Comparing methods for program extraction from classical proofs

Trifon Trifonov
(joint work with Diana Ratiu)

Ludwig Maximilian Universität, München

Logic Colloquium 2009
Sofia, 31.07.2009

Negative Arithmetic (NA^ω)

We consider the negative fragment of Heyting Arithmetic.

$$A, B \quad ::= \quad P(\vec{t}) \mid \mathrm{at}(b^{\mathbb{B}}) \mid A \to B \mid A \wedge B \mid \forall_x A \mid \exists_x A$$

We obtain HA^ω by adding the strong existential $\exists$.

Negative Arithmetic (NA^ω)

We consider the negative fragment of Heyting Arithmetic.

$$A, B ::= P(\vec{t}) \mid \mathrm{at}(b^{\mathbb{B}}) \mid A \to B \mid A \wedge B \mid \forall_x A \mid \exists_x A$$
$$\neg A ::= A \to \bot$$

We obtain HA^ω by adding the strong existential $\exists$.

Negative Arithmetic (NA^ω)

We consider the negative fragment of Heyting Arithmetic.

$$\begin{array}{rcl} A, B & ::= & P(\vec{t}) \mid \mathrm{at}(b^{\mathbb{B}}) \mid A \to B \mid A \wedge B \mid \forall_x A \mid \exists_x A \\ \neg A & ::= & A \to \bot \\ \tilde{\exists}_x A & ::= & \neg \forall_x \neg A \end{array}$$

We obtain HA^ω by adding the strong existential $\exists$.

Heyting Arithmetic (HA^ω)

We consider the negative fragment of Heyting Arithmetic.

$$\begin{array}{rcl} A, B & ::= & P(\vec{t}) \mid \mathrm{at}(b^{\mathbb{B}}) \mid A \to B \mid A \wedge B \mid \forall_x A \mid \exists_x A \\ \neg A & ::= & A \to \bot \\ \tilde{\exists}_x A & ::= & \neg \forall_x \neg A \end{array}$$

We obtain HA^ω by adding the strong existential $\exists$.

Weak and strong existence

- $\exists_x A$
  - To prove: show t and prove $A(t)$
- $\tilde{\exists}_x A$
  - To prove: assume $u : \forall_x (A \to \bot)$ and show $\bot$

Weak and strong existence

- $\exists_x A$
  - To prove: show t and prove $A(t)$
- $\tilde{\exists}_x A$
  - To prove: assume $u : \forall_x(A \to \bot)$ and show $\bot$

Weak and strong existence

- $\exists_x A$
  - To prove: show t and prove $A(t)$
- $\tilde{\exists}_x A$
  - To prove: assume $u : \forall_x (A \to \bot)$ and show $\bot$

Weak and strong existence

- $\exists_x A$
  - To prove: show t and prove $A(t)$
- $\tilde{\exists}_x A$
  - To prove: assume $u : \forall_x (A \to \bot)$ and show $\bot$

Weak and strong existence

- $\exists_x A$
  - To prove: show t and prove $A(t)$
- $\tilde{\exists}_x A$
  - To prove: assume $u : \forall_x (A \to \bot)$ and show $\bot$

Weak existence proofs contain implicit computational content.
Simple idea: look which term t is used with the assumption u.

Weak and strong existence

- $\exists_x A$
  - To prove: show t and prove $A(t)$
- $\tilde{\exists}_x A$
  - To prove: assume $u : \forall_x (A \to \bot)$ and show $\bot$

Weak existence proofs contain implicit computational content.
Simple idea: look which term t is used with the assumption u.
But: u can be used many times with different terms!

Weak and strong existence

- $\exists_x A$
  - To prove: show t and prove $A(t)$
- $\tilde{\exists}_x A$
  - To prove: assume $u : \forall_x (A \to \bot)$ and show $\bot$

Weak existence proofs contain implicit computational content.
Simple idea: look which term t is used with the assumption u.
But: u can be used many times with different terms!
Idea: Try to keep track of *all* terms used for u.

Boolean falsity

Using a general predicate variable $\bot$ we work in a minimal logic setting. We denote the system as HA_0^ω.

However, if we use *decidable falsity* $\mathrm{F} := \mathrm{at}(\mathrm{ff})$, we are able to prove by induction on the definition of formulas

Lemma (ex falso quodlibet)

$\mathrm{HA}^\omega \vdash \mathrm{F} \to A$

Lemma (stability)

$\mathrm{NA}^\omega \vdash ((A \to \mathrm{F}) \to \mathrm{F}) \to A$

if A contains no predicate variables.

Boolean falsity

Using a general predicate variable $\bot$ we work in a minimal logic setting. We denote the system as HA_0^ω.
However, if we use *decidable falsity* $\mathrm{F} := \mathrm{at}(\mathrm{ff})$, we are able to prove by induction on the definition of formulas

Lemma (ex falso quodlibet)

$\mathrm{HA}^\omega \vdash \mathrm{F} \to A$

Lemma (stability)

$\mathrm{NA}^\omega \vdash ((A \to \mathrm{F}) \to \mathrm{F}) \to A$

if A contains no predicate variables.

Boolean falsity

Using a general predicate variable $\bot$ we work in a minimal logic setting. We denote the system as HA_0^ω.
However, if we use *decidable falsity* $\mathrm{F} := \mathrm{at}(\mathrm{ff})$, we are able to prove by induction on the definition of formulas

Lemma (ex falso quodlibet)

$\mathrm{HA}^\omega \vdash \mathrm{F} \to A$

Lemma (stability)

$\mathrm{NA}^\omega \vdash ((A \to \mathrm{F}) \to \mathrm{F}) \to A$

if A contains no predicate variables.

Boolean falsity

Using a general predicate variable $\bot$ we work in a minimal logic setting. We denote the system as HA_0^ω.
However, if we use *decidable falsity* $\mathrm{F} := \mathrm{at}(\mathrm{ff})$, we are able to prove by induction on the definition of formulas

Lemma (ex falso quodlibet)

$\mathrm{HA}^\omega \vdash \mathrm{F} \to A$

Lemma (stability)

$\mathrm{NA}^\omega \vdash ((A \to \mathrm{F}) \to \mathrm{F}) \to A$

if A contains no predicate variables.

Boolean falsity

Using a general predicate variable $\perp$ we work in a minimal logic setting. We denote the system as HA_0^ω.
However, if we use *decidable falsity* $\mathrm{F} := \mathrm{at}(\mathrm{ff})$, we are able to prove by induction on the definition of formulas

Lemma (ex falso quodlibet)

$\mathrm{HA}^\omega \vdash \mathrm{F} \to A$

Lemma (stability)

$\mathrm{NA}^\omega \vdash ((A \to \mathrm{F}) \to \mathrm{F}) \to A$

if A contains no predicate variables.

A-translation

Idea: use $\bot$ to extract computational content of proofs in NA^ω.

Theorem (Extraction via *A*-translation)

Let M be a proof of

$$\mathrm{HA}_0^\omega \vdash D \to \tilde{\exists}_{y^\rho} G$$

with D, G not containing $\bot$. Then

$$\mathrm{HA}^\omega \vdash D \to \exists_y G$$

Idea.

Let $M' := M\left[\bot := \exists_y G\right]$. A witness for y is $[\![M']\!](\lambda_y y)$. □

A-translation

Idea: use $\bot$ to extract computational content of proofs in NA^ω.

Theorem (Extraction via A-translation)

Let M be a proof of

$$\mathrm{HA}_0^\omega \vdash D \to \tilde{\exists}_{y^\rho} G$$

with D, G not containing $\bot$. Then

$$\mathrm{HA}^\omega \vdash D \to \exists_y G$$

Idea.
Let $M' := M\left[\bot := \exists_y G\right]$. A witness for y is $[\![M']\!](\lambda_y y)$. □

A-translation

Idea: use $\bot$ to extract computational content of proofs in NA^ω.

Theorem (Extraction via A-translation)

Let M be a proof of

$$\mathrm{HA}_0^\omega \vdash D \to \forall_{y^\rho}(G \to \bot) \to \bot$$

with D, G not containing $\bot$. Then

$$\mathrm{HA}^\omega \vdash D \to \exists_y G$$

Idea.

Let $M' := M\left[\bot := \exists_y G\right]$. A witness for y is $[\![M']\!](\lambda_y y)$. □

A-translation

Idea: use $\bot$ to extract computational content of proofs in NA^ω.

Theorem (Extraction via A-translation)

Let M be a proof of

$$\mathrm{HA}_0^\omega \vdash D \to \forall_{y^\rho}(G \to \bot) \to \bot$$

with D, G not containing $\bot$. Then

$$\mathrm{HA}^\omega \vdash D \to \exists_y G$$

Idea.
Let $M' := M[\bot := \exists_y G]$. A witness for y is $[\![M']\!](\lambda_y y)$. □

A-translation

Idea: use $\bot$ to extract computational content of proofs in NA^ω.

Theorem (Extraction via A-translation)

Let M be a proof of

$$\mathrm{HA}_0^\omega \vdash D \to \forall_{y^\rho}(G \to \bot) \to \bot$$

with D, G not containing $\bot$. Then

$$\mathrm{HA}^\omega \vdash D \to \exists_y G$$

Idea.
Let $M' := M\left[\bot := \exists_y G\right]$. A witness for y is $[\![M']\!](\lambda_y y)$. □

A-translation

Idea: use $\bot$ to extract computational content of proofs in NA^ω.

Theorem (Extraction via A-translation)

Let M be a proof of

$$\mathrm{HA}_0^\omega \vdash D \to \forall_{y^\rho}(G \to \bot) \to \bot$$

with D, G not containing $\bot$. Then

$$\mathrm{HA}^\omega \vdash D \to \exists_y G$$

Idea.

Let $M' := M\left[\bot := \exists_y G\right]$. A witness for y is $[\![M']\!](\lambda_y y)$. □

Definite and goal formulas

What if $\bot$ appears in D or G?

Bucholz, Berger, Schwichtenberg (2000), Seisenberger (2008):

$$
\begin{array}{rcll}
D & ::= & P \mid G \to D & (\text{if } \tau(D) = \varepsilon \text{ then } \tau(G) = \varepsilon) \\
 & & \mid D_1 \wedge D_2 & (\text{if } \tau(D_1) \neq \varepsilon \text{ then } \tau(D_2) = \varepsilon) \\
 & & \mid \forall_x D &
\end{array}
$$

$$
\begin{array}{rcll}
G & ::= & P \mid D \to G & (\text{if } \tau(G) \neq \varepsilon \text{ and } \tau(D) = \varepsilon \text{ then } D \text{ decidable}) \\
 & & \mid G_1 \wedge G_2 & \\
 & & \mid \forall_x G & (\text{if } \tau(G) = \varepsilon)
\end{array}
$$

Definite and goal formulas

What if $\bot$ appears in D or G?
Bucholz, Berger, Schwichtenberg (2000), Seisenberger (2008):

$$
\begin{array}{lll}
D & ::= & P \mid G \to D \quad (\text{if } \tau(D) = \varepsilon \text{ then } \tau(G) = \varepsilon) \\
 & & \mid D_1 \wedge D_2 \quad (\text{if } \tau(D_1) \neq \varepsilon \text{ then } \tau(D_2) = \varepsilon) \\
 & & \mid \forall_x D
\end{array}
$$

$$
\begin{array}{lll}
G & ::= & P \mid D \to G \quad (\text{if } \tau(G) \neq \varepsilon \text{ and } \tau(D) = \varepsilon \text{ then } D \text{ decidable}) \\
 & & \mid G_1 \wedge G_2 \\
 & & \mid \forall_x G \qquad (\text{if } \tau(G) = \varepsilon)
\end{array}
$$

Dialectica interpretation

Let us have a proof of B from the assumption A.

- In case A is true, we have a function producing a witness for B from a witness for A
- In case B is false, we have a counterexample for A depending on a counterexample for B

Dialectica interpretation

Let us have a proof of B from the assumption A.

- In case A is true, we have a function producing a witness for B from a witness for A
- In case B is false, we have a counterexample for A depending on a counterexample for B

Dialectica interpretation

Let us have a proof of B from the assumption A.

- In case A is true, we have a function producing a witness for B from a witness for A
- In case B is false, we have a counterexample for A depending on a counterexample for B

Contractions in Dialectica

- When A was used more than once, we have a counterexample for each separate use
- Still we need to choose only one of them
- We need to be able to *decide* which instance of the assumption A was false
- Other approaches — finite set of solutions (Diller-Nahm, 1974), monotone Dialectica (Kohlenbach, 1993)

Contractions in Dialectica

- When *A* was used more than once, we have a counterexample for each separate use
- Still we need to choose only one of them
- We need to be able to *decide* which instance of the assumption *A* was false
- Other approaches — finite set of solutions (Diller-Nahm, 1974), monotone Dialectica (Kohlenbach, 1993)

Contractions in Dialectica

- When *A* was used more than once, we have a counterexample for each separate use
- Still we need to choose only one of them
- We need to be able to *decide* which instance of the assumption *A* was false
- Other approaches — finite set of solutions (Diller-Nahm, 1974), monotone Dialectica (Kohlenbach, 1993)

Contractions in Dialectica

- When A was used more than once, we have a counterexample for each separate use
- Still we need to choose only one of them
- We need to be able to *decide* which instance of the assumption A was false
- Other approaches — finite set of solutions (Diller-Nahm, 1974), monotone Dialectica (Kohlenbach, 1993)

The Infinite Pigeon Hole Principle

Theorem (Infinite Pigeon Hole (IPH) Principle)

Any infinite sequence coloured with finitely many colours has an infinite monochromatic subsequence.

Formalisation:

$$\forall_r \forall_f \left(\forall_n (f_n < r) \to \tilde{\exists}_q \forall_n \tilde{\exists}_m (m \geq n \wedge f_m = q) \right)$$

Proof of IPH

$$\forall_r \forall_f \big(\forall_k (f_k < r) \to \tilde{\exists}_q \forall_n \tilde{\exists}_m (m \geq n \wedge f_m = q)\big)$$

Proof.

Induction on r.

- When $r = 0$ we have a false premise.
- Assume the claim for r, and take f with $r + 1$ colours.
- A case distinction on "the colour r appears infinitely often":
  - If yes, then we have found a monochromatic subsequence
  - If not, we take the subsequence after the last appearance of the colour r and apply the induction hypothesis

□

Proof of IPH

$$\forall_r \forall_f \big(\forall_k (f_k < r) \to \tilde{\exists}_q \forall_n \tilde{\exists}_m (m \geq n \wedge f_m = q)\big)$$

Proof.
Induction on r.

- When $r = 0$ we have a false premise.
- Assume the claim for r, and take f with $r + 1$ colours.
- A case distinction on "the colour r appears infinitely often":
  - If yes, then we have found a monochromatic subsequence
  - If not, we take the subsequence after the last appearance of the colour r and apply the induction hypothesis

□

Proof of IPH

$$\forall_r \forall_f \big(\forall_k (f_k < r) \to \tilde{\exists}_q \forall_n \tilde{\exists}_m (m \geq n \wedge f_m = q)\big)$$

Proof.

Induction on r.

- When $r = 0$ we have a false premise.
- Assume the claim for r, and take f with $r + 1$ colours.
- A case distinction on “the colour r appears infinitely often”:
  - If yes, then we have found a monochromatic subsequence
  - If not, we take the subsequence after the last appearance of the colour r and apply the induction hypothesis

□

Proof of IPH

$$\forall_r \forall_f \big(\forall_k (f_k < r) \to \tilde{\exists}_q \forall_n \tilde{\exists}_m (m \geq n \wedge f_m = q)\big)$$

Proof.

Induction on r.

- When $r = 0$ we have a false premise.
- Assume the claim for r, and take f with $r + 1$ colours.
- A case distinction on “the colour r appears infinitely often”:
  - If yes, then we have found a monochromatic subsequence
  - If not, we take the subsequence after the last appearance of the colour r and apply the induction hypothesis

□

Proof of IPH

$$\forall_r \forall_f \big(\forall_k (f_k < r) \to \tilde{\exists}_q \forall_n \tilde{\exists}_m (m \geq n \wedge f_m = q)\big)$$

Proof.

Induction on r.

- When $r = 0$ we have a false premise.
- Assume the claim for r, and take f with $r + 1$ colours.
- A case distinction on “the colour r appears infinitely often”:
  - If yes, then we have found a monochromatic subsequence
  - If not, we take the subsequence after the last appearance of the colour r and apply the induction hypothesis

□

IPH is non-constructive

$$\forall_r \forall_f \left(\forall_k (f_k < r) \to \tilde{\exists}_q \forall_n \tilde{\exists}_m (m \geq n \wedge f_m = q)\right)$$

Thus, we cannot have a program

- taking r and f as inputs
- and providing an *infinite* subsequence f_m of colour q

But: we can have a program

- taking r, f and a number n as inputs
- and providing a *finite* subsequence of length n and colour q

It should reflect the finitary computational meaning of IPH.

IPH is non-constructive

$$\forall_r \forall_f \big(\forall_k (f_k < r) \to \tilde{\exists}_q \forall_n \tilde{\exists}_m (m \geq n \wedge f_m = q)\big)$$

Thus, we cannot have a program

- taking r and f as inputs
- and providing an *infinite* subsequence f_m of colour q

But: we can have a program

- taking r, f and a number n as inputs
- and providing a *finite* subsequence of length n and colour q

It should reflect the finitary computational meaning of IPH.

A finitary corollary of IPH

Corollary (Unbounded Pigeon Hole Principle)

Any infinite sequence coloured with finitely many colours has a finite monochromatic subsequence of any given length.

Proof.

Induction on n, using IPH to provide the next element in the subsequence. □

A constructive proof exists, but explicit construction is needed!

A finitary corollary of IPH

Corollary (Unbounded Pigeon Hole Principle)

Any infinite sequence coloured with finitely many colours has a finite monochromatic subsequence of any given length.

Proof.

Induction on n, using IPH to provide the next element in the subsequence. □

A constructive proof exists, but explicit construction is needed!

A finitary corollary of IPH

Corollary (Unbounded Pigeon Hole Principle)

Any infinite sequence coloured with finitely many colours has a finite monochromatic subsequence of any given length.

Proof.

Induction on n, using IPH to provide the next element in the subsequence. □

A constructive proof exists, but explicit construction is needed!

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[]
b	[]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

a b a c b b c b a a c . . .

Color	List
c	[]
b	[]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[]
b	[]
a	[0]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \dots$

Color	List
c	[]
b	[]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[]
b	[1]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[]
b	[1]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[]
b	[1]
a	[2]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

a b a c b b c b a a c . . .

Color	List
c	[]
b	[1]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c\ldots$

Color	List
c	[]
b	[]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[3]
b	[]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[3]
b	[]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$$

Color	List
c	[3]
b	[]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[3]
b	[4]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[3]
b	[4, 5]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	[3]
b	[]
a	[]

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \dots$

Color	List
c	$[3, 6]$
b	$[]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c\ldots$

Color	List
c	$[3, 6]$
b	$[]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$$

Color	List
c	$[3, 6]$
b	$[]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6]$
b	$[]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6]$
b	$[7]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6]$
b	$[7]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6]$
b	$[7]$
a	$[8]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3,6]$
b	$[7]$
a	$[8,9]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6]$
b	$[7]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6]$
b	$[]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6, 10]$
b	$[]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6, 10]$
b	$[]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6, 10]$
b	$[]$
a	$[]$

- When a higher colour occurs, lists of lower colours are reset
- The program returns the smallest possible indices of the same colour
- between which no higher colour occurs

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6, 10]$
b	$[]$
a	$[]$

- Worst time complexity is $O(n^r)$
- However, average time complexity is $O(n \cdot r)$
- which is the same as the complexity of a naïve algorithm

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6, 10]$
b	$[]$
a	$[]$

- Worst time complexity is $O(n^r)$
- However, average time complexity is $O(n \cdot r)$
- which is the same as the complexity of a naïve algorithm

A-translation: Example run

$a\ b\ a\ c\ b\ b\ c\ b\ a\ a\ c \ldots$

Color	List
c	$[3, 6, 10]$
b	$[]$
a	$[]$

- Worst time complexity is $O(n^r)$
- However, average time complexity is $O(n \cdot r)$
- which is the same as the complexity of a naïve algorithm

A-translation: Specific features

- IPH corresponds to an abstract backtracking scheme
- The type of the final result is determined by the corollary
- Extracted programs follow continuation-passing style
- Computed witnesses are immediately passed to continuations
- Case distinctions on decidable definite formulas determine:
  - Should we accept the witness (identity)
  - or should we backtrack (call an alternative continuation)

A-translation: Specific features

- IPH corresponds to an abstract backtracking scheme
- The type of the final result is determined by the corollary
- Extracted programs follow continuation-passing style
- Computed witnesses are immediately passed to continuations
- Case distinctions on decidable definite formulas determine:
  - Should we accept the witness (identity)
  - or should we backtrack (call an alternative continuation)

A-translation: Specific features

- IPH corresponds to an abstract backtracking scheme
- The type of the final result is determined by the corollary
- Extracted programs follow continuation-passing style
- Computed witnesses are immediately passed to continuations
- Case distinctions on decidable definite formulas determine:
  - Should we accept the witness (identity)
  - or should we backtrack (call an alternative continuation)

A-translation: Specific features

- IPH corresponds to an abstract backtracking scheme
- The type of the final result is determined by the corollary
- Extracted programs follow continuation-passing style
- Computed witnesses are immediately passed to continuations
- Case distinctions on decidable definite formulas determine:
  - Should we accept the witness (identity)
  - or should we backtrack (call an alternative continuation)

A-translation: Specific features

- IPH corresponds to an abstract backtracking scheme
- The type of the final result is determined by the corollary
- Extracted programs follow continuation-passing style
- Computed witnesses are immediately passed to continuations
- Case distinctions on decidable definite formulas determine:
  - Should we accept the witness (identity)
  - or should we backtrack (call an alternative continuation)

A-translation: Specific features

- IPH corresponds to an abstract backtracking scheme
- The type of the final result is determined by the corollary
- Extracted programs follow continuation-passing style
- Computed witnesses are immediately passed to continuations
- Case distinctions on decidable definite formulas determine:
  - Should we accept the witness (identity)
  - or should we backtrack (call an alternative continuation)

A-translation: Specific features

- IPH corresponds to an abstract backtracking scheme
- The type of the final result is determined by the corollary
- Extracted programs follow continuation-passing style
- Computed witnesses are immediately passed to continuations
- Case distinctions on decidable definite formulas determine:
  - Should we accept the witness (identity)
  - or should we backtrack (call an alternative continuation)

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[]
a	[]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[]
a	[]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[]
a	[]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[]
a	[0]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[]
a	[0, 1]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c\ldots$

Color	List
c	[]
b	[]
a	[0, 1, 2]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[]
a	[0, 1, 2]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[1]
a	[]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[1]
a	[2]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[1]
a	[2, 3]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	$[]$
b	$[1]$
a	$[2, 3, 4]$

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	$[]$
b	$[1, 4]$
a	$[]$

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	$[]$
b	$[1, 4]$
a	$[5]$

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[1, 4]
a	[5, 6]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[1, 4]
a	[5, 6, 7]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[]
b	[1, 4, 7]
a	[]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	$[]$
b	$[1, 4, 7]$
a	$[]$

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[]
a	[]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[]
a	[]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$$

Color	List
c	[4]
b	[]
a	[]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[]
a	[5]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[]
a	[5, 6]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \dots$

Color	List
c	[4]
b	[]
a	[5, 6, 7]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[7]
a	[]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[7]
a	[8]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	$[4]$
b	$[7]$
a	$[8, 9]$

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[7]
a	[8, 9, 10]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	$[4]$
b	$[7, 10]$
a	$[]$

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[7, 10]
a	[11]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[7, 10]
a	[11, 12]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	[4]
b	[7, 10]
a	[11, 12, 13]

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	$[4]$
b	$[7, 10, 13]$
a	$[]$

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	$[4, 13]$
b	$[]$
a	$[]$

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Example run

$a\ b\ a\ b\ c\ b\ c\ b\ a\ a\ c\ b\ a\ c \ldots$

Color	List
c	$[4, 13]$
b	$[]$
a	$[]$

- For each colour we store the *last* failure index
- and use it as a candidate witness for the higher colour
- Both worst and average time complexity are $O(n^r)$

Dialectica: Specific features

- IPH corresponds to a concrete backtracking scheme
- Program for IPH expects a “challenging” function
- Programs return
  - Candidate for a witness
  - Candidate for a counterexample
- Backtracking is controlled by checking counterexamples:
  - If the counterexample is valid, the witness is not correct — backtrack
  - If the counterexample is not valid, return the witness

Dialectica: Specific features

- IPH corresponds to a concrete backtracking scheme
- Program for IPH expects a “challenging” function
- Programs return
  - Candidate for a witness
  - Candidate for a counterexample
- Backtracking is controlled by checking counterexamples:
  - If the counterexample is valid, the witness is not correct — backtrack
  - If the counterexample is not valid, return the witness

Dialectica: Specific features

- IPH corresponds to a concrete backtracking scheme
- Program for IPH expects a “challenging” function
- Programs return
  - Candidate for a witness
  - Candidate for a counterexample
- Backtracking is controlled by checking counterexamples:
  - If the counterexample is valid, the witness is not correct — backtrack
  - If the counterexample is not valid, return the witness

Dialectica: Specific features

- IPH corresponds to a concrete backtracking scheme
- Program for IPH expects a “challenging” function
- Programs return
  - Candidate for a witness
  - Candidate for a counterexample
- Backtracking is controlled by checking counterexamples:
  - If the counterexample is valid, the witness is not correct — backtrack
  - If the counterexample is not valid, return the witness

Dialectica: Specific features

- IPH corresponds to a concrete backtracking scheme
- Program for IPH expects a “challenging” function
- Programs return
  - Candidate for a witness
  - Candidate for a counterexample
- Backtracking is controlled by checking counterexamples:
  - If the counterexample is valid, the witness is not correct — backtrack
  - If the counterexample is not valid, return the witness

Dialectica: Specific features

- IPH corresponds to a concrete backtracking scheme
- Program for IPH expects a “challenging” function
- Programs return
  - Candidate for a witness
  - Candidate for a counterexample
- Backtracking is controlled by checking counterexamples:
  - If the counterexample is valid, the witness is not correct — backtrack
  - If the counterexample is not valid, return the witness

Dialectica: Specific features

- IPH corresponds to a concrete backtracking scheme
- Program for IPH expects a “challenging” function
- Programs return
  - Candidate for a witness
  - Candidate for a counterexample
- Backtracking is controlled by checking counterexamples:
  - If the counterexample is valid, the witness is not correct — backtrack
  - If the counterexample is not valid, return the witness

Dialectica: Specific features

- IPH corresponds to a concrete backtracking scheme
- Program for IPH expects a “challenging” function
- Programs return
  - Candidate for a witness
  - Candidate for a counterexample
- Backtracking is controlled by checking counterexamples:
  - If the counterexample is valid, the witness is not correct — backtrack
  - If the counterexample is not valid, return the witness

Is optimisation possible?

- The complexity is high, because we wait for the *last* failure index
- What if we changed the program to find the *first* failure index instead?
- Returned subsequences will be the same as with the *A*-translation program!
- But time complexity is still $O(n^r)$
- Even though we return the first failure index, we recheck its validity on every step
- To obtain faster programs we need to optimise the extraction method internally

Is optimisation possible?

- The complexity is high, because we wait for the *last* failure index
- What if we changed the program to find the *first* failure index instead?
- Returned subsequences will be the same as with the A-translation program!
- But time complexity is still $O(n^r)$
- Even though we return the first failure index, we recheck its validity on every step
- To obtain faster programs we need to optimise the extraction method internally

Is optimisation possible?

- The complexity is high, because we wait for the *last* failure index
- What if we changed the program to find the *first* failure index instead?
- Returned subsequences will be the same as with the *A*-translation program!
- But time complexity is still $O(n^r)$
- Even though we return the first failure index, we recheck its validity on every step
- To obtain faster programs we need to optimise the extraction method internally

Is optimisation possible?

- The complexity is high, because we wait for the *last* failure index
- What if we changed the program to find the *first* failure index instead?
- Returned subsequences will be the same as with the *A*-translation program!
- But time complexity is still $O(n^r)$
- Even though we return the first failure index, we recheck its validity on every step
- To obtain faster programs we need to optimise the extraction method internally

Is optimisation possible?

- The complexity is high, because we wait for the *last* failure index
- What if we changed the program to find the *first* failure index instead?
- Returned subsequences will be the same as with the *A*-translation program!
- But time complexity is still $O(n^r)$
- Even though we return the first failure index, we recheck its validity on every step
- To obtain faster programs we need to optimise the extraction method internally

Is optimisation possible?

- The complexity is high, because we wait for the *last* failure index
- What if we changed the program to find the *first* failure index instead?
- Returned subsequences will be the same as with the *A*-translation program!
- But time complexity is still $O(n^r)$
- Even though we return the first failure index, we recheck its validity on every step
- To obtain faster programs we need to optimise the extraction method internally

Conclusion

- Programs from classical proofs are backtracking schemes
- A-translation extracts an abstract backtracking scheme
- Dialectica extracts a concrete backtracking scheme
- Methods control the backtracking process in specific ways
- Dialectica needs optimisation to match A-translation
- Extract from Ramsey's theorem

Conclusion

- Programs from classical proofs are backtracking schemes
- A-translation extracts an abstract backtracking scheme
- Dialectica extracts a concrete backtracking scheme
- Methods control the backtracking process in specific ways
- Dialectica needs optimisation to match A-translation
- Extract from Ramsey's theorem

Conclusion

- Programs from classical proofs are backtracking schemes
- *A*-translation extracts an abstract backtracking scheme
- Dialectica extracts a concrete backtracking scheme
- Methods control the backtracking process in specific ways
- Dialectica needs optimisation to match *A*-translation
- Extract from Ramsey's theorem

Conclusion

- Programs from classical proofs are backtracking schemes
- A-translation extracts an abstract backtracking scheme
- Dialectica extracts a concrete backtracking scheme
- Methods control the backtracking process in specific ways
- Dialectica needs optimisation to match A-translation
- Extract from Ramsey's theorem

Conclusion

- Programs from classical proofs are backtracking schemes
- *A*-translation extracts an abstract backtracking scheme
- Dialectica extracts a concrete backtracking scheme
- Methods control the backtracking process in specific ways
- Dialectica needs optimisation to match *A*-translation
- Extract from Ramsey's theorem

Conclusion

- Programs from classical proofs are backtracking schemes
- A-translation extracts an abstract backtracking scheme
- Dialectica extracts a concrete backtracking scheme
- Methods control the backtracking process in specific ways
- Dialectica needs optimisation to match A-translation
- Extract from Ramsey's theorem

Thank you

Thank you for your attention!