The Computational Content of Classical Proofs

Extracting programs from classical proofs.

Hans Bugge Grathwohl

Institute for Logic, Language and Computation,
Universiteit van Amsterdam

Cool Logic
April 19th 2013

Outline

Outline

1. Friedman's A-translation
2. Program Extraction

Kreisel's theorem

Theorem (Kreisel (1958))

PA *is a conservative extension of* HA *for* Π^0_2*-sentences.*

Kreisel's theorem

Theorem (Kreisel (1958))

PA *is a conservative extension of* HA *for* Π^0_2*-sentences.*

This means that

$$\vdash_{\mathsf{PA}} \forall x \exists y \,.\, P(x,y) \iff \vdash_{\mathsf{HA}} \forall x \exists y \,.\, P(x,y),$$

where P is a computable predicate.

Corollary

A recursive function is provably total in Peano Arithmetic iff it is provably total in Heyting Arithmetic.

Some preliminaries

We first fix the language.

- $\mathcal{L}$ has logical constants $\bot, \wedge, \vee, \rightarrow, \forall, \exists$, variables $x, y, z, \ldots$, and binary predicate $=$.
- $\neg\varphi$ is an abbreviation of $\varphi \rightarrow \bot$.

Some preliminaries

We first fix the language.

- $\mathcal{L}$ has logical constants $\bot, \wedge, \vee, \rightarrow, \forall, \exists$, variables $x, y, z, \ldots$, and binary predicate $=$.
- $\neg\varphi$ is an abbreviation of $\varphi \rightarrow \bot$.
- Terms and formulas are defined as usual.
- $\vdash_C$ resp. $\vdash_I$ denotes classical resp. intuitionistic derivability in a natural deduction system.

Double-negation translation

Definition (Gödel, Gentzen)

Let φ be a formula. Define the *double-negation translation* φ^- of φ as follows:

$$
\begin{aligned}
\bot^- &:= \bot \\
\alpha^- &:= \neg\neg\alpha, \text{ where } \alpha \neq \bot \text{ is atomic} \\
(\varphi \vee \psi)^- &:= \neg\neg(\varphi^- \vee \psi^-) \\
(\varphi \wedge \psi)^- &:= \varphi^- \wedge \psi^- \\
(\varphi \to \psi)^- &:= \varphi^- \to \psi^- \\
(\forall x.\varphi)^- &:= \forall x.\varphi^- \\
\exists x.\varphi^- &:= \neg\neg\exists x.\varphi^-
\end{aligned}
$$

Double-negation translation

Definition (Gödel, Gentzen)

Let φ be a formula. Define the *double-negation translation* φ^- of φ as follows:

$$
\begin{aligned}
\bot^- &:= \bot \\
\alpha^- &:= \neg\neg\alpha, \text{ where } \alpha \neq \bot \text{ is atomic} \\
(\varphi \lor \psi)^- &:= \neg\neg(\varphi^- \lor \psi^-) \\
(\varphi \land \psi)^- &:= \varphi^- \land \psi^- \\
(\varphi \to \psi)^- &:= \varphi^- \to \psi^- \\
(\forall x.\varphi)^- &:= \forall x.\varphi^- \\
\exists x.\varphi^- &:= \neg\neg\exists x.\varphi^-
\end{aligned}
$$

So φ^- is the result of double-negating all atomic, disjunctive and existential subformulas of φ.

Some properties of the double-negation translation

Lemma

Let φ be a formula, Γ a set of formulas, and $\Gamma^- = \{\psi^- \mid \psi \in \Gamma\}$.

1. $\vdash_C \varphi \leftrightarrow \varphi^-$,
2. $\neg\neg\varphi^- \vdash_I \varphi^-$,
3. *If $\Gamma \vdash_C \varphi$, then $\Gamma^- \vdash_I \varphi^-$ (this justifies calling it a translation),*
4. *In general not $\varphi \vdash_I \varphi^-$.*

Some properties of the double-negation translation

Lemma

Let φ be a formula, Γ a set of formulas, and $\Gamma^- = \{\psi^- \mid \psi \in \Gamma\}$.

1. $\vdash_C \varphi \leftrightarrow \varphi^-$,
2. $\neg\neg\varphi^- \vdash_I \varphi^-$,
3. *If $\Gamma \vdash_C \varphi$, then $\Gamma^- \vdash_I \varphi^-$ (this justifies calling it a translation),*
4. *In general not $\varphi \vdash_I \varphi^-$.*

1, 2 and 3 are not very surprising, and their proofs are easy inductions on the depth of the derivation. 4 is less obvious. A counterexample is $\varphi = \neg\forall x.P(x)$.

Friedman's A-translation

Definition (Friedman)

Let φ and A be formulas such that no bound variable of φ is free in A. We define the *A-translation* φ^A of φ as follows:

$$\begin{aligned}
\bot^A &:= A \\
\alpha^A &:= \alpha \vee A, \text{ where } \alpha \neq \bot \text{ is atomic} \\
(\varphi \wedge \psi)^A &:= \varphi^A \wedge \psi^A \\
(\varphi \vee \psi)^A &:= \varphi^A \vee \psi^A \\
(\varphi \to \psi)^A &:= \varphi^A \to \psi^A \\
(\forall x \varphi)^A &:= \forall x \varphi^A \\
(\exists x \varphi)^A &:= \exists x \varphi^A
\end{aligned}$$

Friedman's A-translation

Definition (Friedman)

Let φ and A be formulas such that no bound variable of φ is free in A. We define the *A-translation* φ^A of φ as follows:

$$
\begin{aligned}
\bot^A &:= A \\
\alpha^A &:= \alpha \vee A, \ \text{where } \alpha \neq \bot \text{ is atomic} \\
(\varphi \wedge \psi)^A &:= \varphi^A \wedge \psi^A \\
(\varphi \vee \psi)^A &:= \varphi^A \vee \psi^A \\
(\varphi \to \psi)^A &:= \varphi^A \to \psi^A \\
(\forall x \varphi)^A &:= \forall x \varphi^A \\
(\exists x \varphi)^A &:= \exists x \varphi^A
\end{aligned}
$$

So φ^A is the result of substituting all atomic subformulas α with $\alpha \vee A$, and replacing any $\bot$ with A.

Friedman's A-translation

Definition (Friedman)

Let φ and A be formulas such that no bound variable of φ is free in A. We define the *A-translation* φ^A of φ as follows:

$$
\begin{aligned}
\bot^A &:= A \\
\alpha^A &:= \alpha \vee A, \ \text{where } \alpha \neq \bot \text{ is atomic} \\
(\varphi \wedge \psi)^A &:= \varphi^A \wedge \psi^A \\
(\varphi \vee \psi)^A &:= \varphi^A \vee \psi^A \\
(\varphi \to \psi)^A &:= \varphi^A \to \psi^A \\
(\forall x \varphi)^A &:= \forall x \varphi^A \\
(\exists x \varphi)^A &:= \exists x \varphi^A
\end{aligned}
$$

So φ^A is the result of substituting all atomic subformulas α with $\alpha \vee A$, and replacing any $\bot$ with A. Note that $(\neg\alpha)^A = \alpha \vee A \to A$.

Some properties of Friedman's A-translation

Lemma

Let φ be formula, Γ a set of formulas and A a formula such that φ^A and Γ^A are defined, where $\Gamma^A = \{\psi^A \mid \psi \in \Gamma\}$.

1. $\vdash_C \varphi^A \leftrightarrow \varphi \vee A$
2. $A \vdash_I \varphi^A$
3. *If* $\Gamma \vdash_I \varphi$*, then* $\Gamma^A \vdash_I \varphi^A$
4. *In general not* $\varphi \vdash_I \varphi^A$

Some properties of Friedman's A-translation

Lemma

Let φ be formula, Γ a set of formulas and A a formula such that φ^A and Γ^A are defined, where $\Gamma^A = \{\psi^A \mid \psi \in \Gamma\}$.

1. $\vdash_C \varphi^A \leftrightarrow \varphi \vee A$
2. $A \vdash_I \varphi^A$
3. *If $\Gamma \vdash_I \varphi$, then $\Gamma^A \vdash_I \varphi^A$*
4. *In general not $\varphi \vdash_I \varphi^A$*

Proof of 1 and 2 are straight-forward inductions on the derivation. A counterexample of 4 is $\varphi := \neg\neg A$.

Sketch of proof of 3: If $\Gamma \vdash_I \varphi$, then $\Gamma^A \vdash_I \varphi^A$

The rules $\wedge_I, \wedge_E, \vee_I, \vee_E, \rightarrow_I, \rightarrow_E$ are straightforward. See for example $\rightarrow_I$:

Sketch of proof of 3: If $\Gamma \vdash_I \varphi$, then $\Gamma^A \vdash_I \varphi^A$

The rules $\wedge_I, \wedge_E, \vee_I, \vee_E, \rightarrow_I, \rightarrow_E$ are straightforward. See for example $\rightarrow_I$:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma, \varphi \vdash \psi\end{array}}{\Gamma \vdash \varphi \rightarrow \psi} \rightarrow_I$$

Sketch of proof of 3: If $\Gamma \vdash_I \varphi$, then $\Gamma^A \vdash_I \varphi^A$

The rules $\wedge_I, \wedge_E, \vee_I, \vee_E, \rightarrow_I, \rightarrow_E$ are straightforward. See for example $\rightarrow_I$:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma, \varphi \vdash \psi\end{array}}{\Gamma \vdash \varphi \rightarrow \psi} \rightarrow_I \quad \mapsto \quad \dfrac{\begin{array}{c}\text{IH}\\ \cdots\cdots\cdots\\ \Gamma^A, \varphi^A \vdash \psi^A\end{array}}{\Gamma^A \vdash \varphi^A \rightarrow \psi^A} \rightarrow_I$$

Sketch of proof of 3: If $\Gamma \vdash_I \varphi$, then $\Gamma^A \vdash_I \varphi^A$

The rules $\wedge_I, \wedge_E, \vee_I, \vee_E, \rightarrow_I, \rightarrow_E$ are straightforward. See for example $\rightarrow_I$:

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma, \varphi \vdash \psi}}{\Gamma \vdash \varphi \rightarrow \psi} \rightarrow_I \quad \mapsto \quad \dfrac{\dfrac{\text{IH}}{\Gamma^A, \varphi^A \vdash \psi^A}}{\Gamma^A \vdash \varphi^A \rightarrow \psi^A} \rightarrow_I$$

$\forall_I, \forall_E, \exists_I, \exists_E$ are a bit trickier because of variable bindings. We consider $\exists_I$:

Sketch of proof of 3: If $\Gamma \vdash_I \varphi$, then $\Gamma^A \vdash_I \varphi^A$

The rules $\wedge_I, \wedge_E, \vee_I, \vee_E, \to_I, \to_E$ are straightforward. See for example $\to_I$:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma, \varphi \vdash \psi\end{array}}{\Gamma \vdash \varphi \to \psi}\to_I \quad \mapsto \quad \dfrac{\begin{array}{c}\text{IH}\\ \cdots\cdots\cdots\\ \Gamma^A, \varphi^A \vdash \psi^A\end{array}}{\Gamma^A \vdash \varphi^A \to \psi^A}\to_I$$

$\forall_I, \forall_E, \exists_I, \exists_E$ are a bit trickier because of variable bindings. We consider $\exists_I$:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \varphi[t/x]\end{array}}{\Gamma \vdash \exists x.\varphi}\exists_I$$

Sketch of proof of 3: If $\Gamma \vdash_I \varphi$, then $\Gamma^A \vdash_I \varphi^A$

The rules $\wedge_I, \wedge_E, \vee_I, \vee_E, \rightarrow_I, \rightarrow_E$ are straightforward. See for example $\rightarrow_I$:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma, \varphi \vdash \psi\end{array}}{\Gamma \vdash \varphi \rightarrow \psi}\rightarrow_I \quad \mapsto \quad \dfrac{\begin{array}{c}\text{IH}\\ \overline{\Gamma^A, \varphi^A \vdash \psi^A}\end{array}}{\Gamma^A \vdash \varphi^A \rightarrow \psi^A}\rightarrow_I$$

$\forall_I, \forall_E, \exists_I, \exists_E$ are a bit trickier because of variable bindings. We consider $\exists_I$:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \varphi[t/x]\end{array}}{\Gamma \vdash \exists x.\varphi}\exists_I \quad \mapsto \quad \dfrac{\begin{array}{c}\text{IH}\\ \overline{\Gamma^A \vdash \varphi^A[t/x]}\end{array}}{\Gamma^A \vdash \exists x.\varphi^A}\exists_I$$

because $(\varphi[t/x])^A = \varphi^A[t/x]$ and $(\exists x.\varphi)^A = \exists x.\varphi^A$.

Sketch of proof of 3: If $\Gamma \vdash_I \varphi$, then $\Gamma^A \vdash_I \varphi^A$

The rules $\wedge_I, \wedge_E, \vee_I, \vee_E, \rightarrow_I, \rightarrow_E$ are straightforward. See for example $\rightarrow_I$:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma, \varphi \vdash \psi\end{array}}{\Gamma \vdash \varphi \rightarrow \psi} \rightarrow_I \quad \mapsto \quad \dfrac{\begin{array}{c}\text{IH}\\ \cdots\cdots\cdots\\ \Gamma^A, \varphi^A \vdash \psi^A\end{array}}{\Gamma^A \vdash \varphi^A \rightarrow \psi^A} \rightarrow_I$$

$\forall_I, \forall_E, \exists_I, \exists_E$ are a bit trickier because of variable bindings. We consider $\exists_I$:

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Gamma \vdash \varphi[t/x]\end{array}}{\Gamma \vdash \exists x.\varphi} \exists_I \quad \mapsto \quad \dfrac{\begin{array}{c}\text{IH}\\ \cdots\cdots\cdots\\ \Gamma^A \vdash \varphi^A[t/x]\end{array}}{\Gamma^A \vdash \exists x.\varphi^A} \exists_I$$

because $(\varphi[t/x])^A = \varphi^A[t/x]$ and $(\exists x.\varphi)^A = \exists x.\varphi^A$.
For $\bot_E$: IH is $\Gamma^A \vdash A$, and 2 gives us $A \vdash \varphi^A$.

Arithmetic

- We add new symbols to the language:
  - nullary constant **0**,
  - unary function symbol **S**,
  - symbols $F, G, H, \ldots$ for all primitive recursive functions.

Arithmetic

- We add new symbols to the language:
  - nullary constant **0**,
  - unary function symbol **S**,
  - symbols $F, G, H, \ldots$ for all primitive recursive functions.
- Peano axioms:

(refl) $x = x$

(trans) $x = y \wedge y = z \to x = z$

(cong_F) $x_i = x_i' \to F(x_1, \ldots, x_i, \ldots, x_n) = F(x_1, \ldots, x_i', \ldots, x_n)$ for any n-ary function constant F

(succ_1) $\mathbf{S}(x) \neq \mathbf{0}$

(succ_2) $\mathbf{S}(x) = \mathbf{S}(y) \to x = y$

(ind) $\varphi(0) \wedge \forall x(\varphi(x) \to \varphi(\mathbf{S}(x))) \to \forall x \varphi(x)$

Arithmetic

- We add new symbols to the language:
  - nullary constant **0**,
  - unary function symbol **S**,
  - symbols $F, G, H, \ldots$ for all primitive recursive functions.
- Peano axioms:

$$
\begin{array}{rl}
(\mathit{refl}) & x = x \\
(\mathit{trans}) & x = y \wedge y = z \rightarrow x = z \\
(\mathit{cong}_F) & x_i = x_i' \rightarrow F(x_1, \ldots, x_i, \ldots, x_n) = F(x_1, \ldots, x_i', \ldots, x_n) \text{ for any } n\text{-ary} \\
 & \text{function constant } F \\
(\mathit{succ}_1) & \mathbf{S}(x) \neq \mathbf{0} \\
(\mathit{succ}_2) & \mathbf{S}(x) = \mathbf{S}(y) \rightarrow x = y \\
(\mathit{ind}) & \varphi(0) \wedge \forall x(\varphi(x) \rightarrow \varphi(\mathbf{S}(x))) \rightarrow \forall x \varphi(x)
\end{array}
$$

$$
\begin{array}{rl}
(\mathit{proj}_F) & F(x_1, \ldots, x_i, \ldots, x_n) = x_i \\
(\mathit{comp}_F) & F(x_1, \ldots, x_n) = G(H_1(x_1, \ldots, x_n), \ldots, H_m(x_1, \ldots, x_n)) \\
(\mathit{rec}_F) & F(\mathbf{0}, x_1, \ldots, x_n) = G(x_1, \ldots, x_n) \\
 & \quad \wedge F(\mathbf{S}(y), x_1, \ldots, x_n) = H(F(y, x_1, \ldots, x_n), y, x_1, \ldots, x_n)
\end{array}
$$

Definition (Peano Arithmetic, Heyting Arithmetic)

Let Γ be a subset of the Peano axioms and φ be a formula.

- $\Gamma \vdash_C \varphi \implies \vdash_{\mathsf{PA}} \varphi$
- $\Gamma \vdash_I \varphi \implies \vdash_{\mathsf{HA}} \varphi$

Arithmetic

Definition (Peano Arithmetic, Heyting Arithmetic)

Let Γ be a subset of the Peano axioms and φ be a formula.

- $\Gamma \vdash_C \varphi \implies \vdash_{\mathsf{PA}} \varphi$
- $\Gamma \vdash_I \varphi \implies \vdash_{\mathsf{HA}} \varphi$

Fact

For any quantifier-free formula $\varphi(x_1, \ldots, x_n)$ there is a primitive recursive function symbol F such that

$$\vdash_{\mathsf{HA}} \varphi(x_1, \ldots, x_n) \leftrightarrow F(x_1, \ldots, x_n) = \mathbf{0}.$$

Axiom Translations

Lemma

Let φ *be a Peano axiom. Then* $\vdash_{\mathsf{HA}} \varphi^-$ *and* $\vdash_{\mathsf{HA}} \varphi^A$*.*

Axiom Translations

Lemma

Let φ be a Peano axiom. Then $\vdash_{\mathsf{HA}} \varphi^-$ and $\vdash_{\mathsf{HA}} \varphi^A$.

Proof.

Axiom Translations

Lemma

Let φ be a Peano axiom. Then $\vdash_{\mathsf{HA}} \varphi^-$ and $\vdash_{\mathsf{HA}} \varphi^A$.

Proof.

If φ is on one of the forms

- α,
- $\alpha \wedge \beta$,
- $\alpha \rightarrow \beta$ or
- $\alpha \wedge \beta \rightarrow \gamma$,

where α, β, γ are atomic, then $\varphi \vdash_I \varphi^-$ and $\varphi \vdash_I \varphi^A$.

Axiom Translations

Lemma

Let φ be a Peano axiom. Then $\vdash_{\mathsf{HA}} \varphi^-$ and $\vdash_{\mathsf{HA}} \varphi^A$.

Proof.

If φ is on one of the forms

- α,
- $\alpha \wedge \beta$,
- $\alpha \to \beta$ or
- $\alpha \wedge \beta \to \gamma$,

where α, β, γ are atomic, then $\varphi \vdash_I \varphi^-$ and $\varphi \vdash_I \varphi^A$.
Luckily, everything, except instances of the induction scheme, is of this form.

Axiom Translations

Lemma

Let φ be a Peano axiom. Then $\vdash_{\mathsf{HA}} \varphi^-$ and $\vdash_{\mathsf{HA}} \varphi^A$.

Proof.

Let φ be an instance of the induction axiom:

$$\varphi = \psi(0) \wedge \forall x(\psi(x) \to \psi(\mathbf{S}(x))) \to \forall x.\psi(x),$$

for some formula $\psi(x)$.

Axiom Translations

Lemma

Let φ be a Peano axiom. Then $\vdash_{\mathsf{HA}} \varphi^-$ and $\vdash_{\mathsf{HA}} \varphi^A$.

Proof.

Let φ be an instance of the induction axiom:

$$\varphi = \psi(0) \wedge \forall x(\psi(x) \to \psi(\mathbf{S}(x))) \to \forall x.\psi(x),$$

for some formula $\psi(x)$. Now:

$$\varphi^- = \psi^-(0) \wedge \forall x(\psi^-(x) \to \psi^-(\mathbf{S}(x))) \to \forall x.\psi^-(x),$$
$$\varphi^A = \psi^A(0) \wedge \forall x(\psi^A(x) \to \psi^A(\mathbf{S}(x))) \to \forall x.\psi^A(x),$$

which are themselves axioms of HA. □

Axiom Translations II

Corollary

1. *If* $\vdash_{\mathsf{PA}} \varphi$, *then* $\vdash_{\mathsf{HA}} \varphi^{-}$,
2. *if* $\vdash_{\mathsf{HA}} \varphi$ *and* φ^{A} *is defined, then* $\vdash_{\mathsf{HA}} \varphi^{A}$.

Axiom Translations II

Corollary

1. *If* $\vdash_{\mathsf{PA}} \varphi$, *then* $\vdash_{\mathsf{HA}} \varphi^{-}$,
2. *if* $\vdash_{\mathsf{HA}} \varphi$ *and* φ^{A} *is defined, then* $\vdash_{\mathsf{HA}} \varphi^{A}$.

Proof.

1. Let Γ be the axioms used in the derivation $\vdash_{\mathsf{PA}} \varphi$.

$$\Gamma \vdash_{C} \varphi \implies \Gamma^{-} \vdash_{I} \varphi^{-} \implies \vdash_{\mathsf{HA}} \varphi^{-}.$$

Axiom Translations II

Corollary

1. *If* $\vdash_{\mathsf{PA}} \varphi$, *then* $\vdash_{\mathsf{HA}} \varphi^{-}$*,*
2. *if* $\vdash_{\mathsf{HA}} \varphi$ *and* φ^{A} *is defined, then* $\vdash_{\mathsf{HA}} \varphi^{A}$*.*

Proof.

1. Let Γ be the axioms used in the derivation $\vdash_{\mathsf{PA}} \varphi$.

$$\Gamma \vdash_C \varphi \implies \Gamma^{-} \vdash_I \varphi^{-} \implies \vdash_{\mathsf{HA}} \varphi^{-}.$$

2. Let Γ be the axioms used in the derivation $\vdash_{\mathsf{HA}} \varphi$.

$$\Gamma \vdash_I \varphi \implies \Gamma^{A} \vdash_I \varphi^{A} \implies \vdash_{\mathsf{HA}} \varphi^{A}.$$

□

Friedman's proof of Kreisel's theorem

Observation

If φ is a Σ^0_1-formula, then $\vdash_I \varphi^A \leftrightarrow \varphi \vee A$.

Friedman's proof of Kreisel's theorem

Observation

If φ is a Σ^0_1-formula, then $\vdash_I \varphi^A \leftrightarrow \varphi \vee A$.

Proof.

- $(\exists y.F(x,y) = \mathbf{0})^A = \exists y.(F(x,y) = \mathbf{0} \vee A)$

Friedman's proof of Kreisel's theorem

Observation

If φ is a Σ^0_1-formula, then $\vdash_I \varphi^A \leftrightarrow \varphi \vee A$.

Proof.

- $(\exists y.F(x,y) = \mathbf{0})^A = \exists y.(F(x,y) = \mathbf{0} \vee A)$
- $\vdash_I \exists x(\varphi \vee \psi) \leftrightarrow \exists x \varphi \vee \psi$ when x not free in ψ

Friedman's proof of Kreisel's theorem

Observation

If φ is a Σ^0_1-formula, then $\vdash_I \varphi^A \leftrightarrow \varphi \vee A$.

Proof.

- $(\exists y.F(x,y) = \mathbf{0})^A = \exists y.(F(x,y) = \mathbf{0} \vee A)$
- $\vdash_I \exists x(\varphi \vee \psi) \leftrightarrow \exists x\varphi \vee \psi$ when x not free in ψ
- Therefore $\vdash_I (\exists y.F(x,y) = \mathbf{0})^A \leftrightarrow \exists y(F(x,y) = \mathbf{0}) \vee A$

□

Friedman's proof of Kreisel's theorem

Proof of Theorem (Friedman).

- To show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Π^0_2-sentence φ.
- It is sufficient to show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Σ^0_1-formula.

Friedman's proof of Kreisel's theorem

Proof of Theorem (Friedman).

- To show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Π^0_2-sentence φ.
- It is sufficient to show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Σ^0_1-formula.
- Let $A := \exists y.F(x,y) = \mathbf{0}$.

Friedman's proof of Kreisel's theorem

Proof of Theorem (Friedman).

- To show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Π^0_2-sentence φ.
- It is sufficient to show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Σ^0_1-formula.
- Let $A := \exists y.F(x, y) = \mathbf{0}$.
- Assume $\vdash_{\mathsf{PA}} A$.

Friedman's proof of Kreisel's theorem

Proof of Theorem (Friedman).

- To show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Π^0_2-sentence φ.
- It is sufficient to show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Σ^0_1-formula.
- Let $A := \exists y. F(x, y) = \mathbf{0}$.
- Assume $\vdash_{\mathsf{PA}} A$.
- Double-negation translation: $\vdash_{\mathsf{HA}} \neg\neg A$.

Friedman's proof of Kreisel's theorem

Proof of Theorem (Friedman).

- To show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Π^0_2-sentence φ.
- It is sufficient to show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Σ^0_1-formula.
- Let $A := \exists y. F(x, y) = \mathbf{0}$.
- Assume $\vdash_{\mathsf{PA}} A$.
- Double-negation translation: $\vdash_{\mathsf{HA}} \neg\neg A$.
- Friedman's A translation: $\vdash_{\mathsf{HA}} (\neg\neg A)^A$.

Friedman's proof of Kreisel's theorem

Proof of Theorem (Friedman).

- To show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Π^0_2-sentence φ.
- It is sufficient to show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Σ^0_1-formula.
- Let $A := \exists y.F(x,y) = \mathbf{0}$.
- Assume $\vdash_{\mathsf{PA}} A$.
- Double-negation translation: $\vdash_{\mathsf{HA}} \neg\neg A$.
- Friedman's A translation: $\vdash_{\mathsf{HA}} (\neg\neg A)^A$.
- $\vdash_{\mathsf{HA}} (\neg\neg A)^A \leftrightarrow (((A \vee A) \to A) \to A)$

Friedman's proof of Kreisel's theorem

Proof of Theorem (Friedman).

- To show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Π^0_2-sentence φ.
- It is sufficient to show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Σ^0_1-formula.
- Let $A := \exists y.F(x,y) = \mathbf{0}$.
- Assume $\vdash_{\mathsf{PA}} A$.
- Double-negation translation: $\vdash_{\mathsf{HA}} \neg\neg A$.
- Friedman's A translation: $\vdash_{\mathsf{HA}} (\neg\neg A)^A$.
- $\vdash_{\mathsf{HA}} (\neg\neg A)^A \leftrightarrow (((A \vee A) \to A) \to A) \leftrightarrow A$.

Friedman's proof of Kreisel's theorem

Proof of Theorem (Friedman).

- To show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Π^0_2-sentence φ.
- It is sufficient to show: $\vdash_{\mathsf{PA}} \varphi \iff \vdash_{\mathsf{HA}} \varphi$ for any Σ^0_1-formula.
- Let $A := \exists y.F(x,y) = \mathbf{0}$.
- Assume $\vdash_{\mathsf{PA}} A$.
- Double-negation translation: $\vdash_{\mathsf{HA}} \neg\neg A$.
- Friedman's A translation: $\vdash_{\mathsf{HA}} (\neg\neg A)^A$.
- $\vdash_{\mathsf{HA}} (\neg\neg A)^A \leftrightarrow (((A \vee A) \to A) \to A) \leftrightarrow A$.
- $\vdash_{\mathsf{HA}} A$.

□

Outline

1. Friedman's A-translation

2. Program Extraction

Program Extraction I

- Rice's Theorem: It is in general undecidable whether a program meets some specification.
- Proofs can easily be checked.
- From a constructive proof, we can extract a correct program.

Program Extraction I

- Rice's Theorem: It is in general undecidable whether a program meets some specification.
- Proofs can easily be checked.
- From a constructive proof, we can extract a correct program.

Program Extraction

$$\vdash t \;:\; \forall x^A\, \exists y^B \,.\, P(x,y)$$

$$\varepsilon(t) : A \to B \qquad\qquad \vdash \texttt{corr} \;:\; \forall x^A \,.\, P(x, (\varepsilon(t))(x))$$

Program Extraction II

Example

- We want a sorting function $\texttt{sort} : \texttt{list}(\mathsf{N}) \to \texttt{list}(\mathsf{N})$.

Program Extraction II

Example

- We want a sorting function $\texttt{sort} : \texttt{list}(\mathsf{N}) \to \texttt{list}(\mathsf{N})$.
- $\vdash t \ : \ \forall x : \texttt{list}(\mathsf{N}) \exists y : \texttt{list}(\mathsf{N}) \, . \, \texttt{perm}(x, y) \wedge \texttt{sorted}(x, y)$

Program Extraction II

Example

- We want a sorting function $\texttt{sort} : \texttt{list}(\mathsf{N}) \to \texttt{list}(\mathsf{N})$.
- $\vdash t \;:\; \forall x : \texttt{list}(\mathsf{N}) \exists y : \texttt{list}(\mathsf{N}) \,.\, \texttt{perm}(x, y) \wedge \texttt{sorted}(x, y)$
- $\texttt{sort} = \varepsilon(t) : \texttt{list}(\mathsf{N}) \to \texttt{list}(\mathsf{N})$

Program Extraction II

Example

- We want a sorting function $\texttt{sort} : \texttt{list}(\mathsf{N}) \to \texttt{list}(\mathsf{N})$.
- $\vdash t \ : \ \forall x : \texttt{list}(\mathsf{N}) \exists y : \texttt{list}(\mathsf{N}) \,.\, \texttt{perm}(x, y) \wedge \texttt{sorted}(x, y)$
- $\texttt{sort} = \varepsilon(t) : \texttt{list}(\mathsf{N}) \to \texttt{list}(\mathsf{N})$
- $\vdash u \ : \ \forall x : \texttt{list}(\mathsf{N}) \,.\, \texttt{perm}(x, \texttt{sort}(x)) \wedge \texttt{sorted}(x, \texttt{sort}(x))$

Program Extraction II

Example

- We want a sorting function $\texttt{sort} : \texttt{list}(\mathsf{N}) \to \texttt{list}(\mathsf{N})$.
- $\vdash t \ : \ \forall x : \texttt{list}(\mathsf{N}) \exists y : \texttt{list}(\mathsf{N}) \,.\, \texttt{perm}(x, y) \wedge \texttt{sorted}(x, y)$
- $\texttt{sort} = \varepsilon(t) : \texttt{list}(\mathsf{N}) \to \texttt{list}(\mathsf{N})$
- $\vdash u \ : \ \forall x : \texttt{list}(\mathsf{N}) \,.\, \texttt{perm}(x, \texttt{sort}(x)) \wedge \texttt{sorted}(x, \texttt{sort}(x))$

A perfect computer program: It does exactly what we want, and it is provably bug-free.

Extraction from Classical Proofs I

- Using translations:

$$\vdash_{\mathsf{PA}} t \ : \ \forall x \exists y \, P(x, y)$$

Extraction from Classical Proofs I

- Using translations:

$$\vdash_{\mathsf{PA}} t \ : \ \forall x \exists y \, P(x,y) \xrightarrow[\ ]{\text{Double-negation translation, } A\text{-translation}} \vdash_{\mathsf{HA}} t' \ : \ \forall x \exists y \, P(x,y)$$

Extraction from Classical Proofs I

- Using translations:

Double-negation translation, A-translation:

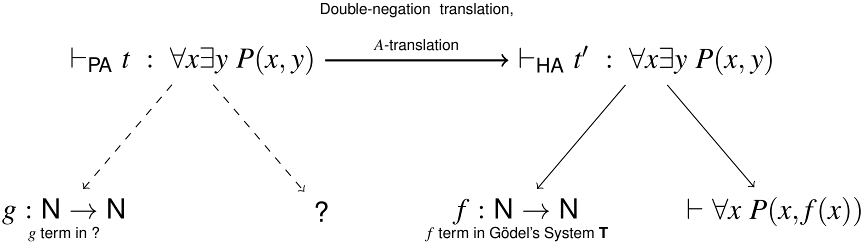
$$\vdash_{\mathsf{PA}} t \;:\; \forall x \exists y\, P(x,y) \longrightarrow \vdash_{\mathsf{HA}} t' \;:\; \forall x \exists y\, P(x,y)$$

$$f : \mathsf{N} \to \mathsf{N} \qquad\qquad \vdash \forall x\, P(x, f(x))$$

f term in Gödel's System **T**

Extraction from Classical Proofs I

- Using translations:



Extraction from Classical Proofs II

- Intuitionistic proofs:
  - Extracts *pure functional* programs.

Extraction from Classical Proofs II

- Intuitionistic proofs:
  - Extracts *pure functional* programs.
- Classical proofs:
  - Needs a more expressive programming language.
  - Griffin (1990): Classical reasoning corresponds to *control operators*.
  - Control operators allow for more flexibility; it compares to adding labels and jumps, `return` or exception handling.

Extraction from Classical Proofs II

- Intuitionistic proofs:
  - Extracts *pure functional* programs.
- Classical proofs:
  - Needs a more expressive programming language.
  - Griffin (1990): Classical reasoning corresponds to *control operators*.
  - Control operators allow for more flexibility; it compares to adding labels and jumps, `return` or exception handling.
- Underlying algorithms in classical proofs are potentially more efficient than ones from intuitionistic proofs.

Programs with control operators

- A traditional functional program $\mathtt{mult} : \mathtt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$\mathtt{mult}[5, 7, 0, 2] \mapsto$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$\texttt{mult}[5,7,0,2] \mapsto 5 \cdot (\texttt{mult}[7,0,2])$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$\begin{aligned}\texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2]))\end{aligned}$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$\begin{aligned} \texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2])) \\ &\mapsto 5 \cdot (7 \cdot 0) \end{aligned}$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$\begin{aligned} \texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2])) \\ &\mapsto 5 \cdot (7 \cdot 0) \end{aligned}$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$\begin{aligned}\texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2])) \\ &\mapsto 5 \cdot (7 \cdot 0) \\ &\mapsto 5 \cdot 0\end{aligned}$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathbb{N}) \to \mathbb{N}$ would have a computation similar to this:

$$\begin{aligned}\texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2])) \\ &\mapsto 5 \cdot (7 \cdot 0) \\ &\mapsto 5 \cdot 0 \\ &\mapsto 0\end{aligned}$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathrm{N}) \to \mathrm{N}$ would have a computation similar to this:

$$\begin{aligned} \texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2])) \\ &\mapsto 5 \cdot (7 \cdot 0) \\ &\mapsto 5 \cdot 0 \\ &\mapsto 0 \end{aligned}$$

- Alternatively, when using control operators, we can make the program behave more like the following:

$$\texttt{mult}'[5,7,0,2] \mapsto$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$\begin{aligned}
\texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\
&\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2])) \\
&\mapsto 5 \cdot (7 \cdot 0) \\
&\mapsto 5 \cdot 0 \\
&\mapsto 0
\end{aligned}$$

- Alternatively, when using control operators, we can make the program behave more like the following:

$$\texttt{mult}'[5,7,0,2] \mapsto 5 \cdot (\texttt{mult}'[7,0,2])$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$\begin{aligned}\texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2])) \\ &\mapsto 5 \cdot (7 \cdot 0) \\ &\mapsto 5 \cdot 0 \\ &\mapsto 0\end{aligned}$$

- Alternatively, when using control operators, we can make the program behave more like the following:

$$\begin{aligned}\texttt{mult}'[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}'[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot ((\texttt{mult}'[0,2])))\end{aligned}$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$\begin{aligned} \texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2])) \\ &\mapsto 5 \cdot (7 \cdot 0) \\ &\mapsto 5 \cdot 0 \\ &\mapsto 0 \end{aligned}$$

- Alternatively, when using control operators, we can make the program behave more like the following:

$$\begin{aligned} \texttt{mult}'[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}'[7,0,2]) \\ &\mapsto 5 \cdot (7 \cdot ((\texttt{mult}'[\color{red}{0}, 2]))) \end{aligned}$$

Programs with control operators

- A traditional functional program $\texttt{mult} : \texttt{list}(\mathsf{N}) \to \mathsf{N}$ would have a computation similar to this:

$$
\begin{aligned}
\texttt{mult}[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}[7,0,2]) \\
&\mapsto 5 \cdot (7 \cdot (\texttt{mult}[0,2])) \\
&\mapsto 5 \cdot (7 \cdot 0) \\
&\mapsto 5 \cdot 0 \\
&\mapsto 0
\end{aligned}
$$

- Alternatively, when using control operators, we can make the program behave more like the following:

$$
\begin{aligned}
\texttt{mult}'[5,7,0,2] &\mapsto 5 \cdot (\texttt{mult}'[7,0,2]) \\
&\mapsto 5 \cdot (7 \cdot ((\texttt{mult}'[0,2]))) \\
&\mapsto {\color{red}0}
\end{aligned}
$$

Extraction from Classical Proofs III

- Double negation translation $\leftrightarrow$ CPS-translation
  - CPS: Continuation Passing Style
  - CPS style function: The control appears explicitly in the form of a *continuation* that is passed to the function.

Extraction from Classical Proofs III

- Double negation translation $\quad \leftrightarrow \quad$ CPS-translation
  - CPS: Continuation Passing Style
  - CPS style function: The control appears explicitly in the form of a *continuation* that is passed to the function.
- Instead, we want to extract to a system that has control as a primitive construct.
- One approach is to interpret classical logics in a control calculus via a Curry-Howard correspondence (proofs-as-terms).
  - This requires a lot of fiddling around with reduction strategies. And program extraction tend to not necessarily be correct.

Extraction from Classical Proofs III

- Double negation translation $\leftrightarrow$ CPS-translation
  - CPS: Continuation Passing Style
  - CPS style function: The control appears explicitly in the form of a *continuation* that is passed to the function.
- Instead, we want to extract to a system that has control as a primitive construct.
- One approach is to interpret classical logics in a control calculus via a Curry-Howard correspondence (proofs-as-terms).
  - This requires a lot of fiddling around with reduction strategies. And program extraction tend to not necessarily be correct.
- Another approach is realisability.
  - Realisability can be seen as a formalisation of the BHK-interpretation: A realiser of an existential formula gives a witness for the formula, and a realiser of a disjunction tells which side of the disjunction is provable.

EM_1: Alwayz into somethin'

- Which fragment of classical logic should we consider?
  - EM_1: Excluded middle restricted to Σ^0_1-formulas.
  - Markov's Principle: $\neg\neg\exists x P(x) \rightarrow \exists x P(x)$

EM_1: Alwayz into somethin'

- Which fragment of classical logic should we consider?
  - EM_1: Excluded middle restricted to Σ^0_1-formulas.
  - Markov's Principle: $\neg\neg\exists x P(x) \rightarrow \exists x P(x)$
- A natural place to start seems to be HA $+$ EM_1
  - HA $+$ EM_1 proves a lot of theorems (Akama, Berardi, Hayashi, Kohlenbach 2004)

EM_1: Alwayz into somethin'

- Which fragment of classical logic should we consider?
  - EM_1: Excluded middle restricted to Σ^0_1-formulas.
  - Markov's Principle: $\neg\neg\exists x P(x) \rightarrow \exists x P(x)$
- A natural place to start seems to be $\mathsf{HA} + \mathsf{EM}_1$
  - $\mathsf{HA} + \mathsf{EM}_1$ proves a lot of theorems (Akama, Berardi, Hayashi, Kohlenbach 2004)
- Traditional realisability cannot be used for $\mathsf{HA} + \mathsf{EM}_1$:
- $\mathsf{HA} + \mathsf{EM}_1 \vdash \forall x \forall y (\exists z Txyz \vee \forall z \neg Txyz)$, where T is Kleene's predicate.
- A (traditional) realiser of this would solve the Halting Problem.

Learning-Based Realisability

Aschieri's Interactive Learning-Based Realisability is based on the idea of learning by counterexamples.

- Knowledge states S.
- At any state s, we have a truth value of all instances $\exists y P(x,y) \vee \forall y \neg P(x,y)$ of EM_1, and in case of $\exists y P(x,y)$ being "true", also a witness m.

Learning-Based Realisability

Aschieri's Interactive Learning-Based Realisability is based on the idea of learning by counterexamples.

- Knowledge states S.
- At any state s, we have a truth value of all instances $\exists y P(x,y) \vee \forall y \neg P(x,y)$ of EM_1, and in case of $\exists y P(x,y)$ being "true", also a witness m.
- The realiser learns:
  - At stage s: It believes $\forall x \neg P(x)$
  - It turns out that $P(n)$ for some n.
  - We backtrack the computation, update to stage s'.
  - At stage s': It believes $\exists x P(x)$, and has witness n.

Learning-Based Realisability

Aschieri's Interactive Learning-Based Realisability is based on the idea of learning by counterexamples.

- Knowledge states S.
- At any state s, we have a truth value of all instances $\exists y P(x,y) \vee \forall y \neg P(x,y)$ of EM_1, and in case of $\exists y P(x,y)$ being "true", also a witness m.
- The realiser learns:
  - At stage s: It believes $\forall x \neg P(x)$
  - It turns out that $P(n)$ for some n.
  - We backtrack the computation, update to stage s'.
  - At stage s': It believes $\exists x P(x)$, and has witness n.
- Since a proof is finite, we only need a finite piece of information about EM_1.
- A learning-based realiser is a self-correcting program.

Learning-Based Realisability

Aschieri's Interactive Learning-Based Realisability is based on the idea of learning by counterexamples.

- Knowledge states S.
- At any state s, we have a truth value of all instances $\exists y P(x,y) \vee \forall y \neg P(x,y)$ of EM_1, and in case of $\exists y P(x,y)$ being "true", also a witness m.
- The realiser learns:
  - At stage s: It believes $\forall x \neg P(x)$
  - It turns out that $P(n)$ for some n.
  - We backtrack the computation, update to stage s'.
  - At stage s': It believes $\exists x P(x)$, and has witness n.
- Since a proof is finite, we only need a finite piece of information about EM_1.
- A learning-based realiser is a self-correcting program.

I will investigate whether we from $HA + EM_1$-proofs of Π^0_2-sentences can extract programs *that uses control.*

Thank you!

Counterexample to 4: In general *not* $\varphi \vdash_I \varphi^-$.

Consider a Kripke model with ω many nodes $k_0 \leq k_1 \leq k_2 \leq \ldots$, with the following domains and valuations.

i	0	1	2	$\ldots$
$D(k_i)$	$\{0\}$	$\{0,1\}$	$\{0,1,2\}$	$\ldots$
P	$\{\}$	$\{0\}$	$\{0,1\}$	$\ldots$

Counterexample to 4: In general *not* $\varphi \vdash_I \varphi^-$.

Consider a Kripke model with ω many nodes $k_0 \leq k_1 \leq k_2 \leq \ldots$, with the following domains and valuations.

i	0	1	2	$\ldots$
$D(k_i)$	$\{0\}$	$\{0,1\}$	$\{0,1,2\}$	$\ldots$
P	$\{\}$	$\{0\}$	$\{0,1\}$	$\ldots$

Clearly $k_n \not\Vdash \forall x.P(x)$ for all n, so especially $k_0 \Vdash \neg\forall x P(x)$.

Counterexample to 4: In general *not* $\varphi \vdash_I \varphi^-$.

Consider a Kripke model with ω many nodes $k_0 \leq k_1 \leq k_2 \leq \ldots$, with the following domains and valuations.

i	0	1	2	$\ldots$
$D(k_i)$	$\{0\}$	$\{0,1\}$	$\{0,1,2\}$	$\ldots$
P	$\{\}$	$\{0\}$	$\{0,1\}$	$\ldots$

Clearly $k_n \nVdash \forall x.P(x)$ for all n, so especially $k_0 \Vdash \neg\forall x P(x)$. Let n be given, and take any $l \leq n$. Then $k_{n+1} \Vdash P(l)$. Therefore $k_n \Vdash \neg\neg P(l)$.

Counterexample to 4: In general *not* $\varphi \vdash_I \varphi^-$.

Consider a Kripke model with ω many nodes $k_0 \leq k_1 \leq k_2 \leq \dots$, with the following domains and valuations.

i	0	1	2	$\dots$
$D(k_i)$	$\{0\}$	$\{0,1\}$	$\{0,1,2\}$	$\dots$
P	$\{\}$	$\{0\}$	$\{0,1\}$	$\dots$

Clearly $k_n \not\Vdash \forall x.P(x)$ for all n, so especially $k_0 \Vdash \neg\forall x P(x)$. Let n be given, and take any $l \leq n$. Then $k_{n+1} \Vdash P(l)$. Therefore $k_n \Vdash \neg\neg P(l)$. Hence $k_0 \Vdash \forall x.\neg\neg P(x)$.

Counterexample to 4: In general *not* $\varphi \vdash_I \varphi^-$.

Consider a Kripke model with ω many nodes $k_0 \leq k_1 \leq k_2 \leq \dots$, with the following domains and valuations.

i	0	1	2	...
$D(k_i)$	$\{0\}$	$\{0,1\}$	$\{0,1,2\}$	...
P	$\{\}$	$\{0\}$	$\{0,1\}$	...

Clearly $k_n \nVdash \forall x.P(x)$ for all n, so especially $k_0 \Vdash \neg\forall x P(x)$. Let n be given, and take any $l \leq n$. Then $k_{n+1} \Vdash P(l)$. Therefore $k_n \Vdash \neg\neg P(l)$. Hence $k_0 \Vdash \forall x.\neg\neg P(x)$.

This proves that we cannot have $\neg\forall x.P(x) \vdash_I \neg\forall x.\neg\neg P(x)$.