An introduction to classical realizability

Alexandre Miquel

UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

FACULTAD DE INGENIERIA

EQUIPO DE LÓGICA UDELAR

January 27th, 2017 – EJCIM'17 – Lyon

The Curry-Howard correspondence

The dictionary:

Proof theory	**Functional programming**

The Curry-Howard correspondence

The dictionary:

Proof theory	**Functional programming**
Proposition (formula)	Data type
Proof (derivation)	Program (or data)

The Curry-Howard correspondence

The dictionary:

Proof theory	**Functional programming**
Proposition (formula)	Data type
Proof (derivation)	Program (or data)
p is a proof of the formula A	p is a program of type A

The Curry-Howard correspondence

The dictionary:

Proof theory	**Functional programming**
Proposition (formula)	Data type
Proof (derivation)	Program (or data)
p is a proof of the formula A	p is a program of type A
$A \wedge B$,	$A \times B$,

The Curry-Howard correspondence

The dictionary:

Proof theory	**Functional programming**
Proposition (formula) Proof (derivation)	Data type Program (or data)
p is a proof of the formula A	p is a program of type A
$A \wedge B, \quad A \vee B,$	$A \times B, \quad A + B,$

The Curry-Howard correspondence

The dictionary:

Proof theory	**Functional programming**
Proposition (formula)	Data type
Proof (derivation)	Program (or data)
p is a proof of the formula A	p is a program of type A
$A \wedge B, \quad A \vee B, \quad A \Rightarrow B$	$A \times B, \quad A + B, \quad A \to B$

The Curry-Howard correspondence

The dictionary:

Proof theory	**Functional programming**
Proposition (formula)	Data type
Proof (derivation)	Program (or data)
p is a proof of the formula A	p is a program of type A
$A \wedge B, \quad A \vee B, \quad A \Rightarrow B$	$A \times B, \quad A + B, \quad A \to B$
Deduction rule	Typing rule
Proof checker	Type checker

The Curry-Howard correspondence

The dictionary:

Proof theory	**Functional programming**
Proposition (formula)	Data type
Proof (derivation)	Program (or data)
p is a proof of the formula A	p is a program of type A
$A \wedge B, \quad A \vee B, \quad A \Rightarrow B$	$A \times B, \quad A + B, \quad A \to B$
Deduction rule	Typing rule
Proof checker	Type checker
Cut elimination	Computation
Cut-free proof	Value

The Curry-Howard correspondence

The dictionary:

Proof theory	**Functional programming**
Proposition (formula)	Data type
Proof (derivation)	Program (or data)
p is a proof of the formula A	p is a program of type A
$A \wedge B, \quad A \vee B, \quad A \Rightarrow B$	$A \times B, \quad A + B, \quad A \rightarrow B$
Deduction rule	Typing rule
Proof checker	Type checker
Cut elimination	Computation
Cut-free proof	Value
Proof of a lemma	Sub-program
Theory (statements & proofs)	Module (interface & implem.)

Core language: the λ-calculus [Church'41]

- A universal language of functions
- Only three constructions: variable, abstraction, application:

Language	**Var.**	**Abstraction**	**Application**
λ-calculus	x	$\lambda x\,.\,\langle expr\rangle$	$\langle expr\rangle\ \langle expr\rangle$
Math.	x	$x \mapsto \langle expr\rangle$	$f(\langle expr\rangle)$
LISP	x	(`lambda` (x) $\langle expr\rangle$)	$(\langle expr\rangle\ \langle expr\rangle)$
Python	x	`lambda` x : $\langle expr\rangle$	$\langle expr\rangle(\langle expr\rangle)$

Core language: the λ-calculus [Church'41]

- A universal language of functions
- Only three constructions: variable, abstraction, application:

Language	**Var.**	**Abstraction**	**Application**
λ-calculus	x	$\lambda x\,.\,\langle expr\rangle$	$\langle expr\rangle\ \langle expr\rangle$
Math.	x	$x \mapsto \langle expr\rangle$	$f(\langle expr\rangle)$
LISP	x	`(lambda` $(x)\ \langle expr\rangle$`)`	$(\langle expr\rangle\ \langle expr\rangle)$
Python	x	`lambda` $x : \langle expr\rangle$	$\langle expr\rangle(\langle expr\rangle)$

- Computation rule = β-reduction

$$\begin{array}{rcl}(\lambda x\,.\,x+x+18)(3\times 4) & \succ_\beta & (3\times 4)+(3\times 4)+18 \\ & \succ & \cdots \quad \succ \quad 42\end{array}$$

- Formally: $(\lambda x\,.\,t)\ u \quad \succ_\beta \quad t\{x := u\}$

From proofs to programs

$$\frac{}{\forall x\,(P(x) \Rightarrow Q(x)) \;\Rightarrow\; \forall x\,(Q(x) \Rightarrow R(x)) \;\Rightarrow\; \forall x\,(P(x) \Rightarrow R(x))}$$

From proofs to programs

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[\forall x\,(Q(x)\Rightarrow R(x))]}}{Q(x)\Rightarrow R(x)} \qquad \dfrac{\dfrac{\dfrac{}{[\forall x\,(P(x)\Rightarrow Q(x))]}}{P(x)\Rightarrow Q(x)} \qquad \dfrac{}{[P(x)]}}{Q(x)}}{R(x)}}{P(x)\Rightarrow R(x)}}{\forall x\,(P(x)\Rightarrow R(x))}}{\forall x\,(Q(x)\Rightarrow R(x))\ \Rightarrow\ \forall x\,(P(x)\Rightarrow R(x))}}{\forall x\,(P(x)\Rightarrow Q(x))\ \Rightarrow\ \forall x\,(Q(x)\Rightarrow R(x))\ \Rightarrow\ \forall x\,(P(x)\Rightarrow R(x))}$$

From proofs to programs

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[\forall x\,(Q(x)\Rightarrow R(x))]}\;\text{Ax.}}{Q(x)\Rightarrow R(x)}\;\forall\text{-elim} \qquad \dfrac{\dfrac{\dfrac{}{[\forall x\,(P(x)\Rightarrow Q(x))]}\;\text{Ax.}}{P(x)\Rightarrow Q(x)}\;\forall\text{-elim} \qquad \dfrac{}{[P(x)]}\;\text{Ax.}}{Q(x)}\;\Rightarrow\text{-elim}}{R(x)}\;\Rightarrow\text{-elim}}{P(x)\Rightarrow R(x)}\;\Rightarrow\text{-intro}}{\forall x\,(P(x)\Rightarrow R(x))}\;\forall\text{-intro}}{\forall x\,(Q(x)\Rightarrow R(x))\;\Rightarrow\;\forall x\,(P(x)\Rightarrow R(x))}\;\Rightarrow\text{-intro}}{\forall x\,(P(x)\Rightarrow Q(x))\;\Rightarrow\;\forall x\,(Q(x)\Rightarrow R(x))\;\Rightarrow\;\forall x\,(P(x)\Rightarrow R(x))}\;\Rightarrow\text{-intro}$$

From proofs to programs

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[\forall x\,(Q(x)\Rightarrow R(x))]}\,g}{Q(x)\Rightarrow R(x)} \qquad \dfrac{\dfrac{\dfrac{}{[\forall x\,(P(x)\Rightarrow Q(x))]}\,f}{P(x)\Rightarrow Q(x)} \qquad \dfrac{}{[P(x)]}\,u}{Q(x)}\,@}{R(x)}\,@}{P(x)\Rightarrow R(x)}\,\lambda u}{\forall x\,(P(x)\Rightarrow R(x))}}{\forall x\,(Q(x)\Rightarrow R(x)) \ \Rightarrow\ \forall x\,(P(x)\Rightarrow R(x))}\,\lambda g}{\forall x\,(P(x)\Rightarrow Q(x)) \ \Rightarrow\ \forall x\,(Q(x)\Rightarrow R(x)) \ \Rightarrow\ \forall x\,(P(x)\Rightarrow R(x))}\,\lambda f
$$

From proofs to programs

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[\forall x\,(Q(x)\Rightarrow R(x))]}\,g}{Q(x)\Rightarrow R(x)} \qquad \dfrac{\dfrac{\dfrac{}{[\forall x\,(P(x)\Rightarrow Q(x))]}\,f}{P(x)\Rightarrow Q(x)} \qquad \dfrac{}{[P(x)]}\,u}{Q(x)}\,@}{R(x)}\,@}{P(x)\Rightarrow R(x)}\,\lambda u}{\forall x\,(P(x)\Rightarrow R(x))}}{\forall x\,(Q(x)\Rightarrow R(x))\ \Rightarrow\ \forall x\,(P(x)\Rightarrow R(x))}\,\lambda g}{\forall x\,(P(x)\Rightarrow Q(x))\ \Rightarrow\ \forall x\,(Q(x)\Rightarrow R(x))\ \Rightarrow\ \forall x\,(P(x)\Rightarrow R(x))}\,\lambda f
$$

$$\lambda f\,.\,\lambda g\,.\,\lambda u\,.\,g\,(f\,u)$$

Significance of the Curry-Howard correspondence

- Theoretical impact on:
  - Proof theory
  - Constructive mathematics
  - Category theory
  - Denotational semantics
  - Functional programming

- Theoretical by-products:
  - Type theory (Martin-Löf),
  - Linear logic (Girard)

Significance of the Curry-Howard correspondence

- Theoretical impact on:
  - Proof theory
  - Constructive mathematics
  - Category theory
  - Denotational semantics
  - Functional programming
- Theoretical by-products:
  - Type theory (Martin-Löf),
  - Linear logic (Girard)
- Applications:
  - Proof assistants: Coq, Agda
  - Program certification
  - Program extraction

From intuitionistic logic to classical logic (1/2)

- For a long time, the Curry-Howard correspondence was limited to intuitionistic logic and constructive mathematics, since it was (thought to be) incompatible with classical reasoning principles, such as for instance:
    - The law of excluded middle: $A \vee \neg A$
    - Double-negation elimination: $\neg\neg A \Rightarrow A$
    - Reductio ad absurdum: from the absurdity of $\neg A$, deduce A
    - Most De Morgan laws, e.g.: $\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$
    - Peirce's law: $((A \Rightarrow B) \Rightarrow A) \Rightarrow A$
    - The full axiom of choice

- However, *a lot* of interesting mathematics can be formalized in intuitionistic logic (i.e. without using classical reasoning)

From intuitionistic logic to classical logic (2/2)

- In 1990, Griffin's discovered a connection between classical reasoning and control operators (call/cc)

$$\mathrm{call/cc} \ : \ ((A \Rightarrow B) \Rightarrow A) \Rightarrow A \qquad\qquad \text{(Peirce's law)}$$

- A new paradigm for the Curry-Howard correspondence:

$$\begin{array}{rcl} \text{Classical reasoning} & = & \text{programming with continuations} \\ & = & \text{computing by trial/error} \end{array}$$

From intuitionistic logic to classical logic (2/2)

- In 1990, Griffin's discovered a connection between classical reasoning and control operators (call/cc)

$$\text{call/cc} \;:\; ((A \Rightarrow B) \Rightarrow A) \Rightarrow A \qquad \text{(Peirce's law)}$$

- A new paradigm for the Curry-Howard correspondence:

$$\begin{array}{rcl} \text{Classical reasoning} & = & \text{programming with continuations} \\ & = & \text{computing by trial/error} \end{array}$$

- Many classical λ-calculi:
    - $\lambda\mu$ [Parigot 1992]
    - λ-sym [Barbanera & Berardi 1996]
    - λ_c [Krivine 1994]
    - $\bar{\lambda}\mu\tilde{\mu}$ [Curien & Herbelin 2000]

From intuitionistic logic to classical logic (2/2)

- In 1990, Griffin's discovered a connection between classical reasoning and control operators (call/cc)

$$\text{call/cc} \;:\; ((A \Rightarrow B) \Rightarrow A) \Rightarrow A \qquad\qquad \text{(Peirce's law)}$$

- A new paradigm for the Curry-Howard correspondence:

$$\begin{aligned} \text{Classical reasoning} \quad &= \quad \text{programming with continuations} \\ &= \quad \text{computing by trial/error} \end{aligned}$$

- Many classical λ-calculi:
    - $\lambda\mu$ [Parigot 1992]
    - λ-sym [Barbanera & Berardi 1996]
    - λ_c [Krivine 1994]
    - $\bar{\lambda}\mu\tilde{\mu}$ [Curien & Herbelin 2000]
- Classical realizability [Krivine '00, '03, '09]

What is classical realizability?

- An operational semantics for the programs extracted from classical proofs, formulated using the tools of model theory
  - Based on the connection between Peirce's law and call/cc
  - Allows to predict the behavior of classical programs
  - Interprets the Axiom of Dependent Choices (DC) [K. 2003]

What is classical realizability?

- An operational semantics for the programs extracted from classical proofs, formulated using the tools of model theory
  - Based on the connection between Peirce's law and call/cc
  - Allows to predict the behavior of classical programs
  - Interprets the Axiom of Dependent Choices (DC) [K. 2003]
- Initially designed for PA2, but extends to:
  - Higher-order arithmetic (PAω)
  - Zermelo-Fraenkel set theory (ZF) [K. 2001, 2012]
  - The calculus of inductive constructions (CIC) [M. 2007]
    (with classical logic in Prop)

What is classical realizability?

- An operational semantics for the programs extracted from classical proofs, formulated using the tools of model theory
  - Based on the connection between Peirce's law and call/cc
  - Allows to predict the behavior of classical programs
  - Interprets the Axiom of Dependent Choices (DC) [K. 2003]
- Initially designed for PA2, but extends to:
  - Higher-order arithmetic (PAω)
  - Zermelo-Fraenkel set theory (ZF) [K. 2001, 2012]
  - The calculus of inductive constructions (CIC) [M. 2007]
    (with classical logic in Prop)
- Deep connections with Cohen forcing [K. 2011]

$\rightsquigarrow$ can be used to define new models of PA2/ZF [K. 2012]

Plan

Plan

1. Introduction
2. Second-order arithmetic (PA2)
3. Extracted programs
4. The classical realizability interpretation
5. Witness extraction

The language of (minimal) second-order logic

- Second-order logic deals with two kinds of objects:
  - 1st-order objects = individuals (i.e. basic objects of the theory)
  - 2nd-order objects = k-ary relations over individuals

First-order terms and formulas

First-order terms $e, e' \;::=\; x \;\mid\; f(e_1, \ldots, e_k)$

Formulas $A, B \;::=\; X(e_1, \ldots, e_k) \;\mid\; A \Rightarrow B \;\mid\; \forall x\, A \;\mid\; \forall X\, A$

- Two kinds of variables
  - 1st-order variables: $x, y, z, \ldots$
  - 2nd-order variables: $X, Y, Z, \ldots$ of all arities $k \geq 0$

The language of (minimal) second-order logic

- Second-order logic deals with two kinds of objects:
  - 1st-order objects = individuals (i.e. basic objects of the theory)
  - 2nd-order objects = k-ary relations over individuals

First-order terms and formulas

First-order terms $\quad e, e' \quad ::= \quad x \quad | \quad f(e_1, \dots, e_k)$

Formulas $\quad A, B \quad ::= \quad X(e_1, \dots, e_k) \quad | \quad A \Rightarrow B \quad | \quad \forall x\, A \quad | \quad \forall X\, A$

- Two kinds of variables
  - 1st-order variables: $x, y, z, \dots$
  - 2nd-order variables: $X, Y, Z, \dots$ of all arities $k \geq 0$

- **2nd-order arithmetic:** individuals represent natural numbers

First-order terms (1/2)

- Defined from a first-order signature Σ (as usual):

First-order terms $\quad e, e' \quad ::= \quad x \quad | \quad f(e_1, \ldots, e_k)$

  - f ranges over k-ary function symbols in Σ
  - constant symbol $=$ function symbol of arity 0

First-order terms (1/2)

- Defined from a first-order signature Σ (as usual):

First-order terms $\quad e, e' \quad ::= \quad x \quad | \quad f(e_1, \ldots, e_k)$

  - f ranges over k-ary function symbols in Σ
  - constant symbol = function symbol of arity 0

- In what follows we assume that the signature Σ contains:
  - a constant symbol 0 (zero)
  - a unary function symbol s (successor)
  - binary function symbols $+$, $\times$, $-$ (truncated subtraction)
  - function symbols for all primitive recursive functions (more generally)

First-order terms (1/2)

- Defined from a first-order signature Σ (as usual):

First-order terms $e, e' \quad ::= \quad x \quad | \quad f(e_1, \ldots, e_k)$

  - f ranges over k-ary function symbols in Σ
  - constant symbol $=$ function symbol of arity 0

- In what follows we assume that the signature Σ contains:
  - a constant symbol 0 (zero)
  - a unary function symbol s (successor)
  - binary function symbols $+$, $\times$, $-$ (truncated subtraction)
  - function symbols for all primitive recursive functions (more generally)

- Peano numerals: $\underbrace{s(\cdots s}_{n}(0)\cdots)$ written n $(n \in \mathbb{N})$

First-order terms (1/2)

- Defined from a first-order signature Σ (as usual):

First-order terms $\qquad e, e' \quad ::= \quad x \quad | \quad f(e_1, \ldots, e_k)$

  - f ranges over k-ary function symbols in Σ
  - constant symbol = function symbol of arity 0

- In what follows we assume that the signature Σ contains:
  - a constant symbol 0 (zero)
  - a unary function symbol s (successor)
  - binary function symbols $+$, $\times$, $-$ (truncated subtraction)
  - function symbols for all primitive recursive functions (more generally)

- Peano numerals: $\underbrace{s(\cdots s}_{n}(0)\cdots)$ written n $\qquad (n \in \mathbb{N})$

- First-order substitution written: $e\{x := e'\}$

First-order terms (2/2)

- Each k-ary function symbol f is interpreted by the corresponding primitive recursive function, written

$$f^{\mathbb{N}} \;:\; \mathbb{N}^k \to \mathbb{N}$$

The constant symbol 0 is interpreted by $\quad 0^{\mathbb{N}} = 0 \ (\in \mathbb{N})$

First-order terms (2/2)

- Each k-ary function symbol f is interpreted by the corresponding primitive recursive function, written

  $$f^{\mathbb{N}} \ : \ \mathbb{N}^k \to \mathbb{N}$$

  The constant symbol 0 is interpreted by $0^{\mathbb{N}} = 0 \ (\in \mathbb{N})$

- The denotation in $\mathbb{N}$ (i.e. the value) of a closed first-order term e is written $e^{\mathbb{N}}$. For instance:

  $$\begin{array}{rcl} 0^{\mathbb{N}} & = & 0 \\ 4^{\mathbb{N}} \ = \ (s(s(s(s(0)))))^{\mathbb{N}} & = & 4 \\ ((2-3)+s(3\times 4))^{\mathbb{N}} & = & 13 \end{array}$$

Formulas (1/2)

- Formulas of minimal second-order logic

Formulas

$$A, B \quad ::= \quad X(e_1, \ldots, e_k) \quad | \quad A \Rightarrow B \quad | \quad \forall x\, A \quad | \quad \forall X\, A$$

only based on implication and 1st/2nd-order universal quantification

Formulas (1/2)

- Formulas of minimal second-order logic

Formulas

$$A, B \quad ::= \quad X(e_1, \ldots, e_k) \quad | \quad A \Rightarrow B \quad | \quad \forall x\, A \quad | \quad \forall X\, A$$

only based on implication and 1st/2nd-order universal quantification

- Implication is right-associative:

$$A_1 \Rightarrow \cdots \Rightarrow A_n \Rightarrow B \quad \text{means} \quad A_1 \Rightarrow (\cdots \Rightarrow (A_n \Rightarrow B) \cdots)$$

The above formula is equivalent to $(A_1 \wedge \cdots \wedge A_n) \Rightarrow B$
but without using conjunction

Formulas (1/2)

- Formulas of minimal second-order logic

Formulas $$\begin{array}{rcl} A, B & ::= & X(e_1, \ldots, e_k) \quad | \quad A \Rightarrow B \\ & | & \forall x\, A \quad | \quad \forall X\, A \end{array}$$

only based on implication and 1st/2nd-order universal quantification

- Implication is right-associative:

$$A_1 \Rightarrow \cdots \Rightarrow A_n \Rightarrow B \quad \text{means} \quad A_1 \Rightarrow (\cdots \Rightarrow (A_n \Rightarrow B) \cdots)$$

The above formula is equivalent to $(A_1 \wedge \cdots \wedge A_n) \Rightarrow B$
but without using conjunction

- Two kinds of substitutions:
  - 1st-order substitution, written $A\{x := e\}$ (capture avoiding)
  - 2nd-order substitution, written $A\{X := P\}$ (postponed)

Formulas (2/2)

- Other connectives/quantifiers defined via second-order encodings:

$$\begin{array}{rcll}
\bot & \equiv & \forall Z\, Z & \text{(absurdity)} \\
\neg A & \equiv & A \Rightarrow \bot & \text{(negation)} \\[1ex]
A \wedge B & \equiv & \forall Z\, ((A \Rightarrow B \Rightarrow Z) \Rightarrow Z) & \text{(conjunction)} \\
A \vee B & \equiv & \forall Z\, ((A \Rightarrow Z) \Rightarrow (B \Rightarrow Z) \Rightarrow Z) & \text{(disjunction)} \\[1ex]
A \Leftrightarrow B & \equiv & (A \Rightarrow B) \wedge (B \Rightarrow A) & \text{(equivalence)} \\[1ex]
\exists x\, A(x) & \equiv & \forall Z\, (\forall x\, (A(x) \Rightarrow Z) \Rightarrow Z) & \text{(1st-order } \exists) \\
\exists X\, A(X) & \equiv & \forall Z\, (\forall X\, (A(X) \Rightarrow Z) \Rightarrow Z) & \text{(2nd-order } \exists) \\[1ex]
e_1 = e_2 & \equiv & \forall Z\, (Z(e_1) \Rightarrow Z(e_2)) & \text{(Leibniz equality)}
\end{array}$$

Formulas (2/2)

- Other connectives/quantifiers defined via second-order encodings:

$$\begin{array}{rcll}
\bot & \equiv & \forall Z\, Z & \text{(absurdity)} \\
\neg A & \equiv & A \Rightarrow \bot & \text{(negation)} \\
A \wedge B & \equiv & \forall Z\, ((A \Rightarrow B \Rightarrow Z) \Rightarrow Z) & \text{(conjunction)} \\
A \vee B & \equiv & \forall Z\, ((A \Rightarrow Z) \Rightarrow (B \Rightarrow Z) \Rightarrow Z) & \text{(disjunction)} \\
A \Leftrightarrow B & \equiv & (A \Rightarrow B) \wedge (B \Rightarrow A) & \text{(equivalence)} \\
\exists x\, A(x) & \equiv & \forall Z\, (\forall x\, (A(x) \Rightarrow Z) \Rightarrow Z) & \text{(1st-order } \exists) \\
\exists X\, A(X) & \equiv & \forall Z\, (\forall X\, (A(X) \Rightarrow Z) \Rightarrow Z) & \text{(2nd-order } \exists) \\
e_1 = e_2 & \equiv & \forall Z\, (Z(e_1) \Rightarrow Z(e_2)) & \text{(Leibniz equality)}
\end{array}$$

- We could also have used the De Morgan laws

$$\begin{array}{rclcrcl}
A \wedge B & \equiv & \neg(A \Rightarrow B \Rightarrow \bot) & \qquad & \exists x\, A(x) & \equiv & \neg \forall x\, \neg A(x) \\
A \vee B & \equiv & \neg A \Rightarrow \neg B \Rightarrow \bot & \qquad & \exists X\, A(x) & \equiv & \neg \forall X\, \neg A(X)
\end{array}$$

that are classically equivalent

Predicates

- Concrete relations are represented using predicates (syntactic sugar)

Predicates $P, Q \quad ::= \quad \hat{x}_1 \cdots \hat{x}_k A_0$ (of arity k)

Predicates

- Concrete relations are represented using predicates (syntactic sugar)

Predicates $P, Q \quad ::= \quad \hat{x}_1 \cdots \hat{x}_k A_0$ (of arity k)

Definition (Predicate application and 2nd-order substitution)

1. $P(e_1, \ldots, e_k)$ is the formula defined by

$$P(e_1, \ldots, e_k) \;\equiv\; A_0\{x_1 := e_1, \ldots, x_k := e_k\}$$

where $P \equiv \hat{x}_1 \cdots \hat{x}_k A_0$, and where $e_1, \ldots, e_k$ are k first-order terms

Predicates

- Concrete relations are represented using predicates (syntactic sugar)

Predicates $P, Q \quad ::= \quad \hat{x}_1 \cdots \hat{x}_k A_0$ (of arity k)

Definition (Predicate application and 2nd-order substitution)

1. $P(e_1, \ldots, e_k)$ is the formula defined by

$$P(e_1, \ldots, e_k) \;\equiv\; A_0\{x_1 := e_1, \ldots, x_k := e_k\}$$

where $P \equiv \hat{x}_1 \cdots \hat{x}_k A_0$, and where $e_1, \ldots, e_k$ are k first-order terms

2. 2nd-order substitution $A\{X := P\}$ (where X and P are of the same arity k) consists to replace in the formula A every atomic sub-formula of the form

$$X(e_1, \ldots, e_k) \qquad \text{by the formula} \qquad P(e_1, \ldots, e_k)$$

Predicates

- Concrete relations are represented using predicates (syntactic sugar)

Predicates $P, Q \quad ::= \quad \hat{x}_1 \cdots \hat{x}_k A_0$ (of arity k)

Definition (Predicate application and 2nd-order substitution)

1. $P(e_1, \ldots, e_k)$ is the formula defined by

$$P(e_1, \ldots, e_k) \;\equiv\; A_0\{x_1 := e_1, \ldots, x_k := e_k\}$$

where $P \equiv \hat{x}_1 \cdots \hat{x}_k A_0$, and where $e_1, \ldots, e_k$ are k first-order terms

2. 2nd-order substitution $A\{X := P\}$ (where X and P are of the same arity k) consists to replace in the formula A every atomic sub-formula of the form

$$X(e_1, \ldots, e_k) \qquad \text{by the formula} \qquad P(e_1, \ldots, e_k)$$

- **Note:** Every k-ary 2nd-order variable X can be seen as a predicate:

$$X \;\equiv\; \hat{x}_1 \cdots \hat{x}_k X(x_1, \ldots, x_k)$$

Unary predicates as sets

- Unary predicates represent sets of individuals

  Syntactic sugar: $\{x : A\} \equiv \hat{x}A, \qquad e \in P \equiv P(e)$

Example: The set N of Dedekind numerals

$$\mathbf{N} \equiv \{x : \forall Z\,(0 \in Z \Rightarrow \forall y\,(y \in Z \Rightarrow s(y) \in Z) \Rightarrow x \in Z\}$$

Unary predicates as sets

- Unary predicates represent sets of individuals

  Syntactic sugar: $\{x : A\} \equiv \hat{x}A, \qquad e \in P \equiv P(e)$

Example: The set N of Dedekind numerals

$$\mathbf{N} \equiv \{x : \forall Z\,(0 \in Z \Rightarrow \forall y\,(y \in Z \Rightarrow s(y) \in Z) \Rightarrow x \in Z\}$$

- Relativized quantifications:

$$\begin{aligned}(\forall x \in P)\,A(x) &\equiv \forall x\,(x \in P \Rightarrow A(x))\\ (\exists x \in P)\,A(x) &\equiv \forall Z\,(\forall x\,(x \in P \Rightarrow A(x) \Rightarrow Z) \Rightarrow Z)\\ &\Leftrightarrow \exists x\,(x \in P \wedge A(x))\end{aligned}$$

Unary predicates as sets

- Unary predicates represent sets of individuals

  Syntactic sugar: $\{x : A\} \equiv \hat{x}A, \qquad e \in P \equiv P(e)$

Example: The set $\mathbf{N}$ of Dedekind numerals

$$\mathbf{N} \equiv \{x : \forall Z\,(0 \in Z \Rightarrow \forall y\,(y \in Z \Rightarrow s(y) \in Z) \Rightarrow x \in Z\}$$

- Relativized quantifications:

$$\begin{array}{rcl} (\forall x \in P)\, A(x) & \equiv & \forall x\,(x \in P \Rightarrow A(x)) \\ (\exists x \in P)\, A(x) & \equiv & \forall Z\,(\forall x\,(x \in P \Rightarrow A(x) \Rightarrow Z) \Rightarrow Z) \\ & \Leftrightarrow & \exists x\,(x \in P \wedge A(x)) \end{array}$$

- Inclusion and extensional equality:

$$\begin{array}{rcl} P \subseteq Q & \equiv & \forall x\,(x \in P \Rightarrow x \in Q) \\ P = Q & \equiv & \forall x\,(x \in P \Leftrightarrow x \in Q) \end{array}$$

- Set constructors: $P \cup Q \equiv \{x : x \in P \vee x \in Q\}$ (etc.)

Sequents

Definition (Sequent)

A sequent is a pair of the form

$$A_1, \ldots, A_n \vdash A \qquad (n \geq 0)$$

where $A_1, \ldots, A_n, A$ are formulas

- $A_1, \ldots, A_n$ are the hypotheses, which form the context
- A is the thesis
- $\vdash$ is the entailment symbol (that reads: 'entails')

- Sequents are usually written $\Gamma \vdash A$ (Γ finite list of formulas)
- $\Gamma \vdash A$ means: "under the hypotheses in Γ, the formula A holds"
- Notations $FV(\Gamma)$, $\Gamma\{x := t\}$ extended to finite lists Γ

Rules of inference & systems of deduction

Definition (Rule of inference)

A rule of inference is a pair formed by a finite set of sequents $\{\Gamma_1 \vdash A_1, \ldots, \Gamma_n \vdash A_n\}$ and a sequent $\Gamma \vdash A$, usually written

$$\frac{\Gamma_1 \vdash A_1 \quad \cdots \quad \Gamma_n \vdash A_n}{\Gamma \vdash A}$$

- $\Gamma_1 \vdash A_1, \ldots, \Gamma_n \vdash A_n$ are the premises of the rule $(n \geq 0)$
- $\Gamma \vdash A$ is the conclusion of the rule

Definition (System of deduction)

A system of deduction is a set of inference rules

Natural deduction for classical 2nd-order logic (NK2)

- Here, we work in system NK2, whose deduction rules are:

(Axiom)
$$\frac{}{\Gamma \vdash A}\ \text{if } A\in\Gamma$$

($\Rightarrow$-intro,elim)
$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \qquad \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

($\forall^1$-intro,elim)
$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x\, A}\ \text{if } x\notin FV(\Gamma) \qquad \frac{\Gamma \vdash \forall x\, A}{\Gamma \vdash A\{x := e\}}$$

($\forall^2$-intro,elim)
$$\frac{\Gamma \vdash A}{\Gamma \vdash \forall X\, A}\ \text{if } X\notin FV(\Gamma) \qquad \frac{\Gamma \vdash \forall X\, A}{\Gamma \vdash A\{X := P\}}$$

(Peirce's law)
$$\frac{}{\Gamma \vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$$

System NK2 contains the usual rules of intuitionistic 2nd-order logic (NJ2), plus Peirce's law, for classical reasoning

Derivations

- Deduction rules are the elementary bricks of reasoning. They can be assembled to form derivations (finite sequent-labelled trees)

Example: derivation of the syllogism Barbara

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash \forall x\,(Q(x) \Rightarrow R(x))}\,(\text{axiom})}{\Gamma_3 \vdash Q(x) \Rightarrow R(x)}\,(\forall^1\text{-elim}) \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash \forall x\,(P(x) \Rightarrow Q(x))}\,(\text{axiom})}{\Gamma_3 \vdash P(x) \Rightarrow Q(x)}\,(\forall^1\text{-elim}) \qquad \dfrac{}{\Gamma_3 \vdash P(x)}\,(\text{axiom})}{\Gamma_3 \vdash Q(x)}\,(\Rightarrow\text{-elim})}{\Gamma_3 \vdash R(x)}\,(\Rightarrow\text{-elim})}{\Gamma_2 \vdash P(x) \Rightarrow R(x)}\,(\Rightarrow\text{-intro})}{\Gamma_2 \vdash \forall x\,(P(x) \Rightarrow R(x))}\,(\forall^1\text{-intro})}{\Gamma_1 \vdash \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}\,(\Rightarrow\text{-intro})}{\vdash \forall x\,(P(x) \Rightarrow Q(x)) \Rightarrow \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}\,(\Rightarrow\text{-intro})
$$

with $\Gamma_1 \equiv \forall x\,(P(x) \Rightarrow Q(x)),\quad \Gamma_2 \equiv \Gamma_1, \forall x\,(Q(x) \Rightarrow R(x)),\quad \Gamma_3 \equiv \Gamma_2, P(x)$

Derivations

- Deduction rules are the elementary bricks of reasoning. They can be assembled to form derivations (finite sequent-labelled trees)

Example: derivation of the syllogism Barbara

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash \forall x\,(Q(x) \Rightarrow R(x))}\,(\text{axiom})}{\Gamma_3 \vdash Q(x) \Rightarrow R(x)}\,(\forall^1\text{-elim}) \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash \forall x\,(P(x) \Rightarrow Q(x))}\,(\text{axiom})}{\Gamma_3 \vdash P(x) \Rightarrow Q(x)}\,(\forall^1\text{-elim}) \qquad \dfrac{}{\Gamma_3 \vdash P(x)}\,(\text{axiom})}{\Gamma_3 \vdash Q(x)}\,(\Rightarrow\text{-elim})}{\Gamma_3 \vdash R(x)}\,(\Rightarrow\text{-elim})}{\Gamma_2 \vdash P(x) \Rightarrow R(x)}\,(\Rightarrow\text{-intro})}{\Gamma_2 \vdash \forall x\,(P(x) \Rightarrow R(x))}\,(\forall^1\text{-intro})}{\Gamma_1 \vdash \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}\,(\Rightarrow\text{-intro})}{\vdash \forall x\,(P(x) \Rightarrow Q(x)) \Rightarrow \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}\,(\Rightarrow\text{-intro})
$$

with $\Gamma_1 \equiv \forall x\,(P(x) \Rightarrow Q(x)),\quad \Gamma_2 \equiv \Gamma_1, \forall x\,(Q(x) \Rightarrow R(x)),\quad \Gamma_3 \equiv \Gamma_2, P(x)$

- A sequent $\Gamma \vdash A$ is derivable when it appears as the conclusion of a derivation. A formula A is derivable when the sequent $\vdash A$ is
- **Remark:** One also uses proof/provable for derivation/derivable

Expressiveness

The 8 deduction rules of system NK2 allow us to derive the usual rules of logic (for all connectives & quantifiers):

Expressiveness

The 8 deduction rules of system NK2 allow us to derive the usual rules of logic (for all connectives & quantifiers):

- Introduction/elimination rules for defined connectives/quantifiers:

$$\begin{gathered}
\bot \Rightarrow A, \qquad A \Rightarrow B \Rightarrow A \wedge B, \qquad A \wedge B \Rightarrow A, \qquad A \wedge B \Rightarrow B, \\
A \Rightarrow A \vee B, \qquad B \Rightarrow A \vee B, \qquad (A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow A \vee B \Rightarrow C, \\
A(e) \Rightarrow \exists x\, A(x), \qquad \forall x\, (A(x) \Rightarrow C) \Rightarrow \exists x\, A(x) \Rightarrow C, \\
A(P) \Rightarrow \exists X\, A(X), \qquad \forall X\, (A(X) \Rightarrow C) \Rightarrow \exists X\, A(X) \Rightarrow C, \\
e = e, \qquad e_1 = e_2 \Rightarrow e_2 = e_1, \qquad e_1 = e_2 \Rightarrow e_2 = e_3 \Rightarrow e_1 = e_3, \qquad \text{etc.}
\end{gathered}$$

Expressiveness

The 8 deduction rules of system NK2 allow us to derive the usual rules of logic (for all connectives & quantifiers):

- Introduction/elimination rules for defined connectives/quantifiers:

$$\bot \Rightarrow A, \qquad A \Rightarrow B \Rightarrow A \wedge B, \qquad A \wedge B \Rightarrow A, \qquad A \wedge B \Rightarrow B,$$
$$A \Rightarrow A \vee B, \qquad B \Rightarrow A \vee B, \qquad (A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow A \vee B \Rightarrow C,$$
$$A(e) \Rightarrow \exists x\, A(x), \qquad \forall x\, (A(x) \Rightarrow C) \Rightarrow \exists x\, A(x) \Rightarrow C,$$
$$A(P) \Rightarrow \exists X\, A(X), \qquad \forall X\, (A(X) \Rightarrow C) \Rightarrow \exists X\, A(X) \Rightarrow C,$$
$$e = e, \qquad e_1 = e_2 \Rightarrow e_2 = e_1, \qquad e_1 = e_2 \Rightarrow e_2 = e_3 \Rightarrow e_1 = e_3, \qquad \text{etc.}$$

- Classical reasoning + De Morgan laws:

$$A \vee \neg A$$
$$\begin{array}{rcl@{\qquad\qquad}rcl} \neg\neg A & \Leftrightarrow & A & \neg(A \wedge B) & \Leftrightarrow & \neg A \vee \neg B \\ (A \Rightarrow B) & \Leftrightarrow & (\neg B \Rightarrow \neg A) & \neg\forall x\, A(x) & \Leftrightarrow & \exists x\, \neg A(x) \end{array}$$

Axioms of classical 2nd-order arithmetic (PA2)

- We have defined (classical) 2nd-order logic (NK2)
  To get 2nd-order arithmetic (PA2), we add the following axioms:

Axioms of classical 2nd-order arithmetic (PA2)

- We have defined (classical) 2nd-order logic (NK2)
  To get 2nd-order arithmetic (PA2), we add the following axioms:

- Defining axioms of primitive recursive function symbols:

$$\begin{array}{ll} \forall x\,(x+0=x) & \forall x\,(x\times 0=0) \\ \forall x\,\forall y\,(x+s(y)=s(x+y)) & \forall x\,\forall y\,(x\times s(y)=x\times y+x) \\ \forall x\,(x-0=x) & \\ \forall y\,(0-y=0) & \text{etc.} \\ \forall x\,\forall y\,(s(x)-s(y)=x-y) & \end{array}$$

Axioms of classical 2nd-order arithmetic (PA2)

- We have defined (classical) 2nd-order logic (NK2)
  To get 2nd-order arithmetic (PA2), we add the following axioms:

- Defining axioms of primitive recursive function symbols:

$$\begin{array}{ll} \forall x\,(x+0=x) & \forall x\,(x\times 0=0) \\ \forall x\,\forall y\,(x+s(y)=s(x+y)) & \forall x\,\forall y\,(x\times s(y)=x\times y+x) \\ \forall x\,(x-0=x) & \\ \forall y\,(0-y=0) & \text{etc.} \\ \forall x\,\forall y\,(s(x)-s(y)=x-y) & \end{array}$$

- Peano axioms:

$$\forall x\,\neg(s(x)=0) \qquad \forall x\,\forall y\,(s(x)=s(y)\Rightarrow x=y)$$

Axioms of classical 2nd-order arithmetic (PA2)

- We have defined (classical) 2nd-order logic (NK2)
  To get 2nd-order arithmetic (PA2), we add the following axioms:

- Defining axioms of primitive recursive function symbols:

$$\begin{array}{ll} \forall x\,(x+0=x) & \forall x\,(x\times 0=0) \\ \forall x\,\forall y\,(x+s(y)=s(x+y)) & \forall x\,\forall y\,(x\times s(y)=x\times y+x) \\ \forall x\,(x-0=x) & \\ \forall y\,(0-y=0) & \text{etc.} \\ \forall x\,\forall y\,(s(x)-s(y)=x-y) & \end{array}$$

- Peano axioms:

$$\forall x\,\neg(s(x)=0) \qquad\qquad \forall x\,\forall y\,(s(x)=s(y)\Rightarrow x=y)$$

- Technically, these axioms are aggregated to the deduction system as new inference rules of the form

$$\frac{}{\Gamma\vdash\forall x\,(x+0=x)} \qquad (\text{etc.})$$

The problem of induction (1/2)

- The above presentation of PA2 contains no induction axiom

The problem of induction (1/2)

- The above presentation of PA2 contains no induction axiom
- The reason is that the property of being a natural number is definable in 2nd-order logic, via the set/predicate:

$$\mathbf{N} \equiv \{x : \forall Z\,(Z(0) \Rightarrow \forall y\,(Z(y) \Rightarrow Z(s(y))) \Rightarrow Z(x))\}$$

The problem of induction (1/2)

- The above presentation of PA2 contains no induction axiom
- The reason is that the property of being a natural number is definable in 2nd-order logic, via the set/predicate:

$$\mathbf{N} \;\equiv\; \{x \;:\; \forall Z\,(Z(0) \;\Rightarrow\; \forall y\,(Z(y) \Rightarrow Z(s(y))) \;\Rightarrow\; Z(x))\}$$

- So we can replace 1st-order quantifications by their versions relativized to **N** (arithmetic quantifications):

$$\begin{array}{rcl} (\forall x \in \mathbf{N})\,A(x) & \equiv & \forall x\,(x \in \mathbf{N} \Rightarrow A(x)) \\ (\exists x \in \mathbf{N})\,A(x) & \equiv & \forall Z\,((\forall x \in \mathbf{N})\,(A(x) \Rightarrow Z) \;\Rightarrow\; Z) \\ & \Leftrightarrow & \exists x\,(x \in \mathbf{N} \;\wedge\; A(x)) \end{array}$$

The problem of induction (2/2)

- Through this process of relativization, induction is derivable:

Relativized principle of induction

$$\forall Z\,(Z(0)\ \Rightarrow\ (\forall x\in\mathbf{N})\,(Z(x)\Rightarrow Z(s(x)))\ \Rightarrow (\forall x\in\mathbf{N})\,Z(x))$$

The problem of induction (2/2)

- Through this process of relativization, induction is derivable:

Relativized principle of induction

$$\forall Z\,(Z(0)\;\Rightarrow\;(\forall x\in\mathbf{N})\,(Z(x)\Rightarrow Z(s(x)))\;\Rightarrow(\forall x\in\mathbf{N})\,Z(x))$$

- In practice, one works with relativized quantification the same way as with unrelativized ones
- However, we need to check that the set/predicate $\mathbf{N}$ is closed under all the operations of the signature Σ:

Proposition (Totality of arithmetic expressions)

For each arithmetic expression $e(x_1,\ldots,x_k)$, the formula

$$\mathrm{Total}(e)\quad\equiv\quad(\forall x_1,\ldots,x_k\in\mathbf{N})\;e(x_1,\ldots,x_k)\in\mathbf{N}$$

is derivable in system NK2 (without an axiom)

Plan

1. Introduction
2. Second-order arithmetic (PA2)
3. Extracted programs
4. The classical realizability interpretation
5. Witness extraction

The λ_c-calculus

Terms, stacks and processes

Terms $t, u \quad ::= \quad x \mid \lambda x\,.\,t \mid tu \mid \text{cc} \mid \mathsf{stop} \mid \mathsf{k}_\pi$

Stacks $\pi, \pi' \quad ::= \quad \diamond \mid t \cdot \pi$ (t closed)

Processes $p, q \quad ::= \quad t \star \pi$ (t closed)

- A λ-calculus with two kinds of constants:
  - Instructions cc (call/cc) and stop
  - Continuation constants k_π, one for every stack π (generated by cc)

The λ_c-calculus

Terms, stacks and processes

Terms $t, u \quad ::= \quad x \;\mid\; \lambda x\,.\,t \;\mid\; tu \;\mid\; \text{\ae} \;\mid\; \mathsf{stop} \;\mid\; \mathsf{k}_\pi$

Stacks $\pi, \pi' \quad ::= \quad \diamond \;\mid\; t \cdot \pi$ (t closed)

Processes $p, q \quad ::= \quad t \star \pi$ (t closed)

- A λ-calculus with two kinds of constants:
  - Instructions $\text{\ae}$ (call/cc) and stop
  - Continuation constants k_π, one for every stack π (generated by $\text{\ae}$)
- **Notations:**

$$\begin{aligned} \Lambda &= \text{set of closed } \lambda_c\text{-terms} \\ \Pi &= \text{set of stacks (closed)} \\ \Lambda \star \Pi &= \text{set of processes (closed)} \end{aligned}$$

The Krivine Abstract Machine (KAM) (1/2)

- The set of processes $(\Lambda \star \Pi)$ is equipped with a preorder of evaluation $p \succ p'$, that is generated from the following rules:

Krivine Abstract Machine (KAM)

Push	$tu \star \pi$	$\succ$	$t \star u \cdot \pi$
Grab	$\lambda x\,.\,t \star u \cdot \pi$	$\succ$	$t\{x := u\} \star \pi$
Save	$\mathrm{c\!\!c} \star u \cdot \pi$	$\succ$	$u \star \mathrm{k}_{\pi} \cdot \pi$
Restore	$\mathrm{k}_{\pi} \star u \cdot \pi'$	$\succ$	$u \star \pi$

(+ reflexivity & transitivity)

- Extensible machinery: can add extra instructions and rules

The Krivine Abstract Machine (KAM) (2/2)

- Rules **Push** and **Grab** implement weak head β-reduction:

Push	$tu \star \pi$	$\succ$	$t \star u \cdot \pi$
Grab	$\lambda x . t \star u \cdot \pi$	$\succ$	$t\{x := u\} \star \pi$

  - Example: $$\begin{array}{rcl}(\lambda xy . t)\, u\, v \star \pi & \succ & \lambda xy . t \star u \cdot v \cdot \pi \\ & \succ & t\{x := u\}\{y := v\} \star \pi\end{array}$$

The Krivine Abstract Machine (KAM) (2/2)

- Rules **Push** and **Grab** implement weak head β-reduction:

$$\begin{array}{lrcl} \textbf{Push} & tu \star \pi & \succ & t \star u \cdot \pi \\ \textbf{Grab} & \lambda x\,.\,t \star u \cdot \pi & \succ & t\{x := u\} \star \pi \end{array}$$

  - Example: $$\begin{array}{rcl} (\lambda xy\,.\,t)\,u\,v \star \pi & \succ & \lambda xy\,.\,t \star u \cdot v \cdot \pi \\ & \succ & t\{x := u\}\{y := v\} \star \pi \end{array}$$

- Rules **Save** and **Restore** implement backtracking:

$$\begin{array}{lrcl} \textbf{Save} & \mathrm{c\!\!c} \star u \cdot \pi & \succ & u \star \mathrm{k}_\pi \cdot \pi \\ \textbf{Restore} & \mathrm{k}_\pi \star u \cdot \pi' & \succ & u \star \pi \end{array}$$

  - Instruction $\mathrm{c\!\!c}$ most often used in the pattern

$$\begin{array}{rcl} \mathrm{c\!\!c}\,(\lambda k\,.\,t) \star \pi & \succ & \mathrm{c\!\!c} \star (\lambda k\,.\,t) \cdot \pi \\ & \succ & (\lambda k\,.\,t) \star \mathrm{k}_\pi \cdot \pi \\ & \succ & t\{k := \mathrm{k}_\pi\} \star \pi \end{array}$$

The Krivine Abstract Machine (KAM) (2/2)

- Rules **Push** and **Grab** implement weak head β-reduction:

Push	$tu \star \pi$	$\succ$	$t \star u \cdot \pi$
Grab	$\lambda x \,.\, t \star u \cdot \pi$	$\succ$	$t\{x := u\} \star \pi$

  - Example:
$$\begin{array}{rcl} (\lambda xy \,.\, t)\, u\, v \star \pi & \succ & \lambda xy \,.\, t \star u \cdot v \cdot \pi \\ & \succ & t\{x := u\}\{y := v\} \star \pi \end{array}$$

- Rules **Save** and **Restore** implement backtracking:

Save	$\text{œ} \star u \cdot \pi$	$\succ$	$u \star \mathrm{k}_\pi \cdot \pi$
Restore	$\mathrm{k}_\pi \star u \cdot \pi'$	$\succ$	$u \star \pi$

  - Instruction œ most often used in the pattern

$$\begin{array}{rcl} \text{œ}\, (\lambda k \,.\, t) \star \pi & \succ & \text{œ} \star (\lambda k \,.\, t) \cdot \pi \\ & \succ & (\lambda k \,.\, t) \star \mathrm{k}_\pi \cdot \pi \\ & \succ & t\{k := \mathrm{k}_\pi\} \star \pi \end{array}$$

- Instruction stop has no evaluation rule: $\mathrm{stop} \star \pi \not\succ$

A type system for 2nd-order logic: λNK2 (1/2)

- **Aim:** Turning the deduction system NK2 into a type system written λNK2, where:
  - Formulas are used as types
  - The computational contents of proofs is given by λ_c-terms

A type system for 2nd-order logic: λNK2 (1/2)

- **Aim:** Turning the deduction system NK2 into a type system written λNK2, where:
  - Formulas are used as types
  - The computational contents of proofs is given by λ_c-terms

- Typing judgments of the form

$$\underbrace{x_1 : A_1, \ldots, x_n : A_n}_{\text{typing context } \Gamma} \vdash t : A$$

= sequent decorated with computational information

A type system for 2nd-order logic: λNK2 (1/2)

- **Aim:** Turning the deduction system NK2 into a type system written λNK2, where:
  - Formulas are used as types
  - The computational contents of proofs is given by λ_c-terms
- Typing judgments of the form

$$\underbrace{x_1 : A_1, \ldots, x_n : A_n}_{\text{typing context } \Gamma} \vdash t : A$$

= sequent decorated with computational information

- **Note:** We only use proof-like terms, that is: λ_c-terms without continuation constants (k_π) and without the instruction stop

A type system for 2nd-order logic: λNK2 (2/2)

Typing rules of system λNK2

$$\frac{}{\Gamma \vdash x : A}\ \text{if } (x{:}A) \in \Gamma$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x\,.\,t : A \Rightarrow B} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x\, A}\ \text{if } x \notin FV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall x\, A}{\Gamma \vdash t : A\{x := e\}}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X\, A}\ \text{if } X \notin FV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall X\, A}{\Gamma \vdash t : A\{X := P\}}$$

$$\frac{}{\Gamma \vdash \mathrm{cc} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$$

A type system for 2nd-order logic: λNK2 (2/2)

Typing rules of system λNK2

$$\frac{}{\Gamma \vdash x : A}\ \text{if } (x{:}A)\in\Gamma$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x\,.\,t : A \Rightarrow B} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall x\, A}\ \text{if } x \notin FV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall x\, A}{\Gamma \vdash t : A\{x := e\}}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X\, A}\ \text{if } X \notin FV(\Gamma) \qquad \frac{\Gamma \vdash t : \forall X\, A}{\Gamma \vdash t : A\{X := P\}}$$

$$\frac{}{\Gamma \vdash \text{cc} : ((A \Rightarrow B) \Rightarrow A) \Rightarrow A}$$

- **Remarks:**
    - $\forall$ interpreted uniformly (intersection type)
    - typing derivations defined the same way as logical derivations
    - type checking/inference undecidable

Relation between deduction (NK2) and typing (λNK2)

- Each typing context $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ can be turned into a logical context $\Gamma^* \equiv A_1, \ldots, A_n$
- Each typing judgment $\Gamma \vdash t : A$ can be turned into a sequent:
$$(\Gamma \vdash t : A)^* \equiv \Gamma^* \vdash A$$
- Each typing derivation d is turned into a logical derivation d^*

Relation between deduction (NK2) and typing (λNK2)

- Each typing context $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ can be turned into a logical context $\Gamma^* \equiv A_1, \ldots, A_n$
- Each typing judgment $\Gamma \vdash t : A$ can be turned into a sequent:
$$(\Gamma \vdash t : A)^* \equiv \Gamma^* \vdash A$$
- Each typing derivation d is turned into a logical derivation d^*

Equivalence between systems NK2 and λNK2

Relation between deduction (NK2) and typing (λNK2)

- Each typing context $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ can be turned into a logical context $\Gamma^* \equiv A_1, \ldots, A_n$
- Each typing judgment $\Gamma \vdash t : A$ can be turned into a sequent:
$$(\Gamma \vdash t : A)^* \;\equiv\; \Gamma^* \vdash A$$
- Each typing derivation d is turned into a logical derivation d^*

Equivalence between systems NK2 and λNK2

1. If d is a typing derivation of $\Gamma \vdash t : A$ in system λNK2, then d^* is a logical derivation of $\Gamma^* \vdash A$ in system NK2

Relation between deduction (NK2) and typing (λNK2)

- Each typing context $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$ can be turned into a logical context $\Gamma^* \equiv A_1, \ldots, A_n$
- Each typing judgment $\Gamma \vdash t : A$ can be turned into a sequent:
$$(\Gamma \vdash t : A)^* \ \equiv\ \Gamma^* \vdash A$$
- Each typing derivation d is turned into a logical derivation d^*

Equivalence between systems NK2 and λNK2

1. If d is a typing derivation of $\Gamma \vdash t : A$ in system λNK2, then d^* is a logical derivation of $\Gamma^* \vdash A$ in system NK2
2. Every logical derivation d of a sequent $\Gamma \vdash A$ in system NK2 comes from a typing derivation d_0 of a judgment of the form $\Gamma_0 \vdash t : A$ in system λNK2 (with $\Gamma_0^* \equiv \Gamma$ and $d_0^* \equiv d$)

Relation between deduction (NK2) and typing (λNK2)

- Each typing context $\Gamma \equiv x_1 : A_1, \dots, x_n : A_n$ can be turned into a logical context $\Gamma^* \equiv A_1, \dots, A_n$
- Each typing judgment $\Gamma \vdash t : A$ can be turned into a sequent:
$$(\Gamma \vdash t : A)^* \;\equiv\; \Gamma^* \vdash A$$
- Each typing derivation d is turned into a logical derivation d^*

Equivalence between systems NK2 and λNK2

1. If d is a typing derivation of $\Gamma \vdash t : A$ in system λNK2, then d^* is a logical derivation of $\Gamma^* \vdash A$ in system NK2
2. Every logical derivation d of a sequent $\Gamma \vdash A$ in system NK2 comes from a typing derivation d_0 of a judgment of the form $\Gamma_0 \vdash t : A$ in system λNK2 (with $\Gamma_0^* \equiv \Gamma$ and $d_0^* \equiv d$)

   The typing derivation d_0 is unique, up to the names of variables

   The term t is called the program extracted from the derivation d

Example: extracting a program from a proof

Example: derivation of the syllogism Barbara

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{(axiom)}
\UnaryInfC{$\Gamma_3 \vdash \forall x\,(Q(x) \Rightarrow R(x))$}
\RightLabel{($\forall^1$-elim)}
\UnaryInfC{$\Gamma_3 \vdash Q(x) \Rightarrow R(x)$}
\AxiomC{}
\RightLabel{(axiom)}
\UnaryInfC{$\Gamma_3 \vdash \forall x\,(P(x) \Rightarrow Q(x))$}
\RightLabel{($\forall^1$-elim)}
\UnaryInfC{$\Gamma_3 \vdash P(x) \Rightarrow Q(x)$}
\AxiomC{}
\RightLabel{(axiom)}
\UnaryInfC{$\Gamma_3 \vdash P(x)$}
\RightLabel{($\Rightarrow$-elim)}
\BinaryInfC{$\Gamma_3 \vdash Q(x)$}
\RightLabel{($\Rightarrow$-elim)}
\BinaryInfC{$\Gamma_3 \vdash R(x)$}
\RightLabel{($\Rightarrow$-intro)}
\UnaryInfC{$\Gamma_2 \vdash P(x) \Rightarrow R(x)$}
\RightLabel{($\forall^1$-intro)}
\UnaryInfC{$\Gamma_2 \vdash \forall x\,(P(x) \Rightarrow R(x))$}
\RightLabel{($\Rightarrow$-intro)}
\UnaryInfC{$\Gamma_1 \vdash \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))$}
\RightLabel{($\Rightarrow$-intro)}
\UnaryInfC{$\vdash \forall x\,(P(x) \Rightarrow Q(x)) \Rightarrow \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))$}
\end{prooftree}
$$

$$\text{with}\quad \Gamma_1 \equiv \forall x\,(P(x) \Rightarrow Q(x)),\quad \Gamma_2 \equiv \Gamma_1, \forall x\,(Q(x) \Rightarrow R(x)),\quad \Gamma_3 \equiv \Gamma_2, P(x)$$

Example: extracting a program from a proof

Example: typing derivation of the syllogism Barbara

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash g : \forall x\,(Q(x) \Rightarrow R(x))}}{\Gamma_3 \vdash g : Q(x) \Rightarrow R(x)} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash f : \forall x\,(P(x) \Rightarrow Q(x))}}{\Gamma_3 \vdash f : P(x) \Rightarrow Q(x)} \quad \dfrac{}{\Gamma_3 \vdash z : P(x)}}{\Gamma_3 \vdash f\ z : Q(x)}}{\Gamma_3 \vdash g\,(f\ z) : R(x)}}{\Gamma_2 \vdash \lambda z\,.\,g\,(f\ z) : P(x) \Rightarrow R(x)}}{\Gamma_2 \vdash \lambda z\,.\,g\,(f\ z) : \forall x\,(P(x) \Rightarrow R(x))}}{\Gamma_1 \vdash \lambda g\,.\,\lambda z\,.\,g\,(f\ z) : \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}}{\vdash \lambda f\,.\,\lambda g\,.\,\lambda z\,.\,g\,(f\ z) : \forall x\,(P(x) \Rightarrow Q(x)) \Rightarrow \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}
$$

with $\Gamma_1 \equiv f : \forall x\,(P(x) \Rightarrow Q(x))$, $\Gamma_2 \equiv \Gamma_1, g : \forall x\,(Q(x) \Rightarrow R(x))$, $\Gamma_3 \equiv \Gamma_2, z : P(x)$

Example: extracting a program from a proof

Example: typing derivation of the syllogism Barbara

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash g : \forall x\,(Q(x) \Rightarrow R(x))}}{\Gamma_3 \vdash g : Q(x) \Rightarrow R(x)} \quad \dfrac{\dfrac{\dfrac{}{\Gamma_3 \vdash f : \forall x\,(P(x) \Rightarrow Q(x))}}{\Gamma_3 \vdash f : P(x) \Rightarrow Q(x)} \quad \dfrac{}{\Gamma_3 \vdash z : P(x)}}{\Gamma_3 \vdash f\,z : Q(x)}}{\Gamma_3 \vdash g\,(f\,z) : R(x)}}{\Gamma_2 \vdash \lambda z\,.\,g\,(f\,z) : P(x) \Rightarrow R(x)}}{\Gamma_2 \vdash \lambda z\,.\,g\,(f\,z) : \forall x\,(P(x) \Rightarrow R(x))}}{\Gamma_1 \vdash \lambda g\,.\,\lambda z\,.\,g\,(f\,z) : \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}}{\vdash \lambda f\,.\,\lambda g\,.\,\lambda z\,.\,g\,(f\,z) : \forall x\,(P(x) \Rightarrow Q(x)) \Rightarrow \forall x\,(Q(x) \Rightarrow R(x)) \Rightarrow \forall x\,(P(x) \Rightarrow R(x))}$$

with $\Gamma_1 \equiv f : \forall x\,(P(x) \Rightarrow Q(x)),\quad \Gamma_2 \equiv \Gamma_1, g : \forall x\,(Q(x) \Rightarrow R(x)),\quad \Gamma_3 \equiv \Gamma_2, z : P(x)$

- Extracted program is: $\lambda f\,.\,\lambda g\,.\,\lambda z\,.\,f\,(g\,z)$ (composition of functions)

Typing examples (1/2)

- Pairing construct and projections associated to conjunction $A \wedge B$ (= Cartesian product):

$$\begin{array}{lcllll}
\langle t, u\rangle & \equiv & \lambda f\,.\,f\,t\,u & : & A \wedge B & (\text{if } t : A,\ \ u : B) \\
\mathbf{pair} & \equiv & \lambda xy\,.\,\langle x, y\rangle & : & \forall X\,\forall Y\,(X \Rightarrow Y \Rightarrow X \wedge Y) & \\
\mathbf{fst} & \equiv & \lambda z\,.\,z\,(\lambda xy\,.\,x) & : & \forall X\,\forall Y\,(X \wedge Y \Rightarrow X) & \\
\mathbf{snd} & \equiv & \lambda z\,.\,z\,(\lambda xy\,.\,y) & : & \forall X\,\forall Y\,(X \wedge Y \Rightarrow Y) &
\end{array}$$

Typing examples (1/2)

- Pairing construct and projections associated to conjunction $A \wedge B$ (= Cartesian product):

$$\begin{array}{lcll}
\langle t, u\rangle & \equiv & \lambda f\,.\,f\;t\;u & : \quad A \wedge B \qquad (\text{if } t : A,\;\; u : B) \\
\mathbf{pair} & \equiv & \lambda xy\,.\,\langle x, y\rangle & : \quad \forall X\,\forall Y\,(X \Rightarrow Y \Rightarrow X \wedge Y) \\
\mathbf{fst} & \equiv & \lambda z\,.\,z\,(\lambda xy\,.\,x) & : \quad \forall X\,\forall Y\,(X \wedge Y \Rightarrow X) \\
\mathbf{snd} & \equiv & \lambda z\,.\,z\,(\lambda xy\,.\,y) & : \quad \forall X\,\forall Y\,(X \wedge Y \Rightarrow Y)
\end{array}$$

- Injections associated to disjunction $A \vee B$ (= direct sum):

$$\begin{array}{lcll}
\mathbf{left} & \equiv & \lambda xfg\,.\,f\;x & : \quad \forall X\,\forall Y\,(X \Rightarrow X \vee Y) \\
\mathbf{right} & \equiv & \lambda yfg\,.\,g\;y & : \quad \forall X\,\forall Y\,(Y \Rightarrow X \vee Y)
\end{array}$$

Typing examples (1/2)

- Pairing construct and projections associated to conjunction $A \wedge B$ (= Cartesian product):

$$
\begin{array}{lcllll}
\langle t, u \rangle & \equiv & \lambda f \,.\, f\, t\, u & : & A \wedge B & (\text{if } t : A,\ \ u : B) \\
\textbf{pair} & \equiv & \lambda xy \,.\, \langle x, y \rangle & : & \forall X\, \forall Y\, (X \Rightarrow Y \Rightarrow X \wedge Y) & \\
\textbf{fst} & \equiv & \lambda z \,.\, z\, (\lambda xy \,.\, x) & : & \forall X\, \forall Y\, (X \wedge Y \Rightarrow X) & \\
\textbf{snd} & \equiv & \lambda z \,.\, z\, (\lambda xy \,.\, y) & : & \forall X\, \forall Y\, (X \wedge Y \Rightarrow Y) &
\end{array}
$$

- Injections associated to disjunction $A \vee B$ (= direct sum):

$$
\begin{array}{lclll}
\textbf{left} & \equiv & \lambda xfg \,.\, f\, x & : & \forall X\, \forall Y\, (X \Rightarrow X \vee Y) \\
\textbf{right} & \equiv & \lambda yfg \,.\, g\, y & : & \forall X\, \forall Y\, (Y \Rightarrow X \vee Y)
\end{array}
$$

- Reflexivity, symmetry and transitivity of equality:

$$
\begin{array}{rclll}
\textbf{eq\_refl} & \equiv & \lambda z \,.\, z & : & \forall x\, (x = x) \\
\textbf{eq\_sym} & \equiv & \lambda z \,.\, z\, (\lambda u \,.\, u) & : & \forall x\, \forall y\, (x = y \Rightarrow y = x) \\
\textbf{eq\_trans} & \equiv & \lambda xyz \,.\, y\, (x\, z) & : & \forall x\, \forall y\, \forall z\, (x = y \Rightarrow y = z \Rightarrow x = z)
\end{array}
$$

Typing examples (2/2)

- **Recall:** injections associated to disjunction $A \vee B$:

$$\begin{array}{lcccl} \textbf{left} & \equiv & \lambda xfg\,.\,f\,x & : & \forall X\,\forall Y\,(X \Rightarrow X \vee Y) \\ \textbf{right} & \equiv & \lambda yfg\,.\,g\,y & : & \forall X\,\forall Y\,(Y \Rightarrow X \vee Y) \end{array}$$

Typing examples (2/2)

- **Recall:** injections associated to disjunction $A \vee B$:

$$\begin{array}{lcl} \textbf{left} & \equiv & \lambda xfg\,.\,f\,x & : & \forall X\,\forall Y\,(X \Rightarrow X \vee Y) \\ \textbf{right} & \equiv & \lambda yfg\,.\,g\,y & : & \forall X\,\forall Y\,(Y \Rightarrow X \vee Y) \end{array}$$

- Computational contents of the law of excluded middle?

$$\textbf{EM} \quad \equiv \qquad\qquad\qquad\qquad : \quad \forall X\,(X \vee \neg X)$$

Typing examples (2/2)

- **Recall:** injections associated to disjunction $A \vee B$:

$$\begin{array}{lclcl} \textbf{left} & \equiv & \lambda xfg\,.\,f\,x & : & \forall X\,\forall Y\,(X \Rightarrow X \vee Y) \\ \textbf{right} & \equiv & \lambda yfg\,.\,g\,y & : & \forall X\,\forall Y\,(Y \Rightarrow X \vee Y) \end{array}$$

- Computational contents of the law of excluded middle:

$$\textbf{EM} \quad \equiv \quad \text{œ}\,(\lambda k\,.\,\textbf{right}\,(\lambda x\,.\,k\,(\textbf{left}\,x))) \quad : \quad \forall X\,(X \vee \neg X)$$

Typing examples (2/2)

- **Recall:** injections associated to disjunction $A \vee B$:

$$\begin{array}{lcl} \textbf{left} & \equiv & \lambda xfg\,.\,f\,x & : & \forall X\,\forall Y\,(X \Rightarrow X \vee Y) \\ \textbf{right} & \equiv & \lambda yfg\,.\,g\,y & : & \forall X\,\forall Y\,(Y \Rightarrow X \vee Y) \end{array}$$

- Computational contents of the law of excluded middle:

$$\mathbf{EM} \quad \equiv \quad \text{œ}\,(\lambda k\,.\,\textbf{right}\,(\lambda x\,.\,k\,(\textbf{left}\,x))) \quad : \quad \forall X\,(X \vee \neg X)$$

- Double-negation elimination & De Morgan laws:

$$\begin{array}{rcl} \lambda z\,.\,\text{œ}\,(\lambda k\,.\,z\,k) & : & \forall X\,(\neg\neg X \Rightarrow X) \\ \lambda zy\,.\,z\,(\lambda x\,.\,yx) & : & \exists x\,A(x) \;\Rightarrow\; \neg\forall x\,\neg A(x) \\ \lambda zy\,.\,\text{œ}\,(\lambda k\,.\,z\,(\lambda x\,.\,k\,(y\,x))) & : & \neg\forall x\,\neg A(x) \;\Rightarrow\; \exists x\,A(x) \end{array}$$

Representing natural numbers

- Encoding zero and successor:

$$\begin{array}{rcll} \overline{0} & \equiv & \lambda zf\,.\,z & : \quad 0 \in \mathbf{N} \\ \overline{s} & \equiv & \lambda nzf\,.\,f\,(n\,z\,f) & : \quad (\forall x \in \mathbf{N})\, s(x) \in \mathbf{N} \end{array}$$

Representing natural numbers

- Encoding zero and successor:

$$\begin{array}{lcll} \overline{0} & \equiv & \lambda zf \,.\, z & : \quad 0 \in \mathbf{N} \\ \overline{s} & \equiv & \lambda nzf \,.\, f\,(n\,z\,f) & : \quad (\forall x \in \mathbf{N})\, s(x) \in \mathbf{N} \end{array}$$

- Each natural number $n \in \mathbb{N}$ is thus represented by the program

$$\overline{n} \;\equiv\; \overline{s}^{n}\,\overline{0} \;\equiv\; \underbrace{\overline{s}\,(\cdots(\overline{s}}_{n}\;\overline{0})\cdots) \quad : \quad n \in \mathbf{N}$$

(= Krivine numeral n)

Representing natural numbers

- Encoding zero and successor:

$$\begin{array}{rcll} \overline{0} & \equiv & \lambda zf\,.\,z & : \quad 0 \in \mathbf{N} \\ \overline{s} & \equiv & \lambda nzf\,.\,f\,(n\,z\,f) & : \quad (\forall x \in \mathbf{N})\, s(x) \in \mathbf{N} \end{array}$$

- Each natural number $n \in \mathbb{N}$ is thus represented by the program

$$\overline{n} \quad \equiv \quad \overline{s}^n\,\overline{0} \quad \equiv \quad \underbrace{\overline{s}\,(\cdots(\overline{s}}_{n}\;\overline{0})\cdots) \quad : \quad n \in \mathbf{N}$$

(= Krivine numeral n)

- Intuitively, the program $\overline{n}$ behaves as an iterator:

$$\begin{array}{rcl} \overline{0} \star u_0 \cdot u_1 \cdot \pi & \succ & u_0 \star \pi \\ \overline{n+1} \star u_0 \cdot u_1 \cdot \pi & \succ & u_1 \star (\overline{n}\,u_0\,u_1) \cdot \pi \end{array}$$

Plan

1. Introduction
2. Second-order arithmetic (PA2)
3. Extracted programs
4. The classical realizability interpretation
5. Witness extraction

Classical realizability: principles

- **Intuitions:**
  - term = “proof” / stack = “counter-proof”
  - process = “contradiction” (slogan: never trust a classical realizer!)

Classical realizability: principles

- **Intuitions:**
  - term = "proof" / stack = "counter-proof"
  - process = "contradiction" (slogan: never trust a classical realizer!)

- Classical realizability model parameterized by a pole $\perp\!\!\!\perp$
  = set of processes closed under anti-evaluation

Classical realizability: principles

- **Intuitions:**
    - term = "proof" / stack = "counter-proof"
    - process = "contradiction" (slogan: never trust a classical realizer!)
- Classical realizability model parameterized by a pole $\perp\!\!\!\perp$
  = set of processes closed under anti-evaluation
- Each formula A is interpreted as two sets:
    - A set of stacks $\|A\|$ (falsity value)
    - A set of terms $|A|$ (truth value)

Classical realizability: principles

- **Intuitions:**
  - term = "proof" / stack = "counter-proof"
  - process = "contradiction" (slogan: never trust a classical realizer!)
- Classical realizability model parameterized by a pole $\perp\!\!\!\perp$ = set of processes closed under anti-evaluation
- Each formula A is interpreted as two sets:
  - A set of stacks $\|A\|$ (falsity value)
  - A set of terms $|A|$ (truth value)
- Falsity value $\|A\|$ defined by induction on A (negative interpretation)
- Truth value $|A|$ defined by orthogonality:

$$|A| \quad = \quad \|A\|^{\perp\!\!\!\perp} \quad = \quad \{t \in \Lambda \ : \ \forall \pi \in \|A\| \ \ t \star \pi \in \perp\!\!\!\perp\}$$

Architecture of the realizability model

- The realizability model $\mathscr{M}_{\perp\!\!\!\perp}$ is defined from:
  - The full standard model $\mathscr{M}$ of PA2: the ground model
    (but we could take any model $\mathscr{M}$ of PA2 as well)
  - A saturated set of processes $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$ (the pole)

Architecture of the realizability model

- The realizability model $\mathscr{M}_{\perp\!\!\!\perp}$ is defined from:
  - The full standard model $\mathscr{M}$ of PA2: the ground model
    (but we could take any model $\mathscr{M}$ of PA2 as well)
  - A saturated set of processes $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$ (the pole)
- Architecture:
  - First-order terms/variables interpreted as natural numbers $n \in \mathbb{N}$
  - Formulas interpreted as falsity values $S \in \mathfrak{P}(\Pi)$
  - k-ary second-order variables (and k-ary predicates) interpreted as falsity functions $F : \mathbb{N}^k \to \mathfrak{P}(\Pi)$.

Architecture of the realizability model

- The realizability model $\mathscr{M}_{\perp\!\!\!\perp}$ is defined from:
  - The full standard model $\mathscr{M}$ of PA2: the ground model
    (but we could take any model $\mathscr{M}$ of PA2 as well)
  - A saturated set of processes $\perp\!\!\!\perp \subseteq \Lambda \star \Pi$ (the pole)
- Architecture:
  - First-order terms/variables interpreted as natural numbers $n \in \mathbb{N}$
  - Formulas interpreted as falsity values $S \in \mathfrak{P}(\Pi)$
  - k-ary second-order variables (and k-ary predicates) interpreted as falsity functions $F : \mathbb{N}^k \to \mathfrak{P}(\Pi)$.

Formulas with parameters $A, B \quad ::= \quad \cdots \quad | \quad \dot{F}(e_1, \ldots, e_k)$

Add a predicate constant $\dot{F}$ for every falsity function $F : \mathbb{N}^k \to \mathfrak{P}(\Pi)$

Interpreting closed formulas with parameters

Let A be a closed formula (with parameters)

- Falsity value $\|A\|$ defined by induction on A:

$$\begin{aligned}
\|\dot{F}(e_1,\ldots,e_k)\| &= F(e_1^{\mathbb{N}},\ldots,e_k^{\mathbb{N}}) \\
\|A \Rightarrow B\| &= |A|\cdot\|B\| = \{t\cdot\pi \ :\ t\in|A|,\ \ \pi\in\|B\|\} \\
\|\forall x\ A\| &= \bigcup_{n\in\mathbb{N}}\|A\{x:=n\}\| \\
\|\forall X\ A\| &= \bigcup_{F:\mathbb{N}^n\to\mathfrak{P}(\Pi)}\|A\{X:=\dot{F}\}\|
\end{aligned}$$

- Truth value $|A|$ defined by orthogonality:

$$|A| \ = \ \|A\|^{\perp\!\!\!\perp} \ = \ \{t\in\Lambda \ :\ \forall\pi\in\|A\| \quad t\star\pi\in\perp\!\!\!\perp\}$$

The realizability relation

Falsity value $\|A\|$ and truth value $|A|$ depend on the pole $\perp\!\!\!\perp$

$\rightsquigarrow$ write them (sometimes) $\|A\|_{\perp\!\!\!\perp}$ and $|A|_{\perp\!\!\!\perp}$ to recall the dependency

Realizability relations

$$t \Vdash A \quad \equiv \quad t \in |A|_{\perp\!\!\!\perp} \qquad \text{(Realizability w.r.t. } \perp\!\!\!\perp\text{)}$$

$$t \Vvdash A \quad \equiv \quad \forall \perp\!\!\!\perp \;\; t \in |A|_{\perp\!\!\!\perp} \qquad \text{(Universal realizability)}$$

From computation to realizability (1/2)

Fundamental idea: The computational behavior of a term determines the formula(s) it realizes:

Example 1: A closed term t is identity-like if:

$$t \star u \cdot \pi \quad \succ \quad u \star \pi \qquad\qquad \text{for all } u \in \Lambda,\ \pi \in \Pi$$

From computation to realizability (1/2)

Fundamental idea: The computational behavior of a term determines the formula(s) it realizes:

Example 1: A closed term t is identity-like if:

$$t \star u \cdot \pi \quad \succ \quad u \star \pi \qquad\qquad \text{for all } u \in \Lambda,\ \pi \in \Pi$$

Proposition

If t is identity-like, then $t \Vdash \forall X\,(X \Rightarrow X)$

Proof: Exercise! (Remark: converse implication holds – exercise!)

From computation to realizability (1/2)

Fundamental idea: The computational behavior of a term determines the formula(s) it realizes:

Example 1: A closed term t is identity-like if:

$$t \star u \cdot \pi \quad \succ \quad u \star \pi \qquad\qquad \text{for all } u \in \Lambda,\ \pi \in \Pi$$

Proposition

If t is identity-like, then $t \Vdash \forall X\,(X \Rightarrow X)$

Proof: Exercise! (Remark: converse implication holds – exercise!)

- Examples of identity-like terms:
  - $\lambda x\,.\,x$, $(\lambda x\,.\,x)\,(\lambda x\,.\,x)$, etc.
  - $\lambda x\,.\,\text{œ}\,(\lambda k\,.\,x)$, $\lambda x\,.\,\text{œ}\,(\lambda k\,.\,k\,x)$, $\lambda x\,.\,\text{œ}\,(\lambda k\,.\,k\,x\,\omega)$, etc.
  - $\lambda x\,.\,\mathrm{quote}\,x\,\lambda n\,.\,\mathrm{unquote}\,n\,(\lambda z\,.\,z)$

From computation to realizability (2/2)

Example 2: Control operators:

$$\begin{array}{rcl} \mathbf{c\!c} \star t \cdot \pi & \succ & t \star k_\pi \cdot \pi \\ k_\pi \star t \cdot \pi' & \succ & t \star \pi \end{array}$$

From computation to realizability (2/2)

Example 2: Control operators:

$$\begin{array}{rcl} \mathfrak{c}\!\!\mathfrak{c} \star t \cdot \pi & \succ & t \star k_\pi \cdot \pi \\ k_\pi \star t \cdot \pi' & \succ & t \star \pi \end{array}$$

- "Typing" k_π: $k_\pi \star t \cdot \pi' \quad \succ \quad t \star \pi$

Lemma

If $\pi \in \|A\|$, then $\mathrm{k}_\pi \Vdash A \Rightarrow B$ (B any)

Proof: Exercise

From computation to realizability (2/2)

Example 2: Control operators:

$$\begin{array}{lcl} \text{æ} \star t \cdot \pi & \succ & t \star k_\pi \cdot \pi \\ k_\pi \star t \cdot \pi' & \succ & t \star \pi \end{array}$$

- "Typing" k_π: $k_\pi \star t \cdot \pi' \quad \succ \quad t \star \pi$

Lemma

If $\pi \in \|A\|$, then $\mathrm{k}_\pi \Vdash A \Rightarrow B$ (B any)

Proof: Exercise

- "Typing" æ: $\text{æ} \star t \cdot \pi \quad \succ \quad t \star k_\pi \cdot \pi$

Proposition (Realizing Peirce's law)

$\text{æ} \Vdash ((A \Rightarrow B) \Rightarrow A) \Rightarrow A$

Proof: Exercise

Anatomy of the model (1/2)

- **Denotation of universal quantification:**

Falsity value: $$\|\forall x\, A\| \;=\; \bigcup_{n\in\mathbb{N}} \|A\{x := n\}\| \qquad \text{(by definition)}$$

Truth value: $$|\forall x\, A| \;=\; \bigcap_{n\in\mathbb{N}} |A\{x := n\}| \qquad \text{(by orthogonality)}$$

(and similarly for 2nd-order universal quantification)

Anatomy of the model (1/2)

- **Denotation of universal quantification:**

  Falsity value: $\|\forall x\,A\| \;=\; \bigcup_{n\in\mathbb{N}} \|A\{x:=n\}\|$ (by definition)

  Truth value: $|\forall x\,A| \;=\; \bigcap_{n\in\mathbb{N}} |A\{x:=n\}|$ (by orthogonality)

  (and similarly for 2nd-order universal quantification)

- **Denotation of implication:**

  Falsity value: $\|A\Rightarrow B\| \;=\; |A|\cdot\|B\|$ (by definition)

  Truth value: $|A\Rightarrow B| \;\subseteq\; |A|\to|B|$ (by orthogonality)

  writing $|A|\to|B| \;=\; \{t\in\Lambda \;:\; \forall u\in|A|\;\; tu\in|B|\}$ (realizability arrow)

Anatomy of the model (2/2)

- **Degenerate case:** $\perp\!\!\!\perp = \varnothing$
    - Classical realizability mimics the Tarski interpretation:

Degenerated interpretation

In the case where $\perp\!\!\!\perp = 0$, for every closed formula A:

$$|A| = \begin{cases} \Lambda & \text{if } \mathscr{M} \models A \\ \varnothing & \text{if } \mathscr{M} \not\models A \end{cases}$$

Anatomy of the model (2/2)

- **Degenerate case:** $\perp\!\!\!\perp = \varnothing$
  - Classical realizability mimics the Tarski interpretation:

Degenerated interpretation

In the case where $\perp\!\!\!\perp = 0$, for every closed formula A:

$$|A| \;=\; \begin{cases} \Lambda & \text{if } \mathscr{M} \models A \\ \varnothing & \text{if } \mathscr{M} \not\models A \end{cases}$$

- **Non degenerate cases:** $\perp\!\!\!\perp \neq \varnothing$
  - Every truth value $|A|$ is inhabited:

    If $t_0 \star \pi_0 \in \perp\!\!\!\perp$, then $\mathsf{k}_{\pi_0} t_0 \in |A|$ for all A (paraproof)
  - We shall only consider realizers that are proof-like terms

Adequacy (1/2)

Aim: Prove the theorem of adequacy

$t : A$ (in the sense of λNK2) implies $t \Vdash A$ (in the sense of realizability)

Adequacy (1/2)

Aim: Prove the theorem of adequacy

$t : A$ (in the sense of λNK2) implies $t \Vdash A$ (in the sense of realizability)

- Closing typing judgments $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$
    - We close logical objects (1st-order terms, formulas, predicates) using semantic objects (natural numbers, falsity values, falsity functions)
    - We close proof-terms using realizers

Adequacy (1/2)

Aim: Prove the theorem of adequacy

$t : A$ (in the sense of λNK2) implies $t \Vdash A$ (in the sense of realizability)

- Closing typing judgments $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$
  - We close logical objects (1st-order terms, formulas, predicates) using semantic objects (natural numbers, falsity values, falsity functions)
  - We close proof-terms using realizers

Definition (Valuations)

1. A valuation is a function ρ such that
   - $\rho(x) \in \mathbb{N}$ for each 1st-order variable x
   - $\rho(X) : \mathbb{N}^k \to \mathfrak{P}(\Pi)$ for each 2nd-order variable X of arity k
2. Closure of A with ρ written $A[\rho]$ (formula with parameters)

Adequacy (2/2)

Definition (Adequate judgment, adequate rule)

Given a fixed pole $\perp\!\!\!\perp$:

1. A judgment $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$ is adequate if for every valuation ρ and for all $u_1 \Vdash A_1[\rho]$, …, $u_n \Vdash A_n[\rho]$ we have:

$$t\{x_1 := u_1, \ldots, x_n := u_n\} \Vdash A[\rho]$$

2. A typing rule is adequate if it preserves the property of adequacy (from the premises to the conclusion of the rule)

Adequacy (2/2)

Definition (Adequate judgment, adequate rule)

Given a fixed pole $\perp\!\!\!\perp$:

1. A judgment $x_1 : A_1, \ldots, x_n : A_n \vdash t : A$ is adequate if for every valuation ρ and for all $u_1 \Vdash A_1[\rho]$, …, $u_n \Vdash A_n[\rho]$ we have:
$$t\{x_1 := u_1, \ldots, x_n := u_n\} \Vdash A[\rho]$$
2. A typing rule is adequate if it preserves the property of adequacy (from the premises to the conclusion of the rule)

Theorem

1. All typing rules of λNK2 are adequate
2. All derivable judgments of λNK2 are adequate

Corollary: If $\vdash t : A$ (A closed formula), then $t \Vdash A$

Extending adequacy to subtyping

Definition (Adequate subtyping judgment)

Judgment $A \leq B$ adequate $\quad\equiv\quad \|B[\rho]\| \subseteq \|A[\rho]\|$ (for all valuations)

Remark: Implies $|A[\rho]| \subseteq |B[\rho]|$ (for all ρ), but strictly stronger

Extending adequacy to subtyping

Definition (Adequate subtyping judgment)

Judgment $A \le B$ adequate $\quad\equiv\quad \|B[\rho]\| \subseteq \|A[\rho]\|$ (for all valuations)

Remark: Implies $|A[\rho]| \subseteq |B[\rho]|$ (for all ρ), but strictly stronger

- Some adequate typing/subtyping rules:

$$\frac{}{A \le A} \qquad \frac{A \le B \qquad B \le C}{A \le C} \qquad \frac{\Gamma \vdash t : A \qquad A \le B}{\Gamma \vdash t : B}$$

$$\frac{}{\forall x\, A \;\le\; A\{x := e\}} \qquad \frac{}{\forall X\, A \;\le\; A\{X := P\}}$$

$$\frac{A \le B}{A \;\le\; \forall x\, B}\ {\scriptstyle x \notin FV(A)} \qquad \frac{A \le B}{A \;\le\; \forall X\, B}\ {\scriptstyle X \notin FV(A)} \qquad \frac{A' \le A \qquad B \le B'}{A \Rightarrow B \;\le\; A' \Rightarrow B'}$$

$$\frac{}{\forall x\,(A \Rightarrow B) \;\le\; A \Rightarrow \forall x\, B}\ {\scriptstyle x \notin FV(A)} \qquad \frac{}{\forall X\,(A \Rightarrow B) \;\le\; A \Rightarrow \forall X\, B}\ {\scriptstyle X \notin FV(A)}$$

Extending adequacy to subtyping

Definition (Adequate subtyping judgment)

Judgment $A \leq B$ adequate $\quad \equiv \quad \|B[\rho]\| \subseteq \|A[\rho]\|$ (for all valuations)

Remark: Implies $|A[\rho]| \subseteq |B[\rho]|$ (for all ρ), but strictly stronger

- Some adequate typing/subtyping rules:

$$\frac{}{A \leq A} \qquad \frac{A \leq B \qquad B \leq C}{A \leq C} \qquad \frac{\Gamma \vdash t : A \qquad A \leq B}{\Gamma \vdash t : B}$$

$$\frac{}{\forall x\, A \ \leq \ A\{x := e\}} \qquad \frac{}{\forall X\, A \ \leq \ A\{X := P\}}$$

$$\frac{A \leq B}{A \ \leq \ \forall x\, B}\ {\scriptstyle x \notin FV(A)} \qquad \frac{A \leq B}{A \ \leq \ \forall X\, B}\ {\scriptstyle X \notin FV(A)} \qquad \frac{A' \leq A \qquad B \leq B'}{A \Rightarrow B \ \leq \ A' \Rightarrow B'}$$

$$\frac{}{\forall x\,(A \Rightarrow B) \ \leq \ A \Rightarrow \forall x\, B}\ {\scriptstyle x \notin FV(A)} \qquad \frac{}{\forall X\,(A \Rightarrow B) \ \leq \ A \Rightarrow \forall X\, B}\ {\scriptstyle X \notin FV(A)}$$

- Example: $\underbrace{\forall X\, \forall Y\, (((X \Rightarrow Y) \Rightarrow X) \Rightarrow X)}_{\text{Peirce's law}} \ \leq \ \underbrace{\forall X\, (\neg\neg X \Rightarrow X)}_{\text{DNE}}$

Realizing equalities

- Equality between individuals defined by

$$e_1 = e_2 \;\equiv\; \forall Z\,(Z(e_1) \Rightarrow Z(e_2)) \qquad \text{(Leibniz equality)}$$

Denotation of Leibniz equality

Given two closed first-order terms e_1, e_2 (and a pole $\perp\!\!\!\perp$)

$$\|e_1 = e_2\| \quad = \quad \begin{cases} \|\mathbf{1}\| \;=\; \{t \cdot \pi \;:\; (t \star \pi) \in \perp\!\!\!\perp\} & \text{if } [\![e_1]\!] = [\![e_2]\!] \\ \|\top \Rightarrow \bot\| \;=\; \Lambda \cdot \Pi & \text{if } [\![e_1]\!] \neq [\![e_2]\!] \end{cases}$$

writing $\mathbf{1} \equiv \forall Z\,(Z \Rightarrow Z)$ and $\top \equiv \dot{\varnothing}$

- Intuitions:
  - A realizer of a true equality (in the ground model $\mathscr{M}$) behaves as the identity function $\lambda z \,.\, z$
  - A realizer of a false equality (in the ground model $\mathscr{M}$) behaves as a point of backtrack (breakpoint)

Realizing axioms

Corollary 1 (Realizing true equations)

If $\mathscr{M} \models \forall\vec{x}\,(e_1(\vec{x}) = e_2(\vec{x}))$ (truth in the ground model)

then $\mathbf{I} \equiv \lambda z\,.\,z \Vdash \forall\vec{x}\,(e_1(\vec{x}) = e_2(\vec{x}))$ (universal realizability)

Corollary 2

All defining equations of primitive recursive function symbols $(+,\ -,\ \times,$ etc.$)$ are universally realized by $\mathbf{I} \equiv \lambda z\,.\,z$

Realizing axioms

Corollary 1 (Realizing true equations)

If $\mathscr{M} \models \forall\vec{x}\,(e_1(\vec{x}) = e_2(\vec{x}))$ (truth in the ground model)

then $\mathbf{I} \equiv \lambda z\,.\,z \Vdash \forall\vec{x}\,(e_1(\vec{x}) = e_2(\vec{x}))$ (universal realizability)

Corollary 2

All defining equations of primitive recursive function symbols $(+,\ -,\ \times,$ etc.) are universally realized by $\mathbf{I} \equiv \lambda z\,.\,z$

Corollary 3 (Realizing Peano axioms)

$$\begin{aligned} \lambda z\,.\,z\,\mathbf{I} &\Vdash \forall x\,\neg(s(x) = 0) \\ \mathbf{I} &\Vdash \forall x\,\forall y\,(s(x) = s(y) \Rightarrow x = y) \end{aligned}$$

Theorem: If PA2 $\vdash A$, then $\theta \Vdash A$ for some proof-like term θ

Provability, universal realizability and truth

- From what precedes:
  1. A provable $\Rightarrow$ A universally realized (by a proof-like term)
  2. A universally realized $\Rightarrow$ A true (in the full standard model)

$\rightsquigarrow$ Universal realizability: an intermediate notion between provability and truth

Provability, universal realizability and truth

- From what precedes:
  1. A provable $\Rightarrow$ A universally realized (by a proof-like term)
  2. A universally realized $\Rightarrow$ A true (in the full standard model)

$\rightsquigarrow$ Universal realizability: an intermediate notion between provability and truth

- **Beware!**

$$\begin{array}{ccc}
\text{Intuitionistic proofs of } A & \subseteq & \text{Classical proofs of } A \\
\cap & & \cap \\
\text{Intuitionistic realizers of } A & \begin{array}{c}\not\subseteq \\ \not\supseteq\end{array} & \text{Classical realizers of } A
\end{array}$$

Plan

The problem of witness extraction

- **Problem:** Extract a witness from a universal realizer (or a proof)

$$t_0 \;\Vdash\; (\exists x \in \mathbf{N})\, A(x)$$

i.e. some $n \in \mathbb{N}$ such that $A(n)$ is true

The problem of witness extraction

- **Problem:** Extract a witness from a universal realizer (or a proof)

$$t_0 \ \Vdash \ (\exists x \in \mathbf{N})\, A(x)$$

i.e. some $n \in \mathbb{N}$ such that $A(n)$ is true

- This is not always possible!

$$t_0 \ \Vdash \ (\exists x \in \mathbf{N})\, ((x = 1 \wedge C) \vee (x = 0 \wedge \neg C))$$

(C = Continuum hypothesis, Goldbach's conjecture, etc.)

The problem of witness extraction

- **Problem:** Extract a witness from a universal realizer (or a proof)

  $$t_0 \Vdash (\exists x \in \mathbf{N})\, A(x)$$

  i.e. some $n \in \mathbb{N}$ such that $A(n)$ is true

- This is not always possible!

  $$t_0 \Vdash (\exists x \in \mathbf{N})\,((x = 1 \wedge C) \vee (x = 0 \wedge \neg C))$$

  (C = Continuum hypothesis, Goldbach's conjecture, etc.)

- Two possible compromises:
  - Intuitionistic logic: restrict the shape of the realizer t_0
    (by only keeping intuitionistic reasoning principles)
  - Classical logic: restrict the shape of the formula $A(x)$
    (typically: Δ^0_0-formulas)

Storage operators (1/2)

- The call-by-value implication:

Formulas $A, B \quad ::= \quad \cdots \quad | \quad \{e\} \Rightarrow A$

with the semantics: $\|\{e\} \Rightarrow A\| \; = \; \{\bar{n} \cdot \pi \; : \; n = e^{\mathbb{N}}, \; \pi \in \|A\|\}$

Storage operators (1/2)

- The call-by-value implication:

Formulas $A, B \quad ::= \quad \cdots \quad | \quad \{e\} \Rightarrow A$

with the semantics: $\|\{e\} \Rightarrow A\| \; = \; \{\bar{n} \cdot \pi \; : \; n = e^{\mathbb{N}}, \; \pi \in \|A\|\}$

- From the definition: $e \in \mathbf{N} \Rightarrow A \;\leq\; \{e\} \Rightarrow A$

  so that: $\mathbf{I} \;\Vdash\; \forall x \, \forall Z \, [(x \in \mathbf{N} \Rightarrow Z) \Rightarrow (\{x\} \Rightarrow Z)]$ (direct implication)

Storage operators (1/2)

- The call-by-value implication:

Formulas $A, B \quad ::= \quad \cdots \quad | \quad \{e\} \Rightarrow A$

with the semantics: $\|\{e\} \Rightarrow A\| \;=\; \{\bar{n} \cdot \pi \;:\; n = e^{\mathbb{N}},\; \pi \in \|A\|\}$

- From the definition: $e \in \mathbf{N} \Rightarrow A \;\leq\; \{e\} \Rightarrow A$

so that: $\mathbf{I} \Vdash \forall x \forall Z\, [(x \in \mathbf{N} \Rightarrow Z) \Rightarrow (\{x\} \Rightarrow Z)]$ (direct implication)

Definition (Storage operator)

A storage operator is a closed proof-like term M such that:

$$M \Vdash \forall x \forall Z\, [(\{x\} \Rightarrow Z) \Rightarrow (x \in \mathbf{N} \Rightarrow Z)] \qquad \text{(converse implication)}$$

Storage operators (1/2)

- The call-by-value implication:

Formulas $A, B \quad ::= \quad \cdots \quad | \quad \{e\} \Rightarrow A$

with the semantics: $\|\{e\} \Rightarrow A\| \; = \; \{\bar{n} \cdot \pi \; : \; n = e^{\mathbb{N}}, \; \pi \in \|A\|\}$

- From the definition: $e \in \mathbf{N} \Rightarrow A \;\leq\; \{e\} \Rightarrow A$

so that: $\mathbf{I} \;\Vdash\; \forall x \forall Z\,[(x \in \mathbf{N} \Rightarrow Z) \Rightarrow (\{x\} \Rightarrow Z)]$ (direct implication)

Definition (Storage operator)

A storage operator is a closed proof-like term M such that:

$$M \;\Vdash\; \forall x \forall Z\,[(\{x\} \Rightarrow Z) \Rightarrow (x \in \mathbf{N} \Rightarrow Z)] \qquad \text{(converse implication)}$$

Theorem (Existence)

Storage operators exist, e.g.: $M \;:=\; \lambda fn\,.\,n\,f\,(\lambda hx\,.\,h\,(\bar{s}\,x))\,\bar{0}$

Storage operators (2/2)

- Intuitively, a storage operator

$$M \ \Vdash\ \forall x \forall Z\, [(\{x\} \Rightarrow Z) \Rightarrow (x \in \mathbf{N} \Rightarrow Z)]$$

is a proof-like term that is intended to be applied to

  - a function f that only accepts values (i.e. intuitionistic integers)
  - a classical integer $t \Vdash n \in \mathbf{N}$ (n arbitrary)

and that evaluates (or 'smoothes') the classical integer t into a value of the form $\bar{n}$ before passing this value to f

Storage operators (2/2)

- Intuitively, a storage operator

$$M \Vdash \forall x\, \forall Z\, [(\{x\} \Rightarrow Z) \Rightarrow (x \in \mathbf{N} \Rightarrow Z)]$$

is a proof-like term that is intended to be applied to

  - a function f that only accepts values (i.e. intuitionistic integers)
  - a classical integer $t \Vdash n \in \mathbf{N}$ (n arbitrary)

and that evaluates (or 'smoothes') the classical integer t into a value of the form $\bar{n}$ before passing this value to f

- By subtyping, we also have:

$$M \Vdash \forall Z\, [\forall x\, (\{x\} \Rightarrow Z(x)) \;\Rightarrow\; (\forall x \in \mathbf{N})\, Z(x)]$$

This means that if a property $Z(x)$ holds for all intuitionistic integers, then it holds for all classical integers too

Storage operators (2/2)

- Intuitively, a storage operator

$$M \ \Vdash\ \forall x\,\forall Z\,[(\{x\} \Rightarrow Z) \Rightarrow (x \in \mathbf{N} \Rightarrow Z)]$$

is a proof-like term that is intended to be applied to

  - a function f that only accepts values (i.e. intuitionistic integers)
  - a classical integer $t \Vdash n \in \mathbf{N}$ (n arbitrary)

and that evaluates (or 'smoothes') the classical integer t into a value of the form $\bar{n}$ before passing this value to f

- By subtyping, we also have:

$$M \ \Vdash\ \forall Z\,[\forall x\,(\{x\} \Rightarrow Z(x)) \ \Rightarrow\ (\forall x \in \mathbf{N})\,Z(x)]$$

This means that if a property $Z(x)$ holds for all intuitionistic integers, then it holds for all classical integers too

- **Conclusion:** $e \in \mathbf{N} \Rightarrow A$ and $\{e\} \Rightarrow A$ interchangeable

Computing with storage operators

- Given a k-ary function symbol f, we let:

$$\begin{array}{lcl}
\mathrm{Total}(f) & := & (\forall x_1 \in \mathbf{N}) \cdots (\forall x_k \in \mathbf{N})(f(x_1, \ldots, x_k) \in \mathbf{N}) \\
\mathrm{Comput}(f) & := & \forall x_1 \cdots \forall x_k \, \forall Z \, [\{x_1\} \Rightarrow \cdots \Rightarrow \{x_k\} \Rightarrow \\
 & & \qquad (\{f(x_1, \ldots, x_k)\} \Rightarrow Z) \Rightarrow Z]
\end{array}$$

Computing with storage operators

- Given a k-ary function symbol f, we let:

$$\begin{array}{lcl} \mathrm{Total}(f) & := & (\forall x_1 \in \mathbf{N}) \cdots (\forall x_k \in \mathbf{N})(f(x_1,\ldots,x_k) \in \mathbf{N}) \\ \mathrm{Comput}(f) & := & \forall x_1 \cdots \forall x_k\, \forall Z\, [\{x_1\} \Rightarrow \cdots \Rightarrow \{x_k\} \Rightarrow \\ & & \qquad (\{f(x_1,\ldots,x_k)\} \Rightarrow Z) \Rightarrow Z] \end{array}$$

Theorem (Specification of the formula Comput(f))

For all $t \in \Lambda$, the following assertions are equivalent:

1. $t \Vdash \mathrm{Comput}(f)$
2. t computes f: for all $(n_1,\ldots,n_k) \in \mathbb{N}^k$, $u \in \Lambda$, $\pi \in \Pi$:

$$t \star \overline{n}_1 \cdots \overline{n}_k \cdot u \cdot \pi \;\succ\; u \star \overline{f(n_1,\ldots,n_k)} \cdot \pi$$

Computing with storage operators

- Given a k-ary function symbol f, we let:

$$\begin{array}{lcl} \mathrm{Total}(f) & := & (\forall x_1 \in \mathbf{N}) \cdots (\forall x_k \in \mathbf{N})(f(x_1, \ldots, x_k) \in \mathbf{N}) \\ \mathrm{Comput}(f) & := & \forall x_1 \cdots \forall x_k \, \forall Z \, [\{x_1\} \Rightarrow \cdots \Rightarrow \{x_k\} \Rightarrow \\ & & \qquad (\{f(x_1, \ldots, x_k)\} \Rightarrow Z) \Rightarrow Z] \end{array}$$

Theorem (Specification of the formula Comput(f))

For all $t \in \Lambda$, the following assertions are equivalent:

1. $t \Vdash \mathrm{Comput}(f)$
2. t computes f: for all $(n_1, \ldots, n_k) \in \mathbb{N}^k$, $u \in \Lambda$, $\pi \in \Pi$:

$$t \star \overline{n}_1 \cdots \overline{n}_k \cdot u \cdot \pi \ \succ \ u \star \overline{f(n_1, \ldots, n_k)} \cdot \pi$$

- Using a storage operator M, we can build proof-like terms:

$$\begin{array}{lclcl} \xi_k & \Vdash & \mathrm{Total}(f) & \Rightarrow & \mathrm{Comput}(f) \\ \xi'_k & \Vdash & \mathrm{Comput}(f) & \Rightarrow & \mathrm{Total}(f) \end{array}$$

The naive extraction method

- A classical realizer $t_0 \Vdash (\exists x \in \mathbf{N})\, A(x)$ always evaluates to a pair witness/justification:

Naive extraction

If $t_0 \Vdash (\exists x \in \mathbf{N})\, A(x)$, then there are $n \in \mathbb{N}$ and $u \in \Lambda$ such that:

$$t_0 \star M(\lambda xy\,.\,\mathrm{stop}\,x\,y) \cdot \diamond \quad \succ \quad \mathrm{stop} \star \overline{n} \cdot u \cdot \diamond$$

(where $u \Vdash A(n)$ w.r.t. the particular pole $\perp\!\!\!\perp$... needed to prove the property)

The naive extraction method

- A classical realizer $t_0 \Vdash (\exists x \in \mathbf{N})\, A(x)$ always evaluates to a pair witness/justification:

Naive extraction

If $t_0 \Vdash (\exists x \in \mathbf{N})\, A(x)$, then there are $n \in \mathbb{N}$ and $u \in \Lambda$ such that:

$$t_0 \star M(\lambda xy\,.\,\mathsf{stop}\,x\,y) \cdot \diamond \quad \succ \quad \mathsf{stop} \star \overline{n} \cdot u \cdot \diamond$$

(where $u \Vdash A(n)$ w.r.t. the particular pole $\perp\!\!\!\perp$... needed to prove the property)

- But $n \in \mathbb{N}$ might be a false witness because the justification $u \Vdash A(n)$ is cheating! (u might contain hidden continuations)

The naive extraction method

- A classical realizer $t_0 \Vdash (\exists x \in \mathbf{N})\, A(x)$ always evaluates to a pair witness/justification:

Naive extraction

If $t_0 \Vdash (\exists x \in \mathbf{N})\, A(x)$, then there are $n \in \mathbb{N}$ and $u \in \Lambda$ such that:

$$t_0 \star M(\lambda xy\,.\,\mathrm{stop}\,x\,y) \cdot \diamond \quad \succ \quad \mathrm{stop} \star \overline{n} \cdot u \cdot \diamond$$

(where $u \Vdash A(n)$ w.r.t. the particular pole $\perp\!\!\!\perp$... needed to prove the property)

- But $n \in \mathbb{N}$ might be a false witness because the justification $u \Vdash A(n)$ is cheating! (u might contain hidden continuations)
- In the case where t_0 comes from an intuitionistic proof, extracted witness $n \in \mathbb{N}$ is always correct

Extraction in the Σ^0_1-case

Extraction in the Σ^0_1-case

If $t_0 \Vdash (\exists x \in \mathbf{N})(f(x) = 0)$, then

$$t_0 \star M(\lambda xy\,.\quad y\,(\mathrm{stop}\,x)) \cdot \diamond \quad \succ \quad \mathrm{stop} \star \overline{n} \cdot \diamond$$

for some $n \in \mathbb{N}$ such that $f(n) = 0$

- Storage operator M used to evaluate 1st component (x)
- 2nd component (y) used as a breakpoint
  (Relies on the particular structure of equality realizers)
- Holds independently from the instruction set
- Supports any representation of numerals
  (One has to implement the storage operator M accordingly)

Extraction in the Σ^0_1-case

Extraction in the Σ^0_1-case (+ display intermediate results)

If $t_0 \Vdash (\exists x \in \mathbf{N})(f(x) = 0)$, then

$$t_0 \star M(\lambda xy\,.\,\mathsf{print}\,x\,y\,(\mathsf{stop}\,x)) \cdot \diamond \quad \succ \quad \mathsf{stop} \star \overline{n} \cdot \diamond$$

for some $n \in \mathbb{N}$ such that $f(n) = 0$

- Storage operator M used to evaluate 1st component (x)
- 2nd component (y) used as a breakpoint
  (Relies on the particular structure of equality realizers)
- Holds independently from the instruction set
- Supports any representation of numerals
  (One has to implement the storage operator M accordingly)

Example: the minimum principle

- Given a unary function symbol f, write:

$$\begin{array}{rcll} \mathrm{Total}(f) & := & (\forall x \in \mathbf{N})(f(x) \in \mathbf{N}) & \text{(totality predicate)} \\ x \leq y & := & x - y = 0 & \text{(truncated subtraction)} \end{array}$$

Example: the minimum principle

- Given a unary function symbol f, write:

$$\begin{array}{rcll} \mathrm{Total}(f) & := & (\forall x \in \mathbf{N})(f(x) \in \mathbf{N}) & \text{(totality predicate)} \\ x \leq y & := & x - y = 0 & \text{(truncated subtraction)} \end{array}$$

Theorem (Minimum principle – MinP)

$$\mathrm{PA2} \;\vdash\; \mathrm{Total}(f) \;\Rightarrow\; (\exists x \in \mathbf{N})\, \underbrace{(\forall y \in \mathbf{N})\,(f(x) \leq f(y))}_{\text{undecidable}}$$

Proof. Reductio ad absurdum + course by value induction

Example: the minimum principle

- Given a unary function symbol f, write:

$$\begin{array}{rcll} \mathrm{Total}(f) & := & (\forall x \in \mathbf{N})(f(x) \in \mathbf{N}) & \text{(totality predicate)} \\ x \leq y & := & x - y = 0 & \text{(truncated subtraction)} \end{array}$$

Theorem (Minimum principle – MinP)

$$\mathrm{PA2} \vdash \mathrm{Total}(f) \Rightarrow (\exists x \in \mathbf{N})\, \underbrace{(\forall y \in \mathbf{N})\, (f(x) \leq f(y))}_{\text{undecidable}}$$

Proof. Reductio ad absurdum + course by value induction

- The minimum principle is not intuitionistically provable (oracle)
- We cannot apply the Σ^0_1-extraction technique to the above proof (applied to a totality proof of f), since the conclusion is Σ^0_2

  The body $(\forall y \in \mathbf{N})\, (f(x) \leq f(y))$ of $\exists$-quantification is undecidable

Implementation of the minimum principle

$$\mathbf{I} \;\equiv\; \lambda x\,.\,x \qquad \mathbf{T} \;\equiv\; \lambda xy\,.\,x \qquad \mathbf{F} \;\equiv\; \lambda xy\,.\,y$$

$$\langle t_1, t_2\rangle \;\equiv\; \lambda z\,.\,z\; t_1\; t_2 \qquad (z \text{ fresh } \lambda\text{-variable})$$

$$\begin{array}{lcl}
\mathrm{pred} & \equiv & \lambda n\,.\,n\;\langle\overline{0},\overline{0}\rangle\;(\lambda p\,.\,p\;(\lambda xy\,.\,\langle x, \overline{s}\;x\rangle))\;(\lambda xy\,.\,x) \\
\mathrm{minus} & \equiv & \lambda n, m\,.\,m\;n\;\mathrm{pred} \\
\mathrm{cmp} & \equiv & \lambda n, m\,.\,\mathrm{minus}\;n\;m\;\mathbf{T}\;(\lambda\_\,.\,\mathbf{F})
\end{array}$$

$$\mathbf{Y} \;\equiv\; (\lambda yf\,.\,f\;(y\;y\;f))\;(\lambda yf\,.\,f\;(y\;y\;f))$$

$$\begin{array}{lcl}
\mathrm{MinP} & \equiv & \lambda f\,.\,\text{œ}\;(\lambda k\,.\,\mathbf{Y}\;(\lambda r, n\,.\,\langle n, \lambda m\,.\,\mathrm{cmp}\;(f\;n)\;(f\;m)\;\mathbf{I}\;(k\;(r\;m))\rangle)\;\overline{0}) \\
 & \Vdash & (\forall x \in \mathbf{N})\; f(x) \in \mathbf{N} \;\;\Rightarrow\;\; (\exists x \in \mathbf{N})(\forall y \in \mathbf{N})\; f(x) \leq f(y)
\end{array}$$

Using the minimum principle to prove a Σ^0_1-formula

- **Idea:** The value x given by the minimum principle can be used to prove a Σ^0_1-formula, so that we can perform program extraction:

Corollary

$$\mathrm{PA2} \vdash \mathrm{Total}(f) \Rightarrow (\exists x \in \mathbf{N})\,\underbrace{(f(x) \leq f(2x+1))}_{\text{decidable}}$$

More generally: $\mathrm{PA2} \vdash \mathrm{Total}(f) \wedge \mathrm{Total}(g) \Rightarrow (\exists x \in \mathbf{N})\,(f(x) \leq f(g(x)))$

Proof. Take the point x given by the minimum principle

Using the minimum principle to prove a Σ^0_1-formula

- **Idea:** The value x given by the minimum principle can be used to prove a Σ^0_1-formula, so that we can perform program extraction:

Corollary

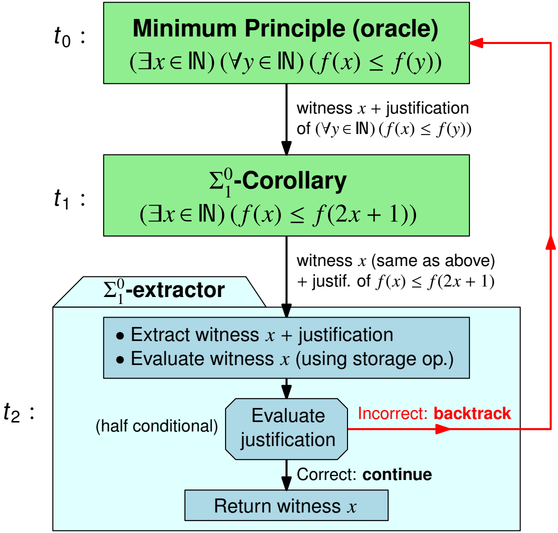
$$\mathrm{PA2} \vdash \mathrm{Total}(f) \Rightarrow (\exists x \in \mathbf{N})\ \underbrace{(f(x) \leq f(2x+1))}_{\text{decidable}}$$

More generally: $\mathrm{PA2} \vdash \mathrm{Total}(f) \wedge \mathrm{Total}(g) \Rightarrow (\exists x \in \mathbf{N})\,(f(x) \leq f(g(x)))$

Proof. Take the point x given by the minimum principle

- Applying Σ^0_1-extraction to the above non-constructive proof, we get a correct witness in finitely many evaluation steps
- How is this witness computed?

The algorithm underlying Σ^0_1-extraction



t_0 :

Minimum Principle (oracle)

$(\exists x \in \mathbb{N})\,(\forall y \in \mathbb{N})\,(f(x) \leq f(y))$

witness x + justification of $(\forall y \in \mathbb{N})\,(f(x) \leq f(y))$

t_1 :

Σ^0_1-Corollary

$(\exists x \in \mathbb{N})\,(f(x) \leq f(2x+1))$

witness x (same as above) + justif. of $f(x) \leq f(2x+1)$

Σ^0_1-extractor

- Extract witness x + justification
- Evaluate witness x (using storage op.)

t_2 :

(half conditional)

Evaluate justification

Incorrect: **backtrack**

Correct: **continue**

Return witness x

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)
and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1 Oracle says: take $x = 0$ since $(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$ (false)

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1 Oracle says:	take $x = 0$	since	$(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$	(false)
Corollary says:	take $x = 0$	since	$f(0) \leq f(1)$	(false)

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1	Oracle says:	take $x = 0$	since	$(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$	(false)
	Corollary says:	take $x = 0$	since	$f(0) \leq f(1)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks				

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1	Oracle says:	take $x = 0$	since	$(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$	(false)
	Corollary says:	take $x = 0$	since	$f(0) \leq f(1)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks				
Step 2	Oracle says:	take $x = 1$	since	$(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$	(false)

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1 Oracle says: take $x = 0$ since $(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$ (false)
Corollary says: take $x = 0$ since $f(0) \leq f(1)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Step 2 Oracle says: take $x = 1$ since $(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$ (false)
Corollary says: take $x = 1$ since $f(1) \leq f(3)$ (false)

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)
and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1 Oracle says: take $x = 0$ since $(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$ (false)
Corollary says: take $x = 0$ since $f(0) \leq f(1)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Step 2 Oracle says: take $x = 1$ since $(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$ (false)
Corollary says: take $x = 1$ since $f(1) \leq f(3)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)
and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1 Oracle says: take $x = 0$ since $(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$ (false)
Corollary says: take $x = 0$ since $f(0) \leq f(1)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Step 2 Oracle says: take $x = 1$ since $(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$ (false)
Corollary says: take $x = 1$ since $f(1) \leq f(3)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Step 3 Oracle says: take $x = 3$ since $(\forall y \in \mathbf{N})\,(f(3) \leq f(y))$ (false)

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1 Oracle says: take $x = 0$ since $(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$ (false)
Corollary says: take $x = 0$ since $f(0) \leq f(1)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Step 2 Oracle says: take $x = 1$ since $(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$ (false)
Corollary says: take $x = 1$ since $f(1) \leq f(3)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Step 3 Oracle says: take $x = 3$ since $(\forall y \in \mathbf{N})\,(f(3) \leq f(y))$ (false)
Corollary says: take $x = 3$ since $f(3) \leq f(7)$ (false)

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1	Oracle says:	take $x = 0$	since $(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$	(false)
	Corollary says:	take $x = 0$	since $f(0) \leq f(1)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks			
Step 2	Oracle says:	take $x = 1$	since $(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$	(false)
	Corollary says:	take $x = 1$	since $f(1) \leq f(3)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks			
Step 3	Oracle says:	take $x = 3$	since $(\forall y \in \mathbf{N})\,(f(3) \leq f(y))$	(false)
	Corollary says:	take $x = 3$	since $f(3) \leq f(7)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks			

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1 Oracle says: take $x = 0$ since $(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$ (false)
Corollary says: take $x = 0$ since $f(0) \leq f(1)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Step 2 Oracle says: take $x = 1$ since $(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$ (false)
Corollary says: take $x = 1$ since $f(1) \leq f(3)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Step 3 Oracle says: take $x = 3$ since $(\forall y \in \mathbf{N})\,(f(3) \leq f(y))$ (false)
Corollary says: take $x = 3$ since $f(3) \leq f(7)$ (false)
Σ^0_1-extractor evaluates incorrect justification and backtracks

Step 4 Oracle says: take $x = 7$ since $(\forall y \in \mathbf{N})\,(f(7) \leq f(y))$ (false)

. . . .

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1	Oracle says:	take $x = 0$	since	$(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$	(false)
	Corollary says:	take $x = 0$	since	$f(0) \leq f(1)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks				
Step 2	Oracle says:	take $x = 1$	since	$(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$	(false)
	Corollary says:	take $x = 1$	since	$f(1) \leq f(3)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks				
Step 3	Oracle says:	take $x = 3$	since	$(\forall y \in \mathbf{N})\,(f(3) \leq f(y))$	(false)
	Corollary says:	take $x = 3$	since	$f(3) \leq f(7)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks				
Step 4	Oracle says:	take $x = 7$	since	$(\forall y \in \mathbf{N})\,(f(7) \leq f(y))$	(false)
		. . . .			
Step 11	Oracle says:	take $x = 1023$	since	$(\forall y \in \mathbf{N})\,(f(1023) \leq f(y))$	(false)

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1	Oracle says:	take $x = 0$	since	$(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$	(false)
	Corollary says:	take $x = 0$	since	$f(0) \leq f(1)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks				
Step 2	Oracle says:	take $x = 1$	since	$(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$	(false)
	Corollary says:	take $x = 1$	since	$f(1) \leq f(3)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks				
Step 3	Oracle says:	take $x = 3$	since	$(\forall y \in \mathbf{N})\,(f(3) \leq f(y))$	(false)
	Corollary says:	take $x = 3$	since	$f(3) \leq f(7)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks				
Step 4	Oracle says:	take $x = 7$	since	$(\forall y \in \mathbf{N})\,(f(7) \leq f(y))$	(false)
		. . . .			
Step 11	Oracle says:	take $x = 1023$	since	$(\forall y \in \mathbf{N})\,(f(1023) \leq f(y))$	(false)
	Corollary says:	take $x = 1023$	since	$f(1023) \leq f(2047)$	(true)

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ_1^0-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1	Oracle says:	take $x = 0$	since $(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$	(false)
	Corollary says:	take $x = 0$	since $f(0) \leq f(1)$	(false)
	Σ_1^0-extractor evaluates incorrect justification and backtracks			
Step 2	Oracle says:	take $x = 1$	since $(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$	(false)
	Corollary says:	take $x = 1$	since $f(1) \leq f(3)$	(false)
	Σ_1^0-extractor evaluates incorrect justification and backtracks			
Step 3	Oracle says:	take $x = 3$	since $(\forall y \in \mathbf{N})\,(f(3) \leq f(y))$	(false)
	Corollary says:	take $x = 3$	since $f(3) \leq f(7)$	(false)
	Σ_1^0-extractor evaluates incorrect justification and backtracks			
Step 4	Oracle says:	take $x = 7$	since $(\forall y \in \mathbf{N})\,(f(7) \leq f(y))$	(false)
		. . . .		
Step 11	Oracle says:	take $x = 1023$	since $(\forall y \in \mathbf{N})\,(f(1023) \leq f(y))$	(false)
	Corollary says:	take $x = 1023$	since $f(1023) \leq f(2047)$	(true)
	Σ_1^0-extractor evaluates correct justification and returns $x = 1023$			

Transcript of the extraction process

Take $f(x) = |x - 1000|$ (real minimum at $x = 1000$)

and apply Σ^0_1-extraction to the proof of $(\exists x \in \mathbf{N})\,(f(x) \leq f(2x+1))$

Step 1	Oracle says:	take $x = 0$	since $(\forall y \in \mathbf{N})\,(f(0) \leq f(y))$	(false)
	Corollary says:	take $x = 0$	since $f(0) \leq f(1)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks			
Step 2	Oracle says:	take $x = 1$	since $(\forall y \in \mathbf{N})\,(f(1) \leq f(y))$	(false)
	Corollary says:	take $x = 1$	since $f(1) \leq f(3)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks			
Step 3	Oracle says:	take $x = 3$	since $(\forall y \in \mathbf{N})\,(f(3) \leq f(y))$	(false)
	Corollary says:	take $x = 3$	since $f(3) \leq f(7)$	(false)
	Σ^0_1-extractor evaluates incorrect justification and backtracks			
Step 4	Oracle says:	take $x = 7$	since $(\forall y \in \mathbf{N})\,(f(7) \leq f(y))$	(false)
		. . . .		
Step 11	Oracle says:	take $x = 1023$	since $(\forall y \in \mathbf{N})\,(f(1023) \leq f(y))$	(false)
	Corollary says:	take $x = 1023$	since $f(1023) \leq f(2047)$	(true)
	Σ^0_1-extractor evaluates correct justification and returns $x = 1023$			

Note that answer $x = 1023$ is correct... but not the point where f reaches its minimum

Extraction in the Σ^0_n-case (1/2)

Definition (Conditional refutation)

$r_A \in \Lambda$ is a conditional refutation of the predicate $A(x)$ if

$$\text{For all } n \in \mathbb{N} \text{ such that } \mathscr{M} \not\models A(n): \quad r_A\,\overline{n} \;\Vdash\; \neg A(n)$$

Extraction in the Σ^0_n-case (1/2)

Definition (Conditional refutation)

$r_A \in \Lambda$ is a conditional refutation of the predicate $A(x)$ if

$$\text{For all } n \in \mathbb{N} \text{ such that } \mathscr{M} \not\models A(n): \quad r_A\,\overline{n} \ \Vdash\ \neg A(n)$$

- Such a conditional refutation can be constructed for every predicate $A(x)$ of 1st-order arithmetic

Extraction in the Σ^0_n-case (1/2)

Definition (Conditional refutation)

$r_A \in \Lambda$ is a conditional refutation of the predicate $A(x)$ if

$$\text{For all } n \in \mathbb{N} \text{ such that } \mathscr{M} \not\models A(n): \quad r_A\,\bar{n} \Vdash \neg A(n)$$

- Such a conditional refutation can be constructed for every predicate $A(x)$ of 1st-order arithmetic

  This result is a consequence of the following

Theorem (Realizing true arithmetic formulas) [Krivine-Miquey]

For every formula $A(x_1, \ldots, x_k)$ of 1st-order arithmetic, there exists a closed proof-like term t_A such that:

$$\text{If} \quad \mathscr{M} \models A(n_1, \ldots, n_k), \quad \text{then} \quad t_A\,\bar{n}_1 \cdots \bar{n}_k \Vdash A(n_1, \ldots, n_k)$$

(for all $n_1, \ldots, n_k \in \mathbb{N}$)

Extraction in the Σ^0_n-case (2/2)

The Kamikaze extraction method [M. 2009]

Let

1. $t_0 \Vdash (\exists x \in \mathbb{N})\, A(x)$
2. r_A a conditional refutation of the predicate $A(x)$

Then the process

$$t_0 \star M\,(\lambda xy\,.\,\mathrm{print}\,x\,(r_A\,x\,y)) \cdot \diamond$$

displays a correct witness after finitely many evaluation steps

Extraction in the Σ^0_n-case (2/2)

The Kamikaze extraction method [M. 2009]

Let

1. $t_0 \Vdash (\exists x \in \mathbb{N})\, A(x)$
2. r_A a conditional refutation of the predicate $A(x)$

Then the process

$$t_0 \star M\,(\lambda xy\,.\,\mathrm{print}\,x\,(r_A\,x\,y)) \cdot \diamond$$

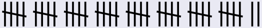
displays a correct witness after finitely many evaluation steps

- **Remark:** No correctness invariant is ensured as soon as the (first) correct witness has been displayed!

  After, anything may happen: crash, infinite loop, displaying incorrect witnesses, etc. (Kamikaze behavior)

Interlude: on numeration systems

- Numeration systems used in the History:

Tally sticks	(35000 BC)	
Babylonian	(3100 BC)	
Egyptian	(3000 BC)	
Roman	(1000 BC)	XLII
Hindu-Arabic	(300 AD)	42

Interlude: on numeration systems

- Numeration systems used in the History:

Tally sticks	(35000 BC)	𝍸𝍸𝍸𝍸𝍸𝍸𝍸𝍸𝍷𝍷
Babylonian	(3100 BC)	𒌋𒌋𒌋𒌋𒁹𒁹
Egyptian	(3000 BC)	𓎆𓎆𓎆𓎆𓏤𓏤
Roman	(1000 BC)	XLII
Hindu-Arabic	(300 AD)	42

- Numeration systems used in Logic:

Interlude: on numeration systems

- Numeration systems used in the History:

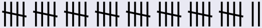
Tally sticks	(35000 BC)	𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 ||
Babylonian	(3100 BC)	𒌋𒌋𒌋𒌋𒁹𒁹
Egyptian	(3000 BC)	𓎆𓎆𓎆𓎆𓏺𓏺
Roman	(1000 BC)	XLII
Hindu-Arabic	(300 AD)	42

- Numeration systems used in Logic:

Peano: $ssssssssssssssssssssssssssssssssssssssssss0$

Interlude: on numeration systems

- Numeration systems used in the History:

Tally sticks	(35000 BC)	
Babylonian	(3100 BC)	
Egyptian	(3000 BC)	
Roman	(1000 BC)	XLII
Hindu-Arabic	(300 AD)	42

- Numeration systems used in Logic:

Peano: $ssssssssssssssssssssssssssssssssssssssssss0$

Church: $\lambda xf\,.\,f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f\,x))$

Interlude: on numeration systems

- Numeration systems used in the History:

Tally sticks	(35000 BC)	𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 𝍸 ||
Babylonian	(3100 BC)	𒌋𒌋𒌋𒌋𒁹𒁹
Egyptian	(3000 BC)	𓎆𓎆𓎆𓎆𓏺𓏺
Roman	(1000 BC)	XLII
Hindu-Arabic	(300 AD)	42

- Numeration systems used in Logic:

Peano: $ssssssssssssssssssssssssssssssssssssssssss0$

Church:

$$\lambda xf\,.\,f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(f(\\ f(f(f(f(f(f(f(f(f(f(f(f\,x)))$$

Krivine:

$$(\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))(\\ (\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))(\\ (\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))(\\ (\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))(\\ (\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))(\\ (\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))(\\ (\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))((\lambda nxf.f(nxf))(\\ (\lambda xf.x))$$

Primitive numerals (1/2)

To get rid of Krivine numerals $\bar{n} = \bar{s}^n\bar{0}$ (cf paleolithic numeration)
we extend the machine with the following instructions:

Primitive numerals (1/2)

To get rid of Krivine numerals $\bar{n} = \bar{s}^n\bar{0}$ (cf paleolithic numeration)
we extend the machine with the following instructions:

- For every natural number $n \in \mathbb{N}$, an instruction $\widehat{n} \in \mathcal{K}$ with no evaluation rule (i.e. inert constant: pure data)

  Intuition: $\widehat{n} \star \pi \succ$ `segmentation fault`

Primitive numerals (1/2)

To get rid of Krivine numerals $\bar{n} \;=\; \bar{s}^n\bar{0}$ (cf paleolithic numeration)
we extend the machine with the following instructions:

- For every natural number $n \in \mathbb{N}$, an instruction $\widehat{n} \in \mathcal{K}$ with no evaluation rule (i.e. inert constant: pure data)

  Intuition: $\widehat{n} \star \pi \;\succ\;$ `segmentation fault`

- An instruction null $\in \mathcal{K}$ with the rules

$$\mathsf{null} \star \widehat{n} \cdot u \cdot v \quad\succ\quad \begin{cases} u \star \pi & \text{if } n = 0 \\ v \star \pi & \text{otherwise} \end{cases}$$

Primitive numerals (1/2)

To get rid of Krivine numerals $\bar{n} \;=\; \bar{s}^n\bar{0}$ (cf paleolithic numeration)
we extend the machine with the following instructions:

- For every natural number $n \in \mathbb{N}$, an instruction $\widehat{n} \in \mathcal{K}$ with no evaluation rule (i.e. inert constant: pure data)

  Intuition: $\widehat{n} \star \pi \;\succ\;$ `segmentation fault`

- An instruction null $\in \mathcal{K}$ with the rules

$$\mathrm{null} \star \widehat{n} \cdot u \cdot v \quad \succ \quad \begin{cases} u \star \pi & \text{if } n = 0 \\ v \star \pi & \text{otherwise} \end{cases}$$

- Instructions $\check{f} \in \mathcal{K}$ with the rules

$$\check{f} \star \widehat{n}_1 \cdots \widehat{n}_k \cdot u \cdot \pi \quad \succ \quad u \star \widehat{m} \cdot \pi \qquad \text{where } m = f(n_1, \ldots, n_k)$$

  for all the usual arithmetic operations

Primitive numerals (2/2)

- Call-by-value implication, yet another definition:

Formulas $A, B \quad ::= \quad \cdots \quad | \quad [e] \Rightarrow A$

with the semantics: $\|\{e\} \Rightarrow A\| \; = \; \{\widehat{n} \cdot \pi \; : \; n = e^{\mathbb{N}}, \; \pi \in \|A\|\}$

Primitive numerals (2/2)

- Call-by-value implication, yet another definition:

Formulas $A, B \quad ::= \quad \cdots \quad | \quad [e] \Rightarrow A$

with the semantics: $\|\{e\} \Rightarrow A\| \; = \; \{\widehat{n} \cdot \pi \;:\; n = e^{\mathbb{N}},\; \pi \in \|A\|\}$

- Redefining the set of natural numbers:

$$\mathbb{N}' \;:=\; \{x \;:\; \forall Z\, (([x] \Rightarrow Z) \Rightarrow Z)\}$$

Primitive numerals (2/2)

- Call-by-value implication, yet another definition:

Formulas $$A, B \quad ::= \quad \cdots \quad \mid \quad [e] \Rightarrow A$$

with the semantics: $$\|\{e\} \Rightarrow A\| \;=\; \{\widehat{n} \cdot \pi \;:\; n = e^{\mathbb{N}},\; \pi \in \|A\|\}$$

- Redefining the set of natural numbers:

$$\mathbb{N}' \;:=\; \{x \;:\; \forall Z\,(([x] \Rightarrow Z) \Rightarrow Z)\}$$

$$\begin{array}{lcl}
\mathrm{box} \;:=\; \lambda k\,.\,k\,x & \Vvdash & \forall x\,([x] \Rightarrow x \in \mathbb{N}') \\
\mathrm{box}\,\widehat{n} & \Vvdash & n \in \mathbb{N}' \\
\lambda n\,.\,n\,\lambda x\,.\,\check{s}\,x\,\mathrm{box} & \Vvdash & (\forall x \in \mathbb{N}')(s(x) \in \mathbb{N}') \\
\lambda nm\,.\,n\,\lambda x\,.\,m\,\lambda y\,.\,(\check{+})\,x\,y\,\mathrm{box} & \Vvdash & (\forall x, y \in \mathbb{N}')(x + y \in \mathbb{N}')
\end{array}$$

Primitive numerals (2/2)

- Call-by-value implication, yet another definition:

Formulas $A, B \quad ::= \quad \cdots \quad | \quad [e] \Rightarrow A$

with the semantics: $\|\{e\} \Rightarrow A\| \;=\; \{\widehat{n} \cdot \pi \;:\; n = e^{\mathbb{N}},\ \pi \in \|A\|\}$

- Redefining the set of natural numbers:

$$\mathbb{N}' \;:=\; \{x \;:\; \forall Z\,(([x] \Rightarrow Z) \Rightarrow Z)\}$$

$$\begin{array}{lcl}
\mathrm{box} \;:=\; \lambda k\,.\,k\,x & \Vdash & \forall x\,([x] \Rightarrow x \in \mathbb{N}') \\
\mathrm{box}\,\widehat{n} & \Vdash & n \in \mathbb{N}' \\
\lambda n\,.\,n\,\lambda x\,.\,\check{s}\,x\,\mathrm{box} & \Vdash & (\forall x \in \mathbb{N}')(s(x) \in \mathbb{N}') \\
\lambda nm\,.\,n\,\lambda x\,.\,m\,\lambda y\,.\,(\check{+})\,x\,y\,\mathrm{box} & \Vdash & (\forall x, y \in \mathbb{N}')(x + y \in \mathbb{N}')
\end{array}$$

$$\begin{array}{rcl}
\mathrm{rec\_cbv} & := & \lambda z_0 z_s\,.\,\mathbf{Y}\,\lambda rx\,.\,\mathrm{null}\,x\,z_0\,((\check{-})\,x\,\hat{1}\,\lambda y\,.\,z_s\,y\,(r\,y)) \\
 & \Vdash & \forall Z\,[Z(0) \;\Rightarrow\; \forall y\,([y] \Rightarrow Z(y) \Rightarrow Z(s(y))) \;\Rightarrow\; \forall x\,([x] \Rightarrow Z(x))] \\
\mathrm{rec} & := & \lambda z_0 z_s n\,.\,n\,\lambda x\,.\,\mathrm{rec\_cbv}\,z_0\,(\lambda yz\,.\,z_s\,(\mathrm{box}\,y)\,z)\,x \\
 & \Vdash & \forall Z\,[Z(0) \Rightarrow (\forall y \in \mathbb{N}')(Z(y) \Rightarrow Z(s(y))) \;\Rightarrow\; (\forall x \in \mathbb{N}')Z(x)]
\end{array}$$

Primitive numerals (2/2)

- Call-by-value implication, yet another definition:

Formulas $\quad A, B \quad ::= \quad \cdots \quad | \quad [e] \Rightarrow A$

with the semantics: $\quad \|\{e\} \Rightarrow A\| \;=\; \{\widehat{n} \cdot \pi \;:\; n = e^{\mathbb{N}},\; \pi \in \|A\|\}$

- Redefining the set of natural numbers:

$$\mathbb{N}' \;:=\; \{x \;:\; \forall Z\,(([x] \Rightarrow Z) \Rightarrow Z)\}$$

$$\begin{array}{lcl}
\mathrm{box} \;:=\; \lambda k\,.\,k\,x & \Vdash & \forall x\,([x] \Rightarrow x \in \mathbb{N}') \\
\mathrm{box}\,\widehat{n} & \Vdash & n \in \mathbb{N}' \\
\lambda n\,.\,n\,\lambda x\,.\,\check{s}\,x\,\mathrm{box} & \Vdash & (\forall x \in \mathbb{N}')(s(x) \in \mathbb{N}') \\
\lambda nm\,.\,n\,\lambda x\,.\,m\,\lambda y\,.\,(\check{+})\,x\,y\,\mathrm{box} & \Vdash & (\forall x, y \in \mathbb{N}')(x + y \in \mathbb{N}')
\end{array}$$

$$\begin{array}{rl}
\mathrm{rec\_cbv} \;:= & \lambda z_0 z_s\,.\,\mathbf{Y}\,\lambda rx\,.\,\mathrm{null}\,x\,z_0\,((\check{-})\,x\,\hat{1}\,\lambda y\,.\,z_s\,y\,(r\,y)) \\
\Vdash & \forall Z\,[Z(0) \;\Rightarrow\; \forall y\,([y] \Rightarrow Z(y) \Rightarrow Z(s(y))) \;\Rightarrow\; \forall x\,([x] \Rightarrow Z(x))] \\
\mathrm{rec} \;:= & \lambda z_0 z_s n\,.\,n\,\lambda x\,.\,\mathrm{rec\_cbv}\,z_0\,(\lambda yz\,.\,z_s\,(\mathrm{box}\,y)\,z)\,x \\
\Vdash & \forall Z\,[Z(0) \Rightarrow (\forall y \in \mathbb{N}')(Z(y) \Rightarrow Z(s(y))) \;\Rightarrow\; (\forall x \in \mathbb{N}')Z(x)]
\end{array}$$

- **Conclusion:** $\quad \Vdash\; \forall x\,(x \in \mathbb{N}' \;\Leftrightarrow\; x \in \mathbb{N})$