Synthesizing nested relational queries from implicit specifications

Cécilia Pradic

j.w.w. Michael Benedikt (Oxford University) and Christoph Wernhard (TU Dresden)

November 10th 2022, Swansea theory seminar

Extracting programs from specifications

The general setting

Input: a *logical* input-output specification $\varphi(i, o)$

Output: a *program* `f` obeying the specification

Extracting programs from specifications

The general setting

Input: a *logical* input-output specification $\varphi(i, o)$
Output: a *program* `f` obeying the specification

Reasonable to constrain specifications:

Extracting programs from specifications

The general setting

Input: a *logical* input-output specification $\varphi(i, o)$

Output: a *program* `f` obeying the specification

Reasonable to constrain specifications:

- **Totality**: at least one output per input $\forall i. \exists o. \varphi(i, o)$
- **Functionality**: at most one output per input $\forall i. \forall o. \forall o'. \varphi(i, o) \wedge \varphi(i, o') \Rightarrow o = o'$

Extracting programs from specifications and proofs

The general setting

Input: a *logical* input-output specification $\varphi(i, o)$ (+ some additional certificate)

Output: a *program* `f` obeying the specification

Reasonable to constrain specifications:

- **Totality**: at least one output per input $\forall i . \exists o . \varphi(i, o)$
- **Functionality**: at most one output per input

$$\forall i . \forall o . \forall o' . \varphi(i, o) \wedge \varphi(i, o') \Rightarrow o = o'$$

Extracting programs from specifications and proofs

The general setting

Input: a *logical* input-output specification $\varphi(i, o)$ (+ some additional certificate)
Output: a *program* `f` obeying the specification

Certificates can help produce programs `f`:

Automatic synthesis: no additional data

Extracting programs from specifications and proofs

The general setting

Input: a *logical* input-output specification $\varphi(i, o)$ (+ some additional certificate)

Output: a *program* `f` obeying the specification

Certificates can help produce programs `f`:

Automatic synthesis: no additional data

Curry-Howard approaches: a proof of totality of φ

- ▶ produce a refinement `f` s.t. $\forall i.\ \varphi(i, \mathtt{f}(i))$

Extracting programs from specifications and proofs

The general setting

Input: a *logical* input-output specification $\varphi(i, o)$ (+ some additional certificate)
Output: a *program* $\texttt{f}$ obeying the specification

Certificates can help produce programs $\texttt{f}$:

Automatic synthesis: no additional data

Curry-Howard approaches: a proof of totality of φ

▶ produce a refinement $\texttt{f}$ s.t. $\forall i.\ \varphi(i, \texttt{f}(i))$

Implicit definitions: a proof of functionality of φ

▶ produce an extension $\texttt{f}$ s.t. $\forall i. \forall o.\ \varphi(i, o) \Rightarrow \texttt{f}(i) = o$

Extracting programs from specifications and proofs

The general setting

Input: a *logical* input-output specification $\varphi(i, o)$ (+ some additional certificate)

Output: a *program* $\mathtt{f}$ obeying the specification

Certificates can help produce programs $\mathtt{f}$:

Automatic synthesis: no additional data

Curry-Howard approaches: a proof of totality of φ

- produce a refinement $\mathtt{f}$ s.t. $\forall i.\ \varphi(i, \mathtt{f}(i))$

Implicit definitions: a proof of functionality of φ

- produce an extension $\mathtt{f}$ s.t. $\forall i. \forall o.\ \varphi(i, o) \Rightarrow \mathtt{f}(i) = o$

Goal

Go from **implicit definitions** to **transformations of nested sets**

Plan of the talk

Goal

Go from **implicit definitions** to **transformations of nested sets**

- Nested relations, set-theoretic specifications and the language (NRC)
  - Our work generalizes Beth's theorem for usual ``flat'' relational queries

Plan of the talk

Goal

Go from **implicit definitions** to **transformations of nested sets**

- Nested relations, set-theoretic specifications and the language (NRC)
  - Our work generalizes Beth's theorem for usual ``flat'' relational queries
- Main result: implicit → explicit NRC definitions
  - One easy method for intuitionistic definitions
  - One harder method for classical definitions
  - Linear-time on *cut-free focused* proofs

Plan of the talk

Goal

Go from **implicit definitions** to **transformations of nested sets**

- Nested relations, set-theoretic specifications and the language (NRC)
  - Our work generalizes Beth's theorem for usual ``flat'' relational queries
- Main result: implicit → explicit NRC definitions
  - One easy method for intuitionistic definitions
  - One harder method for classical definitions
  - Linear-time on *cut-free focused* proofs
- WIP: generalization to effective rigid categoricity

The nested relational model, logic and NRC

The nested relational model

We work with **typed objects**

Types for nested collections

$$T, U ::= \mathfrak{U} \mid \mathsf{Set}(T) \mid 1 \mid T \times U$$

Anonymous base type $\mathfrak{U}$

Semantics $T \mapsto [\![T]\!]$ determined inductively by $[\![\mathfrak{U}]\!]$:

- $\mathsf{Set}(T)$: sets of elements of type T
- finite cartesian products $- \times \ldots \times -$

Examples

Taking $[\![\mathfrak{U}]\!] = \texttt{string}$, we have

$$\begin{array}{lcl}
\{(\texttt{``snake''}, \texttt{``slange''}), (\texttt{``pencil''}, \texttt{``blyant''}), \ldots\} & \in & [\![\mathsf{Set}(\mathfrak{U} \times \mathfrak{U})]\!] \\
\{(\{\texttt{``snake''}, \texttt{``serpent''}\}, \{\texttt{``slange''}, \texttt{``snog''}\}), \ldots\} & \in & [\![\mathsf{Set}(\mathsf{Set}(\mathfrak{U}) \times \mathsf{Set}(\mathfrak{U}))]\!] \\
((), \emptyset, \texttt{``snake''}, \{\texttt{``slange''}, \texttt{``snog''}\}) & \in & [\![1 \times \mathsf{Set}(\mathsf{Set}(1)) \times \mathfrak{U} \times \mathsf{Set}(\mathfrak{U})]\!]
\end{array}$$

The nested relational model

We work with **typed objects**

Types for nested collections

$$T, U ::= \mathfrak{U} \mid \mathsf{Set}(T) \mid 1 \mid T \times U$$

Anonymous base type $\mathfrak{U}$

Semantics $T \mapsto [\![T]\!]$ determined inductively by $[\![\mathfrak{U}]\!]$:

- $\mathsf{Set}(T)$: sets of elements of type T
- finite cartesian products $- \times \ldots \times -$

Examples

Taking $[\![\mathfrak{U}]\!] = \texttt{string}$, we have

$$\begin{array}{lcl}
\{(\texttt{``snake"}, \texttt{``slange"}), (\texttt{``pencil"}, \texttt{``blyant"}), \ldots\} & \in & [\![\mathsf{Set}(\mathfrak{U} \times \mathfrak{U})]\!] \\
\{(\{\texttt{``snake"}, \texttt{``serpent"}\}, \{\texttt{``slange"}, \texttt{``snog"}\}), \ldots\} & \in & [\![\mathsf{Set}(\mathsf{Set}(\mathfrak{U}) \times \mathsf{Set}(\mathfrak{U}))]\!] \\
((), \emptyset, \texttt{``snake"}, \{\texttt{``slange"}, \texttt{``snog"}\}) & \in & [\![1 \times \mathsf{Set}(\mathsf{Set}(1)) \times \mathfrak{U} \times \mathsf{Set}(\mathfrak{U})]\!]
\end{array}$$

Usual relational model: only tuples of relations (sets of tuples)

Nested collection transformations

Types for nested collections

$$T, U ::= \mathfrak{U} \mid \mathsf{Set}(T) \mid 1 \mid T \times U$$

A transformation of nested sets is a function $T \to U$

→ is **not** part of the type system

A transformation of flat relations

Pre-image of a relation R

$$\begin{array}{rccl} \mathsf{fib} : & \mathsf{Set}(\mathfrak{U}) \times \mathsf{Set}(\mathfrak{U} \times \mathfrak{U}) & \to & \mathsf{Set}(\mathfrak{U}) \\ & (A, R) & \mapsto & R^{-1}(A) = \{x \mid \exists y \in A.(x, y) \in R\} \end{array}$$

A transformation of nested collections

Collect all pre-images of individual elements

$$\begin{array}{rccl} \mathsf{fibs} : & \mathsf{Set}(\mathfrak{U} \times \mathfrak{U}) & \to & \mathsf{Set}(\mathfrak{U} \times \mathsf{Set}(\mathfrak{U})) \\ & R & \mapsto & \{(a, \mathsf{fib}(\{a\}, R)) \mid a \in \mathsf{cod}(R)\} \end{array}$$

Logical specfications

Queries can be specified in *multi-sorted* first-order logic:

- variables explicitly typed $x : T$
- basic predicates $x \in_T z$ and $x =_T y$ $\qquad x, y : T$ and $z : \mathsf{Set}(T)$
- terms for tupling and projections $\qquad$ e.g., $\pi_1(((x, z), ()), x)) : (T \times \mathsf{Set}(T)) \times 1$

Logical specfications

Queries can be specified in *multi-sorted* first-order logic:

- variables explicitly typed $x : T$
- basic predicates $x \in_T z$ and $x =_T y$ — $x, y : T$ and $z : \mathsf{Set}(T)$
- terms for tupling and projections — e.g., $\pi_1(((x, z), ()), x)) : (T \times \mathsf{Set}(T)) \times 1$

Consider formulas with only bounded quantifications

Δ_0 formulas

$$\varphi, \psi \quad ::= \quad t =_T u \mid t \in_T u \mid \exists x \in t\ \varphi \mid \forall x \in t\ \varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg \varphi$$

Logical specfications

Queries can be specified in *multi-sorted* first-order logic:

- variables explicitly typed $x : T$
- basic predicates $x \in_T z$ and $x =_T y$ $\qquad x, y : T$ and $z : \mathsf{Set}(T)$
- terms for tupling and projections $\qquad$ e.g., $\pi_1(((x, z), ()), x)) : (T \times \mathsf{Set}(T)) \times 1$

Consider formulas with only bounded quantifications

Δ_0 formulas

$$\varphi, \psi \quad ::= \quad t =_T u \mid t \in_T u \mid \exists x \in t\ \varphi \mid \forall x \in t\ \varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg\varphi$$

Example of **functional** and **total** specifications:

$\varphi_{\mathsf{fib}}(A, R, X)$ for $X = R^{-1}(A)$

- Every $x \in X$ is related to some $a \in A$ $\qquad \forall x \in X. \exists a \in A.\ (x, a) \in R$
- For every $(x, y) \in R$, if $y \in A$, then $x \in X$ $\qquad \forall p \in R.\ \pi_2(p) \in A \Rightarrow \pi_1(p) \in X$

Logical specfications

Queries can be specified in *multi-sorted* first-order logic:

- variables explicitly typed $x : T$
- basic predicates $x \in_T z$ and $x =_T y$ — $x, y : T$ and $z : \mathsf{Set}(T)$
- terms for tupling and projections — e.g., $\pi_1(((x, z), ()), x)) : (T \times \mathsf{Set}(T)) \times 1$

Consider formulas with only bounded quantifications

Δ_0 formulas

$$\varphi, \psi \quad ::= \quad t =_T u \mid t \in_T u \mid \exists x \in t\ \varphi \mid \forall x \in t\ \varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \neg \varphi$$

Example of **functional** and **total** specifications:

$\varphi_{\mathsf{fibs}}(R, O)$ for $O = \{(a, R^{-1}(\{a\})) \mid a \in \mathsf{cod}(R)\}$

- For every $(x, a) \in R$, there is some $(a, X) \in O$ s.t. $x \in X$

  $\forall p \in R. \exists q \in O.\ \pi_1(p) \in \pi_2(O)$

- Every element of $(a, X) \in O$ satisfies $\varphi_{\mathsf{fib}}(\{a\}, R, X)$

  $\forall q \in O.\ (\forall x \in \pi_2(q).(x, \pi_1(q)) \in R) \wedge (\forall p \in R.\ \pi_2(p) = \pi_1(q) \Rightarrow \pi_1(p) \in \pi_2(q))$

Our target for synthesis: the nested relational calculus (NRC)

Our programming language for nested transformations $\Gamma \to T$

$$\frac{}{\Gamma,\ x : T,\ \Gamma' \vdash x : T}$$

$$\frac{}{\Gamma \vdash () : 1} \qquad \frac{\Gamma \vdash e_1 : T_1 \qquad \Gamma \vdash e_2 : T_2}{\Gamma \vdash \langle e_1, e_2 \rangle : T_1 \times T_2} \qquad \frac{\Gamma \vdash e : T_1 \times T_2 \qquad i \in \{1,2\}}{\Gamma \vdash \pi_i(e) : T_i}$$

$$\frac{\Gamma \vdash e : T}{\Gamma \vdash \{e\} : \mathsf{Set}(T)} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Set}(T_1) \qquad \Gamma,\ x : T_1 \vdash e_2 : \mathsf{Set}(T_2)}{\Gamma \vdash \bigcup \{e_2 \mid x \in e_1\} : \mathsf{Set}(T_2)}$$

$$\frac{}{\Gamma \vdash \emptyset_T : \mathsf{Set}(T)} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Set}(T) \qquad \Gamma \vdash e_2 : \mathsf{Set}(T)}{\Gamma \vdash e_1 \cup e_2 : \mathsf{Set}(T)} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Set}(T) \qquad \Gamma \vdash e_2 : \mathsf{Set}(T)}{\Gamma \vdash e_1 \setminus e_2 : \mathsf{Set}(T)}$$

Our target for synthesis: the nested relational calculus (NRC) + Get

Our programming language for nested transformations $\Gamma \to T$

$$\frac{}{\Gamma,\ x : T,\ \Gamma' \vdash x : T}$$

$$\frac{}{\Gamma \vdash () : 1} \qquad \frac{\Gamma \vdash e_1 : T_1 \qquad \Gamma \vdash e_2 : T_2}{\Gamma \vdash \langle e_1, e_2 \rangle : T_1 \times T_2} \qquad \frac{\Gamma \vdash e : T_1 \times T_2 \qquad i \in \{1, 2\}}{\Gamma \vdash \pi_i(e) : T_i}$$

$$\frac{\Gamma \vdash e : T}{\Gamma \vdash \{e\} : \mathsf{Set}(T)} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Set}(T_1) \qquad \Gamma,\ x : T_1 \vdash e_2 : \mathsf{Set}(T_2)}{\Gamma \vdash \bigcup \{e_2 \mid x \in e_1\} : \mathsf{Set}(T_2)}$$

$$\frac{}{\Gamma \vdash \emptyset_T : \mathsf{Set}(T)} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Set}(T) \qquad \Gamma \vdash e_2 : \mathsf{Set}(T)}{\Gamma \vdash e_1 \cup e_2 : \mathsf{Set}(T)} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Set}(T) \qquad \Gamma \vdash e_2 : \mathsf{Set}(T)}{\Gamma \vdash e_1 \setminus e_2 : \mathsf{Set}(T)}$$

$$\frac{\Gamma \vdash e : \mathsf{Set}(T)}{\Gamma \vdash \mathsf{Get}(e) : T}$$

Expressiveness of NRC

Our running examples

- $(A, R) \mapsto \bigcup\{\mathsf{case}(\pi_2(p) \in_{\mathfrak{U}} A, \{\pi_1(p)\}, \emptyset) \mid p \in R\}$
- $R \mapsto \bigcup\{\{\mathsf{fib}(x, R)\} \mid x \in \{\pi_1(p) \mid p \in R\}\}$

Derivable constructs:

- maps $\{e_1(x) \mid x \in e_2\}$
- at type-level, $\mathsf{Bool} := \mathsf{Set}(1)$
- basic predicates $=_T\colon T \times T \to \mathsf{Bool}$, $\in_T\colon T \times \mathsf{Set}(T) \to \mathsf{Bool}$
- case analyses

Expressiveness of NRC

Our running examples

- $(A, R) \mapsto \bigcup\{\mathsf{case}(\pi_2(p) \in_{\mathfrak{U}} A, \{\pi_1(p)\}, \emptyset) \mid p \in R\}$
- $R \mapsto \bigcup\{\{\mathsf{fib}(x, R)\} \mid x \in \{\pi_1(p) \mid p \in R\}\}$

Derivable constructs:

- maps $\{e_1(x) \mid x \in e_2\}$
- at type-level, $\mathsf{Bool} := \mathsf{Set}(1)$
- basic predicates $=_T : T \times T \to \mathsf{Bool}$, $\in_T : T \times \mathsf{Set}(T) \to \mathsf{Bool}$
- case analyses
- Δ_0-separation $\{x \in e \mid \varphi(x)\}$

Proposition

NRC terms $e : T \to \mathsf{Bool}$ correspond exactly to Δ_0 formulas $\varphi(x^T)$

Extraction from Δ_0 specifications

Our effective result

Recall that $\varphi(i, o)$ is an implicit definition when it is functional: $\varphi(i, o) \wedge \varphi(i, o') \implies o = o'$

Extraction from Δ_0 intuitionistic implicit definitions

For every such $\varphi(i, o)$, there is a compatible NRC term $e(i)$

$$\varphi(i, o) \implies o = e(i)$$

Our effective result

Recall that $\varphi(i, o)$ is an implicit definition when it is functional: $\varphi(i, o) \wedge \varphi(i, o') \implies o = o'$

Extraction from Δ_0 intuitionistic implicit definitions

For every such $\varphi(i, o)$, there is a compatible NRC term $e(i)$

$$\varphi(i, o) \implies o = e(i)$$

Further, $e(i)$ may be efficiently computed from a cut-free focused proof

Our effective result

Recall that $\varphi(i, o)$ is an implicit definition when it is functional: $\varphi(i, o) \wedge \varphi(i, o') \implies o = o'$

Extraction from Δ_0 intuitionistic implicit definitions

For every such $\varphi(i, o)$, there is a compatible NRC term $e(i)$

$$\varphi(i, o) \implies o = e(i)$$

Further, $e(i)$ may be efficiently computed from a cut-free focused proof

Nota Bene

- Effectivity w/o efficiency: follows from completeness, compactness and an easy NRC/logical interpreatation correspondence
    - Efficiency is the ultimate goal
- Extension of Beth definability for flat queries $\mathsf{Set}(\mathfrak{U}^k) \times \ldots \times \mathsf{Set}(\mathfrak{U}^m) \to \mathsf{Set}(\mathfrak{U}^n)$
    - Can give some ideas for lower bounds

Potential use-cases for implicit→explicit

Use-case #1: inverting a transformation

Consider an *injective* NRC term such as fibs

$$\begin{array}{rccl} \mathsf{fibs}: & \mathsf{Set}(\mathfrak{U} \times \mathfrak{U}) & \to & \mathsf{Set}(\mathfrak{U} \times \mathsf{Set}(\mathfrak{U})) \\ & R & \mapsto & \{(a, R^{-1}(a)) \mid a \in \mathsf{cod}(f)\} \end{array}$$

Potential use-cases for implicit→explicit

Use-case #1: inverting a transformation

Consider an *injective* NRC term such as fibs

$$\begin{array}{rccl} \mathsf{fibs}: & \mathsf{Set}(\mathfrak{U} \times \mathfrak{U}) & \rightarrow & \mathsf{Set}(\mathfrak{U} \times \mathsf{Set}(\mathfrak{U})) \\ & R & \mapsto & \{(a, R^{-1}(a)) \mid a \in \mathsf{cod}(f)\} \end{array}$$

- can be converted to an implicit $\varphi(R, G)$

Potential use-cases for implicit→explicit

Use-case #1: inverting a transformation

Consider an *injective* NRC term such as fibs

$$\begin{array}{rccl}\mathsf{fibs}: & \mathsf{Set}(\mathfrak{U}\times\mathfrak{U}) & \to & \mathsf{Set}(\mathfrak{U}\times\mathsf{Set}(\mathfrak{U})) \\ & R & \mapsto & \{(a, R^{-1}(a)) \mid a \in \mathsf{cod}(f)\}\end{array}$$

- can be converted to an implicit $\varphi(R, G)$
- $\varphi(R, G)$ defines a partial function $G \mapsto R$

⇝ a NRC-definable inverse of fibs

Potential use-cases for implicit→explicit

Use-case #1: inverting a transformation

Consider an *injective* NRC term such as fibs

$$\begin{array}{rccl} \mathsf{fibs}: & \mathsf{Set}(\mathfrak{U} \times \mathfrak{U}) & \to & \mathsf{Set}(\mathfrak{U} \times \mathsf{Set}(\mathfrak{U})) \\ & R & \mapsto & \{(a, R^{-1}(a)) \mid a \in \mathsf{cod}(f)\} \end{array}$$

- can be converted to an implicit $\varphi(R, G)$
- $\varphi(R, G)$ defines a partial function $G \mapsto R$

$\rightsquigarrow$ a NRC-definable inverse of fibs

Use-case #2: views

Assume an imperative extension and a program

$$x := e_1(i); \ldots; y := e_2(i)$$

When e_2 is functional in terms of e_1:

- Compute $e_2'(x)$ from a proof

$\rightsquigarrow$ Potential optimization if $e_1(i)$ is costly?

Potential use-cases for implicit→explicit

Use-case #1: inverting a transformation

Consider an *injective* NRC term such as fibs

$$\begin{array}{rccl} \mathsf{fibs} : & \mathsf{Set}(\mathfrak{U} \times \mathfrak{U}) & \to & \mathsf{Set}(\mathfrak{U} \times \mathsf{Set}(\mathfrak{U})) \\ & R & \mapsto & \{(a, R^{-1}(a)) \mid a \in \mathsf{cod}(f)\} \end{array}$$

- can be converted to an implicit $\varphi(R, G)$
- $\varphi(R, G)$ defines a partial function $G \mapsto R$

⇝ a NRC-definable inverse of fibs

Use-case #2: views

Assume an imperative extension and a program

$$x := e_1(i); \ldots; y := e_2(i)$$

When e_2 is functional in terms of e_1:

- Compute $e_2'(x)$ from a proof

⇝ Potential optimization if $e_1(i)$ is costly?

Caveat: automation for functionality proofs?

Toward a proof system for nested sets

Wlog, we restrict to the following syntax

$$\begin{array}{lcl} t, u & ::= & x \mid (t, u) \mid \pi_1(t) \mid \pi_2(t) \mid () \\ \varphi, \psi & ::= & t =_{\mathfrak{U}} u \mid t \neq_{\mathfrak{U}} u \mid \exists x \in_T t\ \varphi \mid \forall x \in_T t\ \varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \end{array}$$

Toward a proof system for nested sets

Wlog, we restrict to the following syntax

$$\begin{array}{lll} t, u & ::= & x \mid (t, u) \mid \pi_1(t) \mid \pi_2(t) \mid () \\ \varphi, \psi & ::= & t =_{\mathfrak{U}} u \mid t \neq_{\mathfrak{U}} u \mid \exists x \in_T t\ \varphi \mid \forall x \in_T t\ \varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \end{array}$$

Derived formulas

$$\begin{array}{ccc} t \in_T u & := & \exists x \in u.\, t =_T u \\ t =_{\mathsf{Set}(T)} u & := & t \subseteq_T u \ \wedge \ u \subseteq_T t \end{array} \qquad \begin{array}{ccc} t \subseteq_T u & := & \forall x \in t.\, x \in_T u \\ & \ldots & \end{array}$$

Toward a proof system for nested sets

Wlog, we restrict to the following syntax

$$\begin{array}{lcl} t, u & ::= & x \mid (t, u) \mid \pi_1(t) \mid \pi_2(t) \mid () \\ \varphi, \psi & ::= & t =_{\mathfrak{U}} u \mid t \neq_{\mathfrak{U}} u \mid \exists x \in_T t\ \varphi \mid \forall x \in_T t\ \varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \end{array}$$

Derived formulas

$$\begin{array}{ccccccc} t \in_T u & := & \exists x \in u.\, t =_T u & & t \subseteq_T u & := & \forall x \in t.\, x \in_T u \\ t =_{\mathsf{Set}(T)} u & := & t \subseteq_T u \ \ \wedge \ \ u \subseteq_T t & & & \dots & \end{array}$$

- Bakes the axiom of extensionality in the definition of $=_T$
- **No further set-theoretic axioms**

What is a proof

Straightforward variants of the sequent calculus

- Sequents $\Gamma \vdash \Delta$ with Γ, Δ lists of Δ_0 formulas
- Intended semantics: $\bigwedge_{\phi \in \Gamma} \phi \implies \bigvee_{\psi \in \Delta} \psi$
- Deduction according to proof rules of the shape

$$\frac{\Gamma_1 \vdash \Delta_1 \qquad \ldots \qquad \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta}$$

Left and right rules for each connectives + *structural rules* + *cut*

Examples

$$\exists\text{-R}\ \frac{\Gamma, t \in u \vdash \phi[t/x], \Delta}{\Gamma, t \in u \vdash \exists x \in u.\phi, \Delta} \qquad \textsc{cut}\ \frac{\Gamma \vdash \phi, \Delta \qquad \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta} \qquad \textsc{axiom}\ \frac{}{\Gamma, \phi \vdash \phi, \Delta}$$

Formal proofs of functionality

Certificate that $\varphi(i,o)$ is an implicit definition: a derivation

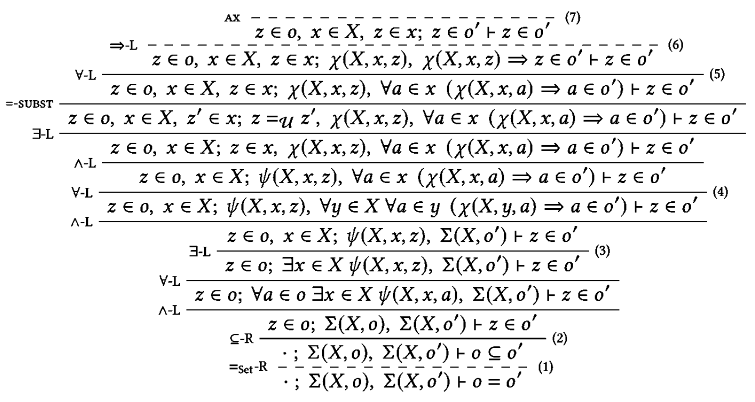
$$\cdot;\ \varphi(i,o),\ \varphi(i,o') \vdash o = o'$$

$$
=_{\text{Set}}\text{-R}\ \dfrac{
\subseteq\text{-R}\ \dfrac{
\wedge\text{-L}\ \dfrac{
\forall\text{-L}\ \dfrac{
\exists\text{-L}\ \dfrac{
\wedge\text{-L}\ \dfrac{
\forall\text{-L}\ \dfrac{
\wedge\text{-L}\ \dfrac{
\exists\text{-L}\ \dfrac{
=\text{-SUBST}\ \dfrac{
\forall\text{-L}\ \dfrac{
\Rightarrow\text{-L}\ \dfrac{
\text{AX}\ \dfrac{}{z \in o,\ x \in X,\ z \in x;\ z \in o' \vdash z \in o'}\ (7)
}{z \in o,\ x \in X,\ z \in x;\ \chi(X,x,z),\ \chi(X,x,z) \Rightarrow z \in o' \vdash z \in o'}\ (6)
}{z \in o,\ x \in X,\ z \in x;\ \chi(X,x,z),\ \forall a \in x\ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o'}\ (5)
}{z \in o,\ x \in X,\ z' \in x;\ z =_{\mathcal{U}} z',\ \chi(X,x,z),\ \forall a \in x\ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o'}
}{z \in o,\ x \in X;\ z \in x,\ \chi(X,x,z),\ \forall a \in x\ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o'}
}{z \in o,\ x \in X;\ \psi(X,x,z),\ \forall a \in x\ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o'}
}{z \in o,\ x \in X;\ \psi(X,x,z),\ \forall y \in X\ \forall a \in y\ (\chi(X,y,a) \Rightarrow a \in o') \vdash z \in o'}\ (4)
}{z \in o,\ x \in X;\ \psi(X,x,z),\ \Sigma(X,o') \vdash z \in o'}
}{z \in o;\ \exists x \in X\ \psi(X,x,z),\ \Sigma(X,o') \vdash z \in o'}\ (3)
}{z \in o;\ \forall a \in o\ \exists x \in X\ \psi(X,x,a),\ \Sigma(X,o') \vdash z \in o'}
}{z \in o;\ \Sigma(X,o),\ \Sigma(X,o') \vdash z \in o'}
}{\cdot\,;\ \Sigma(X,o),\ \Sigma(X,o') \vdash o \subseteq o'}\ (2)
}{\cdot\,;\ \Sigma(X,o),\ \Sigma(X,o') \vdash o = o'}\ (1)
$$

Formal proofs of functionality

Certificate that $\varphi(i, o)$ is an implicit definition: a derivation

$$\cdot;\ \varphi(i,o),\ \varphi(i,o') \vdash o = o'$$



$$
\begin{array}{rcl}
\text{AX} & \text{- -} & (7)\\
 & z \in o,\ x \in X,\ z \in x;\ z \in o' \vdash z \in o' & \\
\Rightarrow\text{-L} & \text{- -} & (6)\\
 & z \in o,\ x \in X,\ z \in x;\ \chi(X,x,z),\ \chi(X,x,z) \Rightarrow z \in o' \vdash z \in o' & \\
\forall\text{-L} & \rule{36em}{0.4pt} & (5)\\
 & z \in o,\ x \in X,\ z \in x;\ \chi(X,x,z),\ \forall a \in x\ \ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o' & \\
=\text{-SUBST} & \rule{40em}{0.4pt} & \\
 & z \in o,\ x \in X,\ z' \in x;\ z =_{\mathcal{U}} z',\ \chi(X,x,z),\ \forall a \in x\ \ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o' & \\
\exists\text{-L} & \rule{40em}{0.4pt} & \\
 & z \in o,\ x \in X;\ z \in x,\ \chi(X,x,z),\ \forall a \in x\ \ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o' & \\
\wedge\text{-L} & \rule{36em}{0.4pt} & \\
 & z \in o,\ x \in X;\ \psi(X,x,z),\ \forall a \in x\ \ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o' & \\
\forall\text{-L} & \rule{36em}{0.4pt} & (4)\\
 & z \in o,\ x \in X;\ \psi(X,x,z),\ \forall y \in X\ \forall a \in y\ \ (\chi(X,y,a) \Rightarrow a \in o') \vdash z \in o' & \\
\wedge\text{-L} & \rule{36em}{0.4pt} & \\
 & z \in o,\ x \in X;\ \psi(X,x,z),\ \Sigma(X,o') \vdash z \in o' & \\
\exists\text{-L} & \rule{24em}{0.4pt} & (3)\\
 & z \in o;\ \exists x \in X\ \psi(X,x,z),\ \Sigma(X,o') \vdash z \in o' & \\
\forall\text{-L} & \rule{28em}{0.4pt} & \\
 & z \in o;\ \forall a \in o\ \exists x \in X\ \psi(X,x,a),\ \Sigma(X,o') \vdash z \in o' & \\
\wedge\text{-L} & \rule{28em}{0.4pt} & \\
 & z \in o;\ \Sigma(X,o),\ \Sigma(X,o') \vdash z \in o' & \\
\subseteq\text{-R} & \rule{18em}{0.4pt} & (2)\\
 & \cdot\ ;\ \Sigma(X,o),\ \Sigma(X,o') \vdash o \subseteq o' & \\
=_{\mathsf{Set}}\text{-R} & \text{- - - - - - - - - - - - - - -} & (1)\\
 & \cdot\ ;\ \Sigma(X,o),\ \Sigma(X,o') \vdash o = o' &
\end{array}
$$

Proof idea for efficient extraction: compute an explicit definition by induction over the proof

Formal proofs of functionality

Certificate that $\varphi(i,o)$ is an implicit definition: a derivation

$$\cdot;\ \varphi(i,o),\ \varphi(i,o') \vdash o = o'$$

$$
\begin{array}{lcr}
\hdashline
\text{AX} & z \in o,\ x \in X,\ z \in x;\ z \in o' \vdash z \in o' & (7) \\
\hdashline
\Rightarrow\text{-L} & z \in o,\ x \in X,\ z \in x;\ \chi(X,x,z),\ \chi(X,x,z) \Rightarrow z \in o' \vdash z \in o' & (6) \\
\hline
\forall\text{-L} & z \in o,\ x \in X,\ z \in x;\ \chi(X,x,z),\ \forall a \in x\ \ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o' & (5) \\
\hline
=\text{-SUBST} & z \in o,\ x \in X,\ z' \in x;\ z =_{\mathcal{U}} z',\ \chi(X,x,z),\ \forall a \in x\ \ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o' & \\
\hline
\exists\text{-L} & z \in o,\ x \in X;\ z \in x,\ \chi(X,x,z),\ \forall a \in x\ \ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o' & \\
\hline
\wedge\text{-L} & z \in o,\ x \in X;\ \psi(X,x,z),\ \forall a \in x\ \ (\chi(X,x,a) \Rightarrow a \in o') \vdash z \in o' & \\
\hline
\forall\text{-L} & z \in o,\ x \in X;\ \psi(X,x,z),\ \forall y \in X\ \forall a \in y\ \ (\chi(X,y,a) \Rightarrow a \in o') \vdash z \in o' & (4) \\
\hline
\wedge\text{-L} & z \in o,\ x \in X;\ \psi(X,x,z),\ \Sigma(X,o') \vdash z \in o' & \\
\hline
\exists\text{-L} & z \in o;\ \exists x \in X\ \psi(X,x,z),\ \Sigma(X,o') \vdash z \in o' & (3) \\
\hline
\forall\text{-L} & z \in o;\ \forall a \in o\ \exists x \in X\ \psi(X,x,a),\ \Sigma(X,o') \vdash z \in o' & \\
\hline
\wedge\text{-L} & z \in o;\ \Sigma(X,o),\ \Sigma(X,o') \vdash z \in o' & \\
\hline
\subseteq\text{-R} & \cdot\,;\ \Sigma(X,o),\ \Sigma(X,o') \vdash o \subseteq o' & (2) \\
\hdashline
=_{\mathsf{Set}}\text{-R} & \cdot\,;\ \Sigma(X,o),\ \Sigma(X,o') \vdash o = o' & (1)
\end{array}
$$

Proof idea for efficient extraction: compute an explicit definition by induction over the proof

Problem: what invariant?

Three standard restrictions

The adjectives

Cut-free, intuitionistic, focused

- All of the proofs we are going to be considering are cut-free
- We will ultimately drop the restriction to intuitionistic proofs...
- ...but ultimately enforce focusing anyway

Cut-freeness

$$\textsc{cut}\ \frac{\Gamma \vdash \phi, \Delta \qquad \Gamma, \phi \vdash \Delta}{\Gamma \vdash \Delta}$$

- Intuition: allows to introduce a lemma ϕ
- Other intuition: allows to *compose* proofs

Cut-elimination (Gentzen)

The cut rule does not allow to prove more sequents

- Effective argument, but cut-elimination is expensive

  (lower bound in $\mathcal{G}_3$ (Buss), i.e. above non-elementary)
- Related to computation in the λ-calculus (Curry-Howard)
- (Easier to define in the sequent calculus than in other systems)
- Cut-free proofs have a nice *subformula property*
- We will require cut-freeness essentially everywhere in the sequel

Intuitionism

At the intuitive level, reject the law of excluded middle/reasoning ad absurdum

$$\phi \vee \neg\phi \qquad\qquad \neg\neg\phi \implies \phi$$

Intuitionism

At the intuitive level, reject the law of excluded middle/reasoning ad absurdum

$$\phi \vee \neg\phi \qquad\qquad \neg\neg\phi \implies \phi$$

Technically: restrict to sequents $\Gamma \vdash \Delta$ with $|\Delta| \leq 1$.

Intuitionism

At the intuitive level, reject the law of excluded middle/reasoning ad absurdum

$$\phi \lor \neg\phi \qquad\qquad \neg\neg\phi \implies \phi$$

Technically: restrict to sequents $\Gamma \vdash \Delta$ with $|\Delta| \leq 1$.

Pros/cons

- Nicer to work with
- Classical logic can be embedded in it anyway

Intuitionism

At the intuitive level, reject the law of excluded middle/reasoning ad absurdum

$$\phi \lor \neg\phi \qquad\qquad \neg\neg\phi \implies \phi$$

Technically: restrict to sequents $\Gamma \vdash \Delta$ with $|\Delta| \leq 1$.

Pros/cons

- Nicer to work with
- Classical logic can be embedded in it anyway

Conservativity for implicit definitions

If $\phi(i, o)$ is functional, then there is a formula $\chi(\vec{x})$ such that the conjoined formula

$$\phi^{\neg\neg}(i, o) \land \forall \vec{x}.\ \chi(\vec{x}) \lor \neg\chi(\vec{x})$$

can be proved to be functional in intuitionistic logic

Intuitionism

At the intuitive level, reject the law of excluded middle/reasoning ad absurdum

$$\phi \vee \neg\phi \qquad\qquad \neg\neg\phi \implies \phi$$

Technically: restrict to sequents $\Gamma \vdash \Delta$ with $|\Delta| \leq 1$.

Pros/cons

- Nicer to work with
- Classical logic can be embedded in it anyway

Conservativity for implicit definitions

If $\phi(i, o)$ is functional, then there is a formula $\chi(\vec{x})$ such that the conjoined formula

$$\phi^{\neg\neg}(i, o) \wedge \forall \vec{x}.\ \chi(\vec{x}) \vee \neg\chi(\vec{x})$$

can be proved to be functional in intuitionistic logic

Actually non-trivial!! (I don't know a corresponding efficient algorithm)

Focusing

A normal form for proofs refining cut-freeness (Andreoli 90s)

Rough idea

Decompose proofs by forcing saturations by certain rules in *positive* and *negative* phase.

- Initially motivated by proof-search
- Like cut-elim, does not change provable statements
- To us: restricts the shape of proofs so much it allows to use simpler inductive invariants

(probably a crutch, but we don't know how to work without it for now)

Focusing

A normal form for proofs refining cut-freeness (Andreoli 90s)

Rough idea

Decompose proofs by forcing saturations by certain rules in *positive* and *negative* phase.

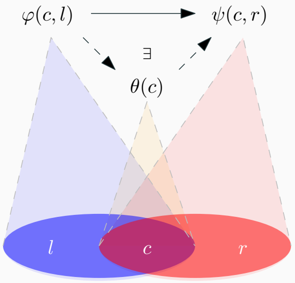
- Initially motivated by proof-search
- Like cut-elim, does not change provable statements
- To us: restricts the shape of proofs so much it allows to use simpler inductive invariants

(probably a crutch, but we don't know how to work without it for now)

Complexity-wise (to the best of my knowledge)

A cut-free proof can be turned into a focused cut-free proof in exponential time.

Preliminary: Craig interpolation



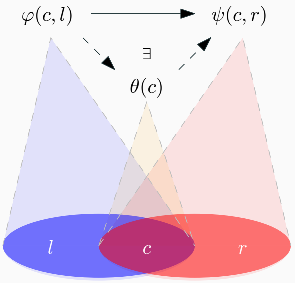
Craig interpolation

If $\varphi \Rightarrow \psi$, there exists θ such that

$$\varphi \Rightarrow \theta \qquad \text{and} \qquad \theta \Rightarrow \psi$$

Further, θ mentions *only* variables/relation symbols common to φ and ψ.

Preliminary: Craig interpolation



Craig interpolation

If $\varphi \Rightarrow \psi$, there exists θ such that

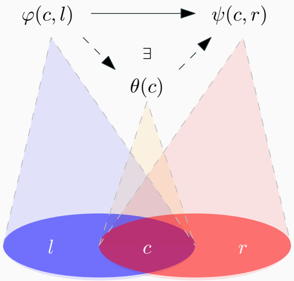
$$\varphi \Rightarrow \theta \qquad \text{and} \qquad \theta \Rightarrow \psi$$

Further, θ mentions *only* variables/relation symbols common to φ and ψ.

- Robust result

Δ_0-interpolation, intuitionistic/linear logic...

Preliminary: Craig interpolation



Craig interpolation

If $\varphi \Rightarrow \psi$, there exists θ such that

$$\varphi \Rightarrow \theta \qquad \text{and} \qquad \theta \Rightarrow \psi$$

Further, θ mentions *only* variables/relation symbols common to φ and ψ.

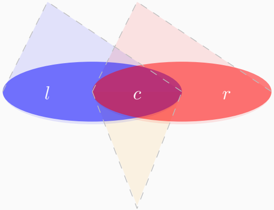
- Robust result

Δ_0-interpolation, intuitionistic/linear logic...

- θ linear-time computable from cut-free proofs
- Interpolation ⇒ effective Beth definability

Our extraction procedure

$\Gamma(c, l),\ \Delta(c, r)\ \ \vdash\ \ l \subseteq r$



$\rightsquigarrow\ \exists\, e.\ \ l \subseteq e(c) \subseteq r$

Suppose $\Gamma(c, l), \Delta(c, r) \vdash \psi$.
Then we can compute $e(c)$ in NRC such that

Higher-type interpolation Lemma

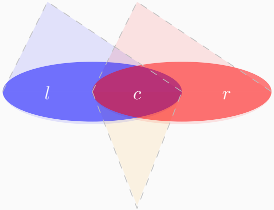
- if ψ is $l = r$, then $\Gamma, \Delta \models l = e \wedge r = e$
- if ψ is $l \subseteq r$, then $\Gamma, \Delta \models l \subseteq e \wedge e \subseteq r$
- if ψ is $l \in r$, then $\Gamma, \Delta \models l \in e$

Stronger than standard interpolation

RHS depends on l

Our extraction procedure

$\Gamma(c, l),\ \Delta(c, r) \ \vdash \ l \subseteq r$



$\rightsquigarrow \exists\, e.\ \ l \subseteq e(c) \subseteq r$

Suppose $\Gamma(c, l), \Delta(c, r) \vdash \psi$.
Then we can compute $e(c)$ in NRC such that

Higher-type interpolation Lemma

- if ψ is $l = r$, then $\Gamma, \Delta \models l = e \wedge r = e$
- if ψ is $l \subseteq r$, then $\Gamma, \Delta \models l \subseteq e \wedge e \subseteq r$
- if ψ is $l \in r$, then $\Gamma, \Delta \models l \in e$

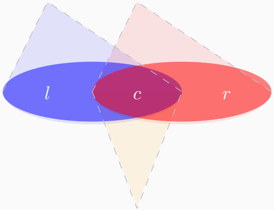
Stronger than standard interpolation

RHS depends on l

Extraction procedure: apply with $\Gamma := \varphi(i, o)$, $\Delta := \varphi(i, o')$ and $\psi := o = o'$

Our extraction procedure

$\Gamma(c,l),\ \Delta(c,r)\ \vdash\ l \subseteq r$



$\rightsquigarrow\ \exists\, e.\ \ l \subseteq e(c) \subseteq r$

Suppose $\Gamma(c,l), \Delta(c,r) \vdash \psi$.
Then we can compute $e(c)$ in NRC such that

Higher-type interpolation Lemma

- if ψ is $l = r$, then $\Gamma, \Delta \models l = e \wedge r = e$
- if ψ is $l \subseteq r$, then $\Gamma, \Delta \models l \subseteq e \wedge e \subseteq r$
- if ψ is $l \in r$, then $\Gamma, \Delta \models l \in e$

Stronger than standard interpolation

RHS depends on l

Extraction procedure: apply with $\Gamma := \varphi(i, o)$, $\Delta := \varphi(i, o')$ and $\psi := o = o'$

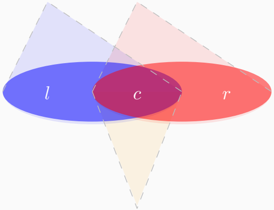
Proof idea

Induction over the proof-tree; at some key steps

- Δ_0 interpolation
- NRC-definability of Δ_0-separation

Our extraction procedure

$$\Gamma(c,l),\ \Delta(c,r)\ \vdash\ l\subseteq r$$



$$\rightsquigarrow\ \exists\, e.\ \ l\subseteq e(c)\subseteq r$$

Suppose $\Gamma(c,l), \Delta(c,r) \vdash \psi$.
Then we can compute $e(c)$ in NRC such that

Higher-type interpolation Lemma

- if ψ is $l = r$, then $\Gamma, \Delta \models l = e \wedge r = e$
- if ψ is $l \subseteq r$, then $\Gamma, \Delta \models l \subseteq e \wedge e \subseteq r$
- if ψ is $l \in r$, then $\Gamma, \Delta \models l \in e$

Stronger than standard interpolation

RHS depends on l

Extraction procedure: apply with $\Gamma := \varphi(i,o)$, $\Delta := \varphi(i,o')$ and $\psi := o = o'$

Proof idea

Induction over the proof-tree; at some key steps

- Δ_0 interpolation
- NRC-definability of Δ_0-separation

Problem: does not generalize well to sequents with multiple conclusions

Handling classical proofs

New strategy: induction over the output type, some tedious proof theory and

(New and somewhat exciting!) NRC parameter collection theorem

Let L, R be sets of variables with $C = L \cap R$ and

- ϕ_L and $\lambda(z)$ Δ_0 formulas over L
- ϕ_R and $\rho(z, y)$ Δ_0 formulas over R
- r a variable of R and c a variable of C.

Suppose that we have a proof of $\phi_L \wedge \phi_R \Rightarrow \exists y \in_p r\, \forall z \in c.\ \ \lambda(z) \iff \rho(z, y)$

Then one may compute in polynomial time a NRC expression E with free variables in C such that

$$\phi_L \wedge \phi_R \quad \Longrightarrow \quad \{z \in c \mid \lambda(z)\} \in E$$

- By induction over *focused proofs*
- E is a *set* of candidate definitions for λ parameterized over the input

(reminiscent of a theorem of Chang and Makkai that yields definabilty from a proof of *fewness* rather than *uniqueness*)

More specifically

Lemma

Let L, R be sets of variables with $C = L \cap R$ and

- ϕ_L and $\lambda(z)$ Δ_0 formulas over L
- ϕ_R and $\rho(z, y)$ Δ_0 formulas over R
- r a variable of R and c a variable of C.

Suppose that we have a proof of $\phi_L \wedge \phi_R \;\Rightarrow\; \exists y \in_p r \, \forall z \in c. \;\; \lambda(z) \iff \rho(z, y)$

Then one may compute in polynomial time a NRC expression E and a Δ_0 θ over C s.t.

$$\phi_L \wedge \theta \quad \Longrightarrow \quad \{z \in c \mid \lambda(z)\} \in E \qquad \text{and} \qquad \phi_R \vdash \theta$$

Intuitions:

- θ is an interpolant for a proof we are also computing on the fly
- Focusing allows to keep the invariant rather specific w.r.t. the r.h.s. formula

Key step: existential rule introducing the ``main" formula

With $\mathcal{G} = \exists y \in_p r.\forall z \in c.\ \lambda(z) \iff \rho(z,y)$

$$\exists\,\dfrac{\forall\,\dfrac{\wedge\,\dfrac{\vee\,\dfrac{\Theta_L, \Theta_R, x \in c \vdash \Delta_L, \Delta_R, \neg\rho(x,w), \lambda(x), \mathcal{G}}{\Theta_L, \Theta_R, x \in c \vdash \Delta_L, \Delta_R, \rho(x,w) \Rightarrow \lambda(x), \mathcal{G}} \qquad \vee\,\dfrac{\Theta_L, \Theta_R, x \in c \vdash \Delta_L, \Delta_R, \neg\lambda(x), \rho(x,w), \mathcal{G}}{\Theta_L, \Theta_R, x \in c \vdash \Delta_L, \Delta_R, \lambda(x) \Rightarrow \rho(x,w), \mathcal{G}}}{\Theta_L, \Theta_R, x \in c \vdash \Delta_L, \Delta_R, \lambda(x) \Leftrightarrow \rho(x,w), \mathcal{G}}}{\Theta_L, \Theta_R \vdash \Delta_L, \Delta_R, \forall z \in c.\ (\lambda(z) \Leftrightarrow \rho(z,w)), \mathcal{G}}}{\Theta_L, \Theta_R \vdash \Delta_L, \Delta_R, \mathcal{G}}$$

- Shape around the root of the tree guaranteed by focusing
- Applying the induction hypothesis we have

$$\begin{array}{llll} & \Theta_L, x \in c \models \lambda(x), \Delta_L, \theta_1^{\mathsf{IH}} \vee \Lambda \in E_1^{\mathsf{IH}} & \text{and} & \Theta_L, x \in c \models \neg\lambda(x), \Delta_L, \theta_2^{\mathsf{IH}} \vee \Lambda \in E_2^{\mathsf{IH}} \\ \text{and} & \Theta_R \models \neg\rho(x,w), \Delta_R, \neg\theta_1^{\mathsf{IH}} & \text{and} & \Theta_R \models \rho(x,w), \Delta_R, \neg\theta_2^{\mathsf{IH}} \end{array}$$

- So $\theta := \exists x \in c.\ \theta_1^{\mathsf{IH}} \wedge \theta_2^{\mathsf{IH}}$ and $E := \left\{\left\{x \in c \mid \theta_2^{\mathsf{IH}}\right\}\right\} \cup \bigcup\left\{E_1^{\mathsf{IH}} \cup E_2^{\mathsf{IH}} \mid x \in c\right\}$ works

Interpretations and multi-sorted definability

Interpretations

Nested collections can be regarded as multi-sorted structures

An object X of sort $\mathsf{Set}(\mathfrak{U} \times \mathsf{Set}(\mathfrak{U}))$

Sorts: $\mathfrak{U}, \mathsf{Set}(\mathfrak{U}), \mathfrak{U} \times \mathsf{Set}(\mathfrak{U})$

Function symbols: $\pi_1, \pi_2, \langle -, - \rangle$

Relation symbol: $\in_{\mathfrak{U}}$

Semantics: subobjects of X

Interpretations

Nested collections can be regarded as multi-sorted structures

An object X of sort $\mathsf{Set}(\mathfrak{U} \times \mathsf{Set}(\mathfrak{U}))$

Sorts: $\mathfrak{U}, \mathsf{Set}(\mathfrak{U}), \mathfrak{U} \times \mathsf{Set}(\mathfrak{U})$

Function symbols: $\pi_1, \pi_2, \langle -, - \rangle$

Relation symbol: $\in_{\mathfrak{U}}$

Semantics: subobjects of X

Interpretations: maps between finite structures defined by FO formulas

Can express

- product, disjoint union of structures $\mathfrak{M}, \mathfrak{N} \mapsto \mathfrak{M} \times \mathfrak{N}, \mathfrak{M} + \mathfrak{N}$
- definable substructures and quotients

Interpretations

Nested collections can be regarded as multi-sorted structures

An object X of sort $\mathsf{Set}(\mathfrak{U} \times \mathsf{Set}(\mathfrak{U}))$

Sorts: $\mathfrak{U}, \mathsf{Set}(\mathfrak{U}), \mathfrak{U} \times \mathsf{Set}(\mathfrak{U})$

Function symbols: $\pi_1, \pi_2, \langle -, - \rangle$

Relation symbol: $\in_{\mathfrak{U}}$

Semantics: subobjects of X

Interpretations: maps between finite structures defined by FO formulas

Can express

- product, disjoint union of structures $\mathfrak{M}, \mathfrak{N} \mapsto \mathfrak{M} \times \mathfrak{N}, \mathfrak{M} + \mathfrak{N}$
- definable substructures and quotients

NRC and interpretations

For structures corresponding to nested collections,
NRC and Δ_0-interpretations coincide

Remark: efficient translation from interpretations to NRC

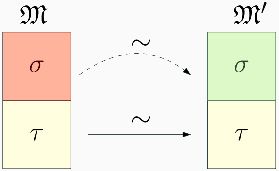
From multi-sorted implicit definitions to explicit interpretations

Fix a theory Σ over two sorts τ and σ

Wlog: two sets of sorts

Multi-sorted implicit definability

σ is implicitly definable from τ when, for every $\mathfrak{M}, \mathfrak{M}' \models \Sigma$ and bijective homomorphism $\mathfrak{M}\big|_{\tau} \cong \mathfrak{M}'\big|_{\tau}$, there is a unique extension $\mathfrak{M} \cong \mathfrak{M}'$



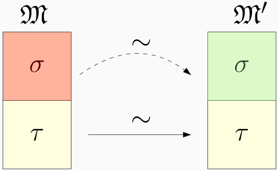
From multi-sorted implicit definitions to explicit interpretations

Fix a theory Σ over two sorts τ and σ

Wlog: two sets of sorts

Multi-sorted implicit definability

σ is implicitly definable from τ when, for every $\mathfrak{M}, \mathfrak{M}' \models \Sigma$ and bijective homomorphism $\mathfrak{M}\big|_\tau \cong \mathfrak{M}'\big|_\tau$, there is a unique extension $\mathfrak{M} \cong \mathfrak{M}'$



Reduction for implicit definition of nested transformations: single model where

- τ contains the input and $\mathfrak{U}$
- σ contains the output

possibly more complex than the input

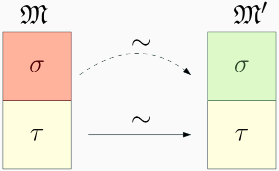
From multi-sorted implicit definitions to explicit interpretations

Fix a theory Σ over two sorts τ and σ

Wlog: two sets of sorts

Multi-sorted implicit definability

σ is implicitly definable from τ when, for every $\mathfrak{M}, \mathfrak{M}' \models \Sigma$ and bijective homomorphism $\mathfrak{M}\big|_{\tau} \cong \mathfrak{M}'\big|_{\tau}$, there is a unique extension $\mathfrak{M} \cong \mathfrak{M}'$



Reduction for implicit definition of nested transformations: single model where

- τ contains the input and $\mathfrak{U}$
- σ contains the output

possibly more complex than the input

Theorem

If σ is implicitly definable from τ, there is an interpretation of Σ into $\Sigma\big|_{\tau}$

Further perspectives

Natural question

Can we make the multi-sorted theorem effective?

- There is a natural notion of implicitly definable (although non-obvious)
- Effectivity is not an issue, but efficiency is
- (the intuitionistic case is easy)

Further topics

- Coq formalization with extraction
- Curry-Howard approach to the extraction of NRC terms

  ``untyped NRC'' treated by Sazonov
- Other settings for extraction from implicit definitions?