and so little to do. Wait a minute. Strike that. Reverse it.

—Willy Wonka

Exercise in duality

$$\frac{\Delta \Vdash A^+ \; true \qquad \Gamma \vdash \Delta}{\Gamma \vdash A^+ \; true} \qquad \frac{\Delta \Vdash A^+ \; true \longrightarrow \Gamma, \Delta \vdash \#}{\Gamma \vdash A^+ \; false}$$

$$\frac{\Delta \Vdash A^- \; false \longrightarrow \Gamma, \Delta \vdash \#}{\Gamma \vdash A^- \; true} \qquad \frac{\Delta \Vdash A^- \; false \qquad \Gamma \vdash \Delta}{\Gamma \vdash A^- \; false}$$

(compare direct/indirect proof/refutation)

(think of V : A→B defined by pattern-matching)

General observations

Language intrinsically forces CPS + pattern-matching

Rules of proof and refutation are *generic* in terms of patterns

⇒ Can often reason about programs in type-generic way

Generalizes (and improves) untyped reasoning

Equational theory

Admissibility of cut = composition of terms (~ β-reduction)

- value V + continuation K ⇒ expression $K \bullet V$
- term t + substitution σ ⇒ term $t[\sigma]$
- (only defined when types line up, but defined generically)

Admissibility of initial sequents = identity terms (~ η-expansion)

- continuation Id_k, substitution $Id_{[\Delta]}$

Satisfy suitable unit and associativity properties

Operational semantics

Small-step, environment semantics (= cut-**elimination** proc.)

Two generic rules for all positive types! (2 more 4 -...)

$$\langle \gamma \mid k\ V \rangle \rightsquigarrow \langle \gamma \mid \gamma(k) \mid V \rangle$$

$$\langle \gamma \mid K \mid p[\sigma] \rangle \rightsquigarrow \langle \gamma;\sigma \mid K(p) \rangle$$

Can extend language with effects...

Definition: $E_1 \cong E_2$ iff same result in all γ

Theorem: $E_1 \neq E_2$ implies $E_1 \ncong E_2$ w/abort + ground state

“Implementation”

Two embeddings:

- In Agda (use IID to directly encode higher-order rules)
- In Twelf (use defunctionalization to get rid of them)

Both piggyback on existing implementations of (dependent) pattern-matching to get “pattern-matching for free”

But a lower-level implementation could be instructive...

Polarization, DNT, CPS

Polarization

Define $|A| = b$:

$$|1| = \mathbf{T} = |\top| \qquad |0| = \mathbf{F} = |\bot|$$

$$|A \otimes B| = |A| \wedge |B| = |A \& B| \qquad |A \oplus B| = |A| \vee |B| = |A \parr B|$$

$$|{\downarrow}A| = |A| = |{\uparrow}A| \qquad |A \to B| = |A| \supset |B|$$

etc.

Definition: *A* is a **polarization** of *b* if $|A|$ and *b* are classically equivalent.

Completeness of focusing

Theorem: Let A^+ be a polarization of b. If b is a classically theorem, then A^+ *false* $\vdash$ #.

Theorem: Let A^- be a polarization of b. If b is a classically theorem, then $\vdash$ A^- *true.*

Translation into minimal logic

Define $A^m = b$:

$$1^m = \mathrm{T} = \bot^m \qquad |0| = \mathrm{F} = \top^m$$

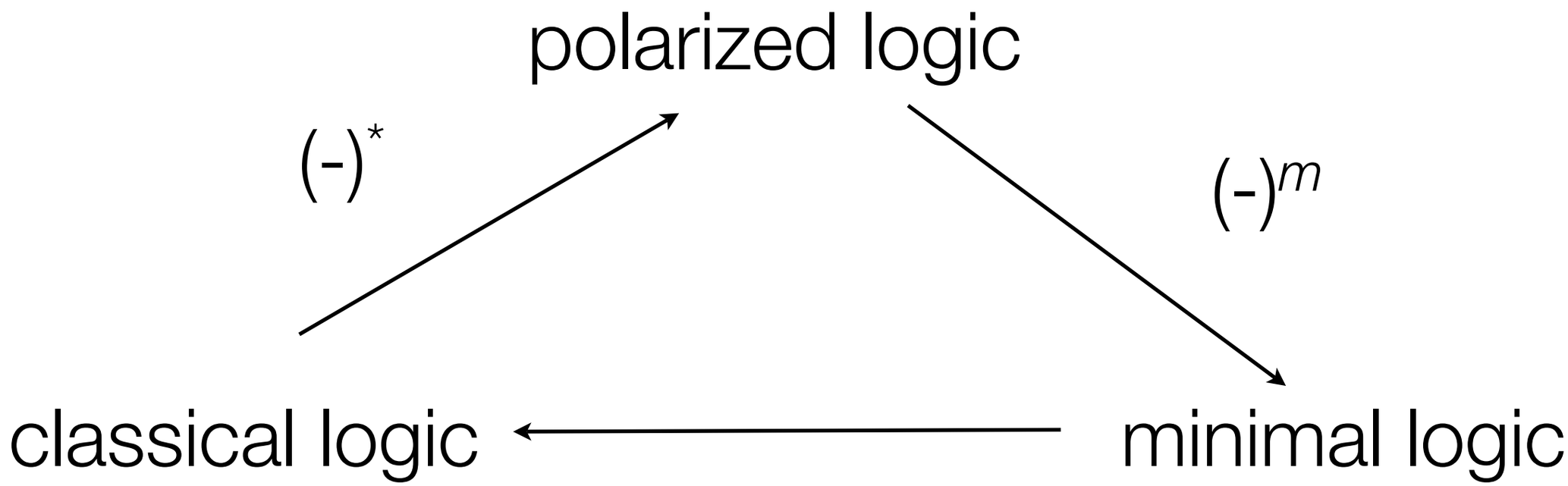
$$(A \otimes B)^m = A^m \wedge B^m = (A^m \parr B^m) \qquad (A \oplus B)^m = A^m \vee B^m = (A \& B)^m$$

$$(\downarrow A)^m = {\sim}A^m = (\uparrow A)^m \qquad (A \multimap B)^m = A^m \wedge B^m$$

etc.

Theorem: If A^+ *false* $\vdash$ # then ${\sim}{\sim}A^{+m}$ is a minimal theorem. If $\vdash A^-$ *true* then ${\sim}A^{-m}$ is a minimal theorem

Double-negation [CPS] translation



different polarizations ⇒ different CPS translations

The polarity jungle

Polarizations of the function space:

Polarization	CPS translation
$\downarrow(A^+ \to \uparrow B^+)$	Reynolds CBV
$\downarrow A^- \to B^-$	Streicher CBN
$\uparrow\downarrow(\downarrow A^- \to B^-)$	Plotkin CBN
$\neg A^+ \oplus B^+$	??
⋮	⋮

Polarity pocket dictionary

Rough guide to ML:

ML type	+ type
A -> B	$\downarrow(A \rightarrow \uparrow B)$
$A * B$	$A \otimes B$
$A + B$	$A \oplus B$

Rough guide to Haskell:

Haskell type	– type
A -> B	$\downarrow A \rightarrow B$
(A , B)	A & B
Either A B	$\uparrow(\downarrow A \oplus \downarrow B)$

From Intrinsic

To Extrinsic Types

What is a type?

Intrinsic (“Church”) vs extrinsic (“Curry”) interpretations

Above: intrinsically-typed terms, but type-free notation

- A type is a promise about the way a program is structured
- But doesn’t promise very much...

Chap. 6: extrinsic **refinement** of intrinsically-typed terms

- Static guarantee of richer properties
- intersection & union types, subtyping, etc.
- Freeman-Pf.91, Freeman94, Xi98, Davies05, Dunfield07...

Sorts refine types

$$S \sqsubseteq A$$

Examples:

$\text{even} \sqsubseteq \text{nat}$ $\quad\quad$ $\text{odd} \sqsubseteq \text{nat}$ $\quad\quad$ $\text{pos} \sqsubseteq \text{nat}$

$\text{even} \cap \text{odd} \sqsubseteq \text{nat}$ $\quad\quad$ $\text{even} \cup \text{pos} \sqsubseteq \text{nat}$

$\text{even} \to \text{even} \sqsubseteq \text{nat} \to \text{nat}$

($\sqsubseteq$ is not subtyping!)

Summary

Sorts **defined** by *pattern-inversion*

Sort-generic refinement typing rules

Sort-generic safety theorem for environment semantics

Value + evaluation context restrictions reconstructed

Two interpretations of subtyping...

Subtyping

Identity coercion interpretation:

$$S \leq T \quad \text{iff} \quad k : T\ \mathit{false} \vdash \mathit{Id}_k : S\ \mathit{false}$$

(or equivalent coercions)

No-counterexamples interpretation:

$$S \preccurlyeq T \quad \text{iff} \quad \text{every } V : S\ \mathit{true} \text{ and } K : T\ \mathit{false} \text{ are } \textbf{safe}$$

Soundness: $S \leq T$ implies $S \preccurlyeq T$ (explicit witness to safety)

Completeness: $S \not\leq T$ implies $S \not\preccurlyeq T$, **given enough effects**

e.g., safety violations of $\neg\neg S \cap \neg\neg T \leq \neg\neg(S \cap T)$, etc.

Conclusions

Thesis Contributions

Focusing proofs as a new take on proofs-as-programs that...

- accounts for important features of modern PLs
- allows mixing of evaluation strategies, and definition of types by either constructors or destructors
- uniformly accounts for untyped computation, intrinsic types, and extrinsic refinement types
- is in many ways *easier* to reason about meta-theoretically, by pattern-generic arguments (cf. infinitary proof theory)

Related Work

Lots of it! See thesis...

Most closely related: ludics, and Levy's call-by-push-value...

- Very ~ distinction between *value* and *computation types*
- Very $\rightarrowtail$ origin (trying to unite models of CBV and CBN)
- Complementary viewpoints

Some unexpected connections

- Realizability (BHK, NuPRL, Krivine, etc.)
- Infinitary proof theory (Buchholz' Ω-rule, Mints78, etc.)

Future Work, abstractly

programming languages

polarized type theory

More concretely...

Extend polarized approach to more high-powered type theory

- Dependent types (meta-circular!) [cf. Licata&Harper09]
- Polymorphism [cf. Lassen&Levy08, for CBPV]
- Pronominal binding [LZH08], module systems, etc.
- Hopefully sheds light on how to do effects and equality...

Full compiler for HOT language w/effects

- Systematic construction, from proof-theoretic principles?
- Relation to line of work initiated by Danvy03?

More concretely...

Better account of asynchronous/pure computation

- Linearity and modalities?
- Delimited continuations?

Categorical/topological/algebraic semantics...

Thanks!

Focusing proofs as a new take on proofs-as-programs that...

- accounts for important features of modern PLs
- allows mixing of evaluation strategies, and definition of types by either constructors or destructors
- uniformly accounts for untyped computation, intrinsic types, and extrinsic refinement types
- is in many ways *easier* to reason about meta-theoretically, by pattern-generic arguments (cf. infinitary proof theory)

any questions?