

ESLLI'03 & CSL'03. Vienna, August 26, 2003

*Back to the Future:
Explicit Logic for Computer Science*

Sergei N. Artemov

Graduate Center of the City University of New York

TIME100 project: "The greatest scientists of the century"
(20 positions):

- Technology - 6 (airplane, rocket, TV, transistor, plastic, WWW)
- Biology & Medicine - 4 (psychoanalysis, penicillin, DNA, polio)
- Physics & Astronomy - 3 (Einstein, Fermi, Hubble)
- Anthropology - 1 (The Leakeys)
- Economy - 1 (Keynes)
- Environment - 1 (Rachel Carson)
- Psychology - 1 (Piaget)

- Computer Science - 1 (?)
- Mathematics - 1 (?)
- Philosophy - 1 (?)

TIME100 project: "The greatest scientists of the century"
(20 positions):

- Technology - 6 (airplane, rocket, TV, transistor, plastic, WWW)
- Biology & Medicine - 4 (psychoanalysis, penicillin, DNA, polio)
- Physics & Astronomy - 3 (Einstein, Fermi, Hubble)
- Anthropology - 1 (The Leakeys)
- Economy - 1 (Keynes)
- Environment - 1 (Rachel Carson)
- Psychology - 1 (Piaget)

- Computer Science - 1 (Turing, a logician)
- Mathematics - 1 (Gödel, a logician. Vienna!)
- Philosophy - 1 (Wittgenstein, who began as a logician. Vienna again!)

Three traditions in Logic

1. *Classical*: Frege, Hilbert, Gödel, Tarski
2. *Constructive*: Brouwer, Heyting, Kolmogorov, Gödel
3. *Explicit*: Skolem, Hilbert, Curry, Gödel, Herbrand, Church
bridge to computing!

Classical Tradition—Propositional Logic: consistent, decidable, not feasible (if $P \neq NP$). Many practical methods reduce to this level.

Boole (1854), Post (1920): boolean circuits, boolean values in programming, verification, duality of proof search and model checking

Gentzen (1933): normalization of proofs basis for modern provers, proof checkers.

Downside: very limited expressive power! This can be somehow improved by introducing new connectives: modalities, temporal operators, epistemic, dynamic, etc.

Classical Tradition—First Order Logic: one sort of objects, quantifiers \forall, \exists . Consistent, undecidable though recursively enumerable, formalizes all "usual" mathematical reasoning via Set Theory. Formal arithmetic, set theory, theory of reals, theory of groups, rings, fields, etc. can be presented as FO theories.

Frege (1879): proofs in first order systems as positive tests of validness.

Gödel (1929): Completeness of proofs in FO logic.

Downside: excessive coding, computationally unfriendly!

Why is excessive coding so bad for specific tasks?

Efficiency is lost, which makes it impossible to use brute force methods. Try to operate natural numbers as finite ordinals, and you will find out that addition is not computable. Gödel numbering of sentences, proofs, computable functions, is notoriously non-efficient.

Structure is lost, which limits using intelligent, e.g. mathematical methods. Our space world is 3D. It takes three real numbers to specify the position of a point in this magnificent hall. With a little bit of coding one can do “better”: take a bijection between the 3D volume of this hall and the unit interval $(0,1)$. Now the position of each point can be determined by just one positive real less than 1. However, the whole of 3D Math has been sacrificed in this coding, no traditional mathematical methods work here any longer.

Classical Tradition - Logic Programming, computing by proving.

J. Robinson (1965): resolution + unification = single rule proof system for the first order logic. A theoretical prototype of PROLOG programming language (1972). Verification problem: “does a program satisfy its specification?” is subsumed by the programming language.

The starting point of computing on the basis of a limited first order specification here is to get rid of the first order quantifiers and replace them by Skolem functions. This step immediately moves the ball to the Explicit Logic area (below).

Downside: Relatively slow. Later refinements improved its efficiency but the logical purity has been lost.

Higher order logic: Higher sorts of quantified variables. “Very” undecidable, not axiomatizable, not compact. HO theories: second order arithmetic (analysis), theory of sets and classes. Closer to the natural language, convenient for proof checking (verification).

Andrews (1982): classical HOL prover

Downside: No efficient proof search, no complete proof systems possible, difficulties with semantics and consistency.

Modal Logic: (Lewis & Langford (1932)) extends the usual logic (typically propositional) by new atoms $\Box F$ - “ F is necessary” to capture different kind of dynamics. Usually preserves decidability

Gödel (1933): *provability calculus* (= **S4**). The most popular modal logic: knowledge representation, dynamic logic, etc.

1. Classical axioms and rules
2. $\Box(F \rightarrow G) \rightarrow (\Box F \rightarrow \Box G)$
3. $\Box F \rightarrow F$
4. $\Box F \rightarrow \Box \Box F$
5. $F / \Box F$ (necessitation rule)

Gödel introduced the above system as a provability calculus where $\Box F$ denotes “ F is provable”, though it took more than 60 years to figure out how to make this idea work.

McKinsey - Tarski (1948): topological semantics $\Box F = \text{interior}(F)$, leads to logics for dynamic systems

Kripke (1959): possible worlds *à la* Leibniz

Hoare (1969): partial correctness statements $A\{G\}B =$ “if A holds before the execution of G then B holds afterwards”, a classic of verification. Some years ago Hoare was knighted by the British Queen.

Pratt (1976): logic of programs, $[C]\phi = \phi$ holds while C is executed, each $[C]$ is an **S4**-modality, Kripke style semantics where possible worlds are machine states

Pnueli (1977): branching temporal logic = logic of concurrency, Turing award in CS.

*Logic of Knowledge: a core AI topic, $K_A(\phi)$ = "agent A knows ϕ ", multiple modalities **S4** and **S5**, negative introspection, common knowledge operator.*

Logical Omniscience: *unrealistic assumption that an agent knows all logical consequences of its knowledge.*

Major Problem: find a logic of knowledge that distinguishes "hard" and "easy" facts

Intuitionistic Tradition: constructive approach to mathematics and logic.

Brouwer (1900s)

“It does not make sense to think of truth or falsity of a mathematical statement independently of our knowledge concerning the statement. A statement is *true* if we have a proof of it, and *false* if we can show that the assumption that there is a proof for the statement leads to a contradiction.”

“Truth Tables” for intuitionistic logic

Brouwer–Heyting–Kolmogorov: a sentence is true if it has a proof, whereas proofs satisfy the following *BHK* conditions:

- a proof of $A \wedge B$ consists of a proof of A and a proof of B ,
- a proof of $A \vee B$ is given by presenting either a proof of A or a proof of B ,
- a proof of $A \rightarrow B$ is a construction which, given a proof of A returns a proof of B ,
- absurdity \perp is a proposition which has no proof, $\neg A$ is $A \rightarrow \perp$.

Crucial for both foundational and practical reasons.

Computational content of intuitionistic proofs

Kleene, Curry, Howard: For each intuitionistic proof one can associate a total computable function (typed lambda-term) calculating an evidence of the conclusion given evidences of assumptions.

Different from *BHK*!

Unlike computational programs proofs normally have a verification mechanism - proof checker. Thus, the predicate " p is a proof of F " is decidable whereas the predicate " r is a realizer of F " is not. Roughly speaking, there are much more realizers than proofs allowed. The proof-based semantics was shown to capture the intuitionistic logic exactly (below). Realizable formulas constitute a larger family; it is still unknown whether this family is computably enumerable!.

Major models for intuitionistic logic

1. Algebraic semantics (Birkhoff, 1935)
2. Topological semantics (Stone, 1937; Tarski, 1938)
3. Realizability semantics (Kleene, 1945)
4. Beth models (1956)
5. Dialectica Interpretation (Gödel, 1958)
6. Curry - Howard isomorphism (1958)
7. Medvedev's logic of problems (1962)
8. Kripke models (1965)
9. Kuznetsov-Muravitsky-Goldblatt provability interpretation (1976)
10. Categorical semantics (Goldblatt, 1979)

None solves the original **BHK**-problem!

Intuitionistic system = classical system + effective \forall and \exists .

Existential property of intuitionistic systems:

a constructive proof of $\forall x \exists y A(x, y)$ yields computable term (program) $f(x)$ such that $\forall x A(x, f(x))$ holds.

*Corollary: intuitionistic correctness proof =
program + correctness proof =
verified program*

Downside: produces correct but not necessarily optimal programs. Programming by proving often reduces to a “reverse engineering”: i.e. writing a proof having in mind a specific extracted algorithm.

Explicit tradition: Functions vs. Quantifiers:

Skolem (1920), Hilbert (1922), Herbrand (1930), Gödel (1930):
quantifiers \forall, \exists are ghosts of functions.:

logic	explicit logic
$\forall x \exists y A(x, y)$	$A(x, f(x))$
$\exists x A(x) \rightarrow \exists y B(y)$	$A(x) \rightarrow B(f(x))$

By its nature the closest to Computer Science. Addresses the right set of questions: whether $f(x)$ is computable, feasible, etc.

Downside: Too many Skolem functions, unification problems.

Shönfinkel (1924), Church (1929, 1930):

λ -calculus = universal functional language, foundational motivations.

Normal form = the result of a computation,

normalization process = computation

McCarthy (1960): λ -calculus implemented in *LISP*,

universal machine = *LISP*-compiler in *LISP*.

Functional programming languages.

Curry (1934), Howard (1969):

typed λ -calculus, implemented in ML

$t:F \sim$ term t has type $F \sim$ t is a proof of the proposition F

proofs \sim *functional programs*

assumptions \sim *initial data*

deduction \sim *execution sequence*

Girard (1971): second order λ -calculus, appeared from foundational studies of consistency of the second order arithmetic, became a prototype of prover *Coq*.

Martin-Löf (1982): type theory with intuitionistic logic, is implemented in Constable's *NuPRL* prover at Cornell, verification, programming via proofs, formally verified mathematics.

A story: How a shift to an explicit language helped solving an old theoretical problem in logic and has provided new promising tools in different areas of Computer Science.

Gödel's provability logic problem.

Gödel embedded **Int** into **S4** in order to provide a provability semantics for the former:

1. translate **Int**-formula F into a classical language \square :

$$tr(F) = \text{“box each subformula of } F\text{”},$$

2. test the translation in **S4**:

$$\mathbf{Int} \text{ proves } F \Leftrightarrow \mathbf{S4} \text{ proves } tr(F)$$

(Gödel (1933), McKinsey & Tarski (1948))

The mission has not been accomplished though, since

S4 itself was left without an exact provability model

$$\mathbf{Int} \hookrightarrow \mathbf{S4} \hookrightarrow ? \hookrightarrow \text{REAL PROOFS}$$

Gödel (1933): **S4** *modality* \neq *Provable*(\cdot)

Indeed, $\Box(\Box F \rightarrow F)$ is provable in **S4**:

$\Box F \rightarrow F$ - reflexivity axiom

$\Box(\Box F \rightarrow F)$ - by Internalization rule

However, under the interpretation of \Box as *Provable* this asserts that reflection is internally provable

$$\text{Provable}_T(\text{Provable}_T(F) \rightarrow F)$$

which is false by Gödel's Incompleteness Theorem.

Gödel's problem: *find an exact provability semantics of S4.*

This problem was discussed by Gödel in 1933 and 1938 in connection with the semantics for intuitionistic logic.

- Source of problem: a nonconstructive character of \exists . The premise in $\exists x Proof(x, F) \rightarrow F$ does not provide a specific proof of F , this “ x ” may well be a nonstandard number which is not a code of a derivation.
- Cure: back to the BHK format!

Gödel, 1938/95, Artemov, Jäger & T. Strassen, 1992 suggested considering format

$t:F$ (“ t is a proof of F ”)

with operations on proofs rather than quantifiers over proof hidden in the modality \Box .

Explicit reflection $Proof(t, F) \rightarrow F$ for each specific t is internally provable.

Indeed, if $Proof(t, F)$ is true, then t is a proof of F and hence F and $Proof(t, F) \rightarrow F$ are both provable. If $Proof(t, F)$ is false, then $\neg Proof(t, F)$ is true and provable, hence $Proof(t, F) \rightarrow F$ is provable. This allows us to circumvent the Incompleteness Theorem here. An appropriate class of BHK style operation on proofs is needed to capture the whole of **S4**.

Principal difficulties:

- Finding the right format of explicit provability. Gödel's suggestion of 1938 remained unpublished till 1995.
- The problem was pronounced unsolvable by Montague in 1963.
- Taming Skolem functions and self-referentiality $t:F(t)$ turned out to be technically difficult. One has to give up many stereotypes coming from related areas, such as modal and combinatory logic and λ -calculus, etc.
- Big distraction and big help: Provability Logic with $\Box F \sim \text{Provable}(F)$ (Solovay, Boolos, de Jongh, Visser, Magari, Sambin, Montagna, S.A., Beklemishev, et al.) is modal system **GL** incompatible with **S4**. Though applications of **GL** have been mostly limited to Proof Theory, its mathematics provided a solid background for the solution of Gödel's problem.

A close relative: typed combinatory logic **CL**.

Combinatory terms can be read as proof terms in a Hilbert style system.

Constant combinators stand for proofs of propositional axioms:

$$\mathbf{k}^{A,B} : (A \rightarrow (B \rightarrow A)), \quad \mathbf{s}^{A,B,C} : [(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))]$$

Variables in **CL** denote unspecified proofs, the operation of application “.” corresponds to the internalized rule of *modus ponens*

$$t:(F \rightarrow G) \rightarrow (s:F \rightarrow (t \cdot s):G)$$

The whole of **CL** corresponds to the fragment of implicative intuitionistic **S4** consisting only of sequents

$$\Box A_1, \dots, \Box A_n \Rightarrow \Box B,$$

where A_1, \dots, A_n, B do not contain modalities.

Proof Polynomials

Basis for all invariant propositional operations on proofs

variables x, y, z, \dots

ranging over proofs

constants a, b, c, \dots

proofs of instances of logical axioms

“.” is **application**:

applies $s:(F \rightarrow G)$ to $t:F$ and returns $(s \cdot t):G$

(New) “!” is **proof checking**:

computes $!t$ a proof of $t:F$

(New) “+” is **union**:

takes union (concatenation) of two proofs

The Logic of Proofs **LP**

The language of **LP** = that of the classical propositional logic + additional atoms $p:F$, where p is a proof polynomial and F is a formula.

A0. classical axioms

A1. $t:(F \rightarrow G) \rightarrow (s:F \rightarrow (t \cdot s):G)$

(application)

A2. $t:F \rightarrow F$

(explicit reflexivity)

A3. $t:F \rightarrow !t:(t:F)$

(proof checker)

A4. $s:F \rightarrow (s \dagger t):F, \quad t:F \rightarrow (s \dagger t):F$

(union)

R1. Modus Ponens

R2. $\vdash c:A$, where $A \in A0-A4$, c is a proof constant. (axiom necessitation)

Combinatory Logic **CL** \approx the implicational fragment of *A1 + R2*.

Corollary. Constructive Necessitation is admissible in **LP**:

$$\frac{\vdash B}{\vdash p:B}$$

for some proof polynomial p depending on the derivation of B .

Examples of derivations

Derivation in **S4**

$$A \rightarrow (B \rightarrow A \wedge B)$$

$$\Box(A \rightarrow (B \rightarrow A \wedge B))$$

$$\Box A \rightarrow \Box(B \rightarrow A \wedge B)$$

$$\Box A \rightarrow (\Box B \rightarrow \Box(A \wedge B))$$

$$(\Box A \wedge \Box B) \rightarrow \Box(A \wedge B)$$

Derivation in **LP**

Examples of derivations

Derivation in **S4**

$$A \rightarrow (B \rightarrow A \wedge B)$$

$$\Box(A \rightarrow (B \rightarrow A \wedge B))$$

$$\Box A \rightarrow \Box(B \rightarrow A \wedge B)$$

$$\Box A \rightarrow (\Box B \rightarrow \Box(A \wedge B))$$

$$(\Box A \wedge \Box B) \rightarrow \Box(A \wedge B)$$

Derivation in **LP**

$$A \rightarrow (B \rightarrow A \wedge B)$$

Examples of derivations

Derivation in **S4**

$$\begin{aligned} & A \rightarrow (B \rightarrow A \wedge B) \\ \Box(A \rightarrow (B \rightarrow A \wedge B)) \\ \Box A \rightarrow \Box(B \rightarrow A \wedge B) \\ \Box A \rightarrow (\Box B \rightarrow \Box(A \wedge B)) \\ (\Box A \wedge \Box B) \rightarrow \Box(A \wedge B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} & A \rightarrow (B \rightarrow A \wedge B) \\ c:(A \rightarrow (B \rightarrow A \wedge B)) \end{aligned}$$

Examples of derivations

Derivation in **S4**

$$\begin{aligned} & A \rightarrow (B \rightarrow A \wedge B) \\ \Box(A \rightarrow (B \rightarrow A \wedge B)) \\ \Box A \rightarrow \Box(B \rightarrow A \wedge B) \\ \Box A \rightarrow (\Box B \rightarrow \Box(A \wedge B)) \\ (\Box A \wedge \Box B) \rightarrow \Box(A \wedge B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} & A \rightarrow (B \rightarrow A \wedge B) \\ c:(A \rightarrow (B \rightarrow A \wedge B)) \\ x:A \rightarrow (c \cdot x):(B \rightarrow A \wedge B) \end{aligned}$$

Examples of derivations

Derivation in **S4**

$$\begin{aligned} & A \rightarrow (B \rightarrow A \wedge B) \\ \Box(A \rightarrow (B \rightarrow A \wedge B)) \\ \Box A \rightarrow \Box(B \rightarrow A \wedge B) \\ \Box A \rightarrow (\Box B \rightarrow \Box(A \wedge B)) \\ (\Box A \wedge \Box B) \rightarrow \Box(A \wedge B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} & A \rightarrow (B \rightarrow A \wedge B) \\ c:(A \rightarrow (B \rightarrow A \wedge B)) \\ x:A \rightarrow (c \cdot x):(B \rightarrow A \wedge B) \\ x:A \rightarrow (y:B \rightarrow ((c \cdot x) \cdot y):(A \wedge B)) \end{aligned}$$

Examples of derivations

Derivation in **S4**

$$\begin{aligned} & A \rightarrow (B \rightarrow A \wedge B) \\ \Box(A \rightarrow (B \rightarrow A \wedge B)) \\ \Box A \rightarrow \Box(B \rightarrow A \wedge B) \\ \Box A \rightarrow (\Box B \rightarrow \Box(A \wedge B)) \\ (\Box A \wedge \Box B) \rightarrow \Box(A \wedge B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} & A \rightarrow (B \rightarrow A \wedge B) \\ c:(A \rightarrow (B \rightarrow A \wedge B)) \\ x:A \rightarrow (c \cdot x):(B \rightarrow A \wedge B) \\ x:A \rightarrow (y:B \rightarrow ((c \cdot x) \cdot y):(A \wedge B)) \\ (x:A \wedge y:B) \rightarrow ((c \cdot x) \cdot y):(A \wedge B) \end{aligned}$$

This was an easy ride, straightforward from **S4**.

Examples of derivations.

Some problems on the way from **S4**:

Derivation in **S4**

$$A \rightarrow A \vee B$$

$$\Box(A \rightarrow A \vee B)$$

$$\Box A \rightarrow \Box(A \vee B)$$

$$B \rightarrow A \vee B$$

$$\Box(B \rightarrow A \vee B)$$

$$\Box B \rightarrow \Box(A \vee B)$$

$$(\Box A \vee \Box B) \rightarrow \Box(A \vee B)$$

Derivation in **LP**

Examples of derivations.

Some problems on the way from **S4**:

Derivation in **S4**

$$\begin{aligned} &A \rightarrow A \vee B \\ &\Box(A \rightarrow A \vee B) \\ &\Box A \rightarrow \Box(A \vee B) \\ &B \rightarrow A \vee B \\ &\Box(B \rightarrow A \vee B) \\ &\Box B \rightarrow \Box(A \vee B) \\ &(\Box A \vee \Box B) \rightarrow \Box(A \vee B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} &A \rightarrow A \vee B \\ &a:(A \rightarrow A \vee B) \\ &x:A \rightarrow (a \cdot x):(A \vee B) \end{aligned}$$

Examples of derivations.

Some problems on the way from **S4**:

Derivation in **S4**

$$\begin{aligned} &A \rightarrow A \vee B \\ &\Box(A \rightarrow A \vee B) \\ &\Box A \rightarrow \Box(A \vee B) \\ &B \rightarrow A \vee B \\ &\Box(B \rightarrow A \vee B) \\ &\Box B \rightarrow \Box(A \vee B) \\ &(\Box A \vee \Box B) \rightarrow \Box(A \vee B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} &A \rightarrow A \vee B \\ &a:(A \rightarrow A \vee B) \\ &x:A \rightarrow (a \cdot x):(A \vee B) \\ &B \rightarrow A \vee B \\ &b:(B \rightarrow A \vee B) \\ &y:B \rightarrow (b \cdot y):(A \vee B) \end{aligned}$$

Examples of derivations.

Some problems on the way from **S4**:

Derivation in **S4**

$$\begin{aligned} &A \rightarrow A \vee B \\ &\Box(A \rightarrow A \vee B) \\ &\Box A \rightarrow \Box(A \vee B) \\ &B \rightarrow A \vee B \\ &\Box(B \rightarrow A \vee B) \\ &\Box B \rightarrow \Box(A \vee B) \\ &(\Box A \vee \Box B) \rightarrow \Box(A \vee B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} &A \rightarrow A \vee B \\ &a:(A \rightarrow A \vee B) \\ &x:A \rightarrow (a \cdot x):(A \vee B) \\ &B \rightarrow A \vee B \\ &b:(B \rightarrow A \vee B) \\ &y:B \rightarrow (b \cdot y):(A \vee B) \end{aligned}$$

Orange parts are different, and we cannot just repeat the corresponding **S4** step. Operation \vdash is needed!

Examples of derivations.

Some problems on the way from **S4**:

Derivation in **S4**

$$\begin{aligned} & A \rightarrow A \vee B \\ & \Box(A \rightarrow A \vee B) \\ & \Box A \rightarrow \Box(A \vee B) \\ & B \rightarrow A \vee B \\ & \Box(B \rightarrow A \vee B) \\ & \Box B \rightarrow \Box(A \vee B) \\ & (\Box A \vee \Box B) \rightarrow \Box(A \vee B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} & A \rightarrow A \vee B \\ & a:(A \rightarrow A \vee B) \\ & x:A \rightarrow (a \cdot x):(A \vee B) [\rightarrow (a \cdot x + b \cdot y):(A \vee B)] \\ & B \rightarrow A \vee B \\ & b:(B \rightarrow A \vee B) \\ & y:B \rightarrow (b \cdot y):(A \vee B) [\rightarrow (a \cdot x + b \cdot y):(A \vee B)] \end{aligned}$$

Examples of derivations.

Some problems on the way from **S4**:

Derivation in **S4**

$$\begin{aligned} & A \rightarrow A \vee B \\ & \Box(A \rightarrow A \vee B) \\ & \Box A \rightarrow \Box(A \vee B) \\ & B \rightarrow A \vee B \\ & \Box(B \rightarrow A \vee B) \\ & \Box B \rightarrow \Box(A \vee B) \\ & (\Box A \vee \Box B) \rightarrow \Box(A \vee B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} & A \rightarrow A \vee B \\ & a:(A \rightarrow A \vee B) \\ & x:A \rightarrow (a \cdot x):(A \vee B) [\rightarrow (a \cdot x + b \cdot y):(A \vee B)] \\ & B \rightarrow A \vee B \\ & b:(B \rightarrow A \vee B) \\ & y:B \rightarrow (b \cdot y):(A \vee B) [\rightarrow (a \cdot x + b \cdot y):(A \vee B)] \\ & (x:A \vee y:B) \rightarrow (a \cdot x + b \cdot y):(A \vee B) \end{aligned}$$

Examples of derivations. All three operations are needed

Derivation in **S4**

$\Box A \rightarrow \Box A \vee \Box B$
 $\Box(\Box A \rightarrow \Box A \vee \Box B)$
 $\Box A \rightarrow \Box \Box A$
 $\Box \Box A \rightarrow \Box(\Box A \vee \Box B)$
 $\Box A \rightarrow \Box(\Box A \vee \Box B)$
 $\Box B \rightarrow \Box A \vee \Box B$
 $\Box(\Box B \rightarrow \Box A \vee \Box B)$
 $\Box B \rightarrow \Box \Box B$
 $\Box \Box B \rightarrow \Box(\Box A \vee \Box B)$
 $\Box B \rightarrow \Box(\Box A \vee \Box B)$
 $\Box A \vee \Box B \rightarrow \Box(\Box A \vee \Box B)$

Derivation in **LP**

Examples of derivations. All three operations are needed

Derivation in **S4**

$\Box A \rightarrow \Box A \vee \Box B$
 $\Box(\Box A \rightarrow \Box A \vee \Box B)$
 $\Box A \rightarrow \Box\Box A$
 $\Box\Box A \rightarrow \Box(\Box A \vee \Box B)$
 $\Box A \rightarrow \Box(\Box A \vee \Box B)$
 $\Box B \rightarrow \Box A \vee \Box B$
 $\Box(\Box B \rightarrow \Box A \vee \Box B)$
 $\Box B \rightarrow \Box\Box B$
 $\Box\Box B \rightarrow \Box(\Box A \vee \Box B)$
 $\Box B \rightarrow \Box(\Box A \vee \Box B)$
 $\Box A \vee \Box B \rightarrow \Box(\Box A \vee \Box B)$

Derivation in **LP**

$x:A \rightarrow x:A \vee y:B$
 $a:(x:A \rightarrow x:A \vee y:B)$
 $x:A \rightarrow !x:x:A$
 $!x:x:A \rightarrow (a \cdot !x):(x:A \vee y:B)$
 $x:A \rightarrow (a \cdot !x):(x:A \vee y:B)$

Examples of derivations. All three operations are needed

Derivation in **S4**

$$\begin{aligned} &\Box A \rightarrow \Box A \vee \Box B \\ &\Box(\Box A \rightarrow \Box A \vee \Box B) \\ &\Box A \rightarrow \Box\Box A \\ &\Box\Box A \rightarrow \Box(\Box A \vee \Box B) \\ &\Box A \rightarrow \Box(\Box A \vee \Box B) \\ &\Box B \rightarrow \Box A \vee \Box B \\ &\Box(\Box B \rightarrow \Box A \vee \Box B) \\ &\Box B \rightarrow \Box\Box B \\ &\Box\Box B \rightarrow \Box(\Box A \vee \Box B) \\ &\Box B \rightarrow \Box(\Box A \vee \Box B) \\ &\Box A \vee \Box B \rightarrow \Box(\Box A \vee \Box B) \end{aligned}$$

Derivation in **LP**

$$\begin{aligned} &x:A \rightarrow x:A \vee y:B \\ &a:(x:A \rightarrow x:A \vee y:B) \\ &x:A \rightarrow !x:x:A \\ &!x:x:A \rightarrow (a \cdot !x):(x:A \vee y:B) \\ &x:A \rightarrow (a \cdot !x):(x:A \vee y:B) \\ &y:B \rightarrow x:A \vee y:B \\ &b:(y:B \rightarrow x:A \vee y:B) \\ &y:B \rightarrow !y:y:B \\ &!y:y:B \rightarrow (b \cdot !y):(x:A \vee y:B) \\ &y:B \rightarrow (b \cdot !y):(x:A \vee y:B) \end{aligned}$$

Examples of derivations. All three operations are needed

Derivation in **S4**

$$\begin{aligned}
&\Box A \rightarrow \Box A \vee \Box B \\
&\Box(\Box A \rightarrow \Box A \vee \Box B) \\
&\Box A \rightarrow \Box\Box A \\
&\Box\Box A \rightarrow \Box(\Box A \vee \Box B) \\
&\Box A \rightarrow \Box(\Box A \vee \Box B) \\
&\Box B \rightarrow \Box A \vee \Box B \\
&\Box(\Box B \rightarrow \Box A \vee \Box B) \\
&\Box B \rightarrow \Box\Box B \\
&\Box\Box B \rightarrow \Box(\Box A \vee \Box B) \\
&\Box B \rightarrow \Box(\Box A \vee \Box B) \\
&\Box A \vee \Box B \rightarrow \Box(\Box A \vee \Box B)
\end{aligned}$$

Derivation in **LP**

$$\begin{aligned}
&x:A \rightarrow x:A \vee y:B \\
&a:(x:A \rightarrow x:A \vee y:B) \\
&x:A \rightarrow !x:x:A \\
&!x:x:A \rightarrow (a \cdot !x):(x:A \vee y:B) \\
&x:A \rightarrow (a \cdot !x):(x:A \vee y:B) [\rightarrow (a \cdot !x + b \cdot !y):(\dots)] \\
&y:B \rightarrow x:A \vee y:B \\
&b:(y:B \rightarrow x:A \vee y:B) \\
&y:B \rightarrow !y:y:B \\
&!y:y:B \rightarrow (b \cdot !y):(x:A \vee y:B) \\
&y:B \rightarrow (b \cdot !y):(x:A \vee y:B) [\rightarrow (a \cdot !x + b \cdot !y):(\dots)] \\
&x:A \vee y:B \rightarrow (a \cdot !x + b \cdot !y):(x:A \vee y:B)
\end{aligned}$$

Example of matching derivations in **S4** and in **LP**.

Derivation in **S4**

$\Box A \rightarrow \Box A \vee \Box B$
 $\Box(\Box A \rightarrow \Box A \vee \Box B)$
 $\Box A \rightarrow \Box\Box A$
 $\Box\Box A \rightarrow \Box(\Box A \vee \Box B)$
 $\Box A \rightarrow \Box(\Box A \vee \Box B)$
 $\Box B \rightarrow \Box A \vee \Box B$
 $\Box(\Box B \rightarrow \Box A \vee \Box B)$
 $\Box B \rightarrow \Box\Box B$
 $\Box\Box B \rightarrow \Box(\Box A \vee \Box B)$
 $\Box B \rightarrow \Box(\Box A \vee \Box B)$
 $\Box A \vee \Box B \rightarrow \Box(\Box A \vee \Box B)$

Derivation in **LP**

$x:A \rightarrow x:A \vee y:B$
 $a:(x:A \rightarrow x:A \vee y:B)$
 $x:A \rightarrow !x:x:A$
 $!x:x:A \rightarrow (a \cdot !x):(x:A \vee y:B)$
 $x:A \rightarrow (a \cdot !x):(x:A \vee y:B) [\rightarrow (a \cdot !x + b \cdot !y):(\dots)]$
 $y:B \rightarrow x:A \vee y:B$
 $b:(y:B \rightarrow x:A \vee y:B)$
 $y:B \rightarrow !y:y:B$
 $!y:y:B \rightarrow (b \cdot !y):(x:A \vee y:B)$
 $y:B \rightarrow (b \cdot !y):(x:A \vee y:B) [\rightarrow (a \cdot !x + b \cdot !y):(\dots)]$
 $x:A \vee y:B \rightarrow (a \cdot !x + b \cdot !y):(x:A \vee y:B)$

Realization Theorem, (S.A, 1995) (relates **S4** to **LP**):
***S4** is the forgetful projection of **LP**.*

Do not be misled by the examples above. The proof of the Realization Theorem is not at all trivial. In particular, it is based on the cut-elimination for **S4**.

Completeness w.r.t. the provability semantics, (S.A, 1995):
LP derives all identities in its own language valid in the standard provability semantics.

Corollary: Solution to Gödel's problem in Gödel'38 format

The foundational picture now looks like this:

Int \hookrightarrow **S4** \hookrightarrow **LP** \hookrightarrow *REAL PROOFS*

and all these embeddings are exact.

Comparing formats

Type (logic) derivation

(plain types - propositions)

$$A \rightarrow B, A \vdash B$$

Combinatory Logic (= λ -calculus)

(plain typed combinatory (λ -)terms, explicit, but no proof iterations allowed)

$$s:(A \rightarrow B), t:A \vdash (s \cdot t):B$$

Modal logic **S4**

(provability iterates, but is implicit)

$$\Box A \vee \Box B \vdash \Box(\Box A \vee \Box B)$$

Proof polynomial derivation

(provability is explicit

and iterates freely)

$$x:A \vee y:B \vdash (a \cdot !x + b \cdot !y):(x:A \vee y:B)$$

$$a:(x:A \rightarrow x:A \vee y:B)$$

$$b:(y:B \rightarrow x:A \vee y:B)$$

Major features that differ **LP** from **CL**/ λ -calculus.

- **Polymorphism.** Operation “+” yields multiple types, i.e. multi-conclusion proof terms.
- **Dynamic typing.** Collection of types assigned to a term can grow in a process of term building.
- **A free mixture of types from different levels.** **CL**/ λ corresponds to pure level 1 only.
- **Self-referentiality of terms.** In **LP** a type assignment $t:F(t)$ is legal and supported by the standard arithmetical semantics.
- **Internalization as an admissible rule.** In **CL**/ λ the Curry-Howard isomorphism relates level 0 (intuitionistic derivations) with the pure level 1 only.

An existential semantics for the modal logic, at last!

The intended semantics for **S4** was Gödel's provability one: $\Box F$ should be read as "there exists a proof of F ". The same kind of semantics is built-in into the knowledge (epistemic) reading of the modality: "there is an evidence of F ". We call this kind of semantics **epistemic/existential**. Prior to proof polynomials, this semantics for modal logic did not have an exact model.

A widely accepted Kripke semantics has a "for all" character: "for all accessible worlds F holds". We will suggest calling this kind of semantics **computational/universal**, since it models computational processes, which behave like Kripke structures. By a happy coincidence, this semantics (for reflexive and transitive frames) obeys the same **S4** rules.

Other developments (in more or less historical order)

0. Basic Logic of Proofs = operation free versions for major classes of proof predicates, pre-**LP** studies. Artemov, T. Strassen;
1. Functional Logic of Proofs **FLP**. V. Krupski;
2. Logical models of referential data structures. V. Krupski, Artemov;
3. Joint Logic of Proofs and Provability **LPP**. Tanya Sidon-Yavorskaya;
4. Models for **LP**. Mkrtychev;
5. Realizability of **Int** in the \dashv -free fragment of **LP**. Kopylov;
6. Explicit counterpart of modal logic **S5**. Artemov, Kazakov, Shapiro;
7. Explicit counterparts of major normal modal logics **K**, **K4**, **T**. Brezhnev;
8. Complexity of Logic of Proofs. Kuznets;
9. First order logic of proofs. Artemov, Sidon-Yavorskaya, Yavorsky;
10. Logic with quantifiers over proofs. Yavorsky;
11. Natural derivation system for **LP**. Artemov;
12. Disjunctive Property, complexity of the reflexive fragment. N. Krupski;

13. Reflexive λ -calculus. Artemov, Alt, Kuznets;
14. Explicit provability in verification theory. Artemov;
15. Reflection in **NuPri**. Alen, Artemov, Barzilay, Constable, Nogin;
16. Kripke semantics for **LP**. Fitting;
17. Tableau proof system for **LP**. Paquit, Renne;
18. Game semantics for **LP**. Dean, Paquit, Renne;
19. Epistemic Logic with justifications. Kanazawa, Paquit, Renne;
20. Reflexive Combinatory Logic. Artemov.

Here we will discuss briefly two major directions of applications of this new tool.

1. Logics of Knowledge.

2. Typed Programming Languages.

Knowledge with justifications

Logical Omniscience Problem: *An agent knows the product of two very large prime integers. In what sense does the agent know those primes? Another version: in what sense does an agent know all the tautologies?*

Research program:

Knowledge representation systems on the basis of explicit modal logics

$$p:F \sim \text{"}p \text{ is a justification for } F\text{"}$$

In such proof carrying formulas the size of a proof term corresponds to the complexity of obtaining this chunk of knowledge.

Reflexive Combinatory Logic (S.A.). *Extends typed **CL** and λ -calculi by reflexive typing. A basic model for reflection mechanism in typed theories and programming languages.*

Characteristic features of the Reflexive Combinatory Logic **RCL** are the Combinatory Logic format, a Church style rigid typing, the implicational intuitionistic logic on level 0, and the Internalization Property, which immediately captures the usual **CL** and much more.

The basic **RCL** combinators are

k: $[A \rightarrow (B \rightarrow A)]$

*old combinator **k***

s: $[(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))]$

*old combinator **s***

d: $[t:F \rightarrow F]$

DENOTATE

o: $[t:(F \rightarrow G) \rightarrow (s:F \rightarrow (t \cdot s):G)]$

INTERPRETER

c: $[t:F \rightarrow !t:(t:F)]$

CODING

The set-theoretical semantics of types, where functional types are interpreted as sets of functions, naturally extends to **RCL**. In this semantics some objects have constructive counterparts *names*, e.g. computable functions have names, which are programs that compute them. $t:F$ is interpreted as a name (program) of type F . Under this reading

d: $[t:F \rightarrow F]$ - realizes a fundamental denotational correspondence *name - object*, in particular, *program - function*.

o: $[t:(F \rightarrow G) \rightarrow (s:F \rightarrow (t \cdot s):G)]$ represents an interpreter, which maps a program t and input s to the result $t \cdot s$.

c: $[t:F \rightarrow !t:(t:F)]$ maps a program into its code (alias, name, etc.).

Internalization Rule is admissible in **RCL**:

If $A_1, \dots, A_n \vdash B$, then for some fresh variables x_1, x_2, \dots, x_n and an appropriate term $t(x_1, \dots, x_n)$

$$x_1:A_1, \dots, x_n:A_n \vdash t(x_1, \dots, x_n):B$$

The famous Curry-Howard isomorphism covers only a very simple special case of this rule when A_1, A_2, \dots, A_n, B are boolean formulas, i.e. do not contain terms.