Logic in Computer Science V

Logic in Computer Science V

Logic in Computer Science V

Recommended reading

- Zlatuška, *Lambda-kalkul*
- Barendregt, *Chapter D.7. in Handbook of Logic.*
- Sørensen and Urzyczyn, *Lectures on the Curry-Howard Isomorphism*

Lesson 10, λ-calculus and intuitionistic logic

λ-calculus is an important calculus that can be used (mainly) for

- formalizing computations
- programming languages
- formalizing logic

It is connected with intuitionistic logic. Extensions that are connected with classical logic are also known, but they are not so natural.

Lesson 10, λ-calculus and intuitionistic logic

λ-calculus is an important calculus that can be used (mainly) for

- formalizing computations
- programming languages
- formalizing logic

It is connected with intuitionistic logic. Extensions that are connected with classical logic are also known, but they are not so natural.

We will see that the formalizations based on the λ-calculus are similar to those we have seen.

main types of λ calculus

1. type-free λ-calculus
   - combinatory algebra, a.k.a. *combinatory logic*
   - term rewriting system
2. typed λ-calculus, a.k.a. *type theory*;

main types of λ calculus

1. type-free λ-calculus
   - combinatory algebra, a.k.a. *combinatory logic*
   - term rewriting system
2. typed λ-calculus, a.k.a. *type theory*; this is connected with intuitionistic propositional logic.

main types of λ calculus

1. type-free λ-calculus
   - combinatory algebra, a.k.a. *combinatory logic*
   - term rewriting system
2. typed λ-calculus, a.k.a. *type theory*; this is connected with intuitionistic propositional logic.

For a connection with first order logic, one needs *dependent types*.

combinatory algebra

Idea: every object is a function and an argument at the same time.

combinatory algebra

Idea: every object is a function and an argument at the same time.

- one binary operation, *application*, xy ("x applied to y")
  we will use *association to the left*

combinatory algebra

Idea: every object is a function and an argument at the same time.

- one binary operation, *application*, xy ("x applied to y") we will use *association to the left*
- axioms:
  1. *combinatory completeness:* for every term A,
  $$\exists f \forall x_1 \ldots \forall x_n (f x_1 \ldots x_n = A),$$
  2. *extensionality:*
  $$\forall x (fx = gx) \to f = g.$$
  3. *nontriviality:*
  $$\exists x, y (x \neq y).$$

To get the combinatorial completeness one can use

1. either λ-*terms*, $\lambda x.A$[1] with axioms

$$(\lambda x.A)B = A[x/B],$$

   called β-*conversion*,[2]

2. or constants K, S, called *combinators*, and axioms
   - $Kxy = x$,
   - $Sxyz = xz(yz)$.

[1] Applying λx to a term is called λ-*abstraction*; x is not free in $\lambda x.A$.

[2] in less precise, but more intuitive notation: $(\lambda x.A[x])B = A[B]$

To get the combinatorial completeness one can use

1. either λ-*terms*, $\lambda x.A$[1] with axioms

$$(\lambda x.A)B = A[x/B],$$

called β-*conversion*,[2]

2. or constants K, S, called *combinators*, and axioms
   - $Kxy = x$,
   - $Sxyz = xz(yz)$.

Note: Two special instances suffice for combinatorial completeness!

[1] Applying λx to a term is called λ-*abstraction*; x is not free in $\lambda x.A$.

[2] in less precise, but more intuitive notation: $(\lambda x.A[x])B = A[B]$

To get the combinatorial completeness one can use

1. either λ-*terms*, $\lambda x.A$[1] with axioms

$$(\lambda x.A)B = A[x/B],$$

   called β-*conversion*,[2]

2. or constants K, S, called *combinators*, and axioms
   - $Kxy = x$,
   - $Sxyz = xz(yz)$.

Note: Two special instances suffice for combinatorial completeness!

Example

- $K = \lambda x \lambda y.x$
- $S = \lambda x \lambda y \lambda z.xz(yz)$

[1] Applying λx to a term is called λ-*abstraction*; x is not free in $\lambda x.A$.
[2] in less precise, but more intuitive notation: $(\lambda x.A[x])B = A[B]$

Proof.

ad 1. by iterating $(\lambda x.A)y = A[x/y]$ we get

$$(\lambda x_1 \ldots \lambda x_n.A)y_1 \ldots y_n = A[x_1/y_1, \ldots, x_n/y_n].$$

Recall that we needed an f such that

$$f y_1 \ldots y_n = A[x_1/y_1, \ldots, x_n/y_n].$$

$\square$

Proof.

ad 2. we construct the λ-terms from the combinators K, S.

Proof.

ad 2. we construct the λ-terms from the combinators K, S.

- define the combinator $I := SKK$ and show $Ix = x$ (Exercise!)

Proof.

ad 2. we construct the λ-terms from the combinators K, S.

- define the combinator $I := SKK$ and show $Ix = x$ (Exercise!)
- prove combinatorial completeness by induction
  - base cases:
    $\lambda x.x \mapsto I$,
    $\lambda x.y \mapsto Ky$.
  - induction step: $\lambda x.AB \mapsto S(\lambda x.A)(\lambda x.B)$; then
    $(S(\lambda x.A)(\lambda x.B))z =$
    $((\lambda x.A)z)(\lambda x.B)z =$ (by definition of S)
    $A[x/z]B[x/z] = AB[x/z]$ (by induction assumption)

$\square$

Fixed Point Theorem

Theorem

1. *For every λ-term A there exists a λ-term B such that*

$$B = AB.$$

Fixed Point Theorem

Theorem

1. For every λ-term A there exists a λ-term B such that

$$B = AB.$$

2. Moreover, there exists a λ-term F that produces fixed-points for every term A

$$FA = A(FA)$$

Fixed Point Theorem

Theorem

1. For every λ-term A there exists a λ-term B such that

$$B = AB.$$

2. Moreover, there exists a λ-term F that produces fixed-points for every term A

$$FA = A(FA)$$

Proof.

1. Define $C := \lambda x.A(xx)$ and $B := CC$. Then

Fixed Point Theorem

Theorem

1. For every λ-term A there exists a λ-term B such that

$$B = AB.$$

2. Moreover, there exists a λ-term F that produces fixed-points for every term A

$$FA = A(FA)$$

Proof.

1. Define $C := \lambda x.A(xx)$ and $B := CC$. Then

$$B = CC = (\lambda x.A(xx))C = A(CC) = AB.$$

2. (Exercise) $\square$

Fixed Point Theorem

Theorem

1. For every λ-term A there exists a λ-term B such that

$$B = AB.$$

2. Moreover, there exists a λ-term F that produces fixed-points for every term A

$$FA = A(FA)$$

Proof.

1. Define $C := \lambda x.A(xx)$ and $B := CC$. Then

$B = CC = (\lambda x.A(xx))C = A(CC) = AB.$

2. (Exercise) $\square$

Intuition: $C \leftrightarrow$ "x written twice has property A"

Exercise

- *Prove 2.*
- *Write the fixed-point using combinators I, K, S.*

term rewriting

Often we can simplify λ-terms by rewriting:

[3]terminology: "conversion" for $=$, "reduction" for $\to$

[4]we will not use η-reduction in the sequel

term rewriting

Often we can simplify λ-terms by rewriting:

- $(\lambda x.A)B \to A[x/B]$ (β-reduction)[3]
- $\lambda x.Ax \to A$ (η-reduction) if $x \notin Var(A)$,[4]

[3] terminology: "conversion" for $=$, "reduction" for $\to$

[4] we will not use η-reduction in the sequel

term rewriting

Often we can simplify λ-terms by rewriting:

- $(\lambda x.A)B \rightarrow A[x/B]$ (β-reduction)[3]
- $\lambda x.Ax \rightarrow A$ (η-reduction) if $x \notin Var(A)$,[4]

but not always.

Example

$\Omega := (\lambda x.xx)(\lambda x.xx)$ *remains the same after β-reduction.*

[3] terminology: "conversion" for $=$, "reduction" for $\rightarrow$

[4] we will not use η-reduction in the sequel

term rewriting

Often we can simplify λ-terms by rewriting:

- $(\lambda x.A)B \rightarrow A[x/B]$ (β-reduction)[3]
- $\lambda x.Ax \rightarrow A$ (η-reduction) if $x \notin Var(A)$,[4]

but not always.

Example

$\Omega := (\lambda x.xx)(\lambda x.xx)$ *remains the same after* β*-reduction.*

β-reduction can *increase* the size.

Example

Suppose B *is a long term, then*

- $(\lambda x.xx)B \rightarrow BB$

produces almost a twice long term.

---

[3] terminology: "conversion" for $=$, "reduction" for $\rightarrow$

[4] we will not use η-reduction in the sequel

Definition

1. A λ-term is in a normal form if it does not contain a subterm $(\lambda x.A)B$ (called *redex*).

Definition

1. A λ-term is in a normal form if it does not contain a subterm $(\lambda x.A)B$ (called *redex*).

2. Normalization is a sequence of β-reductions that produces a term in the normal form.

Definition

1. A λ-term is in a normal form if it does not contain a subterm $(\lambda x.A)B$ (called *redex*).

2. Normalization is a sequence of β-reductions that produces a term in the normal form.

We will see that

- redex $\leftrightarrow$ cut
- normailzation $\leftrightarrow$ cut-elimination

Definition

1. A λ-term is in a normal form if it does not contain a subterm $(\lambda x.A)B$ (called *redex*).

2. Normalization is a sequence of β-reductions that produces a term in the normal form.

We will see that

- redex $\leftrightarrow$ cut
- normailzation $\leftrightarrow$ cut-elimination

Also very important (but we will not deal with it)

- normailzation $\leftrightarrow$ computation

Theorem

If a λ-term can be reduced to a normal form, then the normal form is unique.

Theorem

If a λ-term can be reduced to a normal form, then the normal form is unique.

Proof.

is based on the Chruch-Rosser property:

- if $A \to B_1$ and $A \to B_2$, then there exists C such that $B_1 \to C$ and $B_2 \to C$.

□

typed λ-calculus

Idea: *one can only apply x to y if they have appropriate types.*

typed λ-calculus

Idea: *one can only apply x to y if they have appropriate types.*

Simple types:

- type variables $u, v, \ldots,$
- if σ and τ are types, $\sigma \to \tau$ is a type.

Notation:

- "A has type σ" is abbreviated by $A : \sigma$ (sometimes also A^{σ}).

typed λ-calculus

Idea: *one can only apply x to y if they have appropriate types.*

Simple types:

- type variables $u, v, \ldots,$
- if σ and τ are types, $\sigma \to \tau$ is a type.

Notation:

- "A has type σ" is abbreviated by $A : \sigma$ (sometimes also A^σ).

Rule:

- AB is well-formed if $A : \sigma \to \tau$ and $B : \sigma$,
- then $AB : \tau$.

Given an untyped λ-term it may not be possible to assign types to variables and combinators so that it is a well-formed typed term.

Given an untyped λ-term it may not be possible to assign types to variables and combinators so that it is a well-formed typed term.

If it it possible, we say that the term is typable.

Given an untyped λ-term it may not be possible to assign types to variables and combinators so that it is a well-formed typed term.

If it it possible, we say that the term is typable.

According to *"typing a là Church"*, one should always declare the types of variables and combinators to prevent untypability.

examples

1. For every types ρ, σ, τ we have combinators
   - $I_\rho = \lambda x.x \ : \ \rho \to \rho$
     where $x : \rho$,
   - $K_{\rho,\sigma} = \lambda x \lambda y.x \ : \ \rho \to (\sigma \to \rho)$
     where $x : \rho$, $y : \sigma$,
   - $S_{\rho,\sigma,\tau} = \lambda x \lambda y \lambda z.xz(yz) \ : \ (\rho \to (\sigma \to \tau)) \to ((\rho \to \sigma) \to (\rho \to \tau))$
     where $x : \rho \to (\sigma \to \tau)$, $y : \rho \to \sigma$, $z : \rho$.

examples

1. For every types ρ, σ, τ we have combinators
   - $I_\rho = \lambda x.x \;:\; \rho \to \rho$
     where $x : \rho$,
   - $K_{\rho,\sigma} = \lambda x \lambda y.x \;:\; \rho \to (\sigma \to \rho)$
     where $x : \rho$, $y : \sigma$,
   - $S_{\rho,\sigma,\tau} = \lambda x \lambda y \lambda z.xz(yz) \;:\; (\rho \to (\sigma \to \tau)) \to ((\rho \to \sigma) \to (\rho \to \tau))$
     where $x : \rho \to (\sigma \to \tau)$, $y : \rho \to \sigma$, $z : \rho$.

2. $II := (\lambda x.x)(\lambda y.y)$ is typable:
   - let the first $I : (\tau \to \tau) \to (\tau \to \tau)$
   - the second $I : \tau \to \tau$
   - then $II : \tau \to \tau$

examples

1. For every types ρ, σ, τ we have combinators
   - $I_\rho = \lambda x.x \ : \ \rho \to \rho$
     where $x : \rho$,
   - $K_{\rho,\sigma} = \lambda x \lambda y.x \ : \ \rho \to (\sigma \to \rho)$
     where $x : \rho$, $y : \sigma$,
   - $S_{\rho,\sigma,\tau} = \lambda x \lambda y \lambda z.xz(yz) \ : \ (\rho \to (\sigma \to \tau)) \to ((\rho \to \sigma) \to (\rho \to \tau))$
     where $x : \rho \to (\sigma \to \tau)$, $y : \rho \to \sigma$, $z : \rho$.

2. $II := (\lambda x.x)(\lambda y.y)$ is typable:
   - let the first $I : (\tau \to \tau) \to (\tau \to \tau)$
   - the second $I : \tau \to \tau$
   - then $II : \tau \to \tau$

3. $\Omega := (\lambda x.xx)(\lambda x.xx)$ is not typable, for it remains the same after β-reduction.

Algorithms for typing λ-terms are based on unification (of types).

- $\lambda x.x \ : \ \rho \to \rho$
- $\lambda x \lambda y.x \ : \ \rho \to (\sigma \to \rho)$
- $\lambda x \lambda y \lambda z.xz(yz) \ : \ (\rho \to (\sigma \to \tau)) \to ((\rho \to \sigma) \to (\rho \to \tau))$

- $\lambda x.x \ : \ \rho \to \rho$
- $\lambda x \lambda y.x \ : \ \rho \to (\sigma \to \rho)$
- $\lambda x \lambda y \lambda z.xz(yz) \ : \ (\rho \to (\sigma \to \tau)) \to ((\rho \to \sigma) \to (\rho \to \tau))$

Note: The types are propositional tautologies.

- $\lambda x.x \ : \ \rho \to \rho$
- $\lambda x \lambda y.x \ : \ \rho \to (\sigma \to \rho)$
- $\lambda x \lambda y \lambda z.xz(yz) \ : \ (\rho \to (\sigma \to \tau)) \to ((\rho \to \sigma) \to (\rho \to \tau))$

Note: The types are propositional tautologies.

Furthermore, the rule about application

- if $A : \sigma \to \tau$ and $B : \sigma$, then $AB : \tau$.

is modus ponens.

- $\lambda x.x \ : \ \rho \to \rho$
- $\lambda x \lambda y.x \ : \ \rho \to (\sigma \to \rho)$
- $\lambda x \lambda y \lambda z.xz(yz) \ : \ (\rho \to (\sigma \to \tau)) \to ((\rho \to \sigma) \to (\rho \to \tau))$

Note: The types are propositional tautologies.

Furthermore, the rule about application

- if $A : \sigma \to \tau$ and $B : \sigma$, then $AB : \tau$.

is modus ponens.

Hence, λ-calculus defines some propositional logic.

the Curry-Howard correspondence/isomorphism

λ-terms	proofs
types	formulas
combinators	axioms
application	modus ponens
and more ...	

the Curry-Howard correspondence/isomorphism

λ-terms	proofs
types	formulas
combinators	axioms
application	modus ponens
and more ...	

Example

Recall that $SKK = I$ and $I : \tau \to \tau$. Hence SKK is a proof of $\tau \to \tau$, if it can be properly typed.

Exercise

Find the types for SKK!

Theorem

The λ-calculus defines intuitionistic logic of implication.

Theorem

The λ-calculus defines intuitionistic logic of implication.

Proof-idea

Theorem

The λ-calculus defines intuitionistic logic of implication.

Proof-idea

1. Completeness: Show that the formulas corresponding to the types of K and S and modus ponens axiomatize intuitionistic logic of implication.

Theorem

The λ-calculus defines intuitionistic logic of implication.

Proof-idea

1. Completeness: Show that the formulas corresponding to the types of K and S and modus ponens axiomatize intuitionistic logic of implication.
2. Soundness: Since every λ-term can be constructed from K and S, only intuitionistic tautologies are provable.

$\square$

intuitionistic logic

The standard logic is called classical logic to be distinguished from intuitionistic logic which is a.k.a. constructive logic.

intuitionistic logic

The standard logic is called classical logic to be distinguished from intuitionistic logic which is a.k.a. constructive logic.

- language: $\to, \wedge, \vee, \neg$ and $\forall, \exists$;
  (often $\perp$ instead of $\neg$ and $\neg A$ is expressed by $A \to \perp$)
- weaker than classical logic, e.g. t.f.a. not provable in int. logic:
  - $A \vee \neg A$
  - $\neg\neg A \to A$
  - $\neg\forall x.A \to \exists x.\neg A$
- the connectives $\to, \wedge, \vee, \neg$ and quantifiers $\forall, \exists$ are independent (one cannot be defined from the others)

some constructive properties of intuitionistic logic

- if $\vdash A \vee B$, then either $\vdash A$ or $\vdash B$
- if $\vdash \exists x A(x)$, then $\vdash A(t)$ for some term t

some constructive properties of intuitionistic logic

- if $\vdash A \vee B$, then either $\vdash A$ or $\vdash B$
- if $\vdash \exists x A(x)$, then $\vdash A(t)$ for some term t
- one cannot use proofs by contradiction to prove non-negated sentences
  - if we assume $\neg A$ and get $\perp$, we only can deduce $\neg\neg A$;
  - however, to prove $\neg B$, we can assume B a and prove $\perp$.

Propositional intuitionistic logic of implication is also weaker:

$$((p \to q) \to p) \to p$$

(*Peirce Law*) is a classical tautology, but not intuitionistic.

proof systems for intuitionistic logic

1. Hilbert style with carefully chosen axioms and rules.
   - this corresponds to the λ-calculus formalized using combinators

proof systems for intuitionistic logic

1. Hilbert style with carefully chosen axioms and rules.
   - this corresponds to the λ-calculus formalized using combinators

2. Sequent calculus with the restriction: at most one formula in the consequent, i.e.,

$$A_1, \ldots, A_n \to B \quad \text{or} \quad A_1, \ldots, A_n \to$$

proof systems for intuitionistic logic

1. Hilbert style with carefully chosen axioms and rules.
   - this corresponds to the λ-calculus formalized using combinators

2. Sequent calculus with the restriction: at most one formula in the consequent, i.e.,

$$A_1, \ldots, A_n \to B \quad \text{or} \quad A_1, \ldots, A_n \to$$

3. Natural deduction system with the negation elimination rule (="proof by contradiction") omitted.
   - this corresponds to the λ-calculus formalized using λ-terms.

natural deduction and λ-calculus

Again we restrict ourselves to the implicational fragment of propositional logic.

natural deduction and λ-calculus

Again we restrict ourselves to the implicational fragment of propositional logic.

Recall the nat. ded. rules for $\to$.

$\to$ introduction $\qquad$ $\to$ elimination

$$\frac{\begin{matrix}[A]\\ \vdots \\ B\end{matrix}}{A\to B} \qquad\qquad \frac{A \qquad A\to B}{B}$$

$$\dfrac{\begin{matrix}[A]\\ \vdots \\ B\end{matrix}}{A \to B} \qquad\qquad \dfrac{A \quad A \to B}{B}$$

Suppose we have a term $M : \beta$ with a free variable $x : \alpha$. Then

$$\lambda x.M \; : \; \alpha \to \beta$$

So λ-abstraction corresponds to $\to$ introduction. The object variable x is the assumption.

$$\dfrac{\begin{matrix}[A]\\ \vdots\\ B\end{matrix}}{A \to B} \qquad\qquad \dfrac{A \quad A \to B}{B}$$

Suppose we have a term $M : \beta$ with a free variable $x : \alpha$. Then

$$\lambda x.M \ : \ \alpha \to \beta$$

So λ-abstraction corresponds to $\to$ introduction. The object variable x is the assumption.

We already know: application corresponds to $\to$ elimination (= modus ponens).

In the system of natural deduction we have normailzation instead of cut-elimination. Normal proofs are, essentially, proofs without elimination rules.

In the system of natural deduction we have normailzation instead of cut-elimination. Normal proofs are, essentially, proofs without elimination rules.

Thus we can extend ...

the Curry-Howard correspondence/isomorphism

λ-terms	proofs
types	formulas
combinators	axioms
application	$\rightarrow$ elimination
object variable	assumption
λ-abstraction	$\rightarrow$ introduction
normalization of terms	normalization of proofs
and more ...	

Lesson 11, theories and complexity classes

For missing definitions and proofs see:

- S. Buss, Chapter 2, Handbook of Proof Theory
- P. Hájek and P. Pudlák, Metamathematics of First Order Arithmetic, Chapter V.

fragments of Peano Arithmetic

- $PA := Q$ plus induction axioms for all arithmetical formulas
- $I\Sigma_n := Q$ plus induction axioms for all Σ_n formulas

fragments of Peano Arithmetic

- $PA := Q$ plus induction axioms for all arithmetical formulas
- $I\Sigma_n := Q$ plus induction axioms for all Σ_n formulas

Theorem

The hierarchy

$$I\Sigma_1, I\Sigma_2, I\Sigma_3 \ldots$$

is strictly increasing.

fragments of Peano Arithmetic

- $PA :=$ Q plus induction axioms for all arithmetical formulas
- $I\Sigma_n :=$ Q plus induction axioms for all Σ_n formulas

Theorem

The hierarchy

$$I\Sigma_1, I\Sigma_2, I\Sigma_3 \ldots$$

is strictly increasing.

This means

$$Thm(I\Sigma_1) \subsetneq Thm(I\Sigma_2) \subsetneq Thm(I\Sigma_3) \subsetneq \ldots$$

where $Thm(T)$ is the set of all sentences provable in T.

Proof

The inclusions are trivially true, so we only need to show

$$I\Sigma_1 \neq I\Sigma_2 \neq I\Sigma_3 \neq \dots$$

To this end, we show for $n = 1, 2, 3 \dots$

1. $I\Sigma_n \nvdash Con(I\Sigma_n)$,
2. $I\Sigma_{n+1} \vdash Con(I\Sigma_n)$.

Proof

The inclusions are trivially true, so we only need to show

$$I\Sigma_1 \neq I\Sigma_2 \neq I\Sigma_3 \neq \dots$$

To this end, we show for $n = 1, 2, 3 \dots$

1. $I\Sigma_n \nvdash Con(I\Sigma_n)$,
2. $I\Sigma_{n+1} \vdash Con(I\Sigma_n)$.

1. by the 2nd inco. thm.

Proof

The inclusions are trivially true, so we only need to show

$$I\Sigma_1 \neq I\Sigma_2 \neq I\Sigma_3 \neq \dots$$

To this end, we show for $n = 1, 2, 3 \dots$

1. $I\Sigma_n \nvdash Con(I\Sigma_n)$,
2. $I\Sigma_{n+1} \vdash Con(I\Sigma_n)$.

1. by the 2nd inco. thm.

2. Idea:
   - use cut-elimination to show in $I\Sigma_{n+1}$: "if a contradiction is derivable in $I\Sigma_n$, then it is derivable only using Σ_n formulas;
   - prove in $I\Sigma_{n+1}$ that the universal closure of every formula in such a proof is true, hence $I\Sigma_n \nvdash \perp$.

Problem: if $\phi \in \Sigma_n$, then

$$\phi(0) \wedge \forall x(\phi(x) \rightarrow \phi(Sx)) \rightarrow \forall y \phi(y)$$

is a Δ_{n+2} formula. So we would need Π_{n+2} induction, i.e., $I\Sigma_{n+2}$.

Problem: if $\phi \in \Sigma_n$, then

$$\phi(0) \wedge \forall x(\phi(x) \rightarrow \phi(Sx)) \rightarrow \forall y \phi(y)$$

is a Δ_{n+2} formula. So we would need Π_{n+2} induction, i.e., $I\Sigma_{n+2}$.

In order to get proofs with sequents of Σ_n formulas, we need

1. replace induction axioms by a rule,
2. use free-cut elimination.

Problem: if $\phi \in \Sigma_n$, then

$$\phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \forall y \phi(y)$$

is a Δ_{n+2} formula. So we would need Π_{n+2} induction, i.e., $I\Sigma_{n+2}$.

In order to get proofs with sequents of Σ_n formulas, we need

1. replace induction axioms by a rule,
2. use free-cut elimination.

The induction rule in the sequent calculus

$$\frac{\Gamma, \phi(a) \;\to\; \Delta, \phi(S(a))}{\Gamma, \phi(0) \;\to\; \Delta, \phi(t)}$$

where a is an eigenvariable and t is an arbitrary term.

Problem: if $\phi \in \Sigma_n$, then

$$\phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \forall y \phi(y)$$

is a Δ_{n+2} formula. So we would need Π_{n+2} induction, i.e., $I\Sigma_{n+2}$.

In order to get proofs with sequents of Σ_n formulas, we need

1. replace induction axioms by a rule,
2. use free-cut elimination.

The induction rule in the sequent calculus

$$\frac{\Gamma, \phi(a) \ \to \ \Delta, \phi(S(a))}{\Gamma, \phi(0) \ \to \ \Delta, \phi(t)}$$

where a is an eigenvariable and t is an arbitrary term.

$I\Sigma_n$ can be axiomatized by Q and the induction rule for $\phi \in \Sigma_n$.

- A *free-cut* is a cut with a formula that is not a subformula of an axiom nor of a formula in an instance of the induction rule.
- A *free-cut free* proof is a proof without free cuts.
- One can show, already in $I\Sigma_1$, that free-cuts can be eliminated.

- A *free-cut* is a cut with a formula that is not a subformula of an axiom nor of a formula in an instance of the induction rule.
- A *free-cut free* proof is a proof without free cuts.
- One can show, already in $I\Sigma_1$, that free-cuts can be eliminated.

If all formulas in

$$\Gamma \ \rightarrow \ \Delta$$

are Σ_n or Π_n, then the universal closure

$$\forall \ \ldots \ \bigwedge \Gamma \rightarrow \bigvee \Delta \ \in \ \Pi_{n+1}.$$

Hence Π_{n+1} induction, which is derivable from Σ_{n+1} induction, suffices. $\square$

weak fragments

- $I\Delta_0$ (also denoted by $I\Sigma_0$)

Theorem (R. Parikh)

Let $\phi(x, y)$ be a bounded formula. If

$$I\Delta_0 \ \vdash \ \forall x \exists y . \phi(x, y),$$

then there exists a polynomial $p(x)$ such that

$$I\Delta_0 \ \vdash \ \forall x \exists y \leq p(x) . \phi(x, y).$$

weak fragments

- $I\Delta_0$ (also denoted by $I\Sigma_0$)

Theorem (R. Parikh)

Let $\phi(x, y)$ be a bounded formula. If

$$I\Delta_0 \vdash \forall x \exists y . \phi(x, y),$$

then there exists a polynomial $p(x)$ such that

$$I\Delta_0 \vdash \forall x \exists y \leq p(x) . \phi(x, y).$$

If x encodes (in binary) a string of length $\ell \approx \log x$, then $y \leq x^k$ encodes a string of length $\leq k\ell \approx k \log x$.

As we can only extend strings linearly, we cannot formalize polynomial time computations.

J. Paris and A. Wilkie

- $I\Delta_0 + \Omega_1$

where Ω_1 is an axiom saying $\forall x \exists y. y = x^{\lfloor \log(x+1) \rfloor}$.
(the relation $y = x^{\lfloor \log(x+1) \rfloor}$ is definable in $I\Delta_0$)

J. Paris and A. Wilkie

- $I\Delta_0 + \Omega_1$

where Ω_1 is an axiom saying $\forall x \exists y . y = x^{\lfloor \log(x+1) \rfloor}$.
(the relation $y = x^{\lfloor \log(x+1) \rfloor}$ is definable in $I\Delta_0$)

$\log(x^{\lfloor \log(x+1) \rfloor}) \approx (\log x)^2$

Hence we can increase the length of encoded sequences quadratically, consequently by any polynomial.

J. Paris and A. Wilkie

- $I\Delta_0 + \Omega_1$

where Ω_1 is an axiom saying $\forall x \exists y. y = x^{\lfloor \log(x+1) \rfloor}$.
(the relation $y = x^{\lfloor \log(x+1) \rfloor}$ is definable in $I\Delta_0$)

$\log(x^{\lfloor \log(x+1) \rfloor}) \approx (\log x)^2$

Hence we can increase the length of encoded sequences quadratically, consequently by any polynomial.

This enables one to formalize polynomial time computations.

the hierarchy of weak fragments (Bounded Arithmetic)

S. Buss, 1986

the hierarchy of weak fragments (Bounded Arithmetic)

S. Buss, 1986

We will focus on fragments $T_2^0, T_2^1, \ldots$.

- $T_2^i := BASIC + \Sigma_i^b\text{-}IND$.

BASIC is a finite set that determines the meaning of function symbols.

For $i = 0$ it is more natural to extend the original Buss's BASIC with a new function symbol and defining relations so that polynomial time computations are formalizable in it.

the hierarchy of weak fragments (Bounded Arithmetic)

S. Buss, 1986

We will focus on fragments $T_2^0, T_2^1, \ldots$.

- $T_2^i := BASIC + \Sigma_i^b\text{-}IND$.

BASIC is a finite set that determines the meaning of function symbols.

For $i = 0$ it is more natural to extend the original Buss's BASIC with a new function symbol and defining relations so that polynomial time computations are formalizable in it.

All function symbols define in $\mathbb{N}$ polynomial time computable functions.

the Polynomial Hierarchy

P$:= \Sigma_0^p$, **NP**$:= \Sigma_1^p$, **coNP**$:= \Pi_1^b$, Σ_2^p, Π_2^p,...

the Polynomial Hierarchy

$\mathbf{P}$:= Σ_0^p, $\mathbf{NP}$:= Σ_1^p, $\mathbf{coNP}$:= Π_1^b, Σ_2^p, Π_2^p,...

Hypothesis The Polynomial hierarchy is strictly increasing; in symbols:

$$\Sigma_0^p \subsetneq \Sigma_1^p \subsetneq \Sigma_2^p \subsetneq \dots$$

the Polynomial Hierarchy

$\mathbf{P} := \Sigma_0^p$, $\mathbf{NP} := \Sigma_1^p$, $\mathbf{coNP} := \Pi_1^b$, Σ_2^p, Π_2^p,...

Hypothesis The Polynomial hierarchy is strictly increasing; in symbols:

$$\Sigma_0^p \subsetneq \Sigma_1^p \subsetneq \Sigma_2^p \subsetneq \ldots$$

Theorem

If the Polynomial Hierarchy is strictly increasing, then so is the Bounded Arithmetic hierarchy.

the Polynomial Hierarchy

P$:= \Sigma^p_0$, **NP**$:= \Sigma^p_1$, **coNP**$:= \Pi^b_1$, Σ^p_2, Π^p_2,...

Hypothesis The Polynomial hierarchy is strictly increasing; in symbols:

$$\Sigma^p_0 \subsetneq \Sigma^p_1 \subsetneq \Sigma^p_2 \subsetneq \dots$$

Theorem

If the Polynomial Hierarchy is strictly increasing, then so is the Bounded Arithmetic hierarchy. More precisely, for all $i = 0, 1, \dots,$

$$\Sigma^p_{i+2} \neq \Pi^p_{i+2} \;\Rightarrow\; Thm(T^i_2) \neq Thm(T^{i+1}_2).$$

the Polynomial Hierarchy

P$:= \Sigma_0^p$, **NP**$:= \Sigma_1^p$, **coNP**$:= \Pi_1^b$, Σ_2^p, Π_2^p,...

Hypothesis The Polynomial hierarchy is strictly increasing; in symbols:

$$\Sigma_0^p \subsetneq \Sigma_1^p \subsetneq \Sigma_2^p \subsetneq \dots$$

Theorem

If the Polynomial Hierarchy is strictly increasing, then so is the Bounded Arithmetic hierarchy. More precisely, for all $i = 0, 1, \dots,$

$$\Sigma_{i+2}^p \neq \Pi_{i+2}^p \;\Rightarrow\; Thm(T_2^i) \neq Thm(T_2^{i+1}).$$

We do not know how to prove that the Bounded Arithmetic Hierarchy is strictly increasing without the hypothesis. More about it later.

We will prove the theorem only for $i = 0$. A generalization for all i is easy. Our main tool will be Herbrnad's Theorem.

We will prove the theorem only for $i = 0$. A generalization for all i is easy. Our main tool will be Herbrnad's Theorem.

Overview of the proof:

1. Skolemize T_2^0 using polynomial time computable functions to get a universal theory.
2. Apply Herbrand's Theorem.
3. Interpret the Herbrand disjunction as a program for interactive computation.
4. Interpret $\Sigma_1^b - Ind$ as a computational problem Max.
5. Show that Max cannot be solved by the interactive computation unless $\Sigma_2^p = \Pi_2^p$.

We will prove the theorem only for $i = 0$. A generalization for all i is easy. Our main tool will be Herbrnad's Theorem.

Overview of the proof:

1. Skolemize T_2^0 using polynomial time computable functions to get a universal theory.
2. Apply Herbrand's Theorem.
3. Interpret the Herbrand disjunction as a program for interactive computation.
4. Interpret $\Sigma_1^b - Ind$ as a computational problem Max.
5. Show that Max cannot be solved by the interactive computation unless $\Sigma_2^p = \Pi_2^p$.

Idea of the proof: Suppose that $T_2^0 = T_2^1$. Then Max can be solved by interactive computation. But this is not possible if $\Sigma_2^p \neq \Pi_2^p$.

Skolemization of T_2^0

- All axioms of BASIC are already universal.
- It remains to Skolemize Σ_0^b induction axioms.

Skolemization of T_2^0

- All axioms of BASIC are already universal.
- It remains to Skolemize Σ_0^b induction axioms.

Write the induction axiom for $\phi(x) \in \Sigma_0^b$ as

$$\forall x(\neg\phi(0) \vee \exists y(\phi(y) \wedge \neg\phi(y+1)) \vee \phi(x))$$

Skolemization of T_2^0

- All axioms of BASIC are already universal.
- It remains to Skolemize Σ_0^b induction axioms.

Write the induction axiom for $\phi(x) \in \Sigma_0^b$ as

$$\forall x(\neg\phi(0) \vee \exists y(\phi(y) \wedge \neg\phi(y+1)) \vee \phi(x))$$

So we need a poly. time function f such that for a given a,

- if $\phi(0) \wedge \neg\phi(a)$,
- then $\phi(f(a)) \wedge \neg\phi(f(a)+1)$.

Skolemization of T_2^0

- All axioms of BASIC are already universal.
- It remains to Skolemize Σ_0^b induction axioms.

Write the induction axiom for $\phi(x) \in \Sigma_0^b$ as

$$\forall x(\neg\phi(0) \vee \exists y(\phi(y) \wedge \neg\phi(y+1)) \vee \phi(x))$$

So we need a poly. time function f such that for a given a,

- if $\phi(0) \wedge \neg\phi(a)$,
- then $\phi(f(a)) \wedge \neg\phi(f(a)+1)$.

Since $\phi(x)$ is decidable in polynomial time, we can compute $f(a)$ using **binary search** in polynomial time.

Herbrand's Theorem for $\forall\exists\forall$ formulas

Recall (we only mentioned $\exists\forall$, but it is easy to generalize it):

Theorem

1. $\forall x \exists y \forall z . \phi(x, y, z)$ *is logically valid, iff*
2. *there exist terms* $t_1, \ldots, t_n$ *such that*

$$\phi(a, t_1(a), b_1) \vee \phi(a, t_2(a, b_1), b_2) \vee \cdots \vee \phi(a, t_n(a, b_1, \ldots, b_{n-1}), b_n)$$

is a propositional tautology.

Herbrand's Theorem for ∀∃∀ formulas

Recall (we only mentioned ∃∀, but it is easy to generalize it):

Theorem

1. $\forall x \exists y \forall z . \phi(x, y, z)$ *is logically valid, iff*
2. *there exist terms* $t_1, \ldots, t_n$ *such that*

$$\phi(a, t_1(a), b_1) \vee \phi(a, t_2(a, b_1), b_2) \vee \cdots \vee \phi(a, t_n(a, b_1, \ldots, b_{n-1}), b_n)$$

is a propositional tautology.

A generalization (Proof – Exercise!):

Theorem

Let T be a universal theory. Then

1. *T proves* $\forall x \exists y \forall z . \phi(x, y, z)$ *iff*
2. *there exist terms* $t_1, \ldots, t_n$ *such that T proves*

$$\phi(a, t_1(a), b_1) \vee \phi(a, t_2(a, b_1), b_2) \vee \cdots \vee \phi(a, t_n(a, b_1, \ldots, b_{n-1}), b_n).$$

Recall *the Teacher-Student Game*

- given a formula $\phi(x, y, z)$ and a number a,
- Teacher asks student to find t such that $\forall y.\phi(a, t, y)$ holds true.
- Student tries t_1, Teacher gives a counterexample b_1; $\neg\phi(a, t_1, b_1)$
- knowing b_1, Student tries t_2, Teacher gives a counterexample b_2, $\neg\phi(a, t_2, b_2)$;
- etc.
- eventually, for some $i \leq n$, there is no counterexample, hence t_i is a solution.

In our case

- the relation $\phi(x, y, z)$ defines a set in $\mathbf{P}$ and terms define polynomial time computable functions,
- so Student is polynomial time computable and Teacher represents an oracle,
- also note that the number of counterexamples is bounded by a constant.

a computational problem

Let $R(x, y)$ be a relation in **P** such that

1. $R(x, 0)$ for all x,
2. $R(x, y) \to y \leq x$ for all x, y,

and let f be a function computable in poly. time.

Problem *Max*:

- given a, find b such that $R(a, b)$ and $f(b)$ is maximal.

a computational problem

Let $R(x, y)$ be a relation in **P** such that

1. $R(x, 0)$ for all x,
2. $R(x, y) \rightarrow y \leq x$ for all x, y,

and let f be a function computable in poly. time.

Problem *Max*:

- given a, find b such that $R(a, b)$ and $f(b)$ is maximal.

Lemma

T_2^1 proves that problem Max always has a solution.

Proof.

The existence of a solution to *Max* is essentially the maximization principle and we (should) know that

$$\Sigma_1^b - IND \ \equiv \ \Pi_0^b - MAX$$

Formulas in $\Pi_0^b = \Sigma_0^b$ define sets in **P**. □

Lemma

If $T_2^0 \equiv T_2^1$, then Max can be solved using the Student-Teacher interactive computation.

Lemma

If $T_2^0 \equiv T_2^1$, then Max can be solved using the Student-Teacher interactive computation.

Proof.

The condition that b is a solution for a is

$$R(a, b) \wedge \forall z(R(a, z) \rightarrow f(z) \leq f(b))$$

The fact that *Max* always has a solution is expressed by

$$\forall x \exists y \forall z(R(x, y) \wedge (R(x, z) \rightarrow f(z) \leq f(y)))$$

which has the form required in the previous lemma. $\square$

how do we get a piece of relevant information?

Student is asked to find b such that

$$R(a, b) \wedge \forall z(R(a, z) \to f(z) \leq f(b))$$

[5] a.k.a. *copycat* strategy

how do we get a piece of relevant information?

Student is asked to find b such that

$$R(a,b) \wedge \forall z (R(a,z) \to f(z) \leq f(b))$$

Consider a stupid strategy[5] for Student:

- Student starts with $b_1 = 0$;
- in round $i+1$, if Teacher gave a counterexample c_i in the previous round, Student answers with $b_{i+1} = c_i$.

[5] a.k.a. *copycat* strategy

how do we get a piece of relevant information?

Student is asked to find b such that

$$R(a, b) \wedge \forall z(R(a, z) \to f(z) \leq f(b))$$

Consider a stupid strategy[5] for Student:

- Student starts with $b_1 = 0$;
- in round $i + 1$, if Teacher gave a counterexample c_i in the previous round, Student answers with $b_{i+1} = c_i$.

If the range of f is not bounded by a constant, Student cannot find a solution in a constant number of rounds. Should he find one, he must do something nontrivial.

[5] a.k.a. *copycat* strategy

a special instance of Max

Define *MaxSatSeq* by

$R(a, b)$ holds true if

1. a is a sequence of Boolean formulas $(a_1, \ldots, a_m)$,
2. b is a sequence of satisfying assignments $(b_1, \ldots, b_{m'})$ for formulas $a_1, \ldots, a_{m'}$, $m' \leq m$;
3. we allow the pair of empty sequences.

$f(b) := m'$.

a special instance of Max

Define *MaxSatSeq* by

$R(a, b)$ holds true if

1. a is a sequence of Boolean formulas $(a_1, \ldots, a_m)$,
2. b is a sequence of satisfying assignments $(b_1, \ldots, b_{m'})$ for formulas $a_1, \ldots, a_{m'}$, $m' \leq m$;
3. we allow the pair of empty sequences.

$f(b) := m'$.

We know that if the number of counterexamples $k < m'$, then the polynomial time computation of Student must produce:

- a satisfying assignment for some formula, from satisfying assignments of $\leq k$ other formulas.

Lemma

Suppose MaxSatSeq can be solved with k counterexamples. Then for every n, there is a set S_n of $\leq k^2 n$ formulas of length n and their satisfying assignments such that a satisfying assignment for any satisfiable formula of length n can be computed in polynomial time from S_n.

Proof.

We know that for every k-tuple of satisfiable formulas $(a_1, \ldots, a_k)$ there exists $1 \leq i \leq k$ such that a satisfying assignment for a_i can be computed from satisfying assignments for a_j, $j < i$.

Proof.

We know that for every k-tuple of satisfiable formulas $(a_1, \ldots, a_k)$ there exists $1 \leq i \leq k$ such that a satisfying assignment for a_i can be computed from satisfying assignments for a_j, $j < i$.

Let N_1 be the number of satisfiable formulas of length n. By a simple counting argument (Exercise), there exists a k-tuple D_1 of formulas and their satisfying assignments from which one can compute satisfying assignments for at least

$$\frac{N_1 - k + 1}{k}$$

formulas of length n.

Proof.

We know that for every k-tuple of satisfiable formulas $(a_1, \ldots, a_k)$ there exists $1 \leq i \leq k$ such that a satisfying assignment for a_i can be computed from satisfying assignments for a_j, $j < i$.

Let N_1 be the number of satisfiable formulas of length n. By a simple counting argument (Exercise), there exists a k-tuple D_1 of formulas and their satisfying assignments from which one can compute satisfying assignments for at least

$$\frac{N_1 - k + 1}{k}$$

formulas of length n. Repeat the argument for the remaining

$$N_2 := \left(1 - \frac{1}{k}\right) N_1 + \frac{k-1}{k}$$

satisfiable formulas to get a k-tuple D_2 and so on.

Proof.

We know that for every k-tuple of satisfiable formulas $(a_1, \ldots, a_k)$ there exists $1 \leq i \leq k$ such that a satisfying assignment for a_i can be computed from satisfying assignments for a_j, $j < i$.

Let N_1 be the number of satisfiable formulas of length n. By a simple counting argument (Exercise), there exists a k-tuple D_1 of formulas and their satisfying assignments from which one can compute satisfying assignments for at least

$$\frac{N_1 - k + 1}{k}$$

formulas of length n. Repeat the argument for the remaining

$$N_2 := \left(1 - \frac{1}{k}\right) N_1 + \frac{k-1}{k}$$

satisfiable formulas to get a k-tuple D_2 and so on. After

$$t \leq \log N_1 / \log(k/(k-1)) \leq n / \log(k/(k-1)) \approx nk$$

steps we have $N_t \leq k$.

Let D_{t+1} be the remaining $\leq k$ formulas and their satisfying assignments. Set

$$S_n := D_1 \cup \cdots \cup D_t \cup D_{t+1}$$

Let D_{t+1} be the remaining $\leq k$ formulas and their satisfying assignments. Set

$$S_n := D_1 \cup \cdots \cup D_t \cup D_{t+1}$$

How do we compute a satisfying assignment of ϕ from S_n?

Let D_{t+1} be the remaining $\leq k$ formulas and their satisfying assignments. Set

$$S_n := D_1 \cup \cdots \cup D_t \cup D_{t+1}$$

How do we compute a satisfying assignment of ϕ from S_n?

Try all D_i and for each of them take the formulas $\psi_1, \ldots, \psi_{k-1}$ from D_i. Try to insert ϕ to all possible positions into this string and play the Student-Teacher. At least for one i and one position of ϕ, Student must produce a satisfying assignment for ϕ.

$\square$

The previous lemma implies:

Lemma

If MaxSatSeq can be solved with a constant number of counterexamples then

$$\mathsf{NP} \subseteq \mathsf{P}/poly$$

Proof.

S_n is the advice and the Student-Teacher game provides a poly-time algorithm. □

The previous lemma implies:

Lemma

If MaxSatSeq can be solved with a constant number of counterexamples then

$$\mathsf{NP} \subseteq \mathsf{P}/poly$$

Proof.

S_n is the advice and the Student-Teacher game provides a poly-time algorithm. □

Theorem (well-known)

$$\mathsf{NP} \subseteq \mathsf{P}/poly \;\Rightarrow\; \Pi_2^p = \Sigma_2^p$$

The previous lemma implies:

Lemma

If MaxSatSeq can be solved with a constant number of counterexamples then

$$\mathsf{NP} \subseteq \mathsf{P}/\mathit{poly}$$

Proof.

S_n is the advice and the Student-Teacher game provides a poly-time algorithm. □

Theorem (well-known)

$$\mathsf{NP} \subseteq \mathsf{P}/\mathit{poly} \ \Rightarrow \ \Pi_2^p = \Sigma_2^p$$

Theorem

$$\Pi_2^p \neq \Sigma_2^p \ \Rightarrow \ \mathsf{NP} \not\subseteq \mathsf{P}/\mathit{poly} \ \Rightarrow \ \mathit{Thm}(T_2^0) \neq \mathit{Thm}(T_2^1)$$

Why can't we use the Gödel Theorem to separate T_2^i from T_2^{i+1}?

By Gödel's theorem we have for all i

$$T_2^i \nvdash Con(T_2^i).$$

Why can't we use the Gödel Theorem to separate T_2^i from T_2^{i+1}?

By Gödel's theorem we have for all i

$$T_2^i \nvdash Con(T_2^i).$$

But in fact, for all i

$$T_2^i \nvdash Con(Q)$$

(Q is Robinson's Arithmetic). Hence $T_2^j \nvdash Con(T_2^i)$ for any i, j.

Why can't we use the Gödel Theorem to separate T_2^i from T_2^{i+1}?

By Gödel's theorem we have for all i

$$T_2^i \nvdash Con(T_2^i).$$

But in fact, for all i

$$T_2^i \nvdash Con(Q)$$

(Q is Robinson's Arithmetic). Hence $T_2^j \nvdash Con(T_2^i)$ for any i, j.

This follows from

1. If T is interpretable in S, then $T \nvdash Con(S)$,
2. every T_2^i is interpretable in Q.

Definition

Let S, T be theories. An interpretation of T in S is a set of S-formulas

- a formula "defining" the universe of T,
- for every relation symbol of T, a formula "defining" the relation in S,
- for every function symbol of T, a formula "defining" the function in S.

"Defining in S" means

- if we translate the axioms of T using these formulas, the translations are provable in S.

Definition

Let S, T be theories. An interpretation of T in S is a set of S-formulas

- a formula "defining" the universe of T,
- for every relation symbol of T, a formula "defining" the relation in S,
- for every function symbol of T, a formula "defining" the function in S.

"Defining in S" means

- if we translate the axioms of T using these formulas, the translations are provable in S.

Proposition

If there is an interpretation of T in S, then

$$S_2^1 \vdash Con(S) \to Con(T)$$

Exercise. Prove the proposition.

interpretation of $I\Delta_0$ in Q (idea)

Let $\phi(x)$ be a Δ_0 formula. We want to interpret induction

$$\phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \forall x \phi(x) \qquad (*)$$

interpretation of $I\Delta_0$ in Q (idea)

Let $\phi(x)$ be a Δ_0 formula. We want to interpret induction

$$\phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \forall x \phi(x) \qquad (*)$$

Define

$$\theta(y) := \ \phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \phi(y)$$

interpretation of $I\Delta_0$ in Q (idea)

Let $\phi(x)$ be a Δ_0 formula. We want to interpret induction

$$\phi(0) \land \forall x(\phi(x) \to \phi(Sx)) \to \forall x \phi(x) \qquad (*)$$

Define

$$\theta(y) := \ \phi(0) \land \forall x(\phi(x) \to \phi(Sx)) \to \phi(y)$$

The universe defined by $\theta(x)$ is closed under S (Exercise):

$$Q \vdash \ \theta(0) \land \theta(x) \to \theta(Sx)$$

interpretation of $I\Delta_0$ in Q (idea)

Let $\phi(x)$ be a Δ_0 formula. We want to interpret induction

$$\phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \forall x \phi(x) \qquad (*)$$

Define

$$\theta(y) := \ \phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \phi(y)$$

The universe defined by $\theta(x)$ is closed under S (Exercise):

$$Q \vdash \ \theta(0) \wedge \theta(x) \to \theta(Sx)$$

To interpret $(*)$ we furthermore need a universe closed under $+$ and $\times$.

interpretation of $I\Delta_0$ in Q (idea)

Let $\phi(x)$ be a Δ_0 formula. We want to interpret induction

$$\phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \forall x \phi(x) \qquad (*)$$

Define

$$\theta(y) := \ \phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \phi(y)$$

The universe defined by $\theta(x)$ is closed under S (Exercise):

$$Q \vdash \ \theta(0) \wedge \theta(x) \to \theta(Sx)$$

To interpret $(*)$ we furthermore need a universe closed under $+$ and $\times$. Define

$$\chi(x) := \ \forall y(\theta(y) \to \theta(x+y))$$

Then the universe defined by $\theta(x)$ is closed under $+$:

$$Q \vdash \ \chi(x) \wedge \chi(y) \to \chi(x+y)$$

interpretation of $I\Delta_0$ in Q (idea)

Let $\phi(x)$ be a Δ_0 formula. We want to interpret induction

$$\phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \forall x \phi(x) \qquad (*)$$

Define

$$\theta(y) := \phi(0) \wedge \forall x(\phi(x) \to \phi(Sx)) \to \phi(y)$$

The universe defined by $\theta(x)$ is closed under S (Exercise):

$$Q \vdash \theta(0) \wedge \theta(x) \to \theta(Sx)$$

To interpret $(*)$ we furthermore need a universe closed under $+$ and $\times$. Define

$$\chi(x) := \forall y(\theta(y) \to \theta(x+y))$$

Then the universe defined by $\theta(x)$ is closed under $+$:

$$Q \vdash \chi(x) \wedge \chi(y) \to \chi(x+y)$$

In a similar way we define a universe τ that is closed also under $\times$. Since $\phi(x)$ is bounded, for an x in τ, $\phi(x)$ holds true iff it holds true with quantifiers restricted to τ. $\square$

Exp is the axiom $\forall x \exists y (y = 2^x)$

(the relation $y = 2^x$ is definable in $I\Delta_0$)

Exp is the axiom $\forall x \exists y (y = 2^x)$
(the relation $y = 2^x$ is definable in $I\Delta_0$)

Theorem
$Con(Q)$ is not provable in $I\Delta_0 + Exp$.

Theorem
$I\Delta_0 + Exp$ is not interpretable in $I\Delta_0$.

Theorem
$I\Delta_0 + Exp + Con(I\Delta_0)$ does not prove $Con(I\Delta_0 + Exp)$.

Thank you!