Lightweight Realms

Michael Kohlhase, Florian Rabe, and Marcel Schütz

FAU Erlangen-Nürnberg, Germany

Abstract. During formalization – e.g. of Mathematics – we have to take many decisions that informal mathematics leaves (and can leave) open. In particular, often there are multiple isomorphic ways of formalizing a set of axioms between which mathematicians can switch seamlessly. But this can impede beginners from fully understanding a domain, and it has proved difficult to mimic the same seamlessness in formalized mathematics, hindering interoperability between systems and libraries.
Realms have been proposed as an explicit representation of collections of isomorphic theories and conservative extensions, but have proven difficult to implement and manage. Therefore, here we introduce a more specialized definition that, in our experience, covers a large set of practically relevant examples. The central concept is that of a base of a theory: a subtheory that uniquely determines the entire theory. This allows us to represent an entire realm as a single theory with multiple bases. We show that many foundational concepts can be elegantly represented as such basic realms. The resulting formalism offers a good abstraction level to deal with (the consequences of) differing choices in the literature and in formal libraries, thus reducing interoperability problems, while keeping the formalizations simple.

1 Introduction

It is the very nature of formalization that it makes implicit knowledge and ideas sufficiently explicit such that they can be treated by formal methods: algorithms and interactions that only rely on the form of the representation – nothing else. During the formalization process, we have to take quite a few choices that are usually left open in informal communication of ideas. The choices that are induced by particular formal systems – we call them **foundational choices** – are relatively well-understood and are generally unavoidable. For example, a quotient set can be represented by its canonical projection, by the partition of the base set into equivalence classes, by its defining equivalence relation, or in some cases by a function that returns canonical representatives. Sometimes only some of these choices can be represented by the underlying logical system. E.g., if undefinedness is not a primitive feature, we have to represent partial functions via workarounds like option types, functional relations, or default values.

A particular motivating example from our teaching was the definition of a transition model δ as used in Turing machines or automata. If we want to make this rigorous, we have to choose among several distinct options that include:

1. δ is a relation in $(\mathcal{S} \times \mathcal{A}) \times \mathcal{S}$ with elements of the form ((current state, action), successor state) (e.g. [Sak09, chapter 1.1.1]),
2. δ is a function from $\mathcal{S} \times \mathcal{A}$ to $\mathcal{PS}$ mapping (current state, action) to (set of possible successor states) (e.g. [HMU07, chapter 2.3.2]).

While obviously isomorphic, each particular choice entails a different treatment down the line. For example, to later define deterministic transition systems, we say that δ is a partial function (case 1) or that $|\delta(s,a)| \leq 1$ for all s, a (case 2). In the latter case, an additional definition is now needed for the partial function δ' mapping (s,a) to the unique element of $\delta(s,a)$ (if any), together with a remark that δ' will – by abuse of notation – also be written as δ. These down-the-line choices can have major influence on the overall exposition in, e.g., a textbook.

In [Tao22], Terence Tao divides mathematical education into three phases:

(S1) **Pre-rigorous** stage: Mathematics is taught in an informal, intuitive manner. Here the step towards a rigorous exposition as in the choices for δ is a true "formalization step".

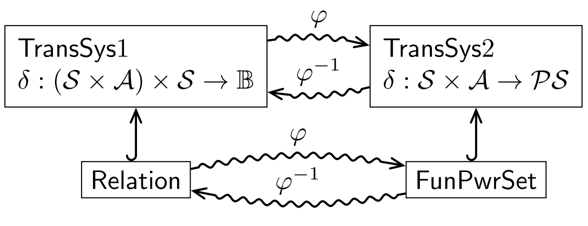
(S2) **Rigorous** stage: One thinks in a precise and rigorous manner and can deal with consequences, including comprehending, but not necessarily overlooking the equivalence of the choices.

(S3) **Post-rigorous** stage: One has grown comfortable with all the rigorous foundations and can switch between rigorous expositions at will without much cognitive effort and in fact does not need rigor to reliably assess the validity of statements.

Arguably, formal methods systems should support users/learners at all three stages of understanding: They should support beginners in achieving and appreciating rigor, give rigorous practitioners access to their accustomed expositions, and give post-rigorous experts direct access to all knowledge irrespective of the expositions.

In this situation, a good definition/implementation of **realms** can help to bundle and bridge between the different equivalent definitions. The concept of realms [CFK14; Ian17] was introduced as an extension of the theory graph paradigm where the definitions themselves are theories (systems of object declarations, axioms, and definitions) and the equivalences are expressed as theory isomorphisms.

Case 1 and 2 above would be formalized as different theories together with two isomorphisms between them. In fact, because the isomorphism is ultimated induced by the isomorphism between relations and set-valued functions, we would start with two theories Relation and FunPwrSet connected by a pair φ/φ^{-1} of theory isomorphisms. Instantiating the involved sets then



$$\begin{tikzcd}
\boxed{\begin{array}{l}\mathsf{TransSys1}\\ \delta : (\mathcal{S} \times \mathcal{A}) \times \mathcal{S} \to \mathbb{B}\end{array}} \arrow[r, squiggly, shift left, "\varphi"] & \boxed{\begin{array}{l}\mathsf{TransSys2}\\ \delta : \mathcal{S} \times \mathcal{A} \to \mathcal{PS}\end{array}} \arrow[l, squiggly, shift left, "\varphi^{-1}"] \\
\boxed{\mathsf{Relation}} \arrow[u, hook] \arrow[r, squiggly, shift left, "\varphi"] & \boxed{\mathsf{FunPwrSet}} \arrow[u, hook] \arrow[l, squiggly, shift left, "\varphi^{-1}"]
\end{tikzcd}$$

Fig. 1. The Realm of Transition Systems

yields the two realms of transition systems corresponding of two corresponding isomorphic theories. These isomorphisms encapsulate the intuition that the rigorous exposition choices do not matter mathematically. And realm-aware formalization could be a basis for "Tao stage"-independent support tools.

Our example already shows another interesting phenomenon: The mathematical equivalence between relations and set-valued functions, which sits at the foundation of mathematics, carries over to the exposition of concepts higher-up in the definitional hierarchy like transition systems. In the common situation where the reader is (post-)rigorous for the concept lower in the hierarchy but pre-rigorous for the one higher up, it is critical for support tools to be able to switch between explicit and implicit representations of realms at different levels of the hierarchy.

While the idea of realms fits very well into this situation, the definition of [CFK14] has resisted implementation. In particular, that definition distinguished the isomorphic theories themselves, conservative extensions thereof, and a "face theory" that merges all of the others. Thus, every realm is represented as a complex theory graph, an approach that turned out to be too heavyweight to easily combine with all the other design constraints on practical formal systems. In response, [RW23] identified minimal language features that allow formalizing realm-like objects. It identified, in particular, sets $T_1 \overset{\cong}{\longleftrightarrow} \dots \overset{\cong}{\longleftrightarrow} T_n$ of isomorphic theories as the single most important special case, on which to focus tool support. It argued that there are two key ways to leverage such a realm R: To create an instance of R, implementing the interface of any T_i should suffice; and when using an instance of R, the union of all T_i (i.e., the face) should be available. But it was not yet able to sketch the design of a formal system that actually allows this.

Contribution Following [RW23], we define realms as sets of isomorphic theories. But to make realms more tractable in practice, we advocate a *face-first* approach to formalizing realms where we identify the realm R with its face theory. Thus, we first formalize the union of all T_i and eliminate the resulting redundancy by relating the primitive concepts of the T_i through axioms. We introduce the concept of a *base* theory as a subtheory that uniquely determines the entire theory, and that allows recovering the various T_i as different bases of R.

We call this "lightweight realms" because the realm is formalized as a single theory, possibly with some annotations that make the various bases explicit. We show that lightweight realms permit elegant formalization of a surprisingly big class of realms, including many foundational concepts that pose difficulties to pre-rigorous readers.

Moreover, we describe multiple algorithms that leverage lightweight realms in ways that are straightforward to add to typical implementations of formal logics. One of these is the concept of realm-induced coercions: functions that embody (aspects of) the view cycles and can be added to the user-supplied underspecified/informal formulae in type-checking-driven reconstruction. Knowing and dealing with these coercions is one of the aspects of mathematical competence, where systems can adapt to the user and thus create value in computer-supported interaction with mathematical knowledge and documents.

Overview In Section 2 we set the stage by introducing a simple language in which we can make our ideas work. Sections 3 and 4 develop two alternative representations of lightweight realms. Section 5 introduces the notion of coercions and discusses some immediate applications. Section 6 concludes the paper and discusses future work.

Acknowledgments The work reported in this article was conducted as part of the VoLL-KI project (see `https://voll-ki.de`) funded by the German Research/Education Ministry under grant 16DHBKI089.

2 LR: A Simple Language For Theories and Structures

$\boxed{\text{Theory and Morphism Definitions}}$		
Thy	$::= \vartheta[a^*]\{\mathit{Decl}^*\}$	theory definition (with type parameters a)
Morph	$::= \mu : T \to T\{(c := t)^*\}$	morphism definition
Decl	$::= c : A$	symbol declaration
	$\mid\ c := t$	symbol definition
	$\mid\ \vdash t$	axiom asserting t (which must be of type $\mathbb{B}$)
$\boxed{\text{Theories}}$		
T	$::= \vartheta[A^*]$	instantiated parametric theory
$\boxed{\text{Types}}$		
A	$::= a$	type variables
	$\mid\ T$	type of structures/models of T
	$\mid\ \mathbb{B}$	Booleans
	$\mid\ A \to A$	function types
	$\mid\ A \times A$	product types
	$\mid\ \mathcal{P}A$	power types
	$\mid\ A^?$	option types
$\boxed{\text{Objects}}$		
t	$::= c \mid x$	reference to a symbol or variable
	$\mid\ T((c := t)^*)$	a structure of type T
	$\mid\ t.c$	projecting out a component of a structure
	$\mid\ \lambda x : A.t \mid t(t^*)$	function formation, application
	$\mid\ t = t \mid t \Rightarrow t \mid \forall x : A.t \mid \ldots$	logic as usual
	$\mid\ t^{\checkmark}$	statement that t is defined
	$\mid\ \ldots$	other productions as needed

Fig. 2. Syntax of LR

To make our ideas precise, we introduce the syntax of a simple language LR that we can use to formulate our abstract definitions and concrete examples of realms. The grammar is given in Fig. 2. It is meant to capture a reasonable fragment of mathematical structures while staying uncommitted on the choice of underlying formal type system and logic. Concrete languages that can be extended to realize our ideas include both fully formal languages, e.g. suitable extensions of theorem prover languages, as well as flexiformal ones like our sTeX. We do not fix a type or proof system for it, instead assuming that readers are post-rigorous for such inference systems and can easily make a reasonable choice.

Generally, we err on the side of simplicity assuming only minimal language features needed to spell our definitions and examples. While we see the general ideas as universally reusable, we expect any implementation to tweak our definitions as needed to trade-off with other design criteria. Most importantly, our choice of type system and concrete syntax should be seen as an example of a concrete language for lightweight realms rather than a requirement for them.

We assume that there are three kinds of expressions:

- Objects are the primary mathematical objects. They include the formulas and truth values as objects of type $\mathbb{B}$.
- Types occur as the classifiers of objects. We ignore the fundamental questions of whether types can occur as input or output of functions or whether they are themselves typed by higher types.
- Theories bundle a set of typed objects, definitions, and axioms into a named scope. They are related by morphisms.

We leave open if these expressions form some kind of axiomatic set theory (in which objects and types jointly form the sets) or type theory (where objects and types are separated and possibly further subdivided by kinds, universes, etc.).

A **theory** is a list of symbol declarations $c : A$, symbol definitions $c := A$, and axioms $\vdash t$ for a Boolean t. A symbol may have multiple definitions.

We make a subtle design choice here: We treat the definition of c as separate from its declaration. Alternatively, we could (i) change the grammar to $Decl ::= c : A[= t]$ to make definitions part of the declarations, or (ii) treat $c := t$ as a special case of the axiom $\vdash c = t$. Not committing to either (i) or (ii) allows **cyclic definitions** where we first declare some constants and later give them mutually recursive definitions. Note that such definition cycles are harmless if we simply think of definitions as axioms, rather than as computation rules. We call a theory **acyclic** if the relation on symbols defined by "occurs in a definition of" is acyclic.

For simplicity, we assume that theories may not introduce any type symbols and that all types needed to state the theory are provided as type parameters $[a_1, \ldots, a_n]$. Thus, references to a theory named ϑ must always be of the form $\vartheta[A_1, \ldots, A_n]$ providing values for all type parameters of ϑ. Generalizations to more complicated versions of parametric theories or to type declarations inside theories are possible, but not needed in the sequel. Given a theory $\vartheta[a_1, \ldots, a_n]\{\ldots\}$, any instantiation $T = \vartheta[A_1, \ldots, A_n]$ can be normalized into a list of declarations by substituting every a_i with A_i in the body of ϑ. In the sequel, we will assume that this normalization always takes place implicitly.

Our grammar spells out only a selection of useful **types** that are relevant in the sequel. For instantiations of our formalism with a specific language, we do not require that all of these are present, let alone be present as primitive features of the language. Moreover, we do not require that there are no other types than those. Similarly, for the **objects** t, we only introduce syntax for the fragment of mathematical expressions that we actually use in the sequel.

Of particular importance in the sequel is that every theory T (normalizing to $\{c_1 : A_1, \ldots\}$), can be used as a type. Semantically, this is the **type of structures** of shape T or of **models** of T. We can also think of every theory T as a record type, and of the concrete structures as the records. The introduction form of this type are of the structure $T(c_1 := t_1, \ldots)$ that provide a definition for each constant of T that does not have a definition yet. Given such a structure $s : T$, the elimination form $s.c_i$ projects out the respective field.

A **morphism** $\mu : S \to T$ is a list of definitions $c := t$ where each c is an S-symbol and each t is a T-object. A morphism must give exactly one definition for every S-symbol with the following exception for defined symbols: If repeated expansion of definitions allows simplifying symbol c to object t and μ defines all symbols in t, then we define $\mu(c)$ as $\mu(t)$. Thus, as usual, a morphism μ induces a homomorphic extension $\mu(-)$ that maps S-expressions to T-expressions.

The definitions in μ must be such that $\mu(-)$ preserves all type declarations, definitions, and axioms of S: If S contains $c : A$, then we require $\mu(c) : \mu(A)$; if it contains $c := t$, we require $\mu(c) = \mu(t)$ (which holds by definition if μ does not define c at all); if it contains $\vdash t$, we require that T can prove $\mu(t)$.

As usual, a morphism $\mu : S \to T$ is an **isomorphism** if there is a morphism $\nu : T \to S$ such that $\mu;\nu = id_S$ and $\nu;\mu = id_T$. Here the identity id_S maps $c = c$ and the composition $\mu;\nu$ maps $c = \nu(\mu(c))$ for every S-symbol c. And two morphisms $\mu, \mu' : S \to S'$ are equal if S' can prove $\mu(c) = \mu'(c)$ for every S-symbol c.

We call S a **subtheory** of T if every S-declaration is also a T-declaration. Note that every set C of T-symbols induces a subtheory of T, which we write $T|_C$, by taking only the symbols of C and the axioms mentioning only those symbols. A morphism $S \to T$ is called an **extension** if every one of its definitions is of the form $c = c$. If an extension exists, it is uniquely determined, and in that case we also call T an extension of S. In particular, a theory extends every one of its subtheories. Extension is a reflexive and transitive relation on theories.

3 Realms as Isomorphism Graphs

Bases The idea of a base of a theory T is that all symbols of T can be canonically defined in terms of the base symbols:

Definition 1 (Base). An acyclic subtheory B of T is a **base** for T if the inclusion $B \to T$ is an isomorphism.

Thus, we can think of the base as the minimal set of symbols that a structure must define in order to be fully determined, and of T as a conservative extension of the base. In particular, B is a base for T if T contains a definition (without slipping into cyclic definition expansions) for every symbol not declared in B. Note that the relation "is a base for" is a reflexive and transitive relation on acyclic theories.

As always, the inverse of an isomorphism is uniquely determined: Given a base B for T, let $i : T \to B$ be the inverse of the inclusion $B \to T$. We can construct i by putting $i(c) := c$ for symbols c of B, and putting for other symbols c of T that $i(c)$ is the canonical definition of c.

Bases are not unique. In fact, we can now rephrase the motivation for realms as the need to flexibly and seamlessly switch bases.

We use the word **bases** in analogy to vector spaces. If we think of a theory as a space, and its terms as the analogues of vectors, then a base theory corresponds to the base of a vector space in the sense that both pick a primitive subset from which the rest can be defined. It also carries over analogously that a vector space can be studied without choosing a base. But a lot of operations become a lot simpler if we do choose one. And there are multiple choices of base. A major difference to vector spaces, however, is that a theory may only have a select few bases.

Example 1 (Bases of Propositional Logic). Consider theory

$$\mathtt{PL}[o]\{\top : o, \bot : o, \wedge : o \times o \to o, \vee : o \times o \to o, \Rightarrow : o \times o \to o, \ldots\}$$

where we omit the axioms that govern classical propositional logic.

We also add an axiom $\forall a, b : o.(a \Rightarrow b) \wedge (b \Rightarrow a) \Rightarrow a =_o b$ to identify equivalent formulas. Then the subtheory declaring only $\top, \neg, \wedge$ (with the respective axioms) is a base for $\mathtt{PL}(o)$. Indeed, the rules for the other connectives already constrain their definitions up to provable equivalence, which makes the morphism formed from them an isomorphism.

Faces As a running example, we use the realm of a quotient on a type A:

Example 2 (Isomorphic Definitions of Quotients). We can define quotients in multiple different ways, e.g., by *i*) the equivalence relation on A, *ii*) the set of equivalence classes, or *iii*) the function that maps each element to its class. This yields the following theories:

$\mathtt{EqRel}[A]\,\{$
  $\mathit{equiv} : A \times A \to \mathbb{B}$
  $\vdash$ "*equiv* is an equivalence on A"
$\}$

$\mathtt{Partition}[A]\,\{$
  $\mathit{classes} : \mathcal{P}\mathcal{P}A$
  $\vdash$ "*classes* is a partition of A"
$\}$

$\mathtt{ClassProjection}[A]\,\{$
  $\mathit{class} : A \to \mathcal{P}A$
  $\vdash \forall a : A.\, a \in \mathit{class}(a)$
  $\vdash$ "the image of *class* is a partition of A"
$\}$

For every choice of A, we obtain an isomorphism cycle $\texttt{EqRel}[A] \xrightarrow{\mu_1} \texttt{Partition}[A] \xrightarrow{\mu_2} \texttt{ClassProjection}[A] \xrightarrow{\mu_3} \texttt{EqRel}[A]$. For example, μ_1 defines $equiv := \lambda x, y : A.\, \exists p \in classes.\, x \in p \wedge y \in p$. Moreover, we can prove that they are indeed isomorphisms, e.g., prove $\mu_1; \mu_2; \mu_3 = id_{\texttt{EqRel}[A]}$ and $\mu_2; \mu_3; \mu_1 = id_{\texttt{Partition}[A]}$ and $\mu_3; \mu_1; \mu_2 = id_{\texttt{ClassProjection}[A]}$.

This shows what we mean when we say that realms can be *heavyweight*: Already we need to maintain three theories, three isomorphisms, and three proofs of morphism equality. Even though the various base theories are often independently interesting anyway, this design can cause a relatively large amount of bureaucracy for both the user and the tool.

[CFK14] in addition assumes support tools to generate the so-called face theory that merges the above. While the existence of the face can be shown by applying co-limits, [CFK14] never spells out how a concrete co-limit can be chosen canonically (a non-trivial problem as shown in [CMR17]). A manual construction could result in the following:

Example 3 (The Face of Quotients). The theory on the right merges all three theories from Ex. 2, which can be recovered as subtheories (again, some objects remain informal for readability). The definitions in the three morphisms are included as definitions here as well (at the end). Thus, each of these three subtheories determines the other fields uniquely and is thus a base.

$\texttt{Quot}[A]\,\{$
- $equiv : A \times A \to \mathbb{B}$
- $\vdash$ "$equiv$ is an equivalence on A"
- $classes : \mathcal{P}\mathcal{P}A$
- $\vdash$ "$classes$ is a partition of A"
- $class : A \to \mathcal{P}A$
- $\vdash \forall a : A.\, a \in class(a)$
- $\vdash$ "the image of $class$ is a partition of A"
- $equiv := \lambda x, y : A.\, \exists p \in classes.\, x \in p \wedge y \in p$
- $classes := \{class\ x \mid x \in A\}$
- $class := \lambda x : A.\, \{y : A \mid equiv(x, y)\}$

$\}$

Note that $\texttt{Quot}$ critically uses cyclic definitions. This allows putting the definitions of *equiv*, *classes*, and *class* at the end of the theory. This kind of recursion is harmless if we simply think of these definitions as axioms, rather than as computation rules.

But to show that these cyclic definitions do not threaten the consistency of $\texttt{Quot}[A]$, we would like to show that repeated expansion of definitions terminates at least up to provable equality. For example, we want to show that the expansion of *equiv* eventually yields *equiv* again as in

$$\begin{aligned} equiv &\rightsquigarrow \lambda x, y : A.\, \exists p \in classes.\, x \in p \wedge y \in p \\ &\rightsquigarrow \lambda x, y : A.\, \exists p \in \{class\ x \mid x \in A\}.\, x \in p \wedge y \in p \\ &\rightsquigarrow \lambda x, y : A.\, \exists p \in \{\{y : A \mid equiv(x, y)\} \mid x \in A\}.\, x \in p \wedge y \in p \\ &= equiv. \end{aligned}$$

The resulting proof obligations are exactly the same as the ones needed to show the morphism equalities mentioned in Ex. 2.

Realms The above leads us to the following definitions:

Definition 2 (Realm). A **realm** is a connected commutative diagram of theories and morphisms, in which all edges are isomorphisms.

A **cyclic realm** is one in which the diagram is a single cycle.

Note that for any two theories S, T in a realm R, there is a unique isomorphism $R^{S,T} : S \to T$ obtained by composing appropriate edges. If there are multiple paths from S to T, the resulting morphisms are equal because the diagram commutes.

Without loss of generality, we can assume that every realm is cyclic by choosing an appropriate set of isomorphisms. If that results in the discarding of any edges of the realm, those edges must have been redundant – otherwise, the diagram would not commute.

Therefore, from now, we will assume that a realm R is given as a cycle $R_1 \xrightarrow{R^1} R_2 \xrightarrow{R^2} \dots R_{n-1} \xrightarrow{R^{n-1}} R_n \xrightarrow{R^n} R_1$. In other words, we write R^i for the isomorphism $R_i \to R_{i+1}$ with the understanding that $R_{n+1} = R_1$. In particular, the unique isomorphism $R^{i,j} : R_i \to R_j$ is given by $R^i; \dots; R^{j-1}$ (with the understanding that we loop around using $R^i; \dots R^n; R^1; \dots R^{j-1}$ if $j < i$).

Definition 3. Given a cyclic realm R and a theory T in R, the **basing** of T at R, written R_T is defined as the theory containing

- a copy of all theories in R
- for every theory S of R except T, and every undefined symbol c of S: a definition $c := \mu_{S,T}(c)$, where $\mu_{S,T} = R^{i,j}$ for $S = R^i$ and $T = R^j$.

Note that R_T is acyclic: T is assumed to be primitive and all other symbols are defined in terms of T.

Definition 4 (Face). Given a cyclic realm R, we define its **face** $\overline{R}$ as the theory containing

- a copy of each theory in R
- for every i and every undefined symbol c of R_i: a definition $c := R^i(c)$.

Note that $\overline{R}$ is a cyclic theory (except for degenerate cases like $n = 1$).

Our intuitions are confirmed by the following:

Theorem 1. *If the theories in a realm R are acyclic, then every theory T in R (as well as every R_T) is a base of $\overline{R}$.*

Proof. Clearly T is a subtheory of R_T and R_T of $\overline{R}$. By construction every constant in R_T other than those from T has a definition in terms of T. Similarly, every constant of $\overline{R}$ has a definition in terms of those from R_T.

4 Face-First Realms

The previous section showed how we can represent a realm as a set of isomorphic theories and merge those into its face. We now want to devise a language that allows for the opposite construction: extract a realm from a given face theory T. This is the practical problem we often face if we want to curate an existing body of mathematical knowledge into a realm-structured library. Of course, this is an underspecified problem: The realm consisting only of T would be a trivial but useless solution.

Therefore, we expect the user to annotate T lightly to choose the right realm structure. Recalling that bases are essentially determined by the set of symbol names, our key idea is to extend the language of LR in such a way that we can equip a theory declaration with a list of bases. For instance in the theory $\texttt{Poset}[A]$ of posets shown on the right we declare the sets $\{=, \leq\}$, $\{\neq, <\}$ and $\{\leq, <\}$ to be bases of that theory by listing them after the (in this case singleton) list of type parameters.

$$
\begin{array}{l}
\texttt{Poset}[A][\{=,\leq\},\{\neq,<\},\{\leq,<\}]\,\{\\
\quad = : A \times A \to \mathbb{B}\\
\quad \neq : A \times A \to \mathbb{B}\\
\quad < : A \times A \to \mathbb{B}\\
\quad \leq : A \times A \to \mathbb{B}\\
\quad \text{omitted: axioms about} \leq \text{and/or} <\\
\quad = := \lambda x, y : A.\, \neg x \neq y\\
\quad = := \lambda x, y : A.\, x \leq y \wedge y \leq x\\
\quad \neq := \lambda x, y : A.\, \neg x = y\\
\quad < := \lambda x, y : A.\, x \leq y \wedge x \neq y\\
\quad \leq := \lambda x, y : A.\, x = y \vee x < y\\
\}
\end{array}
$$

Given such an asserted base B, it remains to obtain the practical criteria to check that $T|_B$ is indeed a base of T. Obviously this is not guaranteed, e.g., $T|_\emptyset$ is the empty theory – B must be big enough for $T|_B$ to induce definitions for all other symbols. But it is also desirable that B be minimal with this property.

4.1 Base Checking and Realm Reconstruction

To get a sufficient criterion for an arbitrary set B of constants to form a base of a fixed theory T we proceed as follows. Let $\mathcal{G}$ be the graph with two kinds of nodes: one node of kind "constant" for each constant declared in T, and one node of kind "definiens" for each expression that occurs as the definiens of a constant declared in T. For each constant c with definiens δ in T, $\mathcal{G}$ has an edge $c \to \delta$; and for each constant d declared in T that occurs in δ, it has an edge $\delta \to d$. If there are two constants c, c' that have the same expression δ as definiens, we require $\mathcal{G}$ to contain two separate nodes v, v' for δ and two separate edges $c \to v$ and $c' \to v'$. Thus every "definiens" node has exactly one incoming edge. For the theory $\texttt{Poset}[A]$ this graph is shown in Fig. 3.

To check if B forms a base for T, we must check that

1. every constant c declared by T can be defined in terms of B, and
2. all possible such definitions of c are provably equal in T.

Since these depend on equality of objects, there is no decision procedure for this. However, if we restrict the notion of "definable" to "definable via definitional

$$\begin{tikzcd}
 & \boxed{\lambda x,y:A.\,\neg x = y} \arrow[dr] & & \boxed{\lambda x,y:A.\,x \leq y \wedge y \leq x} \arrow[dr] & & \boxed{\lambda x,y:A.\,x = y \vee x < y} \arrow[dlll] \arrow[dr] & \\
\neq \arrow[ur] & & = \arrow[ur] \arrow[dl] & & \leq \arrow[ur] & & < \arrow[dl] \\
 & \boxed{\lambda x,y:A.\,\neg x \neq y} \arrow[ul] & & & & \boxed{\lambda x,y:A.\,x \leq y \wedge x \neq y} \arrow[ul] \arrow[ulllll] &
\end{tikzcd}$$

Fig. 3. The dependency graph for constants and definientia in `Poset`$[A]$: Nodes with round borders contain the symbols of `Poset`$[A]$ and nodes with rectangle borders contain their definientia.

equality", we can carry out step 1 automatically. Step 2 can then be tackled by calling an automated theorem prover or requesting the user to act as an oracle function to check whether all possible definientia δ of any T-symbol c given in terms of B are provably equal. For details and algorithm see Appendix B (not part of the submission).

If we have a theory T with a specified set of bases $B_1, \ldots, B_n$ then we want to ensure that the choice of those bases is in some sense "reasonable". One such reasonability criterion is given by the following theorem:

Theorem 2. *Let R be a realm with theories $T_1, \ldots, T_n$ and T be the face of R equipped with a set of bases $B_1, \ldots, B_n$, where B_i is the set of symbols and axioms declared in T_i, such that for every morphism $\mu : T_i \to T_j$ in R and $c \in T_i$ there is a definition of the form $c := \mu(c)$. Then we can recover R from T by defining the morphisms in R via the expansions of the definitions of each base symbol in terms of B_i.*

Proof. In the case where a symbol c has more then one definiens in terms of B_i, any arbitrary choice of them can be taken as the value of c under the respective morphism since the commutativity condition of R ensures that they are all provably equal in T.

In the case where the theory T does not stem from a realm R, we can extend our base checking algorithm with an additional functionality that checks whether any two definientia of a symbol are provably equal. This allow us then to turn any theory T equipped with a set of bases into a realm that is compatible with T in the sense of Thm. 2.

To perform such an equality check, we track the set D of definied symbols during the base checking procedure starting with $D = \{b_1 := b_1, \ldots, b_n := b_n\}$ for each base symbol b_i. Then for each (non-base) symbol c with definiens δ whose T-symbols are all in D we do the following:

- if $c \in D$, we check if every definiens of c in D is provably equal to δ;
- if $c \notin D$, we add $c := \delta$ to D.

If eventually every equality check succeeds, the commutativity condition of the realm to be constructed is satisfied.

4.2 Unitary Bases

A key observation that led to the design of LR is that many practically important bases consist of effectively a single symbol.

Definition 5 (Unitary). A theory is **unitary** if it consists of a single symbol declaration and (possibly) some axioms. A realm is unitary if all its theories are.

Remark 1 (Type Declarations). For simplicity, LR does not allow for type declarations in theories. If an instance of LR allows for type fields, we could alternatively make A a type field in Ex. 2 and 3. All three isomorphisms would then define $A := A$.

That design choice would slightly affect the definition of unitarity: It is still true that the value of A is determined by the values of *equiv*, *classes*, resp. *class*. In fact, it is already determined by their types. But the declaration of A would have to be part of the base theories to make sure them self-contained theories. So we would need to use a slightly more technically complicated definition of unitary base.

```
Quot[$A$] {
    base $equiv : A \times A \to \mathbb{B}$
    $\vdash$ "$equiv$ is an equivalence on $A$"
    base $classes : \mathcal{P}\mathcal{P}A$
    $\vdash$ "$classes$ is a partition of $A$"
    base $class : A \to \mathcal{P}A$
    $\vdash \forall a : A.\, a \in class(a)$
    $\vdash$ "the image of $class$ is a partition of $A$"
    $equiv := \lambda x, y : A.\, \exists p \in classes.\, x \in p \wedge y \in p$
    $classes := \{class(a) \mid a \in A\}$
    $class := \lambda x : A.\, \{y \in A \mid equiv(x, y)\}$
}
```

Fig. 4. Unitary Realm for Quotients

Realms with unitary bases are extremely easy to formalize and to provide tool support for because we only need to give the face theory and annotate the base symbols.

We can finalize our formalization of quotients as shown in Fig. 4, where we annotated the base symbols directly in the body of the theory instead of providing the respective (singleton) base sets as a separate listing outside the body.

Once the proof obligation that $T|_{\{c\}}$ is indeed a base for T is discharged, it justifies a typing rule that infers $T(c := e) : T$, i.e., it is sufficient to define c to create a structure of type T. In object-oriented programming terms, we obtain a unary constructor for T that takes only the value of c as its argument.

```
QuotRep[$A$] {
    include Quot[$A$]
    base $repr : A \to A$
    $equiv := \lambda x, y : A.\, repr(x) = repr(y)$
}
```

Fig. 5. Extension of Quot[A]

We conclude with the example of quotients with representatives:

Example 4 (Extending Realms). Consider the theory in Fig. 5 where an include declaration copies over another theory. Here we use a second definition of *equiv* to connect *repr* to the base symbols of `Quot`. Now *repr* is a (unitary) base for `QuotRep`. Note that the bases of `Quot` are not bases of `QuotRep`. Thus, the annotation of a symbol of a base must be a global property of the whole theory, which may or may not be preserved when the theory is extended.

```
TransSys[$\mathcal{S}, \mathcal{A}$] {
    base $\delta_1 : \mathcal{P}((\mathcal{S} \times \mathcal{A}) \times \mathcal{S})$
    base $\delta_2 : \mathcal{S} \times \mathcal{A} \to \mathcal{P}\mathcal{S}$
    base $\delta_3 : (\mathcal{S} \times \mathcal{A}) \times \mathcal{S} \to \mathbb{B}$
    $\delta_1 := \{((s,a),s') \in (\mathcal{S} \times \mathcal{A}) \times \mathcal{S} \mid s' \in \delta_2(s,a)\}$
    $\delta_2 := \lambda(s,a) : \mathcal{S} \times \mathcal{A}.\, \{s' \in \mathcal{S} \mid \delta_3((s,a),s')\}$
    $\delta_3 := \lambda((s,a),s') : (\mathcal{S} \times \mathcal{A}) \times \mathcal{S}.\, ((s,a),s') \in \delta_1$
}
```

Fig. 6. Unitary Realm for Transition Systems

A lot of practically relevant theories can be formalized elegantly as unitary realms. Examples include: the isomorphism between relations and subset, relations, partial functions, curried functions, lists, orders, closures, groups, finite sets, lattices, etc. Appendix A (not part of the submission) gives details. In particular, we can represent the realm of transition systems from Fig. 1 as shown in Fig. 6. Note that we obtain `TransSys`$[\mathcal{S}, \mathcal{A}]$ from a realm `Rel`$[A, B]$ of relations listed in Appendix A by instantiating its type parameters A and B with $\mathcal{S} \times \mathcal{A}$ and $\mathcal{S}$, resp., which completes the representation of Fig. 1 as a unitary realm.

5 Coercion via Realms

An immediate application of face-first realms and in particular unitary bases is coercion. We do not want to spell out the details of coercion systems and only give a simple definition that conveys the general idea:

Definition 6 (Coercion). A **coercion system** is a set of a unary functions. A **coercion** from A to B is any function arising by composing these. A coercion system is **unambiguous** if all coercion functions from A to B are equal.

Now consider the common situation in implementations of formal systems, where we use a type inference algorithm that takes a user-written and not necessarily well-formed object e as well as an expected type B, and that returns the well-formed object $e' : B$ that the user is understood to have meant. Then we can define:

Definition 7 (Elaboration with Coercions). Given a sound unambiguous coercion system, the coercion rule is: If a (sub-)object e is checked against type B but is inferred to have type A, and there is a coercion $f : A \to B$, then e is understood as $f(e)$.

Now we can obtain two kinds of coercions from a realm T. Firstly, we obtain *coercions out of the realm*. For each symbol $c : A$ of T, if there is no other symbol in T with that type, we obtain a coercion function $\lambda t : T.\, t.c$. Moreover, if c is a unitary base, we also obtain a *coercion into the realm*, namely $\lambda x : A.\, T(c := x)$.

Combining these two, we obtain *coercions through the realm*: For any two unitary bases $c : A$ and $d : B$, we obtain a coercion $\lambda x : A.T(c := x).d$.

Example 5. Consider the theory $\texttt{Rel}[A, B]$ of relations mentioned above which has two unitary bases $\mathit{graph} : \mathcal{P}(A \times B)$ and $\mathit{range} : A \to \mathcal{P}B$ (cf. Appendix A).

This yields the coercions $\mathcal{P}(A \times B) \leftrightarrow (A \to \mathcal{P}B)$ given by $X : \mathcal{P}(A \times B) \mapsto \texttt{Rel}(\mathit{graph} := X).\mathit{range} = \lambda a : A.\, \{b \in B \,|\, (a, b) \in X\}$ and $f : A \to \mathcal{P}B \mapsto \texttt{Rel}(\mathit{range} := f).\mathit{graph} = \{(a, b) \in A \times B \,|\, b \in f(a)\}$.

Showing unambiguity of coercion systems with the coercions through T is tricky because the resulting coercion has type $A \to B$, which does not mention T at all. It is exactly these coercions that often trip up pre-rigorous readers, who might not be aware that T even exists or that it automatically provides the necessary coercion.

This is particularly problematic for induced coercions: It is not necessary that every coercion function is written explicitly. Often coercions at composed types can be induced from coercions at atomic types. The following cases are particularly relevant:

Definition 8 (Induced Coercions). Given a coercion $c : A \to B$, we can induce coercions

1. $c_X : (X \to A) \to (X \to B)$ given by $\lambda f : X \to A.\, \lambda x : X.\, c(f(x))$, and
2. $c_{T,s} : T \to T'$ where T is a theory containing a symbol $s : A$ and T' is the same theory but with a field $s : B$ given by $\lambda t : T.\, +T'(s := c(t.s), \rho)$ where ρ contains $a := t.a$ for every other field of T.

Example 6. Recall our formalization of the realm of transition systems shown in Fig. 6. Consider the coercion $c : \mathcal{P}((\mathcal{S} \times \mathcal{A}) \times \mathcal{S}) \to (\mathcal{S} \times \mathcal{A} \to \mathcal{P}\mathcal{S})$ given by $c(X) := \lambda(s, a) : \mathcal{S} \times \mathcal{A}.\, \{s' \in \mathcal{S} \,|\, ((s, a), s') \in X\}$ (cf. Fig. 5). Applying rule 2 from Def. 8 induces a coersion $\widetilde{c} : \texttt{TransSys}|_{\{\delta_1\}} \to (\texttt{TransSys}|_{\{\delta_1\}})'$ with

$$\begin{aligned}
\widetilde{c}(\delta_1) &= (\texttt{TransSys}|_{\{\delta_1\}})'(\delta_1 := c(\delta_1)) \\
&= \lambda(s, a) : \mathcal{S} \times \mathcal{A}.\, \{s' \in \mathcal{S} \,|\, ((s, a), s') \in \delta_1\} \\
&= \delta_2
\end{aligned}$$

which makes it explicit that (and how) we can regard a transition *relation* $\delta_1 : \mathcal{P}((\mathcal{S} \times \mathcal{A}) \times \mathcal{S})$ as being essentially the same as a (non-deterministic) transition *function* $\delta_2 : \mathcal{S} \times \mathcal{A} \to \mathcal{P}\mathcal{S}$.

Note that the unambiguity of the coercion system is only needed when machine-interpreting input from the user. It is inessential when presenting fully coerced objects to the user: As long as the applied coercions are marked, the renderer can simply elide them. In informal mathematical texts, it is in fact common that the fully coerced objects in the author's mind are rendered with coercions elided in this way. This is particularly problematic for coercions that a pre-rigorous reader is not aware of.

Using unitary realms, in an interactive document scenario, e.g., when rendering as HTML, we can however do better. Assume we have formalized our coercion $f : A \to B$ as going through a unitary realm R with bases of type A and B, we can render the object $f(a)$ as follow:

- post-rigorous reader: simply render a, not mentioning any coercion
- rigorous reader: render as a but indicate the object as coerced, e.g., with a hover that pops up R and shows that a is converted into a B under the hood
- pre-rigorous reader: render the object as (the normal form of) $f(a)$, not mentioning any coercions.

Even though Tao portrays the three stages as inherent personal development stages it should be noted that the stage may very well be domain-dependent – once the general ability of rigorous and post-rigorous thinking has been established: The (good) intuition that guide post-rigorous work are certainly domain-dependent and need to be freshly established when moving to a new domain. So in situations, where we have a learner competency modeling component (e.g. in the ALeA context) the presentation options should be mediated by competency considerations for accuracy.

6 Conclusion and Future Work

In this paper we have re-examined the notion of realms that had been proposed as a "practical" solution for problems with interoperability and the choice of primitives in formalization tasks, but proved too heavyweight to be practical after all. We identified a class of realms that can be realized with less overhead but cover many/most practical situations, especially in or near the foundations. The motor of our simplification is the concept of a base of a theory which – to the best of our knowledge – was previously unrecognized in module/theory/type-class systems in formalization.

There is however a related concept in programming: *Haskell* [Mar] provides a notion of type classes which can be annotated with the `MINIMAL` *pragma* [Tea, section 7.19.5], a compiler instruction that can be placed in the source code, which specifies minimal complete definitions of a class C, i.e. sets B of methods that must be implemented by all instances of C and from which the definitions of all other methods of C can be derived. The intended application is to minimize the effort of implementing C as the programmer must only provide definitions for the methods in one of the sets B without explicitly defining the remaining ones, which is related to, but not the same as our application in simplifying realms and thus formalization. Also: the Haskell compiler does check that a base B is complete for C. The algorithms we present above can probably be adapted.

The main feature of realms is that they allow to characterize and cluster theories as logically equivalent, which helps control and present variants and establish interoperability. In narrative contexts – e.g. math courses – logical equivalence may not mean narrative equivalence. For instance, the prime number theorem has elementary proofs and proofs involving complex analysis, which are not narratively equivalent since they have vastly different prerequisites.

References

[CFK14] Jacques Carette, William Farmer, and Michael Kohlhase. "Realms: A Structure for Consolidating Knowledge about Mathematical Theories". In: *Intelligent Computer Mathematics 2014*. Conferences on Intelligent Computer Mathematics (Coimbra, Portugal, July 7–11, 2014). Ed. by Stephan Watt et al. LNCS 8543. MKM Best-Paper-Award. Springer, 2014, pp. 252–266. ISBN: 978-3-319-08433-6. URL: `https://kwarc.info/kohlhase/papers/cicm14-realms.pdf`.

[CMR17] Mihai Codescu, Till Mossakowski, and Florian Rabe. "Canonical Selection of Colimits". In: *Recent Trends in Algebraic Development Techniques*. Ed. by Phillip James and Markus Roggenbach. Springer, 2017, pp. 170–188.

[HMU07] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 3rd ed. Pearson Education, 2007.

[Ian17] Mihnea Iancu. "Towards Flexiformal Mathematics". PhD thesis. Bremen, Germany: Jacobs University, 2017. URL: `https://opus.jacobs-university.de/frontdoor/index/index/docId/721`.

[Mar] Simon Marlow. *Haskell 2010. Language Report*. URL: `https://www.haskell.org/onlinereport/haskell2010/` (visited on 03/16/2025).

[RW23] F. Rabe and F. Weber. "Morphism Equality in Theory Graphs". In: *Intelligent Computer Mathematics*. Ed. by C. Dubois and M. Kerber. Springer, 2023, pp. 174–189.

[Sak09] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[Tao22] Terence Tao. *There's more to mathematics than rigour and proofs*. 2022. URL: `https://terrytao.wordpress.com/career-advice/theres-more-to-mathematics-than-rigour-and-proofs/`.

[Tea] The GHC Team. *The Glorious Glasgow Haskell Compilation System User's Guide, Version 7.8.20140130*. URL: `https://downloads.haskell.org/~ghc/7.8.1-rc1/docs/html/users_guide/` (visited on 03/16/2025).

A Lightweight Realm Examples

A lot of practically relevant theories can be formalized elegantly as unitary realms. For example, the isomorphism between relations and subsets is formalized as the realm:

```
Subset[$A$] {
    base $contains : A \to \mathbb{B}$
    base $extension : \mathcal{P}A$
    $contains := \lambda x : A.x \in extension$
    $extension := \{x \in A \mid contains(x)\}$
}
```

We list some more more examples of unitary realms, omitting the well-known definitions of the constants:

– Relation:

```
Rel[$A, B$] {
    base $graph : \mathcal{P}(A \times B)$
    base $in : A \times B \to \mathbb{B}$
    base $range : A \to \mathcal{P}B$
}
```

– Partial function

```
PFun[$A, B$] {
    base $apply : A \to B^?$
    base $extension$ : "set of functional relations on $A$,$B$"
}
```

– Curried function

```
Fun[$A_1, A_2, B$] {
    base $apply_1 : A_1 \to A_2 \to B$
    base $apply_2 : A_1 \times A_2 \to B$
}
```

and accordingly for additional arguments.

We also find many examples of unitary realms in many elementary theories of algebra such as the switch between strict and non-strict orders:

```
Order[$A$] {
    base $nonstrict : Rel[A, A]$
    omitted: axioms that make $nonstrict$ reflexive, symmetric, transitive
    base $strict : Rel[A, A]$
    omitted: axioms that make $strict$ irreflexive, transitive
    $nonstrict := \lambda x, y : strict(x, y) \vee x = y$
    $strict := \lambda x, y : nonstrict(x, y) \wedge \neg x = y$
}
```

Or the different ways to define a closure

$$
\begin{array}{l}
\texttt{Closure}[A]\,\{\\
\quad \textbf{base}\ \mathit{closed} : \mathcal{P}\mathcal{P}A\\
\quad \textbf{base}\ \mathit{close} : \mathcal{P}A \to \mathcal{P}A\\
\quad \textbf{base}\ \mathit{closeTo} : \mathcal{P}A \times A \to \mathbb{B}\\
\}
\end{array}
$$

(again omitting the well-known definitions as well as all axioms).

The realms of groups, with unitary bases for multiplication and division, is more complex because defining the neutral and inverse element may or may not be possible or desirable depending on the underlying logic.

$$
\begin{array}{l}
\texttt{Group}[A]\,\{\\
\quad \textbf{base}\ \mathit{mul} : A \times A \to A\\
\quad \text{omitted: axioms about } \mathit{mul}\\
\quad \textbf{base}\ \mathit{div} : A \times A \to A\\
\quad \text{omitted: axioms about } \mathit{div}\\
\quad \mathit{neut} : A\\
\quad \mathit{inv} : A \to A\\
\quad \mathit{mul} := \lambda x, y : A.\mathit{div}(x, \mathit{div}(\mathit{neut}, y))\\
\quad \mathit{div} := \lambda x, y : A.\mathit{mul}(x, \mathit{inv}(y))\\
\quad \text{omitted: definitions for } \mathit{neut} \text{ and } \mathit{inv}\\
\}
\end{array}
$$

B An Algorithm for Checking Base Completeness

Recall from Section 4.1 that tor base completeness we need to check if B forms a basis for T, we must check that

1. every constant c declared by T can be defined on from B-constants, and
2. all possible such definitions of c are provably equal in T.

Since both steps depend on the decidability of equality, there is no complete procedure for them. However, if we restrict the notion of definability to that of definability via definitional equality, step 1 can be carried out automatically by calling the function DEFINIENTIA defined in Fig. 7: It takes c as argument and returns the set of all definientia δ of c with its T-constants being recursively expanded until all constants in that expansion of δ are elements of B (if such an expansion is possible and terminates). Step 2 can then be tackled by calling an automated theorem prover or requesting the user to act as an oracle function to check whether the definientia given by DEFINIENTIA(c) are provably equal.

For instance, if we fix $B = \{<, \neq\}$ as a (potential) basis of `Poset`$[A]$ then applying DEFINIENTIA to the constant $\leq$ – or, more precisely, to the tree t consisting of a single leaf `Leaf` $(\leq)$ – proceeds as follows:

```
function DEFINIENTIA($t$)
    if all leaves of $t$ are in $B$ then
        $d \leftarrow$ expand the children of the root node of $t$ according to $t$
        return $d$
    else
        ▷ We can assume that all leaves of $t$ are constants. ◁
        ▷ Add definientia to $t$: ◁
        Replace all leaves Leaf $c$ with $c \notin B$ by Node $c$ {Leaf $d$ | $d$ is a $\mathcal{G}$-child of $c$} in $t$
        ▷ All leaves of $t$ are now either definientia or constants $c \in B$. ◁
        ▷ "Cycle pruning": Remove all definientia that yield a cyclic dependency: ◁
        Remove all leaves Leaf $d$ from $t$ with $d$ being a definiens that has a $\mathcal{G}$-child that lies in the $t$-path from the root of $t$ to Leaf $d$
        Remove all $\mathcal{G}$-nodes $d$ from $\mathcal{G}$ for which Leaf $d$ was removed from $t$ in the previous step
        ▷ "Dead-end pruning": Remove all constant $c \notin B$ that have no definiens: ◁
        for all nodes $v$ of the form Node $c\,\emptyset$ in $t$, where $c$ is a constant with $c \notin B$ do
            Remove $v$ and its parent (if it has one) from $t$
            Remove the definiens that corresponds to the parent of $v$ from $\mathcal{G}$
        end for
        Replace all leaves of the form Leaf $d$ in $t$, where $d$ is a definiens, by
        Node $d$ {Leaf $c$ | $c$ is a $\mathcal{G}$-child of $d$}
        ▷ All leaves of $t$ are constants now. ◁
        return DEFINIENTIA($t$)
    end if
end function
```

Fig. 7. Definition of the function DEFINIENTIA, depending on a fixed graph $\mathcal{G}$ of constant/definiens dependencies in a theory T and a set B of T-constants. Its initial argument is intended to be a T-constant c (represented as a tree consisting of a single leaf `Leaf` c). It tracks all definientia of c and, recursively, the T-constants they depend on until we end up with all dependencies lying in B (if possible). Those definientia for which this dependency tracking succeeds are expanded accordingly and the resulting expressions are returned.

1. $\leq \notin B$, hence we extend t to $\texttt{Node}\,(\leq)\,\{(\lambda x, y : A.\, x = y \vee x < y)\}$. The children $=$ and $<$ of $\lambda x, y : A.\, x = y \vee x < y$ do not yield a cycle, hence no cycle pruning (and thus also no dead-end pruning) is necessary, therefore we continue with the tree $t = \texttt{Node}\,(\leq)\,\{\texttt{Node}\,(\lambda x, y : A.\, x = y \vee x < y)\,\{(=), (<)\}\}$.
2. We have $< \in B$, hence we do not have to take any children of $<$ into further consideration. On the other hand, $= \notin B$, hence we have to add its definientia to t which yields $t = \texttt{Node}\,(\leq)\,\{\texttt{Node}\,(\lambda x, y : A.\, x = y \vee x < y)\,\{\texttt{Node}\,(=)\,\{(\lambda x, y : A.\, x \leq y \wedge y \leq x), (\lambda x, y : A.\, \neg x \neq y)\}, (<)\}\}$. However, since $\leq$ is a child of $\lambda x, y : A.\, x \leq y \wedge y \leq x$, we have to remove that definiens from t and hence continue with $t = \texttt{Node}\,(\leq)\,\{\texttt{Node}\,(\lambda x, y : A.\, x = y \vee x < y)\,\{\texttt{Node}\,(=)\,\{(\texttt{Node}\,(\lambda x, y : A.\, \neg x \neq y)\,\{(\neq)\})\}, (<)\}\}$.
3. Now, since all leaves of t lie in B, we have to expand the definiens $\lambda x, y : A.\, x = y \vee x < y$ of $\leq$ according to the subtree $\texttt{Node}\,(\lambda x, y : A.\, x = y \vee x < y)\,\{\texttt{Node}\,(=)\,\{(\texttt{Node}\,(\lambda x, y : A.\, \neg x \neq y)\,\{(\neq)\})\}, (<)\}$ which yields the expression $\lambda x, y : A.\, \neg x \neq y \vee x < y$ (where we silently applied β-reduction to simplify the resulting expression slightly).

Thus $\leq$ is indeed definable via $B = \{<, \neq\}$. If we, however, remove $\neq$ from B, the procedure differs crucially after step 2:

3'. Since the definiens $\lambda x, y : A.\, \neg x = y$ of $\neq$ depends on $=$, we have encountered a cycle. Thus cycle pruning leaves us with $\neq$ as a dead end and hence $\neq$ together with its parent $\lambda x, y : A.\, \neg x \neq y$ are removed from t.
4'. Consequently, $=$ is a dead end in t now. Thus $=$ together with its parent $\lambda x, y : A.\, x = y \vee x < y$ is removed (which in particular also results in removing that parent's child $<$).
5'. Now, t consists only of the leaf $\leq$ which is a dead end as well. So after removing $\leq$ we are left with the empty tree.
6'. Since all leaves of the empty tree are trivially elements of B the evaluation of DEFINIENTIA terminates at this point and returns the empty set which means that $\leq$ is not definable by $B = \{<\}$.