Synthesizing Nested Relational Queries from Implicit Specifications

Michael Benedikt  
University of Oxford  
Oxford, UK  
michael.benedikt@cs.ox.ac.uk

Cécilia Pradic  
Swansea University  
Swansea, UK  
c.pradic@swansea.ac.uk

Christoph Wernhard  
University of Potsdam  
Potsdam, Germany  
christoph.wernhard@uni-potsdam.de

ABSTRACT
Derived datasets can be defined implicitly or explicitly. An implicit definition (of dataset \( O \) in terms of datasets \( \bar{I} \)) is a logical specification involving the source data \( \bar{I} \) and the interface data \( O \). It is a valid definition of \( O \) in terms of \( \bar{I} \), if two models of the specification agreeing on \( \bar{I} \) agree on \( O \). In contrast, an explicit definition is a query that produces \( O \) from \( \bar{I} \). Variants of Beth’s theorem [8] state that one can convert implicit definitions to explicit ones. Further, this conversion can be done effectively given a proof witnessing implicit definability in a suitable proof system. We prove the analogous effective implicit-to-explicit result for nested relations: implicit definitions, given in the natural logic for nested relations, can be effectively converted to explicit definitions in the nested relational calculus (NRC). As a consequence, we can effectively extract rewritings of NRC queries in terms of NRC views, given a proof witnessing that the query is determined by the views.

CCS CONCEPTS
• Theory of computation → Automated reasoning.

KEYWORDS
Nested Relational Calculus, Views, Beth definability

1 INTRODUCTION
One way of describing a virtual datasource is via implicit definition: a specification \( \Sigma \) = e.g. in logic – involving symbols for the "implicitly defined" object \( O \) and another set of data objects \( \bar{I} \) that can be used to define \( O \). The specification may mention other data objects. But to be an implicit definition, any two models of \( \Sigma \) that agree on \( \bar{I} \) must agree on \( O \). That is, \( \Sigma \) must guarantee that \( O \) is a function of \( \bar{I} \). In the case where \( \Sigma \) is in first-order logic, this hypothesis can be expressed as a first-order entailment, using two copies of the vocabulary, primed and unprimed, representing the two models:

\[
\Sigma \land \Sigma' \land \forall \bar{x}_i \left[ I_i(\bar{x}_i) \leftrightarrow I'_i(\bar{x}_i) \right] \models \forall \bar{x} \left[ O(\bar{x}) \leftrightarrow O'(\bar{x}) \right] \tag{\star}
\]

Above \( \Sigma' \) is a copy of \( \Sigma \) with primed versions of each predicate.

A fundamental result in logic states that we can replace an implicit definition with an explicit definition: a first-order query \( Q \) such that whenever \( \Sigma(\bar{I}, O, \ldots) \) holds, \( O = Q(\bar{I}) \). The original result of this kind is Beth’s theorem [8], which deals with classical first-order logic. Segoufin and Vianu’s [33] looks at the case where \( \Sigma \) is in active-domain first-order logic, or equivalently a Boolean relational algebra expression. Their conclusion is that one can produce an explicit definition of \( O \) over \( \bar{I} \) in relational algebra.

Prior work in databases like [33] focused on the case where the \( \bar{I} \) correspond to outputs of views, while \( O \) is a query which is determined by the view outputs. More precisely, \( \Sigma(I_1 \ldots I_j, \bar{B}, O) \) specifies each \( I_i \) as an object defined by an active-domain first-order formula \( \varphi_i \) over base data \( \bar{B} \), and also defines \( O \) as an active-domain first-order query \( \varphi_Q \) over \( \bar{B} \). In this case, \( \Sigma \) implicitly defining \( O \) in terms of \( \bar{I} \) is called “determinacy of the query by the views”. Segoufin and Vianu’s result implies that whenever a relational algebra query is determined by relational algebra views, then the query is rewritable over the views by a relational algebra query. There are other scenarios where one data object can be implicitly defined by others: for example, a set of integrity constraints can enforce that one object is functionally determined by others. Segoufin and Vianu’s results also apply to those settings: when you can imply define one table \( O \) from another set of tables \( \bar{I} \), using first-order logic, there must be a relational algebra recipe for generating \( O \) from \( \bar{I} \).

Previous “Beth-style results”, like [8, 33] mentioned above, are effective. From a proof of the entailment (\( \star \)) in a suitable proof system, one can extract an explicit definition effectively, even in polynomial time. In early proofs of Beth’s theorem the proof systems were custom-designed for the task of proving implicit definitions and the bounds were not stated. Later on standard proof systems such as sequent calculi [34] or resolution [21] were employed, and the polynomial claim was explicit. It is important that in our definition of implicit definability, we require the existence of a proof witness. By the completeness theorem for first-order logic, requiring such a proof witness is equivalent to demanding that implicit definability of \( O \) over \( \bar{I} \) holds for all instances, not just finite ones. The bottom line for the relational setting can then be restated as: from a proof
witness of implicit definability of $O$ by $\tilde{T}$, you can efficiently generate a relational algebra recipe generating $O$ from $\tilde{T}$.

This paper deals with the situation for nested relations, a data model heavily explored in the database community. There is a natural analog of active domain first-order logic, suitable for implicit specification. These are the $\Delta_0$ formulas, logical expressions where quantification is over elements within nested sets defined by terms. The notion of a $\Delta_0$ specification $\Sigma(\tilde{l}, \ldots)$ implicitly defining nested relation $o$ in terms of $\tilde{l}$ is the obvious one: for any two nested relations satisfying $\Sigma$, and agreeing on $\tilde{l}$, they must agree on $o$. There is also a natural proof notion of proof witness for determinacy, using a proof system for $\Delta_0$ formulas. The analog of relational algebra for explicit definitions is nested relational calculus NRC [41], which is the standard query language for nested relations. Our main result is:

From a proof $p$ that $\Sigma$ implicitly defines $o$ in terms of $\tilde{l}$, we can obtain, in PTIME, an NRC expression $E$ that explicitly defines $o$ from $\tilde{l}$, relative to $\Sigma$.

A special case of this result concerns NRC views and queries. Our result implies that if we have NRC views $\tilde{V}$ that determine an NRC query $Q$, then we can generate -- from a suitable proof -- an NRC rewriting of $Q$ in terms of $\tilde{V}$.

The fact that such an NRC rewriting exists whenever there is a functional relationship was proven in [6]. But the argument was model-theoretic, and it was unclear how to obtain any algorithm for producing the NRC definition $E$ from a proof.

Example 1.1. We consider the case where our specification $\Sigma(Q, V, B)$ describes a view $V$, a query $Q$, as well as some constraints on the base data $B$. Our base data $B$ is of type $\text{Set}(\mathbb{U} \times \text{Set}(\mathbb{U}))$, where $\mathbb{U}$ refers to the basic set of elements, the "Ur-elements". That is, $B$ is a set of pairs, where the first item is a data item and the second is a set of data items. View $V$ is of type $\text{Set}(\mathbb{U} \times \mathbb{U})$, a set of pairs, given by the query that is the usual "flattening" of $B$: in NRC this can be expressed as $\{ (\pi_1(b), c) | c \in \pi_2(b) \in B \}$. The view definition can be converted to a specification in our logic.

A query $Q$ might ask for a selection of the pairs in $B$, whose first component is contained in the second: $\{ b \in B | \pi_1(b) \in \pi_2(b) \}$. The definition of $Q$ can also be incorporated into our specification.

View $V$ is not sufficient to answer $Q$ in general. This is the case if we assume as part of $\Sigma$ an integrity constraint stating that the first component of $B$ is a key. We can prove that $\Sigma(Q, V, B)$ implicitly defines $Q$ in terms of $V$, and from this proof our algorithm can produce an NRC rewriting of $Q$ in terms of $V$.

Organization. We overview related work in Section 2 and provide preliminaries in Section 3. Section 4 presents our main result. It is proven in Section 6, making use of infrastructure from Section 5. We close with discussion in Section 7. Due to space constraints, many proofs are deferred to the appendix or the full version.

2 RELATED WORK

In addition to the theorems of Beth and Segoufin-Vianu mentioned in the introduction, there are numerous works on effective Beth-style results for other logics. Some concern fragments of classical first-order logic, such as the guarded fragment [7, 20]; others deal with non-classical logics such as description logics [37]. The Segoufin-Vianu result is closely related to variations of Beth’s theorem and Craig interpolation for relativized quantification, such as Otto’s interpolation theorem [31]. There are also effective interpolation and definability results for logics richer than or incomparable to first-order logic, such as fragments of fixpoint logics [3, 13].

The connection between Beth-style results and rewriting database queries over views originates in [30, 33]. The idea of using effective Beth results to generate view rewritings from proofs appears in [15], and is explored in more detail first in [38] and later in [4]. The study of determinacy and its connections with rewriting has been extended to other settings (e.g. [26]). While works like [26, 30] deal only with decidability/completeness issues, and expressiveness of rewritings, there is also a rich literature on algorithms for view rewriting; see, for example, [1].

Our main result can be seen from the perspective of interpolation and definability research, relating to Beth theorems “up-to-isomorphism”. In the setting of nested relations, our implicit definability hypothesis is that two models that satisfy a specification and agree on the inputs must agree on the output nested relations, where “agree on the output” now means up to extensional equivalence of sets, which is a special (definable) kind of isomorphism. Beth-like theorems up to isomorphism originate in Gaifman’s [16] and are studied extensively by Hodges and his collaborators (e.g. [17–19]). The focus of these last works is model-theoretic, with emphasis on connections with categoricity and classification in classical model theory.

Our effective Beth-like theorem for nested relations extends two results in [6]: an ineffective result, which makes use of an idea in [16], and an effective result, but only for a restricted notion of “constructive proof”, which is not complete for classical logic.

3 PRELIMINARIES

3.1 Nested relations

We deal with schemas that describe objects of various types given by the following grammar.

\[
T, U ::= \mathbb{U} | T \times U | \text{Unit} | \text{Set}(T)
\]

For simplicity throughout the remainder we will assume only two basic types. There is the one-element type Unit, which will be used to construct Booleans. And there is $\mathbb{U}$, the “scalars” or Ur-elements whose inhabitants are not specified further. From these we build up the set of types via the product and the power set operation. We use standard conventions for abbreviating types, with the $n$-ary product abbreviating an iteration of binary products. A nested relational schema consists of declarations of variable names associated to objects of given types.

Example 3.1. An example nested relational schema declares two objects $R : \text{Set}(\mathbb{U} \times \mathbb{U})$ and $S : \text{Set}(\mathbb{U} \times \text{Set}(\mathbb{U}))$. That is, $R$ is a set of pairs of Ur-elements: a standard “flat” binary relation. $S$ is a collection of pairs whose first elements are Ur-elements and whose second elements are sets of Ur-elements.

\*\*
For the schema in Example 3.1 above, assuming that $\mathcal{U} = \mathbb{N}$, one possible instance has $R = \{(4, 6), (7, 3)\}$ and $S = \{(4, 6, 9)\}$.

### 3.2 $\Delta_0$ formulas

We need a logic appropriate for talking about nested relations. A natural and well-known subset of first-order logic formulas with a set membership relation are the $\Delta_0$ formulas. They are built up from equality of $\mathcal{U}$-elements via Boolean operators as well as relativized existential and universal quantification. All terms involving tupling and projections are allowed. Our definition is a variation of a standard notion in set theory [22].

Formally, we deal with multi-sorted first-order logic, with sorts corresponding to each of our types. We use the following syntax for $\Delta_0$ formulas and terms. Terms are built from variables using tupling and projections. All formulas and terms are assumed to be well-typed in the obvious way, with the expected sort of a nested relation, by the simple “Mostowski collapse construction”:

$$
\Delta \text{ formulas, into a model satisfying extensionality: if we have}
$$

changing the notion of entailment.

Note that there are no equalities for sorts other than $\mathcal{U}$. Atomic formulas are either of the form $t =_T u$ or $t \neq_T u$. Negation $\neg \varphi$ will be defined as a macro by induction on $\varphi$ by dualizing every connective. Other connectives can be derived in the usual way on top of negation: $\varphi \rightarrow \psi$ by $\neg \varphi \lor \psi$.

More crucial is the fact that $\Delta_0$ formulas do not allow membership atoms. An extended $\Delta_0$ formula allows also membership literals $x \in_T y, x \notin_T y$ at every type $T$.

The notion of an extended $\Delta_0$ formula $\varphi$ entailing another formula $\psi$ is the standard one in first-order logic, meaning that every model of $\varphi$ is a model of $\psi$. We emphasize here that by every model, we include models where membership is not extensional. An important but simple point, which we will use throughout, is:

**Proposition 1.** When $\varphi$ and $\psi$ are $\Delta_0$, rather than extended $\Delta_0$, then “every model” can be replaced by “every nested relation” without changing the notion of entailment.

**Proof.** The first key point is that we have neither $\epsilon$ nor equality at higher types as a primitive predicate. This guarantees that any model can be modified, without changing the truth value of $\Delta_0$ formulas, into a model satisfying extensionality: if we have $x$ and $y$ with $(\forall z \in_T x \in_T y) \land (\forall z \in_T y \in_T x)$ then $x$ and $y$ must be the same. We can just identify elements that are extensionally the same, and $\Delta_0$ formulas won’t work the difference.

Secondly, a well-typed extensional model is isomorphic to a nested relation, by the simple “Mostowski collapse construction”: we iteratively identify elements that have the same members.

When we consider formulas that are $\Delta_0$, we write $\varphi \models_{\text{nested}} \psi$ for entailment.

The lack of primitive membership and equality relations in $\Delta_0$ formulas allows us to avoid having to consider extensionality axioms, which would require special handling in our proof system.

Equality, inclusion and membership predicates “up to extensionality” may be defined as macros by induction on the involved types, while staying within $\Delta_0$ formulas.

$$
\begin{align*}
\mathcal{T}, \mathcal{U} := & \{ x \mid \emptyset \} \mid \{ E, E' \} \mid \pi_E(E) \mid \pi_T(E) \mid \\
& \{ E \} \mid \mathcal{G}(\mathcal{T}) \mid \bigcup \{ E \mid x \in E' \} \mid \{ \text{fun(nesting, binding union)} \} \\
& | \emptyset | E \cup E' | E \setminus E' \\
\end{align*}
$$

**Figure 1:** $\text{NRC syntax (typing rules omitted)}$

We will use small letters for variables in $\Delta_0$ formulas, except in examples when we sometimes use capitals to emphasize that an object is of set type. We drop the type subscripts $T$ in bounded quantifiers, primitive memberships, and macros $\equiv_T$ when clear. Of course membership-up-to-equivalence $\in_T$ and membership $\in_T$ agree on extensional models. But $\subset_T$ and $\subset$ are not interchangeable on general models, and hence are not interchangeable in $\Delta_0$ formulas. For example:

$$
\begin{align*}
x & \in_T y, x \in_T y' \models \exists z \in_T y \in_T y' \\
\text{But we do not have} & \\
x & \in_T y, x \in_T y' \nest \exists z \in_T y \in_T y'
\end{align*}
$$

A set of primitive membership expressions $t \in_T u$ (i.e. extended $\Delta_0$ formulas) will be called an $\epsilon$-context.

Let us now introduce notation for instantiating a block of bounded quantifiers at a time. A variable membership atom is a membership atom $x \in y$ where $x, y$ are variables. An ordered variable $\epsilon$-context is a list of variable membership atoms.

Given a variable membership atom $x \in y$ and a $\Delta_0$ formula of the form $\varphi_0 = \exists w \in y \varphi_1$, the specialization of $\varphi_0$ using $x \in y$ is simply $\varphi_1[x/w]$. We generalize this to specializing $\varphi_0$ using an ordered variable $\epsilon$-context $x_1 \in y_1 \ldots x_i \in y_i$ by induction on $i$: when $\varphi_0 = \exists x_1 \in y_1 \varphi_1$, we first let $\varphi_1$ be the specialization of $\varphi_0$ using $x_1 \in y_1$ and then let $\varphi'_1$ be the specialization of $\varphi_1$ using $x_2 \in y_2 \ldots x_i \in y_i$, the latter given inductively. If $\varphi_1$ or $\varphi'_1$ is not of the required form, then the specialization is not defined. A specialization of $\varphi$ with respect to a variable $\epsilon$-context is a specialization with respect to an ordering of some subset of the context. A maximal specialization (max. spec.) of $\varphi_0$ with respect to an $\epsilon$-context is a specialization which does not start with an existential. That is, no other variable membership can be applied to perform further specialization.

### 3.3 Nested Relational Calculus

We review the main language for declaratively transforming nested relations, Nested Relational Calculus (NRC). Variables occurring in expressions are typed, and each expression is associated with an output type, both of these being in the type system described above. We let Bool denote the type $\text{Set(Unit)}$. Then Bool has exactly two elements, and will be used to simulate Booleans. The grammar for NRC expressions is presented in Figure 1.
The definition of the free and bound variables of an expression is standard, the union operator $\bigcup\{E \mid x \in R\}$ binding the variable $x$. The semantics of these expressions should be fairly evident, see [41]. If $E$ has type $T$, and has input (i.e. free) variables $x_1 \ldots x_n$ of types $T_1 \ldots T_n$, respectively, then the semantics associates with $E$ a function that given a binding associating each free variable with a value of the appropriate type, returns an object of type $T$. For example, the expression $\emptyset$ always returns the empty tuple, while $\emptyset_T$ returns the empty set of type $T$.

The language NRC as originally defined cannot express certain natural transformations whose output type is $I$. To obtain a canonical language for such transformations, above we included in our NRC syntax a family of operations $\text{GET}_T : \text{Set}(T) \to T$ that extracts the unique element from a singleton. $\text{GET}$ was considered in [41]. The semantics is: if $E$ returns a singleton set $\{x\}$, then $\text{GET}_T(E)$ returns $x$; otherwise it returns some default object of the appropriate type. In [35], it is shown that $\text{GET}$ is not expressible in NRC at sort $I$. However, $\text{GET}_T$ for general $T$ is definable from $\text{GET}_I$ and the other NRC constructs.

As explained in prior work (e.g. [41]), on top of the NRC syntax above we can support richer operations as “macros”. For every type $T$ there is an NRC expression $\equiv_T$ of type $\text{Bool}$ representing equality of elements of type $T$. In particular, there is an expression $\equiv_I$ representing equality between $I$-elements. For every type $T$ there is an NRC expression $\equiv_T$ of type $\text{Bool}$ representing membership between an element of type $T$ in an element of type $\text{Set}(T)$. We can define conditional expressions, joins, projections on $k$-tuples, and $k$-tuple formers. NRC is efficiently closed under composition: given $E(x, \ldots)$ and $F(i)$ with output type matching the type of input variable $x$, we can form an expression $E(F)$ whose free variables are those of $E$ other than $x$, unioned with those of $F$.

Finally, we note that NRC is closed under $\Delta_0$ comprehension: if $E$ is in NRC and $\varphi$ is a $\Delta_0$ formula, then we can efficiently form an expression $\{z \in E \mid \varphi\}$ which returns the subset of $E$ such that $\varphi$ holds. We make use of these macros freely in our examples of NRC, such as Example 1.1.

3.4 Connections between NRC queries and $\Delta_0$ formulas

Given an NRC expression $E$ with input relations $\vec{r}$, we can create a $\Delta_0$ formula $\Sigma_E(i, o)$ that is an input-output specification of $E$: a formula such that $\Sigma_E$ implies $o = E(i)$ and whenever $o = E(i)$ holds there is a set of objects including $i$ and $o$ satisfying $\Sigma_E$. For the “composition-free” fragment – in which comprehensions $\bigcup$ can only be over input variables – this conversion can be done in PTIME. But it cannot be done efficiently for general NRC, under complexity-theoretic hypotheses [23].

We also write entailments that use NRC expressions, e.g.

$$\varphi(x, \vec{c}, \ldots) \models_{\text{nested}} x \in E(\vec{c})$$

for $\varphi$ $\Delta_0$ and $E \in$ NRC. An entailment with $\models_{\text{nested}}$ involving NRC expressions means that in every nested relation satisfying $\varphi$, $x$ is in the output of $E$ on $\vec{c}$. Note that the semantics of NRC expressions is only defined on nested relations.

4 IMPLICIT VS EXPLICIT AND THE STATEMENT OF THE MAIN RESULT

We now formalize our implicit-to-explicit result. A $\Delta_0$ formula $\varphi(\vec{i}, \vec{a}, o)$ implicitly defines variable $o$ in terms of variables $\vec{i}$ up to extensionality if we have

$$\varphi(\vec{i}, \vec{a}, o) \wedge \varphi(\vec{i}, \vec{a}', o') \models_{\text{nested}} o \equiv_{\text{T}} o'$$

Recall that $\equiv_T$ is equivalence-modulo-extensionality. It can be replaced by equality if we add extensionality axioms on the left of the entailment symbol. $\varphi$ will be called an implicit definition up to extensionality of $o$ in terms of $\vec{i}$.

An NRC expression $E$ using free variables in $\vec{i}$ explicitly defines $o$ up to extensionality relative to $\Delta_0$ formula $\varphi(\vec{i}, \vec{a}, o)$ if for every model of $\varphi$, $E$ applied to $\vec{i}$ produces $o'$ with $o' \equiv_T o$. Assuming extensionality, the conclusion is equivalent to $o' = o$.

In [6], it was shown that implicit $\Delta_0$ definitions can be converted to NRC definitions:

**Theorem 2.** $\Delta_0$ formula $\varphi(\vec{i}, \vec{a}, o)$ implicitly defines $o$ with respect to $\vec{i}$ up to extensionality if and only if there is an NRC expression $E(\vec{i})$ that explicitly defines $o$ up to extensionality relative to $\varphi$.

We explain how Theorem 2 implies Segoufin and Vianu’s [33] result for relational algebra. Suppose $\Sigma(\vec{i}, O, \ldots)$ is a single-sorted first-order logic formula over predicates that include $\vec{i} \cup (O)$, using only active domain quantification $\exists$ quantification over the union of projections of predicates $\vec{c}$ – and suppose that any two models of $\Sigma$ that agree on $\vec{i}$ agree on $O$. Such a $\Sigma$ can be considered a special kind of $\Delta_0$ formula, and the hypothesis implies that $O$ is implicitly defined by $\vec{i}$ relative to $\Sigma$. Our conclusion is that there is an NRC expression that produces $O$ from $\vec{i}$. We now use well-known results about the “conservativity” of NRC over relational algebra for set-to-set transformation [32, 40, 41]: NRC expressions transforming relations to relations can be converted to relational algebra expressions.

4.1 Proof systems for $\Delta_0$ formulas

Our main result is an effective version of Theorem 2. For this we need to formalize our proof system for $\Delta_0$ formulas, which will allow us to talk about proof witnesses for implicit definability.

If we want to talk only about effective generation of NRC witnesses from proofs, we can use a basic proof system for $\Delta_0$ formulas, whose inference rules are shown in Figure 2. The node labels are a variation of the traditional rules for first-order logic, with a couple of quirks related to the specifics of $\Delta_0$ formulas. Each node label has shape $\Theta; \Gamma \vdash \Delta$ where

- $\Theta$ is an $\in$-context. Recall that these are sets of membership atoms – the only formulas in our proof system that are extended $\Delta_0$ but not $\Delta_0$. They will emerge during proofs involving $\Delta_0$ formulas when we start breaking down bounded-quantifier formulas.
- $\Gamma$ and $\Delta$ are finite sets of $\Delta_0$ formulas.

For example, $\text{Repl}$ in the figure is a “congruence rule”, capturing that terms that are equal are interchangeable. Informally, it says that to prove conclusion $\Delta$ from a hypothesis that includes a formula $\varphi$ including variable $t$ and an equality $t \equiv_{\text{T}} u$, it suffices to add to the hypotheses a copy of $\varphi$ with $u$ replacing some occurrences of $t$. 

that the focused system is "almost level goal \( \Theta, \Gamma \vdash \varphi \land \Delta \) if it contains only atomic formulas and formulas with existential quantification as top-level connective. Consider a set of formulas \( \Delta \) such that it suffices to have the rules for the connectives only for one side, where we choose the right side.

A proof tree whose root is labelled by \( \Theta, \Gamma \vdash \Delta \) witnesses that, for any given meaning for the free variables, if all the membership relations in \( \Theta \) and all formulas in \( \Gamma \) are satisfied, then there is a formula in \( \Delta \) which is true. We say that we have a proof of a single formula \( \varphi \) when we have a proof of \( \Theta, \varphi \).

The proof system is easily seen to be sound: if \( \Theta, \Gamma \vdash \Delta \), then \( \Theta, \Gamma \vdash \Delta \), where we remind the reader that \( \vdash \) considers all models, not just extensional ones. It can be shown to be complete by a standard technique (a "Henkin construction", see Appendix C).

To generate NRC definitions efficiently from proof witnesses we will require a more restrictive proof system, in which we enforce some ordering on how proof rules can be applied, depending on the shape of the hypotheses. We refer to proofs in this system as focused proofs, the terminology being inspired by the proof search literature [29]. To this end we consider a set of formulas existential-leading (EL) if it contains only atomic formulas and formulas with existential quantification as top-level connective. Our focused proof system is shown in Figure 3. A superficial difference from Figure 2 is that the focused system is "almost 1-sided": \( \Delta \) formulas only occur on the right, with only \( \epsilon \)-contexts on the left. In particular, a top-level goal \( \Theta, \Gamma \vdash \Delta \) in the higher-level system would be expressed as \( \Theta \vdash \neg \Gamma, \Delta \) in this system. We will often abuse notation by referring to focused proofs of a 2-sided sequent \( \Theta, \Gamma \vdash \Delta \), considering them as "macros" for the corresponding 1-sided sequent. For example, the hypothesis of the \( \ast \) rule could be written in 2-sided notation as \( \Theta, \Gamma \vdash \Delta, u, \alpha[\theta/x], \alpha[\theta/x], \Delta_{EL} \) while the conclusion could be written as \( \Theta, \Gamma \vdash \Delta_{EL} \). As with \textbf{Repl} in the prior system, this rule is about duplicating a hypothesis with some occurrences of \( t \) replaced by \( u \).

A major aspect of the restriction, related to the terminology \textit{focused}, is that the \( \exists \text{-R} \) rule enforces that blocks of existentials are instantiated all at once and that the context is EL.

Soundness is evident, since it is a special case of the proof system above. Completeness is not so obvious, since we are restricting the proof rules. But we can translate proofs in the more general system of Figure 2 into a focused proof, however with an exponential blow-up: see Appendix F for details.

Furthermore, since for \( \Delta \) formulas equivalence over all structures is the same as equivalence over nested relations, a \( \Delta \) formula \( \varphi \) is provable exactly when \( \models_{\text{nested}} \varphi \).

**Example 4.1.** Let us look at how to formalize a variation of Example 1.1. The specification \( \Sigma(B, V) \) includes two conjuncts \( C_1(B, V) \) and \( C_2(B, V) \). \( C_1(B, V) \) states that every pair \( \langle k, e \rangle \) of \( V \) corresponds to a \( \langle k, S \rangle \) in \( B \) with \( e \in S \):

\[
\forall v \in V \exists b \in B. \pi_1(v) =_{H} \pi_1(b) \land \pi_2(v) \in \pi_2(b)
\]
\( C_2(B, V) \) is:
\[
\forall b \in B \forall v \in \pi_2(b) \exists v' \in V. \pi_1(b) \equiv \pi_1(b') \wedge \pi_2(b) \equiv v'
\]

Let us assume a stronger constraint, \( \Sigma_{\text{lossless}}(B) \), saying that the first component is a key and second is non-empty:
\[
\forall b \in B \forall b' \in B. \pi_1(b) \equiv \pi_1(b') \rightarrow b \equiv b' \\
\wedge \forall b \in B \exists e \in \pi_2(b). \top
\]

With \( \Sigma_{\text{lossless}} \) we can show something stronger than in Example 1.1: \( \Sigma \land \Sigma_{\text{lossless}} \) implicitly defines \( B \) in terms of \( V \). That is, the view determines the identity query, which is witnessed by a proof of \( \Sigma(B, V) \land \Sigma_{\text{lossless}}(B) \land \Sigma_{\text{lossless}}(B') \rightarrow B \equiv B' \)

Let’s prove this informally. Assuming the premise, it is sufficient to prove \( B \subseteq B' \) by symmetry. So fix \( (k, S) \in B \). By the second conjunct of \( \Sigma_{\text{lossless}}(B) \), we know there is \( e \in S \). Thus by \( C_2(B, V) \), \( V \) contains the pair \( (k, e) \). Then, by \( C_1(B', V) \), there is a \( S' \) such that \( (k, S') \in B' \). To conclude it suffices to show that \( S \equiv S' \). There are two similar directions, let us detail the inclusion \( S \subseteq S' \), so fix \( s \in S \). By \( C_2(B, V) \), we have \( (k, s) \in V \). By \( C_1(B', V) \) there exists \( S'' \) such that \( (k, S'') \in B' \) with \( s \in S'' \). But since we also have \( (k, S') \in B' \), the constraint \( \Sigma_{\text{lossless}}(B) \) implies that \( S' \equiv S'' \), so \( s \in S' \) as desired.

4.2 Main result

A derivation of \( \vdash \) in our proof system will be referred to as a witness to the implicit definability of \( o \) in terms of \( i \) up to extensionality. With these definitions, we now state formally our main result, the effective version of Theorem 2:

**Theorem 3 (Effective implicit to explicit for nested data).** Given a witness for an implicit definition of \( o \) in terms of \( i \) up to extensionality relative to \( \Delta_0 \), \( \varphi_L \), and \( \varphi_R \), one can compute an expression \( E \) such that for any \( i, a \) and \( o \), \( \varphi_L(i, a, o) \) then \( E(i) = o \). Furthermore, if the witness is focused, this can be done in polynomial time.

4.3 Application to views and queries and/or constraints

We have a consequence for rewriting queries over views. Consider a query given by an expression \( E_Q \) over inputs \( B \) and \( E_1 \ldots E_n \) over \( \hat{B} \). Let \( E_0 \) be determined by \( E_1 \ldots E_n \), if every two nested relations (finite or infinite) interpreting \( \hat{B} \) that agree on the output of each \( E_i \) agree on the output of \( E_0 \). An NRC rewriting of \( E_Q \) in terms of \( E_1 \ldots E_n \) is an expression \( R(V_1 \ldots V_n) \) such that for any nested relation \( \hat{B} \), if we evaluate each \( E_i \) on \( \hat{B} \) to obtain \( V_i \) and evaluate \( R \) on the resulting \( V_1 \ldots V_n \) we obtain \( Q(\hat{B}) \).

Given \( E_Q \) and \( E_1 \ldots E_n \), let \( \Sigma_{\text{NRC}}(E_Q) \) conjoin the input-output specifications, as defined in Section 3, for \( E_1 \ldots E_n \) and \( E_Q \). This formula has variables \( \hat{B}, V_1 \ldots V_n, Q \) along with auxiliary variables for subqueries. A proof witnessing determinacy of \( E_Q \) by \( E_1 \ldots E_n \), is a proof that \( \Sigma_{\text{NRC}} \) implicitly defines \( Q \) in terms of \( \hat{V} \).

**Corollary 4.** From a witness that a set of NRC views \( \tilde{V} \) determines an NRC query \( Q \), we can produce an NRC rewriting of \( Q \) in terms of \( \tilde{V} \). If the witness is focused, this can be done in PTIME.

Our \( \Delta_0 \) result also applies to settings where a transformation is described via a combination of logic and views, or via a logical specification alone. For example, it applies when we have integrity constraints in the form of a \( \Delta_0 \) theory, as in the key constraint of Example 1.1. The notion of determinacy of a query over views relative to a \( \Delta_0 \) theory is a straightforward generalization of the definitions above, and we can extend the corollary above:

**Corollary 5.** From a witness that a set of NRC views \( \tilde{V} \) determines an NRC query \( Q \) relative to \( \Delta_0 \) integrity constraints \( \Sigma \), we can produce an NRC rewriting of \( Q \) in terms of \( \tilde{V} \) relative to \( \Sigma \). If the witness is focused, this can be done in PTIME.

In the case where we are dealing with flat relations, the effective version is well-known: see Toman and Weddell’s [38], and the discussion in [4, 15].

We emphasize that the result involves equivalence up to extensionality, which underlines the distinction from the classical Beth theorem. If we wrote out implicit definability up to extensionality as an entailment involving two copies of the signature, we would run into problems in applying the standard proof of Beth’s theorem.

5 TOOLS FOR THE MAIN THEOREM

5.1 Interpolation

The first tool for our main theorem will be an interpolation result. Informally, such results say that if we have an entailment involving two formulas, a “left” formula \( \varphi_L \) and a “right” formula \( \varphi_R \), we can get an “explanation” for the entailment that factors through an expression only involving non-logical symbols (in our case, variables) that are common to \( \varphi_L \) and \( \varphi_R \).

**Theorem 6.** Let \( \Theta \) be an \( e \)-context and \( \Gamma, \Delta \) finite sets of \( \Delta_0 \) formulas. Then from any proof of \( \Theta; \Gamma \vdash \Delta \), we can compute in linear time a \( \Delta_0 \) formula \( \vartheta \) with \( \text{FV}(\vartheta) \subseteq \text{FV}(\Theta, \Gamma) \cap \text{FV}(\Delta) \). such that \( \Theta; \Gamma \vdash \vartheta; \vartheta \vdash \Delta \).

The \( \vartheta \) produced by the theorem is a Craig interpolant. Craig’s interpolation theorem [11] states that when \( \Gamma \vdash \Delta \) in first-order logic, such a \( \vartheta \) exists in first-order logic. Our variant states one can find \( \vartheta \) in \( \Delta_0 \) efficiently from a proof of the entailment in either of our \( \Delta_0 \) proof systems. We have stated the result for the 2-sided system. It holds also for the 1-sided focused system, where the partition of the formulas into left and right of the proof symbol is arbitrary. The argument is induction on proof length, roughly following prior interpolation algorithms [34].

5.2 Some admissible rules

As we mentioned earlier, our focused proof system is extremely low-level, and so it is convenient to have higher-level proof rules as macros. We formalize this below.

**Definition 7.** A rule with premise \( \Theta' \vdash \Delta' \) and conclusion \( \Theta \vdash \Delta \)
\[
\Theta' \vdash \Delta' \\
\Theta \vdash \Delta
\]
is (polytime) admissible in a given calculus if a proof of the conclusion \( \Theta \vdash \Delta \) in that calculus can be computed from a proof of the premise \( \Theta' \vdash \Delta' \) (in polynomial time).
Up to rewriting the sequent to be one-sided, all the rules in Figure 2 are polytime admissible in the focused calculus. Our main theorem will rely on the polytime admissibility within the focused calculus of additional rules that involve chains of existential quantifiers. To state them, we need to introduce a generalization of bounded quantification: “quantifying over subobjects of a variable”. For every type \( T \), define a set of words over the three-letter alphabet \( \{1, 2, m \} \) of subtype occurrences of \( T \) inductively as follows:

- The empty word \( \varepsilon \) is a subtype occurrence of any type
- If \( p \) is a subtype occurrence of \( T \), \( mp \) is a subtype occurrence of \( T \).
- If \( i \in \{1, 2\} \) and \( p \) is a subtype occurrence of \( T_1 \), \( ip \) is a subtype occurrence of \( T_1 \times T_2 \).

Given subtype occurrence \( p \) and quantifier symbol \( Q \in \{\forall, \exists\} \), define the notation \( Q x \in m \ t. \varphi \) by induction on \( p \):

- \( Q x \in m \ t. \varphi \) if \( Q x \in t \)
- \( Q x \in mp \ t. \varphi \) with \( y \) a fresh variable
- \( Q x \in ip \ t. \varphi \) when \( i \in \{1, 2\} \).

Now we are ready to state the results we need on admissibility in the body of the paper, referring in each case to the focused calculus. Some further routine uses are found in the appendices. The first states that if we have proven that there exists a subobject of \( o' \) equivalent to object \( r \), then we can prove that for each element \( z \) of \( r \) there is an equivalent corresponding subobject \( z' \) within \( o' \).

**Lemma 8.** Assume \( p \) is a subtype occurrence for the type of the term \( o' \). The following is polytime admissible

\[
\Theta \vdash \Delta, \exists r. \varphi. r \equiv \text{Set}(T) \quad r' \tag{\text{Lemma 8}}
\]

Furthermore, the size of the output proof is bounded by the size of the input proof.

The second rule states that we can move between an equivalence of \( r, r' \) and a universally-quantified biconditional between memberships in \( r \) and \( r' \). Because we are dealing with \( \Delta_0 \) formulas, the universal quantification has to be bound by some additional variable \( a \).

**Lemma 9.** The following is polytime admissible (where \( p \) is a subtype occurrence of the type of \( o' \))

\[
\Theta \vdash \Delta, \exists r. \varphi. r \equiv \text{Set}(T) \quad r' \tag{\text{Lemma 9}}
\]

5.3 The NRC Parameter Collection Theorem

Our last tool is a kind of interpolation result connecting \( \Delta_0 \) formulas and NRC:

**Theorem 10 (NRC Parameter Collection).** Let \( L, R \) be sets of variables with \( C = L \cap R \) and

- \( \varphi_L \) and \( \lambda(z) \) \( \Delta_0 \) formulas over \( L \)
- \( \varphi_R \) and \( \rho(z, y) \) \( \Delta_0 \) formulas over \( R \)
- \( r \) a variable of \( R \) and \( c \) a variable of \( C \).

Suppose that we have a proof of

\[
\varphi_L \land \varphi_R \rightarrow \exists y \varphi_p \forall z \in c (\lambda(z) \leftrightarrow \rho(z, y))
\]

Then one may compute in polynomial time an NRC expression \( E \) with free variables in \( C \) such that

\[
\varphi_L \land \varphi_R \rightarrow \{ z \in c \mid \lambda(z) \in E \}
\]

If \( \lambda \) was a “common formula” – one using only variables in \( C \) – then the nested relation \( \{ z \in c \mid \lambda(z) \} \) would be definable over \( C \) in NRC via \( \Delta_0 \)-comprehension. Unfortunately \( \lambda \) is a “left formula”, possibly with variables outside of \( C \). Our hypothesis is that it is equivalent to a “parameterized right formula”: a formula with variables in \( R \) and parameters that lie below them. Intuitively, this can happen only if \( \lambda \) can be rewritten to a formula \( \rho'(z, x) \) with variables of \( C \) and a distinguished \( c_0 \in C \) such that

\[
\varphi_L \land \varphi_R \rightarrow \exists x \varphi_p c_0 \forall z \in c (\lambda(z) \leftrightarrow \rho'(z, x))
\]

And if this is true, we can use an NRC expression over \( C \) to define a set that will contain the correct “parameter” value \( x \) defining \( \lambda \). From this we can define a set containing the nested relation \( \{ z \in c \mid \lambda(z) \} \).

5.4 Sketch of the proof of Theorem 10

To get the desired conclusion, we need to prove a more general statement by induction over proof trees. Besides making the obvious generalization to handle two sets of formulas instead of the particular formulas \( \varphi_L \) and \( \varphi_R \), as well as some corresponding left and right contexts, that may appear during the proof, we need to additionally generate a new formula \( \theta \) that only uses common variables, which can replace \( \varphi_R \) in the conclusion. This is captured in the following lemma:

**Lemma 11.** Let \( L, R \) be sets of variables with \( C = L \cap R \) and

- \( \Delta_L, \lambda(z) \) a set of \( \Delta_0 \) formulas over \( L \)
- \( \Delta_R, \rho(z, y) \) a set of \( \Delta_0 \) formulas over \( R \)
- \( \Theta_L \) (respectively \( \Theta_R \)) \( a \) a-context over \( L \) (respectively over \( R \))
- \( r \) a variable of \( R \) and \( c \) a variable of \( C \).

Suppose that we have a proof tree with conclusion

\[
\Theta_L, \Theta_R \vdash \Delta_L, \Delta_R, \exists y \varphi_p r \forall z \in c (\lambda(z) \leftrightarrow \rho(z, y))
\]

Then one may compute in polynomial time an NRC expression \( E \) and a \( \Delta_0 \) formula \( \theta \) using only variables from \( C \) such that

\[
\Theta_L \models \text{nested} \Delta_L, \theta \lor \{ z \in c \mid \lambda(z) \} \in E \quad \text{and} \quad \Theta_R \models \text{nested} \Delta_R, \lnot \theta
\]

The proof of the lemma is an induction on the size of the proof witnessing:

\[
\Theta_L, \Theta_R \vdash \Delta_L, \Delta_R, \exists y \varphi_p r \forall z \in c (\lambda(z) \leftrightarrow \rho(z, y))
\]

We make a case distinction according to which rule is applied last. Many of the cases use standard techniques, we focus in the body of the paper on the most novel case.

Let us write \( G \) for the formula \( \exists y \varphi_p r \forall z \in c (\lambda(z) \leftrightarrow \rho(z, y)) \) and \( \Lambda \) for the set \( \{ z \in c \mid \lambda(z) \} \). The most difficult inductive case is where the last rule applied is \( \exists \), and where \( G \) is the main formula, i.e., when the last step, where we pick some witness \( w \) for \( y \) using the \( \exists \) rule, has shape

\[
\Theta_L, \Theta_R \vdash \Delta_L, \Delta_R, \forall z \in c (\lambda(z) \leftrightarrow \rho(z, w)), G
\]

Now notice that, due to our restriction on the \( \exists \) rule, \( \Delta_L \) and \( \Delta_R \) are EL. Therefore, the only possible shape of the proof, when reasoning
Applying the inductive hypothesis, we obtain NRC expressions $E_{1}^{H}$, $E_{2}^{H}$ and formulas $θ_{1}^{H}, θ_{2}^{H}$ which contain free variables in $C \cup \{ z\}$ such that all of the following hold
\[
\Theta_{L}, z \in c \models_{nester} λ(z), Δ_{L}, θ_{1}^{H} ∨ λ \models Θ_{1}^{H}
\]
and
\[
\Theta_{L}, z \in c \models_{nester} −λ(z), Δ_{L}, θ_{2}^{H} ∨ λ \models Θ_{2}^{H}
\]
and
\[
\Theta_{R} \models_{nester} −ρ(z, w), Δ_{R}, −θ_{1}^{H}
\]
and
\[
\Theta_{R} \models_{nester} ρ(z, w), Δ_{R}, −θ_{2}^{H}
\]
With this in hand, we set
\[
θ := ∃x \in c, θ_{1}^{H} \land θ_{2}^{H}
\]
and
\[
E := \{z \in c \mid θ_{1}^{H} \land θ_{2}^{H}\} \cup \bigcup \{θ_{1}^{H} ∪ θ_{2}^{H} \mid z \in c\}
\]
Note in particular that the free variables of $E$ and $θ$ are contained in $C$, since we bind $z$. The verification that $E$ and $θ$ suffice is routine.

Notice that our construction proceeds via the following structure: (1) by induction, we get NRC expressions $E_{1}^{H}$ satisfying the required property over nested relations; (2) using our witness proof, we get a formula $θ$ satisfying some invariant over all models, and thus in particular on all nested relations; (3) we construct a construction combining $θ$ and $E_{1}^{H}$ and reason with the naive semantics of NRC to argue for correctness. We use this reasoning template in the proof of our main theorem as well.

6 PROOF OF THE MAIN RESULT

We now turn to the proof of our main result, Theorem 3.

We have as input a proof of
\[
φ(\vec{i}, \vec{a}, o) \land φ(\vec{i}, \vec{a}', o') \rightarrow o \equiv T o'
\]
and we want an NRC expression $E(\vec{i})$ such that
\[
φ(\vec{i}, \vec{a}, o) \models_{nester} E(\vec{i}) \equiv T o
\]
This will be a consequence of the following theorem.

Theorem 12. [NRC Collection Theorem] Given $Δ_{0} φ(\vec{i}, \vec{a}, o)$ and $ψ(\vec{i}, \vec{b}, o')$ together with a focused proof with conclusion
\[
(\vec{i}, \vec{a}, r); φ(\vec{i}, \vec{a}, r), ψ(\vec{i}, \vec{b}, o') ⊢ \exists r' \in p \cdot o' \equiv r \equiv T r'
\]
can we compute in PTIME an NRC expression $E(\vec{i})$ such that
\[
θ(\vec{i}, \vec{a}, r); φ(\vec{i}, \vec{a}, r), ψ(\vec{i}, \vec{b}, o') \models_{nester} r \in E(\vec{i})
\]
That is, we can find an NRC query that "collects answers". Assuming Theorem 12, let’s prove the main result.

Proof of Theorem 3. We assume $o$ has a set type, deferring the simple product and Ur-element cases (the latter using GET) to Appendix E. Fix an implicit definition of $o$ up to extensionality relative to $φ(\vec{i}, \vec{a}, o)$ and a focused proof of
\[
φ(\vec{i}, \vec{a}, o) \land φ(\vec{i}, \vec{a}', o') \rightarrow o \equiv t_{Set}(T) o'
\]
We can apply a simple variation of Lemma 8 for the "empty path" $p$ to obtain a focused derivation of
\[
r \in o; φ(\vec{i}, \vec{a}, o), φ(\vec{i}, \vec{a}', o') \vdash \exists r' \in o' \equiv T r'
\]
Then applying Theorem 12 gives a NRC expression $E(\vec{i})$ such that
\[
φ(\vec{i}, \vec{a}, o) \land r \in o \land φ(\vec{i}, \vec{a}', o') \models_{nester} r \in E(\vec{i})
\]
Thus, the object determined by $\vec{i}$ is always contained in $E(\vec{i})$. Coming back to (i), we can obtain a derivation of
\[
r \in o; φ(\vec{i}, \vec{a}, o) \vdash φ(\vec{i}, \vec{a}', o') \rightarrow \exists r' \in o' \equiv T r'
\]
and applying interpolation (Theorem 6) to that gives a $Δ_{0}$ formula $κ(\vec{i}, r)$ such that the following are valid
\[
r \in o \land φ(\vec{i}, \vec{a}, o) \rightarrow κ(\vec{i}, r)
\]
\[
κ(\vec{i}, r) \land φ(\vec{i}, \vec{a}', o') \rightarrow \exists r' \in o' \equiv T r'
\]
We claim that $E_{κ}(\vec{i}) = \{x \in E(\vec{i}) \mid κ(\vec{i}, x)\}$ is the desired NRC expression. To show this, assume $φ(\vec{i}, \vec{a}, o)$ holds. We know already that $o \subseteq E(\vec{i})$ and, by (ii), every $r \in o$ satisfies $κ(\vec{i}, o)$, so $o \subseteq E_{κ}(\vec{i})$. Conversely, if $x \in E_{κ}(\vec{i})$, we have $κ(\vec{i}, x)$, so by (iii), we have that $x \in o$, so $E_{κ}(\vec{i}) \subseteq o$. So $E_{κ}(\vec{i}) = o$, which concludes the proof. □

We now turn to the proof of Theorem 12.

6.1 Proof of Theorem 12

We prove the theorem by induction over the type $T$. We only prove the inductive step for set types: the inductive case for products is straightforward.

For $T = Ω$, the base case of the induction, it is clear that we can take for $E$ an expression computing the set of all $\Omega$-elements in the transitive closure of $\vec{i}$. This can clearly be done in NRC.

So now, we assume $T = Set(T')$ and that Theorem 12 holds up to $T'$. We have a focused derivation of
\[
θ; φ(\vec{i}, r), ψ(\vec{i}, o') \vdash \exists r' \in p \cdot o' \equiv T r'
\]
(omitting the additional variables for brevity).

From our input derivation, we can easily see that each element of $r$ must be equivalent to some element below $o'$. This is reflected by Lemma 8, which allows us to efficiently compute a proof of
\[
θ, z \in r; φ(\vec{i}, r), ψ(\vec{i}, o') \vdash \exists r' \in p \cdot o' \equiv T z'
\]
We can then apply the inductive hypothesis of our main theorem at sort $T'$, which is strictly smaller than $Set(T')$. This yields a NRC expression $E_{n}(\vec{i})$ of type $Set(T')$ such that
\[
θ, z \in r; φ(\vec{i}, r), ψ(\vec{i}, o') \models_{nester} z \in E_{n}(\vec{i})
\]
That is, our original hypotheses entail $r \subseteq E_{n}(\vec{i})$.

Thus, we have used the inductive hypothesis to get a "superset expression". But now we want an expression that has $r$ as an element. We will do this by unioning a collection of definable subsets of $E_{n}(\vec{i})$. To get these, we come back to our input derivation (iv). By Lemma 9, we can efficiently compute a derivation of
\[
θ; φ(\vec{i}, r), ψ(\vec{i}, o') \vdash \exists r' \in p \cdot o' \forall z \in a \cdot (z \in r \leftrightarrow z \in r')
\]
where we take $a$ to be a fresh variable of sort $Set(T')$. Now, applying our NRC Parameter Collection result (Theorem 10) we obtain a NRC expression $E_{collect}(\vec{i}, a)$ satisfying
\[
θ; φ(\vec{i}, r), ψ(\vec{i}, o') \models_{nester} a \cap r \in E_{collect}(\vec{i}, a)
\]
Now, recalling that we have \( r \subseteq E^{IH}(\bar{t}) \) and instantiating \( a \) to be \( E^{IH}(\bar{t}) \), we can conclude that
\[
\Theta; \varphi(\bar{t}, r), \varphi(\bar{t}, o') \models_{\text{nested}} r \in E^{coll}(\bar{t}, E^{IH}(\bar{t}))
\]
Thus we can take \( E^{coll}(\bar{t}, E^{IH}(\bar{t})) \) as an explicit definition. \( \square \)

7 DISCUSSION AND FUTURE WORK

Our effective nested Beth result implies that whenever a set of NRC views determines an NRC query, the query is rewritable over the views in NRC. Further, from a proof witnessing determinacy in our proof system, we can efficiently generate the rewriting. Our result applies to a setting where we have determinacy with respect to constraints and views, as in Example 1.1, or to general \( \Delta_0 \) implicit definitions that may not stem from views.

Towards impact for databases. To apply our results in databases, one needs a means to get proofs of determinacy/functionality. One vision is to find proofs semi-automatically, with the help of a proof assistant. An alternative is to run a theorem prover to search for a proof, and if a proof is found our algorithm can be run on the result. A final option is to isolate restrictions – e.g. class of NRC queries and views, or class of \( \Delta_0 \) formulas – where functionality is decidable. Each of these approaches has been explored in the relational setting [4, 5, 38]. For the first approach, our current system is so low-level that it may be difficult to do proofs by hand even with some support. Indeed, a formal proof of implicit definability for Example 1.1, or even the simpler Example 4.1, would come to several pages. And in terms of full automation, the proof system in this work is not supported in any theorem prover. For the final approach, we do not know of many "islands of decidability" where proofs of determinacy can be found effectively for views and queries in NRC.

To our knowledge, the problem has not been investigated.

Contrast with the situation for finite instances. The implicit-to-explicit methodology requires a proof of implicit definability, which implies implicit definability over all instances, not just finite ones. This requirement is necessary: one cannot hope to convert implicit definitions over finite instances to explicit NRC queries, even ineffectively. See Appendix A for details.

Contrast with the intuitionistic proof system of [6]. Our main result, captured by Theorem 3, is an improvement over a partial result in [6][Theorem 4.2]. The method presented there allowed one to compute NRC definitions from a specialized kind of proof of functionality. These were intuitionistic proofs, which informally means that we cannot use reasoning by case analysis and contradiction. For example, typically, statements like “either a set is empty or contains an element” are unprovable in intuitionistic logic. Unlike the focused proofs given here, the intuitionistic system in [6] is not complete for semantic entailment. That is, certain implicit definitions admit classical proofs of functionality, but no intuitionistic proof; see Appendix G for an example. Furthermore, even in cases where intuitionistic proofs of a functionality statement exist, it might be the case that they are necessarily exponentially larger than some classical proof.

On the other hand, the algorithm presented in [6] does not require the input proof to be focused, it is much simpler, and can be easily adapted to run in linear time.

Our results in the context of interpolation and definability theory. Our key proof tool was the NRC Parameter Collection theorem, Theorem 10. There is an intuition that this is a new kind of interpolation result applicable to a general setting, where we have a first-order theory \( \Sigma \) that factors into a conjunction of two formulas \( \Sigma_L \land \Sigma_R \), and from this we have a notion of a “left formula” (with predicates from \( \Sigma_L \)), a “right formula” (predicates from \( \Sigma_R \)), and a “common formula” (all predicates occur in both \( \Sigma_L \) and \( \Sigma_R \)). Under the hypothesis that a left formula \( \lambda \) is definable from a right formula with parameters, we can conclude that the left formula must actually be definable from a common formula with parameters. See the full version for a formal statement of this more classical parameter collection and the corresponding proof.

On the other hand, both our main result and the NRC Parameter Collection theorem show that NRC arises naturally in settings where we specify transformations using \( \Delta_0 \) formulas. We find this extremely surprising, and an additional piece of evidence for the fundamental role of NRC in reasoning about sets.

Our work also contributes to the broader topic of proof-theoretic vs model-theoretic techniques for interpolation and definability theorems. For Beth’s theorem, there are reasonably short model-theoretic [10, 27] and proof-theoretic arguments [12, 14]. In database terms, you can argue semantically that relational algebra is complete for rewritings of queries determined by views. Doing this effectively – producing a rewriting from a proof of determinacy – is just as simple. But for a number of results on definability proved in the 60’s and 70’s [9, 16, 25, 28], there are short model-theoretic arguments, but no proof-theoretic ones. For our NRC analog of Beth’s theorem, the situation is more similar to the latter case: the model-theoretic proof of completeness [6] is relatively short and elementary, but generating explicit definitions from proofs is much more challenging. We hope our results represent a step towards providing effective versions, and towards understanding the relationship between model-theoretic and proof-theoretic arguments.

ACKNOWLEDGMENTS

This research was funded in whole or in part by EPSRC grant EP/T022124/1. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript (AAM) version arising from this submission. Funded also by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 457292495.

REFERENCES

[1] Foto Afrati and Rada Chirkova. 2019. Answering Queries Using Views. Morgan & Claypool Publishers.
[2] John L. Bell. 2014. Intuitionistic set theory. Vol. 50. College Publications London.
[3] Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. 2019. Definability and Interpolation within Decidable Fixpoint Logics. *Log. Methods Comput. Sci.* 15, 3 (2019), 29:1–29:53.
[4] Michael Benedikt, Balder ten Cate, Julie Leblay, and Efthymia Tasmoura. 2016. Generating Plans from Proofs: The Interpolation-Based Approach to Query Reformulation. Morgan Claypool.
[5] Michael Benedikt, Egor V. Kostylev, Fabio Mogavero, and Efthymia Tasmoura. 2017. Reformulating Queries: Theory and Practice. In *IJCAI*.
[6] Michael Benedikt and Cecilia Pradic. 2021. Generating Collection Transformations from Proofs. In *POPL*.
[7] Michael Benedikt, Balder ten Cate, and Michael Vanden Boom. 2016. Effective Interpolation and Preservation in Guarded Logics. *ACM TOCL* 17, 2 (2016), 8:1–8:46.
A COMPARISON TO THE SITUATION WITH FINITE INSTANCES

Our result concerns a specification $\Sigma(\overline{\alpha}, \overline{\beta}, \ldots)$ such that $\overline{\alpha}$ implicitly defines $O$. This can be defined "syntactically" – via the existence of a proof (e.g. in our own proof system). Thus, the class of queries that we deal with could be called the "provably implicitly definable queries". The same class of queries can also be defined semantically, and this is how implicitly defined queries are often presented. But in order to be equivalent to the proof-theoretic version, we need the implicit definability of the object $O$ over $\overline{\alpha}$ to hold considering all nested relations $\overline{\alpha}, \overline{\beta}, \ldots$ not just finite ones. Of course, the fact that when you phrase the property semantically requires referencing unrestricted instances does not mean that our results depend on the existence of infinite nested relations.

Discussion of finite vs. unrestricted instances appears in many other papers (e.g. [6]). And the results in this submission do not raise any new issues with regard to the topic. But we discuss what happens if we take the obvious analog of the semantic definition, but using only finite instances. Let us say that a $\Delta_0$ specification $\Sigma(\overline{\alpha}, \overline{\beta}, \ldots)$ is implicitly defines $O$ in terms of $\overline{\alpha}$ over finite instances if for any finite nested relations $\overline{\alpha}, \overline{\beta}, \ldots$ and a query $\phi$ such that $\Sigma(\overline{\alpha}, \overline{\beta}, \ldots)$ holds, then $\phi = O$. This holds, then $\Sigma$ defines a query, and we call such a query "finely implicitly definable".

This class of queries is reasonably well understood, and we summarize what is known about it:

- Can finely implicitly definable queries always be defined in NRC? The answer is a resounding "no": one can implicitly define the powerset query over finite nested relations. Bootstrapping this, one can define iterated powersets, and show that the expressiveness of implicit definitions is the same as queries in NRC enhanced with powerset – a query language with non-elementary complexity. Even in the setting of relational queries, considering only finite instances leads to a query class that is not known to be in PTIME [24].

- Can we generate explicit definitions from specifications $\Sigma$, given a proof that $\Sigma$ implicitly defines $O$ in terms of $\overline{\alpha}$ over finite instances? It depends on what you mean by a "proof", but in some sense there is no way to make sense of the question: there is no complete proof system for such definitions. This follows from the fact that the set of finely implicitly definable queries is not computably enumerable.

- Is sticking to specifications $\Sigma$ that are implicit definitions over all inputs – as we do in this work – too strong? Here the answer can not be definitive. But we know of no evidence that this is too restrictive in practice. Implicit specifications suffice to specify any NRC query. And the answer to the first question above says that if we modified the definition in the obvious way to get a larger class, we would allow specification of queries that do not admit efficient evaluation. The answer to the second question above says that we do not have a witness to membership in this larger class.

B CAPTURING NRC EXPRESSIONS WITH $\Delta_0$ FORMULAS

In the body of the paper we mentioned that for every NRC expression $E(\overline{i})$, we can create a $\Delta_0$ expression $q_E(\overline{i}, \overline{o})$ such that $E(\overline{i}) = \alpha$ exactly when $q_E(\overline{i}, \overline{o})$ holds. These were called "input-output specifications". This conversion is needed to reason about determinacy of queries by views in our formalism. If we start with the views...
and queries in NRC, we can use this transformation to get a corresponding \(\Delta_0\) specification.

Note that \(\sigma = E\) is itself an NRC expression of Boolean type. The result then follows from the fact that every NRC expression \(E(\bar{x})\) of Boolean type can be converted to a \(\Delta_0\) formula \(\varphi(\bar{w})\). This conversion can be done in polynomial time for the "composition-free" syntax for NRC [23] mentioned briefly in the body: in composition-free NRC, we restrict \(\bigcup \{E | x \in E'\}\) so that \(E'\) must be a variable. One can normalize every expression to be of this form. The normalization is exponential, and under complexity-theoretic hypotheses one cannot do better [23]. The conversion from NRC Boolean expressions to \(\Delta_0\) is given in [6], although it is similar to results on simulating NRC with flat relations in prior work (e.g. [40]).

### C COMPLETENESS OF PROOF SYSTEMS

In the body of the paper we mentioned that the completeness of the proof systems is argued using a standard method. We outline this for the higher-level system in Figure 2.

One has a sequent \(\Theta; \Gamma \vdash \Delta\) that is not provable. We want to construct a countermodel: one that is satisfying all the formulas in \(\Theta\) and \(\Gamma\) but none of the formulas in \(\Delta\). We construct a tree with \(\Theta; \Gamma \vdash \Delta\) at the root by iteratively applying applicable inference rules in reverse: in "proof search mode", generating subgoals from goals. We apply the rules whenever possible in a given order, enforcing some fairness constraints: a rule that is active must be eventually applied along an infinite search, and if a choice of terms must be made (as with the \(3\)-R rule), all possible choices of terms are eventually made in an application of the rule. For example, if we have a disjunction \(\rho_1 \lor \rho_2\) on the right, we may immediately "apply \(\lor\)-R": we generate a subgoal where on the right hand side we add \(\rho_1, \rho_2\). Finite branches leading to a sequent that does not match the conclusion of any rule or axiom are artificially extended to infinite branches by repeating the topmost sequent.

By assumption, this process does not produce a proof, and thus we have an infinite branch \(b\) of the tree. We create a model \(M_b\) whose elements are the variables that appear on the branch, where an element inherits the type of its variable. The memberships correspond to the membership atoms that appear on the left of any sequent in \(b\), and also the atoms that appear negated on the right hand side of any sequent.

We claim that \(M_b\) is the desired countermodel. It suffices to show that for every sequent \(\Theta; \Gamma \vdash \Delta\) in \(b\), \(M_b\) is a counterexample to the sequent: it satisfies the conjunction of formulas on the left and none of the formulas on the right. We prove this by induction on the logical complexity of the formula. For atoms it is immediate by construction. Each inductive step will involve the assumptions about inference rules not terminating proof search. For example, suppose for some sequent \(b_1\) in \(b\) of the above form, \(\Delta\) contains \(\rho_1 \lor \rho_2\); we want to show that \(M_b\) satisfies \(\neg(\rho_1 \lor \rho_2)\). But we know that in some successor of \(b_1\), we would have applied \(\lor\)-R, and thus have a descendant with \(\rho_1, \rho_2\) within the right. By induction \(M_b\) satisfies \(\neg \rho_1\) and \(\neg \rho_2\). Thus \(M_b\) satisfies \(\neg(\rho_1 \lor \rho_2)\) as desired. The other connectives and quantifiers are handled similarly.

### D \(\Delta_0\) INTERPOLATION: PROOF SKETCH OF THEOREM 6

We recall the statement:

Let \(\Theta\) be an \(\varepsilon\)-context and \(\Gamma, \Delta\) finite sets of \(\Delta_0\) formulas. Then from any proof of \(\Theta; \Gamma \vdash \Delta\) we can compute in linear time a \(\Delta_0\) formula \(\theta\) with \(FV(\theta) \subseteq FV(\Theta, \Gamma) \cap FV(\Delta)\), such that \(\Theta; \Gamma \vdash \theta\) and \(\theta \vdash \Delta\)

Recall also that we claim this for both the higher-level 2-sided system and the 1-sided system, where the 2-sided syntax is a "macro": \(\Theta; \Gamma \vdash \Delta\) is a shorthand for \(\Theta \vdash \neg \Gamma, \Delta\), where \(\neg \Gamma\) is a macro for dualizing connectives. Thus in the 1-sided version, we are arbitrarily classifying some of the \(\Delta_0\) formulas as Left and Right, and our interpolant must be common according to that partition.

The construction for Theorem 6 proceeds exactly as in prior interpolation theorems for similar calculi [34, 36, 39]. For a presentation geared towards a database audience one can check [38] or the later [4].

We explain the argument for the higher-level 2-sided system. We prove a more general statement, where we partition the context and the formulas on both sides of \(\vdash\) into Left and Right. So we have

\[\Theta_L; \Theta_R; \Gamma_L \vdash \Delta_L, \Delta_R\]

And our inductive invariant is that we will compute in linear time a \(\theta\) such that:

\[\Theta_L; \Gamma_L \vdash \theta, \Delta_L\] and \[\Theta_R; \Gamma_R \vdash \theta, \Delta_R\]

And we require that \(FV(\theta) \subseteq FV(\Theta_L, \Gamma_L, \Delta_L) \cap FV(\Theta_R, \Gamma_R, \Delta_R)\). This generalization is used to handle the negation rules, as we explain below.

We proceed by induction on the depth of the proof tree.

One of the base cases is where we have a trivial proof tree, which uses rule (Ax) to derive:

\[\Theta; \Gamma, \varphi \vdash \varphi, \Delta\]

We do a case distinction on where the occurrences of \(\varphi\) sit in our partition. Assume the occurrence on the left is in \(\Gamma_L\) and the occurrence on the right is in \(\Delta_R\). Then we can take our interpolant \(\theta\) to be \(\varphi\). Suppose the occurrence on the left is in \(\Gamma_L\) and the occurrence on the right is in \(\Delta_L\). Then we can take \(\theta\) to be \(\bot\). The other base cases are similar.

The inductive cases for forming the interpolant will work "in reverse" for each proof rule. That is, if we used an inference rule to derive sequent \(S\) from sequents \(S_1\) and \(S_2\), we will partition the sequents \(L_S\) and \(S_2\) based on the partition of \(S\). We will then apply induction to our partitioned sequent for \(S_1\) to get an interpolant \(\theta_1\), and also apply induction to our partitioned version of \(S_2\) to get an interpolant \(\theta_2\). We then put them together to get the interpolant for the partitioned sequent \(S\). This "putting together" will usually reflect the semantics of the connective mentioned in the proof rule.

Consider the case where the last rule applied is the \(\neg\)-L rule: this is the case that motivates the more general invariant involving partitions. We have a partition of the final sequent \(\Theta; \Gamma, \varphi \vdash \Delta\). We form a partition of the sequent \(\Theta; \Gamma \vdash \neg \varphi, \Delta\) by placing \(\neg \varphi\) on the same side (Left, Right) as \(\varphi\) was in the original partition. We then get an interpolant \(\theta\) by induction. We just use \(\theta\) for the final interpolant.
We consider the inductive case for $\land$-$R$. We have two top sequents, one for each conjunct. We partition them in the obvious way: each $\phi_1$ in the top is in the same partition that $\phi_1 \land \phi_2$ was in the bottom. Inductively we take the interpolants $\theta_1$ and $\theta_2$ for each sequent. We again do a case analysis based on whether $\phi_1 \land \phi_2$ was in $\Delta_L$ or in $\Delta_R$.

Suppose $\phi_1 \land \phi_2$ was in $\Delta_L$, so $\Delta_L = \phi_1 \land \phi_2, \Delta'_R$. Then we arranged that each $\phi_i$ was in $\Delta_L$ in the corresponding top sequent. So we know that $\theta_L; \Gamma_1 \vdash \theta_1, \Delta_L$ and $\theta_R; \Gamma_2 \vdash \theta_1, \phi_i, \Delta'_R$ for $i = 1, 2$. Now we can set the interpolant $\theta$ to be $\theta_1 \lor \theta_2$.

In the other case, $\phi_1 \land \phi_2$ was in $\Delta_L$, say $\Delta_L = \phi_1 \land \phi_2, \Delta'_L$. Then we would arrange each $\phi_i$ to be "Left" in the corresponding top sequent, so we know that $\theta_L; \Gamma_1 \vdash \theta_i, \phi_i, \Delta'_L$ and $\theta_R; \Gamma_2 \vdash \theta_i, \Delta'_R$ for $i = 1, 2$. We set $\theta = \theta_1 \lor \theta_2$ in this case.

With the $\exists$ rule, a term in the inductively-assumed $\theta'$ for the top sequent may become illegal for the $\theta$ for the bottom sequent, since it has a free variable that is not common. In this case, the term in $\theta$ is replaced by a quantified variable, where the quantifier is existential or universal, depending on the partitioning, and bounded according to the requirements for $\Delta_0$ formulas.

### E PROOF OF THE MAIN THEOREM FOR NON-SET TYPES

Recall again our main result:

**Theorem 3 (Effective Implicit to Explicit for Nested Data).** Given a witness for an implicit definition $o$ in terms of $\iota$ up to extensionality relative to $\Delta_0 \vdash \phi(i, \bar{a}, o)$, one can compute NRC expression $E$ such that for any $i$, $\bar{a}$ and $o$, if $\phi(i, \bar{a}, o)$ then $E(i) = o$. Furthermore, if the witness is focused, this can be done in polynomial time.

In the body, we focused on the case where the type of the defined object is Set($T$) for any $T$. We now discuss the remaining cases: the base case and the inductive case for product types.

So assume we are given an implicit definition $\phi(i, \bar{a}, o)$ and a focused witness, and proceed by induction over the type $o$:

- If $o$ has type $\text{Unit}$, then, since there is only one inhabitant in $\text{Unit}$ type, then we can take our explicit definition to be the corresponding NRC expression ($\emptyset$).
- If $o$ has type $\underline{\Pi}$, then using interpolation on the entailment $\phi(i, \bar{a}, o) \Rightarrow (\phi(i, \bar{a}, o') \Rightarrow o \equiv \gamma o')$, we obtain $\theta(i, o)$ with $\phi(i, \bar{a}, o) \Rightarrow \theta(i, o)$ and $\theta(i, o) \land \phi(i, \bar{a}, o') \Rightarrow o \equiv \gamma o'$. But then we know that $\phi$ implies $o$ is a subobject of $i$: otherwise we could find a model that contradicts the entailment. There is a NRC definition $\lambda(\bar{i})$ that collects all of the $\underline{\Pi}$-elements lying beneath $\bar{i}$. We can then take $E(\bar{i}) = \text{GET}(\{x \in \lambda(\bar{i}) \mid \theta(i, x)\})$ as our NRC[get] definition of $o$. The correctness of $E$ follows from the properties of $\theta$ above.
- If $o$ has type $T_1 \times T_2$, recalling the definition of $\equiv_{T_1 \times T_2}$, we have $\phi(i, \bar{a}, o) \land \phi(i, \bar{a}, o') \Rightarrow \pi_1(o) \equiv_{T_1} \pi_1(o') \land \pi_2(o) \equiv_{T_2} \pi_2(o')$.

We will make use of a couple of simple lemmas saying that we can use certain proof rules as "macros":

#### Lemma 13. The following inference, witnessing the invertibility of the $\land$ rule, is polytime admissible for both $i \in \{1, 2\}$:

$$\begin{align*}
\Theta \vdash \phi_1 \land \phi_2, \Delta \\
\Theta \vdash \phi_i, \Delta
\end{align*}$$

#### Lemma 14. The following substitution rule is polytime admissible:

$$\Theta \vdash \Delta \\
\Theta[i/x] \vdash \Delta[i/x]$$

By Lemma 13, we have proofs of:

$$\phi(i, \bar{a}, o) \land \phi(i, \bar{a}, o') \Rightarrow \pi_1(o) \equiv_{T_1} \pi_1(o')$$

for $i \in \{1, 2\}$. Take $o_1$ and $o_2$ to be fresh variables of types $T_1$ and $T_2$. Take $\phi(i, \bar{a}, o_1, o_2)$ to be $\phi(i, \bar{a}, (o_1, o_2)).$ By substitutivity (the admissible rule given by Lemma 14) and applying the $\times$-$\beta$ rule, we have focused proofs of:

$$\phi(i, \bar{a}, (o_1, o_2')) \land \phi(i, \bar{a}, (o_1, o'_2)) \Rightarrow o_1 \equiv_{T_1} o_1'$$

We can apply our inductive hypothesis to obtain a definition $E^\bot(i)$ for both $i \in \{1, 2\}$. We can then take our explicit definition to be $(E^\bot(i), E^\bot_2(i)).$

### F FROM UNRESTRICTED TO FOCUSED

In Sect. 4 we presented two proof systems. First a high-level calculus (Fig. 2), which appeals to intuition and easily matches common techniques for analysis, for example, to prove completeness. Second a more restricted low-level calculus (Fig. 3) that addresses convenient and efficient extraction of interpolants and related formulas. We claimed that a proof in the high-level calculus can be translated into a proof of the low-level calculus, although with potentially exponential effort. Here we sketch this translation.

Recall that the essential characteristics of the target system is that the rules $\land, \exists, \times, \times_\beta$ require the context to be existential-leading (EL), which means that the context is not permitted to contain formulas with top level operator $\land, \lor, \land$, $\top$ and $\bot$. We translate the proof by transformation steps that each eliminate an occurrence of a counterexample for these requirements.

In the following, we show the particular transformation steps for some chosen cases of counterexamples. First, we consider the case where $\land$ appears as top-level operator in the context of an application of $\exists$, demonstrated here for a quantifier block with just a single variable. The proof of the conclusion then has the form shown on the left of Fig. 4 and is converted to the form shown on the right. Proof $P'$ is there like proof $P$ except that all applications of $\land$ where the principal formula is $\psi_1 \land \psi_2$ and corresponds (i.e., is passed down) to the shown occurrence of $\psi_1 \land \psi_2$ are removed and all occurrences of $\psi_1 \psi_2$ that correspond to the shown occurrence are replaced by $\psi_1$. Proof $P''$ is defined like $P'$, except that the occurrences of $\psi_1 \land \psi_2$ that correspond to the shown occurrence are replaced by $\psi_2$ instead of $\psi_1$.

As a second example we consider in Figure 5 the case where $\forall$ appears as top-level operator in the context of an application of $\exists$. Proof $P'$ is like proof $P$ except that in all sequents the $e$-context is enriched by $y \in e$, all applications of $\forall$ where the principal formula is $\forall u \in e. \psi$ and corresponds to the shown occurrence of $\forall u \in e. \psi$ are removed and all occurrences of $\forall u \in e. \psi$ that correspond to the shown occurrence are replaced by $\forall[y/u]$.

One can observe that each transformation step increases the size of the proof by a factor of at most 2. The exponential bound of the overall translation follows from this.
Figure 4: Transformation step for $\exists$ with a top-level occurrence of $\wedge$ in the context.

\[
\begin{align*}
\Theta, t \in b \vdash \varphi t/x, \exists x &\in b, \varphi \psi_1, \psi_2, \Lambda \\
\Theta, t \in b \vdash \exists x &\in b, \varphi \psi_1, \psi_2, \Delta \\
\text{is converted to} & \\
\Theta, t \in b \vdash \varphi t/x, \exists x &\in b, \varphi \psi_1, \psi_2, \Lambda \\
\Theta, t \in b \vdash \exists x &\in b, \varphi, \psi_1, \Lambda \\
\Theta, t \in b \vdash &\exists x \in b, \varphi, \psi_1, \psi_2, \Delta
\end{align*}
\]

Figure 5: Transformation step for $\exists$ with a top-level occurrence of $\forall$ in the context.

\[
\begin{align*}
\Theta, t \in b \vdash \varphi t/x, \exists x &\in b, \forall u \in c, \psi_1, \Lambda \\
\Theta, t \in b \vdash \exists x &\in b, \varphi, \forall u \in c, \psi_1, \Delta \\
\text{is converted to} & \\
\Theta, t \in b \vdash \varphi t/x, \exists x &\in b, \varphi \psi_1, \psi_2, \Lambda \\
\Theta, t \in b \vdash \exists x &\in b, \varphi, \psi_1, \psi_2, \Delta \\
\Theta, t \in b \vdash &\exists x \in b, \varphi, \psi_1, \psi_2, \Lambda
\end{align*}
\]

G CLASSICAL VS CONSTRUCTIVE PROOFS OF FUNCTIONALITY

For the purpose of this section, let us give an intuitionistic proof system close to the one in [6]. It consists of the rules of Figure 2 with the following alterations:

- we force the right-hand side to consist of a single formula in both the premises and the conclusion. In practice, this means that in most rules $\Lambda$ consists of either zero or one formula, and we disallow the rules $\exists \mathbb{R}$, $\neg \mathbb{R}$, $\neg\mathbb{L}$ and $\forall \mathbb{R}$.
- to compensate we add the following rules:

\[
\begin{align*}
\exists \mathbb{R} & \quad \Theta, t \in b; \Gamma \vdash \varphi t/x \quad \rightarrow \quad \Gamma, t \not\vdash \varphi \\
\forall \mathbb{R} & \quad \Theta, t \in b; \Gamma \vdash \exists x \in b, \varphi \quad \rightarrow \quad \Theta, \Gamma \vdash \varphi
\end{align*}
\]

In this section, we call a proof intuitionistic if it is carried out in this system. The main restriction of note here is that if we want to prove an existential statement $\exists x \in b, \varphi$ using the $\exists \mathbb{R}$ rule by introducing the witness $t \in b$, we need to commit to $t$. In contrast, the $\exists \mathbb{R}$ rule in the body of the paper allowed us to remember we wanted to prove $\exists x \in b, \varphi$ and backtrack there, offering a another witness that could involve variables that were introduced when trying to prove $\varphi t/x$. Similarly for the replacements of the $\forall \mathbb{R}$ rule, an intuitionistic proof needs to commit to a side of the disjunction being proved, while a classical proof does not need to. As a consequence, there are things provable classically but not intuitionistically. An artificial example is the following statement, that means "If $X \subseteq \{\}$, either $X = \emptyset$ or $\bullet \in X":"

$$\forall x \in X, x \in U \bullet \lor (\forall x \in X, \bot) \lor (\exists x \in X, \top)$$

Let us introduce intuitionistically unprovable statements that are more relevant to us, namely, a family of formulas $\varphi_n(X, P)$ and the statements $\varphi_n(X, P)$ defines a (possibly partial) function”.

$$\varphi_n(X, P), \varphi_n(X, P') \vdash P \equiv \text{Set} (\text{Set} (\cup)) \ P'$$

The idea behind the index $n \in \mathbb{N}$ is that the definitions will be valid when $|X| \leq n$ and $|X| = n$ respectively.

For every $n \in \mathbb{N}$, define a $\Delta_0$ formula $\varphi_n(X, P)$ where $X$ is of sort Set($\cup$) and $P$ of sort Set(Set(\cup)) as the conjunction of:

- the following formulas stating that $P$ contains all finite sets of $X$; it says that $P$ contains the empty set and that for every $Y \in P$ and $x \in X$, we have $Y \cup \{x\} \in P$.

$$\begin{align*}
&\exists \emptyset \in P. \forall x \in \emptyset. \bot \\
&\forall Y \in P. \forall x \in X. \exists Z \in P. \left( (\forall z \in Z, x \in Y \lor z = x) \land \right. x \in Z \\
&\left. \land \forall y \in Y, y \in Z \right) \\
&\bullet \text{ a formula stating that } \bigcup P \subseteq X \\
&\forall Y \in P, Y \subseteq X \\
&\bullet \text{ a formula stating that there is at most } n \text{ elements in } X \\
&\forall \forall x_0 \in X \ldots \forall x_n \in X. \sqrt[n]{x_i = x_j}
\end{align*}$$

It is straightforward to check that this defines a partial function taking sets $X$ of size $n$ to their powerset $P(X)$. Since the proof systems in the body are complete for semantic entailment, they can prove that this formula defines a functional relationship. However, this is not the case in an intuitionistic systme like the one above or the one in [6].

**Proposition 15.** There is no intuitionistic proof of functionality for $\varphi_n(X, P)$ for $n \geq 1$.

**Proof idea.** An intuitionistic proof of functionality of $\varphi_n(X, P)$ would also translate into a proof of functionality of $\overline{\varphi_n}(X, P)$ in intuitionistic set theory, where $\overline{\varphi_n}$ is the formula obtained from $\varphi_n$ by forgetting about sorts. In intuitionistic set theory, functionality of $\overline{\varphi_n}$ implies that the powerset of $\{\}$ is equal to $\{\emptyset, \{\emptyset\}\}$. This is equivalent to having the law of excluded middle [2, Proposition 2-\(\forall\)], which is unprovable in, say, IZF (see e.g., frame-valued models of IZF in [2, pp 59-71]).

\[\text{It is impossible to have one implicit definition of powerset that is valid for all sizes of inputs, since our result implies that all such functions are NRC-definable, and it is well-known that powerset is not definable in NRC.}\]