Orbis Specification Language: a type theory for zk-SNARK programming

Morgan Thomas

Orbis Labs
team@orbislabs.com

August 4, 2022

Abstract

Orbis Specification Language (OSL) is a language for writing statements to be proven by zk-SNARKs. zk-SNARK theories allow for proving wide classes of statements. They usually require the statement to be proven to be expressed as a constraint system, called an arithmetic circuit, which can take various forms depending on the theory. It is difficult to express complex statements in the form of arithmetic circuits. OSL is a language of statements which is similar to type theories used in proof engineering, such as Agda and Coq. OSL has a feature set which is sufficiently limited to make it feasible to compile a statement expressed in OSL to an arithmetic circuit which expresses the same statement. This work builds on Σ_1^1 arithmetization [5] in Halo 2 [3, 4], by defining a frontend for a user-friendly circuit compiler.

1 Introduction

A zk-SNARK is a zero knowledge, non-interactive argument of knowledge. In other words, a zk-SNARK is a message whose existence evidences the truth of a statement, while not revealing additional information not implied by the statement being proven. See [10] for a general introduction to zk-SNARKs.

Every zk-SNARK arises from a zk-SNARK theory. A zk-SNARK theory provides an algorithm for constructing a zk-SNARK to prove a given true statement, and an algorithm for verifying a zk-SNARK. The three key properties of this proving and verification process are (1) *soundness* (it is difficult/unlikely to make the verifier believe a false statement); (2) *completeness* (the prover can make the verifier believe a true statement); and (3) *zero-knowledge* (the proof does not reveal additional information).

zk-SNARK theories vary in the way in which a statement to be proven must be expressed to be input to the theory. Generally, the statement must be expressed as an "arithmetic circuit," but what exactly an arithmetic circuit is varies by theory.

Designing and coding arithmetic circuits manually is laborious and error prone. Therefore, Orbis Labs sought an automated method of creating arithmetic circuits. The goal of this research is to enable the relations to be arithmetized to be written in a specification language, without any extraneous implementation details, and then transform those specifications into code to build arithmetic circuits.

This research is similar in purpose to those of the zk-SNARK programming languages, such as Leo [6], Snarky [7], and Lurk [8]. However, a meaningful distinction exists between a specification and a program. In this context, a specification defines a relation. The relation is the thing that is being arithmetized, and writing a specification of the relation is a means to arithmetizing it.

Crucially, the specification should not include any extraneous content. The only way to validate the specification is to be able to look at it and know intuitively that it does specify the intended relation. Because this is the process for validating the specification, and the entire solution will be wrong if the specification is wrong, the specification must be as obvious and clear as practicable.

A program can be used to specify a relation. However, a program specifying a relation will specify not only the relation but also a means of checking whether something is in the relation. A program specifying a relation takes the form of an algorithm to check the satisfaction of that relation. Therefore, a program by nature contains details that are extraneous to the relation being specified. Thus, programming languages do not necessarily make good specification languages.

For this reason, instead of leveraging an existing zk-SNARK programming language, Orbis Labs opted to create a new specification language for expressing the Orbis rollup validity relations. This specification language must satisfy two design constraints.

1. It must allow the Orbis rollup validity relations to be fluently expressed without extraneous information.

2. A provably correct and reasonably efficient method must exist for translating the specification into a Halo 2 circuit. Here, efficiency means the efficiency of the resulting circuit, in terms of proving time and proof size. Efficiency is related to the number of columns, the polynomial degree bound, the number of rows, and other factors intrinsic to the circuit.

The solution to satisfying the latter constraint builds on research performed by Orbis Labs on the arithmetization of Σ_1^1 formulas over the language of rings. [5] That research has shown how to generate Halo 2 circuits from specifications of relations as logic formulas of a particular type, referred to in this research as Σ_1^1 formulas over the language of rings or more simply as Σ_1^1 formulas. Σ_1^1 formulas are too primitive a language to meet the fluency requirement. To bridge the gap, this section defines a high-level specification language and a translation of that specification language into Σ_1^1 formulas.

OSL is the specification language created for specifying the Orbis rollup circuits. OSL is designed with an aim not to solve all problems but to solve the one problem of specifying the Orbis rollup circuits, in a principled manner with a language of general concepts. Therefore, OSL may be useful in many more applications than specifying the Orbis rollup circuits.

OSL code is not called a program but a spec. OSL specs define types, values, and propositions, as well as functions producing types, values, and propositions. A specification of a relation consists of a definition of a function producing a proposition and the context required to define that function.

2 An example: Sudoku

Here is a simple example of an OSL spec. This spec describes the game of Sudoku. Given a circuit compiler for OSL, this spec could be used to generate zk-SNARKs proving that a given Sudoku problem has a solution, without revealing a solution.

$$\text{data Value} \cong \text{Fin}(9). \tag{1}$$

$$\text{data Row} \cong \text{Fin}(9). \tag{2}$$

$$\text{data Col} \cong \text{Fin}(9). \tag{3}$$

$$\text{data Cell} \cong \text{Row} \times \text{Col}. \tag{4}$$

$$\text{data Problem} \cong \text{Cell} \to \text{Maybe}(\text{Value}). \tag{5}$$

$$\text{data Solution} \cong \text{Cell} \to \text{Value}. \tag{6}$$

$$\text{data Square} \cong \text{Fin}(3) \times \text{Fin}(3). \tag{7}$$

$$\text{data SquareCell} \cong \text{Fin}(3) \times \text{Fin}(3). \tag{8}$$

$$\text{def three} : \mathbb{N} := 1_{\mathbb{N}} + 1_{\mathbb{N}} + 1_{\mathbb{N}} \tag{9}$$

$$\begin{aligned}
&\text{def getCell} : \text{Square} \to \text{SquareCell} \to \text{Cell} \\
&\quad := \lambda s : \text{Square} \mapsto \lambda c : \text{SquareCell} \\
&\quad \text{let } s' : \text{Fin}(3) \times \text{Fin}(3) := \text{from}(\text{Square})(s); \\
&\quad \text{let } c' : \text{Fin}(3) \times \text{Fin}(3) := \text{from}(\text{SquareCell})(c); \\
&\quad \mapsto \text{to}(\text{Cell})((\text{to}(\text{Row})(\text{cast}(\text{three} \times_{\mathbb{N}} \text{cast}(\pi_1(s')) +_{\mathbb{N}} \text{cast}(\pi_1(c')))), \\
&\qquad \text{to}(\text{Col})(\text{cast}(\text{three} \times_{\mathbb{N}} \text{cast}(\pi_2(s')) +_{\mathbb{N}} \text{cast}(\pi_2(c'))))).
\end{aligned} \tag{10}$$

$$\begin{aligned}
&\text{def solutionIsWellFormed} : \text{Solution} \to \text{Prop} \\
&\quad := \lambda s : \text{Solution} \mapsto \text{let } f : \text{Cell} \to \text{Value} := \text{from}(\text{Solution})(s); \\
&\quad (\forall r : \text{Row}, \forall v : \text{Value}, \exists c : \text{Col}, f(\text{to}(\text{Cell})((r, c))) = v) \\
&\quad \wedge\ (\forall c : \text{Col}, \forall v : \text{Value}, \exists r : \text{Row}, f(\text{to}(\text{Cell})((r, c))) = v) \\
&\quad \wedge\ (\forall r : \text{Square}, \forall v : \text{Value}, \exists c : \text{SquareCell}, f(\text{getCell}(r, c)) = v).
\end{aligned} \tag{11}$$

$$\begin{aligned}
&\text{def solutionMatchesProblem} : \text{Problem} \to \text{Solution} \to \text{Prop} \\
&\quad := \lambda p : \text{Problem} \mapsto \lambda s : \text{Solution} \\
&\quad \mapsto \text{let } f : \text{Cell} \to \text{Maybe}(\text{Value}) := \text{from}(\text{Problem})(p); \\
&\quad \text{let } g : \text{Cell} \to \text{Value} := \text{from}(\text{Solution})(s); \\
&\quad \forall c : \text{Cell}, f(c) = \text{nothing} \vee f(c) = \text{just}(g(c)).
\end{aligned} \tag{12}$$

$$\begin{aligned}
&\text{def problemIsSolvable} : \text{Problem} \to \text{Prop} \\
&\quad := \lambda p : \text{Problem} \mapsto \exists s : \text{Solution}, \text{solutionMatchesProblem}(p, s) \wedge \text{solutionIsWellFormed}(s).
\end{aligned} \tag{13}$$

3 OSL grammar

The grammar of OSL specs is defined by a sequent calculus. This syntactic definition of the language characterizes type correctness as well as what is more usually considered part of syntax in programming language theory. In specifying a programming language, the syntax is often first defined by using a context-free grammar, and then the semantic rules that determine which syntactically valid programs denote semantically valid, type-correct programs are separately defined. The definition of OSL combines these separate concerns, which are sometimes viewed as syntax and semantics, into one context-sensitive grammar defined as a sequent calculus.

For an introduction to reading and understanding sequent calculus systems, readers are referred to [9].

Three syntactic sorts exist in OSL: expressions, judgments, and sequents. An expression may, in a given context, denote a type, a value, or a proposition. A judgment expresses that an expression belongs to a type. A sequent expresses either that a context is valid or that a judgment is true in a given context.

The different expression forms are implied by the following sequent calculus definition of OSL. A judgment takes the form $\bar{x} : \bar{A}$, where $\bar{x}$ and $\bar{A}$ are metavariables denoting expressions. A context is a comma-separated list of zero or more declarations, with the different declaration forms being implied by the sequent calculus definition of OSL. An empty context is written as $()$.

Expressions may contain variable names, which are denoted in the following sequent calculus definitions by metavariables written as lowercase Latin letters: $a, b, ...$ Metavariables denoting expressions are written either as lowercase Greek letters $(\alpha, \beta, ...)$, or as lowercase Latin letters with overlines $(\bar{a}, \bar{b}, ...)$.

By convention, Greek letters are used for metavariables for expressions denoting propositions, and Latin letters with overlines are used for metavariables for expressions denoting values.

In this sequent calculus definition, Γ is a metavariable denoting an arbitrary context.

A sequent takes either the form Γ ctx (denoting that the context Γ is valid) or the form $\Gamma \vdash \bar{x} : \alpha$ (denoting that in the context Γ, the judgment $\bar{x} : \alpha$ is true).

Given an expression $\bar{x}$, free($\bar{x}$) denotes the set of variable names that occur freely in $\bar{x}$. A free occurrence of a variable name is one that is not bound by a quantifier ($\forall$ or $\exists$), a lambda abstraction (λ), or a let binding.

In OSL expressions, $\to$ associates to the right. Function application is curried, but written in the usual mathematical notation, so that $f(x)$ denotes the application of f to x, $f(x)(y)$ denotes the application of f to x and y, and $f(x, y)$ denotes the same as $f(x)(y)$.

The first rule allows for derivation with no premises that an empty context is valid:

$$\frac{}{() \text{ ctx}}. \tag{14}$$

The following rules introduce basic typing judgments, stating that Prop (the type of propositions) is a type in any context, and types are closed under formation of function types:

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \text{Prop} : \text{Type}} \tag{15}$$

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type}}{\Gamma \vdash (\alpha \to \beta) : \text{Type}}. \tag{16}$$

The following rule enables introduction of a variable of a given type into a context:

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad x \notin \text{free}(\Gamma)}{(\Gamma, x : \alpha) \text{ ctx}}. \tag{17}$$

The following rules declare the primitive scalar types. In OSL, there are three primitive scalar types: natural numbers ($\mathbb{N}$), integers ($\mathbb{Z}$), and finite sets of n distinct values (Fin(n), for $n \in \mathbb{N}$). Throughout this paper, $0 \in \mathbb{N}$.

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathbb{N} : \text{Type}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathbb{Z} : \text{Type}} \qquad \frac{\Gamma \text{ ctx} \qquad n \in \mathbb{N}}{\Gamma \vdash \text{Fin}(n) : \text{Type}} \tag{18}$$

The following rules declare the types of the related functions and constants for the scalar types.

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash +_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash \times_{\mathbb{N}} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}} \tag{19}$$

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash 0_{\mathbb{N}} : \mathbb{N}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash 1_{\mathbb{N}} : \mathbb{N}} \tag{20}$$

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash +_{\mathbb{Z}} : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash \times_{\mathbb{Z}} : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}} \tag{21}$$

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash 0_{\mathbb{Z}} : \mathbb{Z}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash 1_{\mathbb{Z}} : \mathbb{Z}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash -1_{\mathbb{Z}} : \mathbb{Z}} \tag{22}$$

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \text{cast} : \mathbb{N} \to \mathbb{Z}} \qquad \frac{\Gamma \text{ ctx} \qquad n \in \mathbb{N}}{\Gamma \vdash \text{cast} : \text{Fin}(n) \to \mathbb{N}} \qquad \frac{\Gamma \text{ ctx} \qquad n \in \mathbb{N}}{\Gamma \vdash \text{cast} : \mathbb{N} \to \text{Fin}(n)} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash \text{cast} : \mathbb{Z} \to \mathbb{N}} \tag{23}$$

Note that casting from integers to naturals, or a finite type to naturals, is a partial function, where the result is defined only if the argument is in the codomain.

$$\frac{\Gamma \text{ ctx} \qquad n, m \in \mathbb{N} \qquad m \geq n}{\Gamma \vdash \text{cast} : \text{Fin}(n) \to \text{Fin}(m)} \tag{24}$$

$$\frac{\Gamma \text{ ctx} \qquad n, m \in \mathbb{N} \qquad m < n}{\Gamma \vdash \text{fin}(m) : \text{Fin}(n)} \tag{25}$$

The following rules introduce typing judgments for (Cartesian) products, coproducts (i.e., sum types), and functions, and their related primitives.

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type}}{\Gamma \vdash \alpha \times \beta : \text{Type}} \qquad \frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type}}{\Gamma \vdash (,) : \alpha \to \beta \to \alpha \times \beta} \tag{26}$$

At times, we write a function application $(,)(x, y)$ in the abbreviated form (x, y).

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type}}{\Gamma \vdash \pi_1 : (\alpha \times \beta) \to \alpha} \qquad \frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type}}{\Gamma \vdash \pi_2 : (\alpha \times \beta) \to \beta} \tag{27}$$

$$\frac{\Gamma \vdash \bar{f} : \gamma \to \alpha \qquad \Gamma \vdash \bar{g} : \gamma \to \beta}{\bar{f} \times \bar{g} : \gamma \to (\alpha \times \beta)} \tag{28}$$

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type}}{\Gamma \vdash \alpha \oplus \beta : \text{Type}} \tag{29}$$

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type}}{\Gamma \vdash \iota_1 : \alpha \to (\alpha \oplus \beta)} \qquad \frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type}}{\Gamma \vdash \iota_2 : \beta \to (\alpha \oplus \beta)} \tag{30}$$

$$\frac{\Gamma \vdash \bar{f} : \alpha \to \gamma \qquad \Gamma \vdash \bar{g} : \beta \to \gamma \qquad \Gamma \vdash \gamma : \text{FiniteDim}}{\Gamma \vdash \bar{f} \oplus \bar{g} : (\alpha \oplus \beta) \to \gamma} \tag{31}$$

The restriction $\Gamma \vdash \gamma : \text{FiniteDim}$ is present to satisfy a technical restriction in the translation to Σ^1_1 formulas; translating this function is easier when the result is a series of first-order variables.

The following rules define which types are finite-dimensional. The judgment $\Gamma \vdash \alpha : \text{FiniteDim}$ appears to be a type judgment, but like Type itself, FiniteDim is not a type; it can appear only by itself on the right-hand side of a judgment.

$$\text{Scalar} = \{\mathbb{N}, \mathbb{Z}\} \cup \{\text{Fin}(n) \mid n \in \mathbb{N}\} \tag{32}$$

$$\frac{\Gamma \text{ ctx} \qquad \alpha \in \text{Scalar}}{\Gamma \vdash \alpha : \text{FiniteDim}} \qquad \frac{\Gamma \vdash \alpha : \text{FiniteDim}}{\Gamma \vdash \text{Maybe}(\alpha) : \text{FiniteDim}} \tag{33}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{FiniteDim}}{\Gamma \vdash \alpha \times \beta : \text{FiniteDim}} \tag{34}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{FiniteDim}}{\Gamma \vdash \alpha \oplus \beta : \text{FiniteDim}} \tag{35}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad (\text{data } x \cong \alpha) \in \Gamma}{\Gamma \vdash x : \text{FiniteDim}} \tag{36}$$

Finally, the function rules are:

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type}}{\Gamma \vdash \alpha \to \beta : \text{Type}} \tag{37}$$

$$\frac{\Gamma, x : \alpha \vdash \bar{y} : \beta}{\Gamma \vdash (\lambda x : \alpha \mapsto \bar{y}) : \alpha \to \beta} \tag{38}$$

$$\frac{\Gamma \vdash \bar{f} : \alpha \to \beta \qquad \Gamma \vdash \bar{x} : \alpha}{\Gamma \vdash \bar{f}(\bar{x}) : \beta}. \tag{39}$$

The following rules allow for the introduction of new data types that are isomorphic to existing data types (similar to newtypes in Haskell):

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad x \notin \text{free}(\Gamma)}{(\Gamma, \text{data } x \cong \alpha) \text{ ctx}} \qquad \frac{\Gamma \text{ ctx} \qquad (\text{data } x \cong \alpha) \in \Gamma}{\Gamma \vdash x : \text{Type}} \tag{40}$$

$$\frac{\Gamma \text{ ctx} \qquad (\text{data } x \cong \alpha) \in \Gamma}{\Gamma \vdash \text{to}(x) : \alpha \to x} \qquad \frac{\Gamma \text{ ctx} \qquad (\text{data } x \cong \alpha) \in \Gamma}{\Gamma \vdash \text{from}(x) : x \to \alpha}. \tag{41}$$

The following rules enable the introduction of equational definitions and their use for type synonyms:

$$\frac{\Gamma \vdash \bar{a} : \alpha \qquad x \notin \text{free}(\Gamma)}{(\Gamma, \text{def } x : \alpha := \bar{a}) \text{ ctx}} \qquad \frac{\Gamma \text{ ctx} \qquad (\text{def } x : \alpha := \bar{a}) \in \Gamma}{\Gamma \vdash x : \alpha} \tag{42}$$

$$\frac{\Gamma \vdash t : \text{Type} \qquad (\text{def } t : \beta := \alpha) \in \Gamma \qquad \Gamma \vdash x : t}{\Gamma \vdash x : \alpha} \tag{43}$$

$$\frac{\Gamma \vdash t : \text{Type} \qquad (\text{def } t : \beta := \alpha) \in \Gamma \qquad \Gamma \vdash \bar{x} : \alpha}{\Gamma \vdash \bar{x} : t}. \tag{44}$$

The following rule allows for the formation of let expressions:

$$\frac{\Gamma, \text{def } x : \alpha := \bar{y} \vdash \bar{z} : \beta}{\Gamma \vdash (\text{let } x : \alpha := \bar{y}; \bar{z}) : \beta}. \tag{45}$$

The following rules provide the typing rules for the useful functors Maybe and List, along with several basic functions for working with these functors:

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable}}{\Gamma \vdash \text{Maybe}(\alpha) : \text{Type}} \qquad \frac{\Gamma \vdash \bar{f} : \alpha \to \beta}{\Gamma \vdash \text{Maybe}(\bar{f}) : \text{Maybe}(\alpha) \to \text{Maybe}(\beta)} \tag{46}$$

$$\frac{\Gamma \vdash \alpha : \text{Type}}{\Gamma \vdash \text{just} : \alpha \to \text{Maybe}(\alpha)} \qquad \frac{\Gamma \vdash \alpha : \text{Type}}{\Gamma \vdash \text{nothing} : \text{Maybe}(\alpha)} \tag{47}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{FiniteDim} \qquad \Gamma \vdash \bar{f} : \alpha \to \beta}{\Gamma \vdash \text{maybe}(\bar{f}) : \beta \to \text{Maybe}(\alpha) \to \beta}. \tag{48}$$

The following function is useful for extracting a value from a Maybe when one is known to be present (with undefined result if no value is present):

$$\frac{\Gamma \vdash \alpha : \text{Type}}{\Gamma \vdash \text{exists} : \text{Maybe}(\alpha) \to \alpha}. \tag{49}$$

The rules for List have a circular dependency with the rules for Quantifiable judgments, defined in (87)-(93).

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable}}{\Gamma \vdash \text{List}(\alpha) : \text{Type}} \tag{50}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable}}{\Gamma \vdash \text{length} : \text{List}(\alpha) \to \mathbb{N}} \qquad \frac{\Gamma \vdash \alpha : \text{Type}}{\Gamma \vdash \text{nth} : \text{List}(\alpha) \to \mathbb{N} \to \alpha} \tag{51}$$

The result of nth when the index is out of range is undefined.

Because translating the general functor operation of List on morphisms (functions) is difficult, OSL does not include the general functor operation of List on morphisms. However, OSL does include several special cases of the List functor that are useful and easy to translate:

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \text{List}(\pi_1) : \text{List}(\alpha \times \beta) \to \text{List}(\alpha)} \tag{52}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \text{List}(\pi_2) : \text{List}(\alpha \times \beta) \to \text{List}(\beta)} \tag{53}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable} \qquad (\text{data } x \ \cong \alpha) \in \Gamma}{\Gamma \vdash \text{List}(\text{to}(x)) : \text{List}(\alpha) \to \text{List}(x)} \tag{54}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable} \qquad (\text{data } x \cong \alpha) \in \Gamma}{\Gamma \vdash \text{List}(\text{from}(x)) : \text{List}(x) \to \text{List}(\alpha)} \tag{55}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable}}{\Gamma \vdash \text{List}(\text{length}) : \text{List}(\text{List}(\alpha)) \to \text{List}(\mathbb{N})} \tag{56}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \text{List}(\text{Maybe}(\pi_1)) : \text{List}(\text{Maybe}(\alpha \times \beta)) \to \text{List}(\text{Maybe}(\alpha))} \tag{57}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \text{List}(\text{Maybe}(\pi_2)) : \text{List}(\text{Maybe}(\alpha \times \beta)) \to \text{List}(\text{Maybe}(\beta))} \tag{58}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable}}{\Gamma \vdash \text{List}(\text{Maybe}(\text{length})) : \text{List}(\text{Maybe}(\text{List}(\alpha))) \to \text{List}(\text{Maybe}(\mathbb{N}))}. \tag{59}$$

Lists of numeric types support a sum operation. The following rules define what a numeric type is:

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathbb{N} : \text{Num}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathbb{Z} : \text{Num}} \tag{60}$$

$$\frac{\Gamma \vdash \alpha : \text{Num} \qquad (\text{data } a \cong \alpha) \in \Gamma}{\Gamma \vdash a : \text{Num}}. \tag{61}$$

The following rules define the types of the sum operation:

$$\frac{\Gamma \vdash \alpha : \text{Num}}{\Gamma \vdash \text{sum} : \text{List}(\alpha) \to \alpha} \qquad \frac{\Gamma \vdash \alpha : \text{Num}}{\Gamma \vdash \text{sum} : \text{List}(\text{Maybe}(\alpha)) \to \alpha} \tag{62}$$

$$\frac{\Gamma \vdash \alpha : \text{Num}}{\Gamma \vdash \text{sum} : \text{List}(\text{List}(\alpha)) \to \alpha} \qquad \frac{\Gamma \vdash \alpha : \text{Num}}{\Gamma \vdash \text{sum} : \text{List}(\text{List}(\text{Maybe}(\alpha))) \to \alpha}. \tag{63}$$

The following are the map-related rules:

$$\frac{\Gamma \text{ ctx} \qquad \Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \text{Map}(\alpha, \beta) : \text{Type}} \tag{64}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \text{lookup} : \alpha \to \text{Map}(\alpha, \beta) \to \text{Maybe}(\beta)} \tag{65}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \text{keys} : \text{Map}(\alpha, \beta) \to \text{List}(\alpha)} \tag{66}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable} \qquad \Gamma \vdash \gamma : \text{Quantifiable}}{\Gamma \vdash \text{Map}(\pi_1) : \text{Map}(\alpha, \beta \times \gamma) \to \text{Map}(\alpha, \beta)} \tag{67}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable} \qquad \Gamma \vdash \gamma : \text{Quantifiable}}{\Gamma \vdash \text{Map}(\pi_2) : \text{Map}(\alpha, \beta \times \gamma) \to \text{Map}(\alpha, \gamma)} \tag{68}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable} \qquad (\text{data } x \cong \beta) \in \Gamma}{\Gamma \vdash \text{Map}(\text{to}(x)) : \text{Map}(\alpha, \beta) \to \text{Map}(\alpha, x)} \tag{69}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable} \qquad (\text{data } x \cong \beta) \in \Gamma}{\Gamma \vdash \text{Map}(\text{from}(x)) : \text{Map}(\alpha, \beta) \to \text{Map}(\alpha, x)} \tag{70}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Num}}{\Gamma \vdash \text{sum} : \text{Map}(\alpha, \beta) \to \beta}. \tag{71}$$

The following function compositions are defined as primitives, because they are useful and easy to translate into Σ^1_1 formulas:

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash (\text{sum} \circ \text{Map}(\text{length})) : \text{Map}(\alpha, \text{List}(\beta)) \to \mathbb{N}} \tag{72}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Num} \qquad \Gamma \vdash \bar{k} : \alpha}{\Gamma \vdash (\text{sum} \circ \text{List}(\text{lookup}(\bar{k}))) : \text{List}(\text{Map}(\alpha, \beta)) \to \beta}. \tag{73}$$

The following rule defines the types for which equality propositions can be formed in OSL:

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim}}{\Gamma \vdash \alpha : \text{Eq}}. \tag{74}$$

Of note, although Eq appears to be a type, it is not a type. It cannot appear except by itself on the right-hand side of a judgment.

The following rules pertain to the formation of propositions:

$$\frac{\Gamma \vdash \bar{x} : \alpha \qquad \Gamma \vdash \bar{y} : \alpha \qquad \Gamma \vdash \alpha : \text{Eq}}{\Gamma \vdash (\bar{x} = \bar{y}) : \text{Prop}} \tag{75}$$

$$\frac{\Gamma \vdash \bar{x} : \alpha \qquad \Gamma \vdash \bar{y} : \alpha \qquad \Gamma \vdash \alpha : \text{Num}}{\Gamma \vdash (\bar{x} \leq \bar{y}) : \text{Prop}} \tag{76}$$

$$\frac{\Gamma \vdash \phi : \text{Prop} \qquad \Gamma \vdash \psi : \text{Prop}}{\Gamma \vdash (\phi \wedge \psi) : \text{Prop}} \qquad \frac{\Gamma \vdash \phi : \text{Prop} \qquad \Gamma \vdash \psi : \text{Prop}}{\Gamma \vdash (\phi \vee \psi) : \text{Prop}} \tag{77}$$

$$\frac{\Gamma \vdash \phi : \text{Prop}}{\Gamma \vdash \neg\phi : \text{Prop}} \qquad \frac{\Gamma \vdash \phi : \text{Prop} \qquad \Gamma \vdash \psi : \text{Prop}}{\Gamma \vdash (\phi \to \psi) : \text{Prop}}. \tag{78}$$

Because OSL allows for only universal quantification over finite types, the following rules define what a finite type is:

$$\frac{\Gamma \text{ ctx} \qquad n \in \mathbb{N}}{\Gamma \vdash \text{Fin}(n) : \text{Finite}} \qquad \frac{\Gamma \vdash \alpha : \text{Finite}}{\Gamma \vdash \text{Maybe}(\alpha) : \text{Finite}} \tag{79}$$

$$\frac{\Gamma \vdash \alpha : \text{Finite} \qquad \Gamma \vdash \beta : \text{Finite}}{\Gamma \vdash \alpha \times \beta : \text{Finite}} \qquad \frac{\Gamma \vdash \alpha : \text{Finite} \qquad \Gamma \vdash \beta : \text{Finite}}{\Gamma \vdash \alpha \oplus \beta : \text{Finite}} \tag{80}$$

$$\frac{\Gamma \vdash \alpha : \text{Finite} \qquad (\text{data } x \cong \alpha) \in \Gamma}{\Gamma \vdash x : \text{Finite}}. \tag{81}$$

Although Finite appears to be a type, it is not a type. It cannot appear except by itself on the right-hand side of a judgment.

Universal quantifiers can be applied only around propositions that do not contain existential quantifiers over infinite types. To capture this notion, the following rules extend the Finite judgment to propositions.

$$\frac{\Gamma \vdash (\bar{x} = \bar{y}) : \text{Prop}}{\Gamma \vdash (\bar{x} = \bar{y}) : \text{Finite}} \qquad \frac{\Gamma \vdash (\bar{x} \leq \bar{y}) : \text{Prop}}{\Gamma \vdash (\bar{x} \leq \bar{y}) : \text{Finite}} \tag{82}$$

$$\frac{\Gamma \vdash \phi : \text{Finite} \qquad \Gamma \vdash \phi : \text{Prop} \qquad \Gamma \vdash \psi : \text{Finite} \qquad \Gamma \vdash \psi : \text{Prop}}{\Gamma \vdash (\phi \wedge \psi) : \text{Finite}} \tag{83}$$

$$\frac{\Gamma \vdash \phi : \text{Finite} \qquad \Gamma \vdash \phi : \text{Prop} \qquad \Gamma \vdash \psi : \text{Finite} \qquad \Gamma \vdash \psi : \text{Prop}}{\Gamma \vdash (\phi \vee \psi) : \text{Finite}} \tag{84}$$

$$\frac{\Gamma \vdash \phi : \text{Finite} \qquad \Gamma \vdash \phi : \text{Prop}}{\Gamma \vdash \neg\phi : \text{Finite}} \tag{85}$$

$$\frac{\Gamma \vdash \phi : \text{Finite} \qquad \Gamma \vdash \phi : \text{Prop} \qquad \Gamma \vdash \psi : \text{Finite} \qquad \Gamma \vdash \psi : \text{Prop}}{\Gamma \vdash (\phi \to \psi) : \text{Finite}} \tag{86}$$

Existential quantifiers can quantify only over types that are built by using only finite-dimensional types in the domains of function types. This class of types is called Quantifiable, and is defined by the following rules.

$$\frac{\Gamma \text{ ctx} \qquad \alpha \in \text{Scalar}}{\Gamma \vdash \text{Fin}(n) : \text{Quantifiable}} \qquad \frac{\Gamma \vdash \alpha : \text{Quantifiable}}{\Gamma \vdash \text{Maybe}(\alpha) : \text{Quantifiable}} \tag{87}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \alpha \times \beta : \text{Quantifiable}} \tag{88}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \alpha \oplus \beta : \text{Quantifiable}} \tag{89}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable} \qquad (\text{data } x \cong \alpha) \in \Gamma}{\Gamma \vdash x : \text{Quantifiable}} \tag{90}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \alpha \to \beta : \text{Quantifiable}} \tag{91}$$

$$\frac{\Gamma \vdash \alpha : \text{Quantifiable}}{\Gamma \vdash \text{List}(\alpha) : \text{Quantifiable}} \tag{92}$$

$$\frac{\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable}}{\Gamma \vdash \text{Map}(\alpha, \beta) : \text{Quantifiable}} \tag{93}$$

The following rules pertain to the formation of quantified propositions:

$$\frac{\Gamma, x : \alpha \vdash \phi : \text{Prop} \qquad \Gamma, x : \alpha \vdash \phi : \text{Finite} \qquad \Gamma \vdash \alpha : \text{Finite}}{\Gamma \vdash (\forall x : \alpha.\ \phi) : \text{Prop}} \tag{94}$$

$$\frac{\Gamma, x : \alpha \vdash \phi : \text{Prop} \qquad \Gamma \vdash \alpha : \text{Quantifiable}}{\Gamma \vdash (\exists x : \alpha.\ \phi) : \text{Prop}}. \tag{95}$$

The following rules allow for the formation of Finite judgments of quantified propositions:

$$\frac{\Gamma \vdash (\forall x : \alpha.\ \phi) : \text{Prop}}{\Gamma \vdash (\forall x : \alpha.\ \phi) : \text{Finite}} \tag{96}$$

$$\frac{\Gamma, x : \alpha \vdash \phi : \text{Prop} \qquad \Gamma, x : \alpha \vdash \phi : \text{Finite} \qquad \Gamma \vdash \alpha : \text{Finite}}{\Gamma \vdash (\exists x : \alpha.\ \phi) : \text{Finite}}. \tag{97}$$

4 Denotational semantics for OSL

The denotational semantics for OSL assigns denotations to expressions relative to denotational contexts. A denotation is a set. A denotational context consists of a context Γ, a universe of sets U, and a partial function $C : \text{Name} \to U$ that assigns denotations to each unbound variable in Γ. Here, Name denotes the set of OSL variable names. OSL contexts consist of three types of declarations: unbound variable declarations $x : \alpha$, equational definitions $\text{def } x : \alpha := \bar{y}$, and data declarations $\text{data } x \cong \alpha$. The unbound variables in a context Γ are those occurring on the left-hand sides of variable declarations $(x : \alpha) \in \Gamma$.

The universe U is required to be a Grothendieck universe. A Grothendieck universe is a set in which mathematics can be performed. Specifically, by definition, a set U is a Grothendieck universe if and only if:

1. For all x, y, if $x \in U$ and $y \in x$, then $y \in U$.
2. For all $x, y \in U$, $\{x, y\} \in U$.
3. For all $x \in U$, $\mathcal{P}(X) \in U$. $\mathcal{P}(X)$ is the power set of X, i.e.,
$$\mathcal{P}(X) = \{Y \mid Y \subseteq X\}. \tag{98}$$
4. For all sets $I \in U$ and families $\{x_i\}_{i \in I}$ of elements of U,
$$\left(\bigcup_{i \in I} x_i\right) \in U. \tag{99}$$

Any Grothendieck universe U can be used as the universe of a denotational context. A denotational context is written in the form Γ, U, C, where Γ is the context, U is the universe, and C is the map from names to elements of U.

The following sequent calculus delineates the denotational semantics of OSL, by providing rules for the formation of denotational contexts and judgments of denotations relative to denotational contexts. These rules implicitly define, relative to a denotational context, a partial denotation function Δ from expressions to denotations (i.e., elements of the universe U of the related denotational context). The sequent Γ, U, C dctx expresses that Γ, U, C is a valid denotational context. The sequent $\Gamma, U, C \vdash \Delta(\bar{x}) = X$ expresses that, relative to the denotational context Γ, U, C, the denotation of $\bar{x}$ is X (and $X \in U$ is implied).

$$\frac{\Gamma \text{ ctx} \qquad U \text{ is a Grothendieck universe}}{\Gamma, U, \emptyset \text{ dctx}} \tag{100}$$

$$\frac{\Gamma \text{ ctx} \qquad (x : \alpha) \in \Gamma \qquad \Gamma, U, C \vdash \Delta(\alpha) = A \qquad a \in A}{((\Gamma, x : \alpha), U, C \cup \{(x, a)\}) \text{ dctx}} \tag{101}$$

The various sequent rules all preserve the invariant that if Γ, U, C is a denotational context, then $\text{dom}(C)$ is a subset of the set of variables in $\text{free}(\Gamma)$ that are not bound by data or def declarations.

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\text{Type}) = \{A \in U \mid \exists \alpha, (\Gamma \vdash \alpha : \text{Type}) \wedge (\Gamma, U, C \vdash \Delta(\alpha) = A)\}} \tag{102}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\text{Prop}) = \{0, 1\}} \tag{103}$$

In the following rule, B^A represents the set of functions with domain A and codomain B, where a function is represented as the set of ordered pairs of its inputs and outputs.

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type} \qquad \Gamma, U, C \vdash \Delta(\alpha) = A \qquad \Gamma, U, C \vdash \Delta(\beta) = B}{\Gamma, U, C \vdash \Delta(\alpha \to \beta) = B^A} \tag{104}$$

In the following rules for basic numeric types, the mentioned denotations, $\mathbb{N}, +_{\mathbb{N}}$, and so forth, are defined within U in the usual manner, with the set of natural numbers being equated with the set of finite von Neumann ordinals, and the set of integers being equated with the set $\{-1, 1\} \times \mathbb{N}$.

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\mathbb{N}) = \mathbb{N}} \qquad \frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\mathbb{Z}) = \mathbb{Z}} \qquad \frac{\Gamma, U, C \text{ dctx} \qquad n \in \mathbb{N}}{\Gamma, U, C \vdash \Delta(\text{Fin}(n)) = \{0, ..., n-1\}} \tag{105}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(+_{\mathbb{N}}) = +_{\mathbb{N}}} \qquad \frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(+_{\mathbb{Z}}) = +_{\mathbb{Z}}} \tag{106}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\times_{\mathbb{N}}) = \times_{\mathbb{N}}} \qquad \frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\times_{\mathbb{Z}}) = \times_{\mathbb{Z}}} \tag{107}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(0_{\mathbb{N}}) = 0_{\mathbb{N}}} \qquad \frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(1_{\mathbb{N}}) = 1_{\mathbb{N}}} \tag{108}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(0_{\mathbb{Z}}) = 0_{\mathbb{Z}}} \qquad \frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(1_{\mathbb{Z}}) = 1_{\mathbb{Z}}} \qquad \frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(-1_{\mathbb{Z}}) = -1_{\mathbb{Z}}} \tag{109}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\text{cast}) = \{(x, x) \mid x \in \mathbb{Z}\} \cup \{(x, (1, x)) \mid x \in \mathbb{Z}\}} \tag{110}$$

$$\frac{\Gamma, U, C \text{ dctx} \qquad n \in \mathbb{N}}{\Gamma, U, C \vdash \Delta(\text{fin}(n)) = n} \tag{111}$$

The following rules provide denotations for product types and related functions:

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type} \qquad \Gamma, U, C \vdash \Delta(\alpha) = A \qquad \Gamma, U, C \vdash \Delta(\beta) = B}{\Gamma, U, C \vdash \Delta(\alpha \times \beta) = A \times B}. \tag{112}$$

Here, the set $A \times B$ is defined in the usual manner, as the set of ordered pairs (a, b) where $a \in A$ and $b \in B$. Ordered pairs are defined in the usual manner, with the pair (a, b) being encoded as the set $\{\{a\}, \{a, b\}\}$ (which contains sufficient information to discern the elements of the pair and which one is first).

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\pi_1) = \{((x, y), x) \mid \exists A, B \in \Delta(\text{Type}), x \in A \wedge y \in B\}} \tag{113}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\pi_2) = \{((x, y), y) \mid \exists A, B \in \Delta(\text{Type}),\ x \in A \wedge y \in B\}} \tag{114}$$

$$\frac{\Gamma \vdash \bar{f} : \gamma \to \alpha \qquad \Gamma \vdash \bar{G} : \gamma \to \beta \qquad \Gamma, U, C \vdash \Delta(\bar{f}) = F \qquad \Gamma, U, C \vdash \Delta(\bar{g}) = G}{\Gamma, U, C \vdash \Delta(\bar{f} \times \bar{g}) = \{(x, (y, z)) \mid (x, y) \in F \wedge (x, z) \in G\}} \tag{115}$$

The following rules provide denotations for coproduct types and related functions:

$$\frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \beta : \text{Type} \qquad \Gamma, U, C \vdash \Delta(\alpha) = A \qquad \Gamma, U, C \vdash \Delta(\beta) = B}{\Gamma, U, C \vdash \Delta(\alpha \oplus \beta) = (\{0\} \times A) \cup (\{1\} \times B)} \tag{116}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\iota_1) = \{(x, (0, x)) \mid \exists A \in \Delta(\text{Type}),\ x \in A\}} \tag{117}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\iota_2) = \{(x, (1, x)) \mid \exists A \in \Delta(\text{Type}),\ x \in A\}} \tag{118}$$

$$\frac{\Gamma \vdash \bar{f} : \alpha \to \gamma \qquad \Gamma \vdash \bar{G} : \beta \to \gamma \qquad \Gamma, U, C \vdash \Delta(\bar{f}) = F \qquad \Gamma, U, C \vdash \Delta(\bar{g}) = G}{\Gamma, U, C \vdash \Delta(\bar{f} \oplus \bar{g}) = \{((0, x), y) \mid (x, y) \in F\} \cup \{((1, x), y) \mid (x, y) \in G\}}. \tag{119}$$

For a function $C : \text{Name} \to U$, the notation $C[v \mapsto x]$, where $v \in \text{Name}$ and $x \in U$, denotes the partial function:

$$C[v \mapsto x](u) = \begin{cases} x & \text{if } u = v, \\ C(u) & \text{otherwise.} \end{cases} \tag{120}$$

The following rules provide denotations for lambda abstractions and function applications:

$$\frac{\Gamma, v : \alpha \vdash \bar{y} : \beta \qquad \Gamma, U, C \vdash \Delta(\alpha) = A \qquad \Gamma, U, C \vdash \Delta(\beta) = B}{\Gamma, U, C \vdash \Delta(\lambda v : \alpha \mapsto \bar{y}) = \{(x, y) \mid x \in A \wedge y \in B \wedge (\Gamma, U, C[v \mapsto x]) \vdash \Delta(\bar{y}) = y)\}} \tag{121}$$

$$\frac{\Gamma \vdash \bar{f} : \alpha \to \beta \qquad \Gamma \vdash \bar{x} : \alpha \qquad \Gamma \vdash \Delta(\bar{f}) = F \qquad \Gamma \vdash \Delta(\bar{x}) = x \qquad (x, y) \in F}{\Gamma, U, C \vdash \Delta(\bar{f}(\bar{x})) = y}. \tag{122}$$

The following rules define denotations of names introduced by equational definitions and data declarations:

$$\frac{\Gamma, U, C \vdash \Delta(\alpha) = A \qquad (\text{data } x \cong \alpha) \in \Gamma}{\Gamma, U, C \vdash \Delta(x) = A} \qquad \frac{\Gamma, U, C \vdash \Delta(\bar{y}) = Y \qquad (\text{def } x : \alpha := \bar{y}) \in \Gamma}{\Gamma, U, C \vdash \Delta(x) = Y}. \tag{123}$$

The following rules define denotations of to/from isomorphisms, which are quite trivial, because these isomorphisms are always the identity function over the denotation set:

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\text{to}(x)) = \{(x, x) \mid \exists A \in \Delta(\text{Type}), x \in A\}} \tag{124}$$

$$\frac{\Gamma, U, C \vdash \qquad (\text{data } x \cong \alpha) \in \Gamma}{\Gamma, U, C \vdash \Delta(\text{from}(x)) = \{(x, x) \mid x \in A\}}. \tag{125}$$

The following rule defines denotations of let expressions:

$$\frac{\Gamma, U, C \vdash \Delta(\bar{y}) = Y \qquad \Gamma, U, C[x \mapsto Y] \vdash \Delta(\bar{z}) = Z}{\Gamma, U, C \vdash \Delta(\text{let } x : \alpha := \bar{y}; \bar{z}) = Z}. \tag{126}$$

The following rules define denotations of the Maybe functor and related functions:

$$\frac{\Gamma, U, C \text{ dctx}}{\begin{array}{rcll} \Gamma, U, C \vdash \Delta(\text{Maybe}) & = & \{(A, \{0\} \cup (\{1\} \times A)) & \mid \quad A \in \Delta(\text{Type})\} \\ & \cup & \{(f, \{(0,0)\} \cup (\{1\} \times f)) & \mid \quad \exists \alpha, \beta, (\Gamma \vdash \alpha : \text{Type}) \\ & & & \wedge \quad (\Gamma \vdash \beta : \text{Type}) \\ & & & \wedge \quad (\Gamma, U, C \vdash \Delta(\alpha \to \beta) = D) \\ & & & \wedge \quad f \in D\} \end{array}} \tag{127}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\text{just}) = \{(x, (1, x)) \mid \exists A \in \Delta(\text{Type}), x \in A\}} \qquad \frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\text{nothing}) = 0}. \tag{128}$$

For a function $f : A \times B \to C$, let $\text{curry}(f) : A \to B \to C$ be a function defined as follows:

$$\text{curry}(f) = \{(a, \{(b, f(a, b)) \mid b \in B\}) \mid a \in A\}. \tag{129}$$

Similarly, for a function $f : A \times B \times C \to D$, let $\text{curry}^2(f) : A \to B \to C \to D$ be a function defined as follows:

$$\text{curry}^2(f) = \text{curry}(\text{curry}(f)). \tag{130}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\begin{array}{lll}\Gamma, U, C \vdash \Delta(\text{maybe}) = \text{curry}^2(\{(f, d, x, y) & | & \exists \alpha,\ \Gamma \vdash \alpha : \text{FiniteDim} \\ & \wedge & \exists \beta,\ \Gamma \vdash \beta : \text{FiniteDim} \\ & \wedge & f \in \Delta(\alpha \to \beta) \\ & \wedge & d \in \Delta(\beta) \\ & \wedge & x \in \Delta(\text{Maybe}(\alpha)) \\ & \wedge & ((x = \text{nothing} \wedge y = d) \vee (x, y) \in f)\})\end{array}} \tag{131}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\text{exists}) = \{((1, x), x) \mid \exists A \in \Delta(\text{Type}), x \in A\}} \tag{132}$$

The following rules define denotations of the List functor and related functions:

$$\frac{\Gamma, U, C \text{ dctx}}{\begin{array}{lcll}\Gamma, U, C \vdash \Delta(\text{List}) & = & \{(A, \mathbb{N} \times A^{\mathbb{N}}) & | \quad A \in \Delta(\text{Type})\} \\ & \cup & \{(f, (\text{id}_{\mathbb{N}}, g \mapsto f \circ g)) & | \quad \exists \alpha, \beta, f \in \Delta(\alpha \to \beta) \\ & & & \wedge \quad \Gamma \vdash \alpha : \text{Quantifiable} \\ & & & \wedge \quad \Gamma \vdash \beta : \text{Quantifiable}\}\end{array}} \tag{133}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\text{length}) = \{((\ell, f), \ell) \mid \exists \alpha, (\Gamma \vdash \alpha : \text{Quantifiable}) \wedge (\ell, f) \in \Delta(\text{List}(\alpha))\}} \tag{134}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\begin{array}{lll}\Gamma, U, C \vdash \Delta(\text{nth}) = \text{curry}(\{((\ell, f), i, x) & | & \exists \alpha, (\Gamma \vdash \alpha : \text{Quantifiable}) \\ & \wedge & (\ell, f) \in \Delta(\text{List}(\alpha)) \\ & \wedge & i \in \mathbb{N} \wedge (i, x) \in f \wedge i < \ell\}).\end{array}} \tag{135}$$

The following rules define denotations of the Map functor and related functions:

$$\frac{\Gamma, U, C \text{ dctx}}{\begin{array}{l}\Gamma, U, C \vdash \Delta(\text{Map}) = \\ \quad \text{curry}(\{(A, B, \mathbb{N} \times A^{\mathbb{N}} \times B^A) \mid \exists \alpha, (\Gamma \vdash \alpha : \text{FiniteDim}) \wedge \exists \beta, (\Gamma \vdash \beta : \text{Quantifiable}) \\ \qquad \wedge (\Gamma, U, C \vdash \Delta(\alpha) = A) \wedge (\Gamma, U, C \vdash \Delta(\beta) = B)\}) \\ \quad \cup \{(f, \text{id}_{\mathbb{N}} \times \text{id}_{A^{\mathbb{N}}} \times (g \mapsto f \circ g)) \\ \qquad \mid \exists \alpha, \Gamma \vdash \alpha : \text{FiniteDim} \wedge \exists \beta, (\Gamma \vdash \beta : \text{Quantifiable}) \wedge \exists \gamma, (\Gamma \vdash \gamma : \text{Quantifiable}) \\ \qquad \wedge (\Gamma, U, C \vdash \Delta(\alpha) = A) \wedge f \in \Delta(\beta \to \gamma)\}\end{array}} \tag{136}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\begin{array}{l}\Gamma, U, C \vdash \Delta(\text{lookup}) = \text{curry}(\{(i, (\ell, k, v), x) \\ \qquad \mid \exists \alpha, (\Gamma \vdash \alpha : \text{FiniteDim}) \wedge \exists \beta, (\Gamma \vdash \beta : \text{FiniteDim}) \\ \qquad \wedge (\ell, k, v) \in \Delta(\text{Map}(\alpha, \beta)) \\ \qquad \wedge (x = \text{nothing} \vee \exists j \in \mathbb{N}, j < \ell \wedge (j, i) \in k \wedge (i, x') \in v \wedge x = \text{just}(x')\}))\end{array}} \tag{137}$$

$$\frac{\Gamma, U, C \text{ dctx}}{\begin{array}{l}\Gamma, U, C \vdash \Delta(\text{keys}) = \{((\ell, k, v), (\ell, k)) \\ \qquad \mid \exists \alpha, (\Gamma \vdash \alpha : \text{FiniteDim}) \wedge \exists \beta, (\Gamma \vdash \beta : \text{Quantifiable}) \\ \qquad \wedge (\ell, k, v) \in \Delta(\text{Map}(\alpha, \beta))\}.\end{array}} \tag{138}$$

This semantics does not define a denotation for the sum function, because that function is polymorphic, and it will not necessarily work the same on all values regardless of the type that those values are interpreted as belonging to. Therefore, this semantics merely defines the denotations of applications of the sum function:

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(\alpha) \qquad \Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma, U, C \vdash \Delta(\bar{x}) = (\ell, f)}{\Gamma, U, C \vdash \Delta(\mathrm{sum}(\bar{x})) = \sum_{i=0}^{\ell-1} f(i)} \tag{139}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{Maybe}(\alpha)) \qquad \Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma, U, C \vdash \Delta(\bar{x}) = (\ell, f)}{\Gamma, U, C \vdash \Delta(\mathrm{sum}(\bar{x})) = \sum_{i=0}^{\ell-1} \begin{cases} 0 & \text{if } f(i) = 0, \\ \pi_2(f(i)) & \text{otherwise} \end{cases}} \tag{140}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{List}(\alpha)) \qquad \Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma, U, C \vdash \Delta(\bar{x}) = (\ell, f)}{\Gamma, U, C \vdash \Delta(\mathrm{sum}(\bar{x})) = \sum_{i=0}^{\ell-1} \sum_{j=0}^{\pi_1(f(i))-1} \pi_2(f(i))(j)} \tag{141}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{List}(\mathrm{Maybe}(\alpha))) \qquad \Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma, U, C \vdash \Delta(\bar{x}) = (\ell, f)}{\Gamma, U, C \vdash \Delta(\mathrm{sum}(\bar{x})) = \sum_{i=0}^{\ell-1} \sum_{j=0}^{\pi_1(f(i))-1} \begin{cases} 0 & \text{if } \pi_2(f(i))(j) = 0, \\ \pi_2(\pi_2(f(i))(j)) & \text{otherwise} \end{cases}} \tag{142}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{Map}(\alpha, \beta) \qquad \Gamma, U, C \vdash \Delta(\bar{x}) = (\ell, k, v)}{\Gamma, U, C \vdash \Delta(\mathrm{sum}(\bar{x})) = \sum_{i=0}^{\ell-1} v(k(i))} \tag{143}$$

$$\frac{\Gamma \vdash \beta : \mathrm{Num} \qquad \Gamma \vdash \bar{x} : \mathrm{Map}(\alpha, \mathrm{List}(\beta)) \qquad \Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta((\mathrm{sum} \circ \mathrm{Map}(\mathrm{length}))(\bar{x})) = \Delta(\mathrm{sum}(\mathrm{Map}(\mathrm{length})(\bar{x})))} \tag{144}$$

$$\frac{\Gamma \vdash \beta : \mathrm{Num} \qquad \Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{Map}(\alpha, \beta)) \qquad \Gamma \vdash \bar{k} : \alpha \qquad \Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta((\mathrm{sum} \circ \mathrm{List}(\mathrm{lookup}(\bar{k})))(\bar{x})) = \Delta(\mathrm{sum}(\mathrm{List}(\mathrm{lookup}(\bar{k}))(\bar{x})))}. \tag{145}$$

The following rules define denotations of propositions, which are truth values (1 = true, 0 = false):

$$\frac{\Gamma \vdash \alpha : \mathrm{FiniteDim} \qquad \Gamma \vdash \bar{x} : \alpha \qquad \Gamma \vdash \bar{y} : \alpha \qquad \Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\bar{x} = \bar{y}) = \begin{cases} 1 & \text{if } \Delta(\bar{x}) = \Delta(\bar{y}), \\ 0 & \text{otherwise} \end{cases}} \tag{146}$$

$$\frac{\Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma \vdash \bar{x} : \alpha \qquad \Gamma \vdash \bar{y} : \alpha \qquad \Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\bar{x} \leq \bar{y}) = \begin{cases} 1 & \text{if } \Delta(\bar{x}) \leq \Delta(\bar{y}), \\ 0 & \text{otherwise} \end{cases}} \tag{147}$$

$$\frac{\Gamma \vdash \phi : \mathrm{Prop} \qquad \Gamma \vdash \psi : \mathrm{Prop} \qquad \Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\phi \wedge \psi) = \Delta(\phi) \cdot \Delta(\psi)} \tag{148}$$

$$\frac{\Gamma \vdash \phi : \mathrm{Prop} \qquad \Gamma \vdash \psi : \mathrm{Prop} \qquad \Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\phi \vee \psi) = \Delta(\phi) + \Delta(\psi) - (\Delta(\phi) \cdot \Delta(\psi))} \tag{149}$$

$$\frac{\Gamma \vdash \phi : \mathrm{Prop} \qquad \Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\neg\phi) = 1 - \Delta(\phi)} \tag{150}$$

$$\frac{\Gamma \vdash \phi : \mathrm{Prop} \qquad \Gamma \vdash \psi : \mathrm{Prop} \qquad \Gamma, U, C \text{ dctx}}{\Gamma, U, C \vdash \Delta(\phi \to \psi) = \Delta(\neg\phi \vee \psi)} \tag{151}$$

$$\frac{\Gamma, x : \alpha \vdash \phi : \mathrm{Prop} \qquad \Gamma, U, C \vdash \Delta(\alpha) = A \qquad v \in \{0,1\}^A \qquad \forall i \in A, (\Gamma, U, C[x \mapsto i] \vdash \Delta(\phi) = v(i))}{\Gamma, U, C \vdash \Delta(\forall x : \alpha, \phi) = \begin{cases} 1 & \text{if } v = A \times \{1\} \\ 0 & \text{otherwise} \end{cases}} \tag{152}$$

$$\frac{\Gamma, x : \alpha \vdash \phi : \text{Prop} \qquad \Gamma, U, C \vdash \Delta(\alpha) = A \qquad v \in \{0,1\}^A \qquad \forall i \in A, (\Gamma, U, C[x \mapsto i] \vdash \Delta(\phi) = v(i))}{\Gamma, U, C \vdash \Delta(\exists x : \alpha, \phi) = \begin{cases} 0 & \text{if } v = A \times \{0\} \\ 1 & \text{otherwise.} \end{cases}} \tag{153}$$

5 Translation of OSL into Σ_1^1 formulas

The rules for translating OSL into Σ_1^1 formulas are expressed as a sequent calculus, in which sequents can take the form $\Gamma \vdash \phi : \alpha \rightsquigarrow \psi$, where Γ is a context, ϕ and α are OSL expressions, and ψ is a Σ_1^1 formula. This translation calculus incorporates the rules of the sequent calculus definition of OSL in the preceding section. Contexts are extended with declarations mapping OSL variable names to sequences of Σ_1^1 formula variable names; these declarations take the form $x \rightsquigarrow \vec{y}$, where x is an OSL variable name, and $\vec{y}$ is a sequence of Σ_1^1 formula variable names.

The language of Σ_1^1 formulas is defined in [5]. We assume that this language has been extended with a binary function symbol $\text{ind}_<$, and that the arithmetization construction in [5] has been extended to support terms containing applications of this function. The function call $\text{ind}(n, m)$ is interpreted as returning 1 when $0 \leq n < m$ and 0 otherwise. Extending the construction of [5] with this function is straightforward, because it can be implemented by using lookup tables.

The following rules express translations of logical connectives.

$$\frac{\Gamma \vdash \phi_0 : \text{Prop} \rightsquigarrow \psi_0 \qquad \Gamma \vdash \phi_1 : \text{Prop} \rightsquigarrow \psi_1}{\Gamma \vdash \phi_0 \wedge \phi_1 : \text{Prop} \rightsquigarrow \psi_0 \wedge \psi_1} \tag{154}$$

$$\frac{\Gamma \vdash \phi_0 : \text{Prop} \rightsquigarrow \psi_0 \qquad \Gamma \vdash \phi_1 : \text{Prop} \rightsquigarrow \psi_1}{\Gamma \vdash \phi_0 \vee \phi_1 : \text{Prop} \rightsquigarrow \psi_0 \vee \psi_1} \tag{155}$$

$$\frac{\Gamma \vdash \phi : \text{Prop} \rightsquigarrow \psi}{\Gamma \vdash \neg\phi : \text{Prop} \rightsquigarrow \neg\psi} \tag{156}$$

$$\frac{\Gamma \vdash \phi_0 : \text{Prop} \rightsquigarrow \psi_0 \qquad \Gamma \vdash \phi_1 : \text{Prop} \rightsquigarrow \psi_1}{\Gamma \vdash \phi_0 \to \phi_1 : \text{Prop} \rightsquigarrow \psi_0 \to \psi_1} \tag{157}$$

The following rule expresses translations of equality statements in which both sides are of finite-dimensional type. In this rule, $\vec{\tau}$ and $\vec{v}$ denote same-length vectors of terms (necessarily the same length, because a finite type requires a fixed number of first-order variables to represent it). $\vec{\tau}$ is shorthand for $\tau_1, ..., \tau_n$. In addition, $\vec{\tau} = \vec{v}$ is shorthand for $\bigwedge_{i \in [n]} \tau_i = v_i$.

$$\frac{\Gamma \vdash \bar{x} : \alpha \rightsquigarrow \vec{\tau} \qquad \Gamma \vdash \bar{y} : \alpha \rightsquigarrow \vec{v} \qquad \Gamma \vdash \alpha : \text{FiniteDim}}{\Gamma \vdash \bar{x} = \bar{y} \rightsquigarrow \vec{\tau} = \vec{v}} \tag{158}$$

To provide translations for variables, we allow contexts to contain declarations of the form $x \rightsquigarrow y$, where x denotes an OSL variable name, and y denotes a Σ_1^1 formula variable name (either a first-order variable x_i or a second-order variable f_i, where i is a de Bruijn index). The Σ_1^1 formula variable names contained in these declarations are added to the output of free(Γ). Let Var denote the set of OSL variable names, and let DeBruijn denote the set of Σ_1^1 variable names. The following rules allow for the introduction and use of these declarations:

$$\frac{\Gamma \text{ ctx} \qquad \Gamma \vdash x : \alpha \qquad \alpha \in \text{Scalar} \qquad y \in \text{DeBruijn}}{(\Gamma, x \rightsquigarrow y) \text{ ctx}} \tag{159}$$

$$\frac{\Gamma \vdash x : \alpha \qquad (x \rightsquigarrow y) \in \Gamma}{\Gamma \vdash x : \alpha \rightsquigarrow y}. \tag{160}$$

In later rules, contexts will be extended with declarations of the more general form $x \rightsquigarrow \vec{y}$, where x is an OSL variable name, and $\vec{y}$ is a sequence of Σ_1^1 formula variable names.

The following rule allows for introduction of variable mappings for types introduced by data declarations:

$$\frac{(\text{data } a \cong \alpha) \in \Gamma \qquad \Gamma \vdash x : a \qquad (\Gamma, y : \alpha, y \rightsquigarrow \vec{z}) \text{ ctx}}{(\Gamma, x \rightsquigarrow \vec{z}) \text{ ctx}}. \tag{161}$$

The following rules, together with the preceding rules, allow for the introduction of variable mappings for arbitrary finite-dimensional types:

$$\frac{\begin{array}{c}\Gamma \vdash x : \text{Maybe}(\alpha) \quad (\Gamma, y : \alpha, y \rightsquigarrow \vec{z}) \text{ ctx} \\ w \in \text{DeBruijn}\end{array}}{(\Gamma, x \rightsquigarrow w, \vec{z}) \text{ ctx}} \tag{162}$$

$$\frac{\Gamma \vdash x : \alpha \times \beta \qquad (\Gamma, y : \alpha, y \rightsquigarrow \vec{u}) \text{ ctx} \qquad (\Gamma, y : \beta, y \rightsquigarrow \vec{v}) \text{ ctx}}{(\Gamma, x \rightsquigarrow \vec{u}, \vec{v}) \text{ ctx}} \tag{163}$$

$$\frac{\begin{array}{c}\Gamma \vdash x : \alpha \oplus \beta \quad (\Gamma, y : \alpha, y \rightsquigarrow \vec{u}) \text{ ctx} \quad (\Gamma, y : \beta, y \rightsquigarrow \vec{v}) \text{ ctx} \\ w \in \text{DeBruijn}\end{array}}{(\Gamma, x \rightsquigarrow w, \vec{u}, \vec{v}) \text{ ctx}}. \tag{164}$$

Finite-dimensional types can be translated into sequences of first-order variables. Infinite-dimensional (i.e., non-finite-dimensional) types consist of function, list, set, and map types, and product and coproduct types made by using infinite-dimensional types (and also possibly finite-dimensional types). Infinite-dimensional types can be translated into sequences of second-order variables (and also possibly first-order variables).

Any List, Map, or Set type can be considered as a function type from a finite-dimensional type to a finite-dimensional type. This can in turn be modeled as a collection of functions in second-order logic, wherein the inputs are one or more scalars, and the outputs are individual scalars, as captured by the following rules.

This next rule uses a new notation for variables of Σ_1^1 formulas, which are denoted as x_i^n, where i is a de Bruijn index, and n is the arity, with first-order variables having arity 0. This notation allows for keeping track of the arities of Σ_1^1 formula variables in various contexts, which is necessary. When superscripts are not present, the arities of the variables can be anything or can be inferred from context.

$$\frac{\begin{array}{c}\Gamma \vdash x : \alpha \to \beta \\ \Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable} \\ (\Gamma, y : \alpha, y \rightsquigarrow x_{j_1}^0, ..., x_{j_k}^0) \text{ ctx} \quad (\Gamma, y : \beta, y \rightsquigarrow x_{i_1}^{n_1}, ..., x_{i_m}^{n_m}) \text{ ctx}\end{array}}{(\Gamma, x \rightsquigarrow x_{i_1}^{n_1+k}, ..., x_{i_m}^{n_m+k}) \text{ ctx}} \tag{165}$$

A List variable is translated by a first-order variable denoting the length of the list (x_j^0) and a collection of function variables denoting the function from indices to list elements ($x_{i_1}^{n_1+1}, ..., x_{i_m}^{n_m+1}$):

$$\frac{\begin{array}{c}\Gamma \vdash x : \text{List}(\alpha) \qquad \Gamma \vdash \alpha : \text{Quantifiable} \\ (\Gamma, y : \alpha, y \rightsquigarrow x_{i_1}^{n_1}, ..., x_{i_m}^{n_m}) \text{ ctx}\end{array}}{(\Gamma, x \rightsquigarrow x_j^0, x_{i_1}^{n_1+1}, ..., x_{i_m}^{n_m+1}) \text{ ctx}}. \tag{166}$$

A Map variable is translated by variables denoting the list of keys and the function from keys to values:

$$\frac{\begin{array}{c}\Gamma \vdash x : \text{Map}(\alpha, \beta) \qquad \Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{Quantifiable} \\ (\Gamma, y : \alpha, y \rightsquigarrow x_{j_1}^0, ..., x_{j_k}^0) \text{ ctx} \qquad (\Gamma, y : \beta, y \rightsquigarrow x_{i_1}^{n_1}, ..., x_{i_m}^{n_m}) \text{ ctx}\end{array}}{(\Gamma, x \rightsquigarrow x_\ell^0, x_{j_1}^1, ..., x_{j_k}^1, x_p^k, x_{i_1}^{n_1+k}, ..., x_{i_m}^{n_m+k}) \text{ ctx}}. \tag{167}$$

Here, $x^0_\ell, x^1_{j_1}, ..., x^1_{j_k}$ is the list of keys, with length x^0_ℓ. x^k_p holds a $\{0,1\}$-valued function where $x^k_p(\vec{v}) = 1$ iff $\vec{v}$ is a key of the map. $x^{n_1+k}_{i_1}, ..., x^{n_m+k}_{i_m}$ is the key-value map.

The following rules provide translations for basic arithmetic expressions:

$$\frac{\Gamma \vdash \bar{x} : \alpha \rightsquigarrow \tau \qquad \Gamma \vdash \bar{y} : \alpha \rightsquigarrow \upsilon \qquad \alpha \in \{\mathbb{N}, \mathbb{Z}\}}{\Gamma \vdash \bar{x} +_\alpha \bar{y} : \alpha \rightsquigarrow \tau + \upsilon} \tag{168}$$

$$\frac{\Gamma \vdash \bar{x} : \alpha \rightsquigarrow \tau \qquad \Gamma \vdash \bar{y} : \alpha \rightsquigarrow \upsilon \qquad \alpha \in \{\mathbb{N}, \mathbb{Z}\}}{\Gamma \vdash \bar{x} \times_\alpha \bar{y} : \alpha \rightsquigarrow \tau \cdot \upsilon} \tag{169}$$

$$\frac{\Gamma \text{ ctx} \qquad \alpha \in \{\mathbb{N}, \mathbb{Z}\} \qquad k \in \{0,1\}}{\Gamma \vdash k_\alpha : \alpha \rightsquigarrow k} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash -1_\mathbb{Z} : \mathbb{Z} \rightsquigarrow -1} \tag{170}$$

$$\frac{\Gamma \vdash \bar{x} : \alpha \rightsquigarrow \tau \qquad \Gamma \vdash \text{cast} : \alpha \to \beta}{\Gamma \vdash \text{cast}(\bar{x}) : \beta \rightsquigarrow \tau} \tag{171}$$

$$\frac{\Gamma \text{ ctx} \qquad n \in \mathbb{N} \qquad 0 < n}{\Gamma \vdash \text{fin}(0) : \text{Fin}(n) \rightsquigarrow 0} \qquad \frac{\Gamma \vdash \text{fin}(m) : \text{Fin}(n) \rightsquigarrow \tau}{\Gamma \vdash \text{fin}(m+1) : \text{Fin}(n+1) \rightsquigarrow \tau + 1}. \tag{172}$$

The following rules allow for translation of basic product and coproduct functions:

$$\frac{\Gamma \vdash \bar{x} : \alpha \rightsquigarrow \vec{\tau} \qquad \Gamma \vdash \bar{y} : \beta \rightsquigarrow \vec{\upsilon}}{\Gamma \vdash (\bar{x}, \bar{y}) : \alpha \times \beta \rightsquigarrow \vec{\tau}, \vec{\upsilon}} \tag{173}$$

$$\frac{\Gamma \vdash \bar{x} : \alpha \times \beta \rightsquigarrow \vec{\tau}, \vec{\upsilon} \qquad (\Gamma, y : \alpha, y \rightsquigarrow \vec{\tau}) \text{ ctx}}{\Gamma \vdash \pi_1(\bar{x}) : \alpha \rightsquigarrow \vec{\tau}} \tag{174}$$

$$\frac{\Gamma \vdash \bar{x} : \alpha \times \beta \rightsquigarrow \vec{\tau}, \vec{\upsilon} \qquad (\Gamma, y : \beta, y \rightsquigarrow \vec{\upsilon}) \text{ ctx}}{\Gamma \vdash \pi_2(\bar{x}) : \beta \rightsquigarrow \vec{\upsilon}} \tag{175}$$

$$\frac{\begin{array}{c}\Gamma \vdash \bar{f} : \gamma \to \alpha \qquad \Gamma \vdash \bar{g} : \gamma \to \beta \\ \Gamma \vdash \bar{f}(\vec{\bar{x}}) : \alpha \rightsquigarrow \vec{\tau} \qquad \Gamma \vdash \bar{g}(\vec{\bar{y}}) : \beta \rightsquigarrow \vec{\upsilon}\end{array}}{\Gamma \vdash (\bar{f} \times \bar{g})((\vec{\bar{x}}, \vec{\bar{y}})) : \alpha \times \beta \rightsquigarrow \vec{\tau}, \vec{\upsilon}} \tag{176}$$

$$\frac{\Gamma \vdash \bar{x} : \alpha \rightsquigarrow \vec{\tau} \qquad \Gamma \vdash \bar{y} : \beta \rightsquigarrow \vec{\upsilon}}{\Gamma \vdash \iota_1(\bar{x}) : \alpha \oplus \beta \rightsquigarrow 0, \vec{\tau}, \vec{\upsilon}} \tag{177}$$

$$\frac{\Gamma \vdash \bar{x} : \alpha \rightsquigarrow \vec{\tau} \qquad \Gamma \vdash \bar{y} : \beta \rightsquigarrow \vec{\upsilon}}{\Gamma \vdash \iota_2(\bar{y}) : \alpha \oplus \beta \rightsquigarrow 1, \vec{\tau}, \vec{\upsilon}}. \tag{178}$$

The following rule relies on $\vec{\tau}$ and $\vec{\upsilon}$ being like-length sequences of first-order terms (because γ is finite-dimensional), to be able to use vector mathematics (scalar multiplication and addition) to combine these vectors of terms:

$$\frac{\begin{array}{c}\Gamma \vdash \gamma : \text{FiniteDim} \qquad \Gamma \vdash \bar{f} : \alpha \to \gamma \qquad \Gamma \vdash \bar{g} : \beta \to \gamma \\ \Gamma \vdash \bar{f}(\vec{\bar{x}}) : \gamma \rightsquigarrow \vec{\tau} \qquad \Gamma \vdash \bar{g}(\vec{\bar{x}}) : \gamma \rightsquigarrow \vec{\upsilon}\end{array}}{\Gamma \vdash (\bar{f} \oplus \bar{g})(\vec{\bar{x}}) : \gamma \rightsquigarrow u, ((1-u) \cdot \vec{\tau} + u \cdot \vec{\upsilon})}. \tag{179}$$

The following function application translation rule relies on the arities of the terms $\vec{\tau}$ and $\vec{\upsilon}$ aligning consistently. The vectorized function application $\vec{\tau}(\vec{\upsilon})$ denotes the vector of results from the pointwise curried function application of the elements of $\vec{\tau}$ to the elements of $\vec{\upsilon}$. Thus, all elements of $\vec{\tau}$ must be second-order, and all elements of $\vec{\upsilon}$ must be first-order. This aspect is true not by definition but as a theorem. The translation set is thus expanded to include curried function applications, whereas in Σ^1_1

formulas, no curried function applications exist. However, the curried function applications are used only in translation of expressions and not translation of propositions (which is a theorem that must be proven).

$$\frac{\Gamma \vdash \bar{f} : \alpha \to \beta \rightsquigarrow \vec{\tau} \qquad \Gamma \vdash \bar{x} : \alpha \rightsquigarrow \vec{v}}{\Gamma \vdash \bar{f}(\bar{x}) : \beta \rightsquigarrow \vec{\tau}(\vec{v})} \tag{180}$$

The following rule allows for translation of function applications in which the head is a lambda abstraction:

$$\frac{\Gamma, x : \alpha, x \rightsquigarrow \vec{v} \vdash \bar{y} : \beta \to \vec{v} \qquad \Gamma \vdash \bar{w} : \alpha \rightsquigarrow \vec{\tau}}{\Gamma \vdash (\lambda x : \alpha \mapsto \bar{y})(\bar{w}) : \beta \rightsquigarrow \vec{v}[\vec{v} \mapsto \vec{\tau}]}. \tag{181}$$

The substitution notation $v[\vec{v} \mapsto \vec{\tau}]$ actually denotes a series of substitutions, with each occurrence of v_i substituted with τ_i.

The following rules allow for translation of applications of isomorphisms of defined data types with their definientia:

$$\frac{\Gamma \vdash \bar{x} : \alpha \rightsquigarrow \vec{\tau} \qquad (\text{data } a \cong \alpha) \in \Gamma}{\text{to}(a, \bar{x}) : a \rightsquigarrow \vec{\tau}} \tag{182}$$

$$\frac{\Gamma \vdash \bar{x} : a \rightsquigarrow \vec{\tau} \qquad (\text{data } a \cong \alpha) \in \Gamma}{\text{from}(a, \bar{x}) : \alpha \rightsquigarrow \vec{\tau}}. \tag{183}$$

The following rules allow for translation of names defined by equational definitions:

$$\frac{\begin{array}{c}\Gamma \vdash \alpha : \text{Type} \\ (x : \alpha := \bar{a}) \in \Gamma \\ \Gamma \vdash (\bar{a} : \alpha \rightsquigarrow \vec{\tau})\end{array}}{\Gamma \vdash x : \alpha \rightsquigarrow \vec{\tau}} \qquad \frac{\begin{array}{c}\Gamma \vdash \phi : \text{Prop} \rightsquigarrow \psi \\ (p : \text{Prop} := \phi) \in \Gamma\end{array}}{\Gamma \vdash p : \text{Prop} \rightsquigarrow \psi}. \tag{184}$$

The following rules allow for translation of basic functor operations:

$$\frac{\Gamma \vdash \bar{f} : \alpha \to \beta \qquad \Gamma, x : \alpha, x \rightsquigarrow \vec{v} \vdash \bar{f}(x) \rightsquigarrow \vec{\mu} \qquad \Gamma \vdash \bar{y} : \text{Maybe}(\alpha) \rightsquigarrow \sigma, \vec{\tau}}{\Gamma \vdash \text{Maybe}(\bar{f}, \bar{y}) \rightsquigarrow \sigma, \vec{\mu}[\vec{v} \mapsto \vec{\tau}]} \tag{185}$$

$$\frac{\Gamma \vdash \bar{x} : \alpha \rightsquigarrow \vec{\tau}}{\Gamma \vdash \text{just}(\bar{x}) : \text{Maybe}(\alpha) \rightsquigarrow 1, \vec{\tau}} \qquad \frac{\Gamma \vdash \alpha : \text{Type} \qquad \Gamma \vdash \bar{x} : \alpha \rightsquigarrow \vec{\tau}}{\Gamma \vdash \text{nothing} : \text{Maybe}(\alpha) \rightsquigarrow 0, \vec{\tau}} \tag{186}$$

$$\frac{\begin{array}{c}\Gamma \vdash \alpha : \text{FiniteDim} \qquad \Gamma \vdash \beta : \text{FiniteDim} \qquad \Gamma \vdash \bar{z} : \beta \rightsquigarrow \vec{\tau} \\ \Gamma, x : \alpha, x \rightsquigarrow \vec{v} \vdash \bar{y} : \beta \rightsquigarrow \vec{\mu} \\ \Gamma \vdash \bar{x} : \text{Maybe}(\alpha) \rightsquigarrow \sigma, \vec{\eta}\end{array}}{\Gamma \vdash \text{maybe}((\lambda x : \alpha \mapsto \bar{y}), \bar{z}, \bar{x}) : \beta \rightsquigarrow (1 - \sigma) \cdot \vec{\tau} + \sigma \cdot \vec{\mu}[\vec{v} \mapsto \vec{\eta}]} \tag{187}$$

$$\frac{\Gamma \vdash \bar{x} : \text{Maybe}(\alpha) \rightsquigarrow \sigma^0, \vec{\tau}}{\Gamma \vdash \text{exists}(\bar{x}) : \alpha \rightsquigarrow \vec{\tau}} \tag{188}$$

$$\frac{\Gamma \vdash \bar{x} : \text{List}(\alpha) \rightsquigarrow \sigma, \vec{\tau}}{\Gamma \vdash \text{length}(\bar{x}) : \mathbb{N} \rightsquigarrow \sigma} \qquad \frac{\Gamma \vdash \bar{x} : \text{List}(\alpha) \rightsquigarrow \sigma, \vec{\tau} \qquad \Gamma \vdash \bar{i} : \mathbb{N} \rightsquigarrow \iota}{\Gamma \vdash \text{nth}(\bar{x}, \bar{i}) : \alpha \rightsquigarrow \vec{\tau}(\iota)} \tag{189}$$

$$\frac{\begin{array}{c}\Gamma \vdash \bar{x} : \text{List}(\alpha \times \beta) \rightsquigarrow \sigma^0, \vec{\tau}^{+1}, \vec{\mu}^{+1} \\ (\Gamma, y : \alpha, y \rightsquigarrow \vec{\tau}) \text{ ctx}\end{array}}{\Gamma \vdash \text{List}(\pi_1)(\bar{x}) : \text{List}(\alpha) \rightsquigarrow \sigma^0, \vec{\tau}^{+1}} \tag{190}$$

$$\frac{\begin{array}{c}\Gamma \vdash \bar{x} : \text{List}(\alpha \times \beta) \rightsquigarrow \sigma^0, \vec{\tau}^{+1}, \vec{\mu}^{+1} \\ (\Gamma, y : \beta, y \rightsquigarrow \vec{\mu}) \text{ ctx}\end{array}}{\Gamma \vdash \text{List}(\pi_2)(\bar{x}) : \text{List}(\beta) \rightsquigarrow \sigma^0, \vec{\mu}^{+1}} \tag{191}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{Maybe}(\alpha \times \beta)) \rightsquigarrow \sigma^0, \upsilon^1, \vec{\tau}^{+1}, \vec{\mu}^{+1} \quad (\Gamma, y : \alpha, y \rightsquigarrow \vec{\tau})\ \mathrm{ctx}}{\Gamma \vdash \mathrm{List}(\mathrm{Maybe}(\pi_1)(\bar{x}) : \mathrm{List}(\mathrm{Maybe}(\alpha)) \rightsquigarrow \sigma^0, \upsilon^1, \vec{\tau}^{+1}} \tag{192}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{Maybe}(\alpha \times \beta)) \rightsquigarrow \sigma^0, \upsilon^1, \vec{\tau}^{+1}, \vec{\mu}^{+1} \quad (\Gamma, y : \beta, y \rightsquigarrow \vec{\mu})\ \mathrm{ctx}}{\Gamma \vdash \mathrm{List}(\mathrm{Maybe}(\pi_2))(\bar{x}) : \mathrm{List}(\mathrm{Maybe}(\beta)) \rightsquigarrow \sigma^0, \upsilon^1, \vec{\mu}^{+1}} \tag{193}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(\alpha) \rightsquigarrow \vec{\tau} \qquad (\mathrm{data}\ a \cong \alpha) \in \Gamma}{\Gamma \vdash \mathrm{List}(\mathrm{to}(a))(\bar{x}) : \mathrm{List}(a) \rightsquigarrow \vec{\tau}} \tag{194}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(a) \rightsquigarrow \vec{\tau} \qquad (\mathrm{data}\ a \cong \alpha) \in \Gamma}{\Gamma \vdash \mathrm{List}(\mathrm{from}(a))(\bar{x}) : \mathrm{List}(\alpha) \rightsquigarrow \vec{\tau}} \tag{195}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{List}(\alpha)) \rightsquigarrow \sigma^0, \delta^1, \vec{\tau}}{\Gamma \vdash \mathrm{List}(\mathrm{length})(\bar{x}) : \mathrm{List}(\mathbb{N}) \rightsquigarrow \sigma^0, \delta^1} \tag{196}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{Maybe}(\mathrm{List}(\alpha))) \rightsquigarrow \sigma^0, \iota^1, \delta^1, \vec{\tau}}{\Gamma \vdash \mathrm{List}(\mathrm{Maybe}(\mathrm{length}))(\bar{x}) : \mathrm{List}(\mathbb{N}) \rightsquigarrow \sigma^0, \iota^1, \delta^1}. \tag{197}$$

The sum operation is translated by using unrolling up to some constant maximum list size:

$$\frac{\beta \in \mathbb{N} \qquad \Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma \vdash \bar{x} : \mathrm{List}(\alpha) \rightsquigarrow \sigma, \tau}{\Gamma \vdash \mathrm{sum}(\bar{x}) : \alpha \rightsquigarrow \sum_{i=0}^{\beta} \mathrm{ind}_<(i, \sigma) \cdot \tau(i)} \tag{198}$$

$$\frac{\beta \in \mathbb{N} \qquad \Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{Maybe}(\alpha)) \rightsquigarrow \sigma^0, \iota^1, \tau^1}{\Gamma \vdash \mathrm{sum}(\bar{x}) : \alpha \rightsquigarrow \sum_{i=0}^{\beta} \mathrm{ind}_<(i, \sigma^0) \cdot \iota^1(i) \cdot \tau^1(i)} \tag{199}$$

$$\frac{\beta_1, \beta_2 \in \mathbb{N} \qquad \Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{List}(\alpha)) \rightsquigarrow \sigma^0, \delta^1, \tau^2}{\Gamma \vdash \mathrm{sum}(\bar{x}) : \alpha \rightsquigarrow \sum_{i=0}^{\beta_1} \mathrm{ind}_<(i, \sigma^0) \cdot \left(\sum_{j=0}^{\beta_2} \mathrm{ind}_<(j, \delta^1(i)) \cdot \tau^2(i, j) \right)} \tag{200}$$

$$\frac{\beta_1, \beta_2 \in \mathbb{N} \qquad \Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{List}(\mathrm{Maybe}(\alpha))) \rightsquigarrow \sigma^0, \delta^1, \iota^2, \tau^2}{\Gamma \vdash \mathrm{sum}(\bar{x}) : \alpha \rightsquigarrow \sum_{i=0}^{\beta_1} \mathrm{ind}_<(i, \sigma^0) \cdot \left(\sum_{j=0}^{\beta_2} \mathrm{ind}_<(j, \delta^1(i)) \cdot \iota^2(i, j) \cdot \tau^2(i, j) \right)}. \tag{201}$$

The rules above contain the first uses of the indicator function $\mathrm{ind}_<$ mentioned at the start of this subsection.

The following rules translate the map-related functions:

$$\frac{\Gamma \vdash \bar{m} : \mathrm{Map}(\alpha, \beta) \rightsquigarrow \sigma^0, \vec{\eta}^1, \iota^k, \vec{\tau}^{+k} \qquad \Gamma \vdash \bar{x} : \alpha \rightsquigarrow \vec{\theta}^0 \quad (\Gamma, y : \alpha, y \rightsquigarrow \vec{\eta}^0)\ \mathrm{ctx} \qquad (\Gamma, y : \beta, y \rightsquigarrow \vec{\tau})\ \mathrm{ctx} \qquad k = |\vec{\eta}| = |\vec{\theta}|}{\Gamma \vdash \mathrm{lookup}(\bar{x}, \bar{m}) : \mathrm{Maybe}(\beta) \rightsquigarrow \iota^k(\vec{\theta}), \vec{\tau}^{+k}(\vec{\theta})} \tag{202}$$

$$\frac{\Gamma \vdash \bar{m} : \mathrm{Map}(\alpha, \beta) \rightsquigarrow \sigma^0, \vec{\eta}^1, \vec{\tau} \quad (\Gamma, y : \alpha, y \rightsquigarrow \vec{\eta}^0)\ \mathrm{ctx}}{\Gamma \vdash \mathrm{keys}(\bar{m}) : \mathrm{List}(\alpha) \rightsquigarrow \sigma^0, \vec{\eta}^1} \tag{203}$$

$$\frac{\Gamma \vdash \bar{x} : \mathrm{Map}(\alpha, \beta \times \gamma) \rightsquigarrow \sigma^0, \vec{\tau}^1, \iota^\ell, \mu^{+\ell}, \kappa^{+\ell} \quad (\Gamma, y : \alpha, y \rightsquigarrow \vec{\tau}^0)\ \mathrm{ctx} \quad (\Gamma, y : \beta, y \rightsquigarrow \vec{\mu})\ \mathrm{ctx} \quad (\Gamma, y : \gamma, y \rightsquigarrow \vec{\kappa})\ \mathrm{ctx}}{\Gamma \vdash \mathrm{Map}(\pi_1)(\bar{x}) : \mathrm{Map}(\alpha, \beta) \rightsquigarrow \sigma^0, \vec{\tau}^1, \iota^\ell, \vec{\mu}^{+\ell}} \tag{204}$$

$$\frac{\begin{array}{c}\Gamma \vdash \bar{x} : \mathrm{Map}(\alpha, \beta \times \gamma) \rightsquigarrow \sigma^0, \vec{\tau}^1, \iota^\ell, \mu^{+\ell}, \kappa^{+\ell} \\ (\Gamma, y : \alpha, y \rightsquigarrow \vec{\tau}^0)\ \mathrm{ctx} \\ (\Gamma, y : \beta, y \rightsquigarrow \vec{\mu})\ \mathrm{ctx} \\ (\Gamma, y : \gamma, y \rightsquigarrow \vec{\kappa})\ \mathrm{ctx}\end{array}}{\Gamma \vdash \mathrm{Map}(\pi_1)(\bar{x}) : \mathrm{Map}(\alpha, \gamma) \rightsquigarrow \sigma^0, \vec{\tau}^1, \iota^\ell, \vec{\kappa}^{+\ell}} \tag{205}$$

$$\frac{\Gamma \vdash \bar{y} : \mathrm{Map}(\alpha, x) \rightsquigarrow \vec{\tau} \qquad (\mathrm{data}\ x \cong \beta) \in \Gamma}{\Gamma \vdash \mathrm{Map}(\mathrm{from}(x))(\bar{y}) : \mathrm{Map}(\alpha, \beta) \rightsquigarrow \vec{\tau}} \tag{206}$$

$$\frac{\Gamma \vdash \bar{y} : \mathrm{Map}(\alpha, \beta) \rightsquigarrow \vec{\tau} \qquad (\mathrm{data}\ x \cong \beta) \in \Gamma}{\Gamma \vdash \mathrm{Map}(\mathrm{to}(x))(\bar{y}) : \mathrm{Map}(\alpha, x) \rightsquigarrow \vec{\tau}} \tag{207}$$

$$\frac{\beta \in \mathbb{N} \qquad \Gamma \vdash \gamma : \mathrm{Num} \qquad \Gamma \vdash \bar{x} : \mathrm{Map}(\alpha, \gamma) \rightsquigarrow \sigma^0, \vec{\tau}^1, \iota^k, \mu^k}{\Gamma \vdash \mathrm{sum}(\bar{x}) : \gamma \rightsquigarrow \sum_{i=0}^{\beta} \mathrm{ind}_<(i, \sigma^0) \cdot \mu^k(\vec{\tau}^1(i))} \tag{208}$$

$$\frac{\begin{array}{c}\Gamma \vdash \bar{x} : \mathrm{Map}(\alpha, \mathrm{List}(\beta)) \rightsquigarrow \sigma^0, \vec{\tau}^1, \iota^k, \delta^k, \vec{\mu}^{+k} \\ \omega \in \mathbb{N} \qquad (\Gamma, y : \beta, y \rightsquigarrow \vec{\mu})\ \mathrm{ctx} \qquad (\Gamma, y : \alpha, y \rightsquigarrow \vec{\tau}^0)\ \mathrm{ctx}\end{array}}{\Gamma \vdash (\mathrm{sum} \circ \mathrm{Map}(\mathrm{length}))(\bar{x}) : \mathbb{N} \rightsquigarrow \sum_{i=0}^{\omega} \mathrm{ind}_<(i, \sigma^0) \cdot \delta^k(\vec{\tau}^1(i))} \tag{209}$$

$$\frac{\begin{array}{c}\omega \in \mathbb{N} \qquad \Gamma \vdash \beta : \mathrm{Num} \\ \Gamma \vdash \bar{k} : \alpha \rightsquigarrow \vec{\tau}^0 \\ \Gamma \vdash \bar{x} : \mathrm{List}(\mathrm{Map}(\alpha, \beta)) \rightsquigarrow \sigma^0, \gamma^1, \vec{\upsilon}^1, \iota^k, \mu^k\end{array}}{\Gamma \vdash (\mathrm{sum} \circ \mathrm{lookup}(\bar{k}))(x) : \beta \rightsquigarrow \sum_{i=0}^{\omega} \iota^k(i, \vec{\tau}^0) \cdot \mu^k(i, \vec{\tau}^0)}. \tag{210}$$

The following rule allows for translation of let expressions:

$$\frac{\Gamma \vdash \bar{z}[x \mapsto \bar{y}] : \beta \rightsquigarrow \vec{\tau}}{\Gamma \vdash (\mathrm{let}\ x : \alpha := \bar{y}; \bar{z}) : \beta \rightsquigarrow \vec{\tau}}. \tag{211}$$

The following rule allows for translation of inequalities:

$$\frac{\Gamma \vdash \alpha : \mathrm{Num} \qquad \Gamma \vdash \bar{x} : \alpha \rightsquigarrow \tau^0 \qquad \Gamma \vdash \bar{y} : \alpha \rightsquigarrow \mu^0}{\Gamma \vdash \bar{x} \leq \bar{y} : \mathrm{Prop} \rightsquigarrow \mathrm{ind}_<(\tau^0, \mu^0 + 1) = 1}. \tag{212}$$

The following rules allow for translation of quantified statements. The following rule allows for translating quantified statements over numeric types, with some constant upper bound:

$$\frac{\Gamma \vdash \alpha : \mathrm{Num} \qquad n \in \mathbb{N} \qquad \Gamma, x : \alpha, x \rightsquigarrow x_0 \vdash \phi : \mathrm{Prop} \rightsquigarrow \psi}{\Gamma \vdash (\forall x : \alpha, \phi) : \mathrm{Prop} \rightsquigarrow \forall < n.\ \psi} \tag{213}$$

$$\frac{\begin{array}{c}\Gamma \vdash \alpha : \mathrm{Finite} \qquad x \notin \mathrm{free}(\phi) \qquad (\mathrm{data}\ a \cong \alpha) \in \Gamma \\ \Gamma \vdash (\forall x : \alpha, \phi[y \mapsto \mathrm{to}(a, x)]) : \mathrm{Prop} \rightsquigarrow \psi\end{array}}{\Gamma \vdash (\forall y : a, \phi) : \mathrm{Prop} \rightsquigarrow \psi} \tag{214}$$

$$\frac{\begin{array}{rcl} & & x \notin \mathrm{free}(\phi) \\ \Gamma & \vdash & (\phi[y \mapsto \mathrm{nothing}]) \\ & \wedge & (\forall x : \alpha, \phi[y \mapsto \mathrm{just}(x)])) \\ & : & \mathrm{Prop} \rightsquigarrow \psi\end{array}}{\Gamma \vdash (\forall y : \mathrm{Maybe}(\alpha), \phi) : \mathrm{Prop} \rightsquigarrow \psi} \qquad \frac{\begin{array}{rcl} & & x \notin \mathrm{free}(\phi) \\ \Gamma & \vdash & ((\forall x : \alpha, \phi[y \mapsto \iota_1(x)]) \\ & \wedge & (\forall x : \beta, \phi[y \mapsto \iota_2(x)])) \\ & : & \mathrm{Prop} \rightsquigarrow \psi\end{array}}{\Gamma \vdash (\forall y : \alpha \oplus \beta, \phi) : \mathrm{Prop} \rightsquigarrow \psi} \tag{215}$$

$$\frac{x, z \notin \mathrm{free}(\phi) \qquad \Gamma \vdash (\forall x : \alpha, \forall z : \beta, \phi[y \mapsto (x, z)]) : \mathrm{Prop} \rightsquigarrow \psi}{\Gamma \vdash (\forall y : \alpha \times \beta, \phi) : \mathrm{Prop} \rightsquigarrow \psi} \tag{216}$$

$$\frac{\Gamma \vdash \alpha : \mathrm{Num} \qquad n \in \mathbb{N} \qquad \Gamma, x : \alpha, x \rightsquigarrow x_0 \vdash \phi : \mathrm{Prop} \rightsquigarrow \psi}{\Gamma \vdash (\exists x : \alpha, \phi) : \mathrm{Prop} \rightsquigarrow \exists < n.\ \psi} \tag{217}$$

$$\frac{\Gamma \vdash \alpha : \text{Finite} \qquad x \notin \text{free}(\phi) \qquad (\text{data } a \cong \alpha) \in \Gamma \qquad \Gamma \vdash (\exists x : \alpha, \phi[y \mapsto \text{to}(a, x)]) : \text{Prop} \rightsquigarrow \psi}{\Gamma \vdash (\exists y : a, \phi) : \text{Prop} \rightsquigarrow \psi} \tag{218}$$

$$\frac{\begin{array}{rcl} & & x \notin \text{free}(\phi) \\ \Gamma & \vdash & (\phi[y \mapsto \text{nothing}] \\ & \vee & (\exists x : \alpha, \phi[y \mapsto \text{just}(x)])) \\ & : & \text{Prop} \rightsquigarrow \psi \end{array}}{\Gamma \vdash (\exists y : \text{Maybe}(\alpha), \phi) : \text{Prop} \rightsquigarrow \psi} \qquad \frac{\begin{array}{rcl} & & x \notin \text{free}(\phi) \\ \Gamma & \vdash & ((\exists x : \alpha, \phi[y \mapsto \iota_1(x)]) \\ & \vee & (\exists x : \beta, \phi[y \mapsto \iota_2(x)])) \\ & : & \text{Prop} \rightsquigarrow \psi \end{array}}{\Gamma \vdash (\exists y : \alpha \oplus \beta, \phi) : \text{Prop} \rightsquigarrow \psi} \tag{219}$$

$$\frac{x, z \notin \text{free}(\phi) \qquad \Gamma \vdash (\exists x : \alpha, \exists z : \beta, \phi[y \mapsto (x, z)]) : \text{Prop} \rightsquigarrow \psi}{\Gamma \vdash (\exists y : \alpha \times \beta, \phi) : \text{Prop} \rightsquigarrow \psi} \tag{220}$$

$$\frac{f, n \notin \text{free}(\phi) \qquad \Gamma \vdash (\exists f : \mathbb{N} \to \alpha, \exists n : \mathbb{N}, \phi[\text{length}(x) \mapsto n, \text{nth}(x) \mapsto f]) : \text{Prop} \rightsquigarrow \psi}{\Gamma \vdash (\exists x : \text{List}(\alpha), \phi) : \text{Prop} \rightsquigarrow \psi} \tag{221}$$

$$\frac{\begin{array}{rcl} & & s, f \in \text{free}(\phi) \\ \Gamma & \vdash & (\exists s : \text{List}(\alpha), \exists f : \alpha \to \text{Maybe}(\beta), \\ & & \phi[\text{keys}(x) \mapsto s, \text{lookup}(\_, x) \mapsto f(\_)] \\ & \wedge & \forall y : \alpha, (\exists i : \mathbb{N}, i +_{\mathbb{N}} 1_{\mathbb{N}} \leq \text{length}(s) \wedge \text{nth}(s, i) = y) \leftrightarrow \neg(f(y) = \text{nothing})) \\ & : & \text{Prop} \rightsquigarrow \psi \end{array}}{\Gamma \vdash (\exists x : \text{Map}(\alpha, \beta), \phi) : \text{Prop} \rightsquigarrow \psi} \tag{222}$$

$$\frac{\beta_0, ..., \beta_j \in \mathbb{N} \qquad \Gamma, f : \alpha \to \beta, f \rightsquigarrow x^j_{k-1}, ..., x^j_0 \vdash \phi : \text{Prop} \rightsquigarrow \psi}{\Gamma \vdash (\exists f : \alpha \to \beta, \phi) : \text{Prop} \rightsquigarrow \overbrace{\exists_f < \beta_0 (< \beta_1, ..., < \beta_j)}^{k \text{ times}}. \psi}. \tag{223}$$

That completes the sequent calculus definition of the process of translating OSL specs into Σ^1_1 formulas. An OSL spec, by definition, is a derivable sequent of the form:

$$\Gamma \vdash \phi : \alpha \to \text{Prop}, \tag{224}$$

where $\Gamma \vdash \alpha : \text{Type}$. The process of translating an OSL spec into a Σ^1_1 formula is the process of finding a context $\Sigma \supseteq \Gamma$ and a derivation of a sequent of the form

$$\Sigma, x : \alpha, x \rightsquigarrow \vec{v} \vdash \phi(x) : \text{Prop} \rightsquigarrow \psi, \tag{225}$$

where ψ is the translation itself. $x \rightsquigarrow \vec{v}$ explains how the inputs to the relation ϕ are mapped onto the free variables in the translation formula ψ.

Proving that the translation process can always be performed would be desirable. That is, for any OSL spec, there is a translation, and a derivation of that translation can be found by an efficient algorithm taking only the spec as input, and that translation has the correct denotation within the domain of application. Because OSL translation is so complex, proving these theorems would appear onerous and mistake prone unless performed as formally verified proofs. Therefore, this paper does not attempt to prove that the translation process has any of the desired properties. This is left for future formal verification work, which may uncover gaps and/or errors in the definition of the translation calculus. If so, any such gaps and/or errors may be corrected, because by design, OSL is just a more convenient notation for writing Σ^1_1 formulas.

References

[1] Wilfrid Hodges. *Model Theory.* Stanford Encyclopedia of Philosophy, 2020. https://plato.stanford.edu/entries/model-theory/

[2] George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and Logic* (5th ed.). Cambridge: Cambridge University Press, 2007. https://www.cambridge.org/core/books/computability-and-logic/440B4178B7CBF1C241694233716AB271

[3] The Electric Coin Company. *The halo2 Book.* 2021. https://zcash.github.io/halo2/index.html

[4] The Electric Coin Company. *halo2.* 2022. https://github.com/zcash/halo2

[5] Orbis Labs, Morgan Thomas. *Arithmetization of* Σ_1^1 *Relations in Halo 2.* Cryptology ePrint Archive, Report 2022/777. https://eprint.iacr.org/2022/777.pdf

[6] Collin Chin, Howard Wu, Raymond Chu, Alessandro Coglio, Eric McCarthy, and Eric Smith. *Leo: A Programming Language for Formally Verified, Zero-Knowledge Applications.* Cryptology ePrint Archive, Report 2021/651. https://eprint.iacr.org/2021/651.pdf

[7] O(1) Labs. *Snarky.* GitHub. Accessed May 2022. https://github.com/o1-labs/snarky

[8] Filecoin. *Introducing Lurk: A Programming Language for Recursive zk-SNARKs.* Filecoin.io Blog. April 2022. https://filecoin.io/blog/posts/introducing-lurk-a-programming-language-for-recursive-zk-snarks/

[9] Edward Z. Yang. *Logitext Tutorial.* Accessed May 2022. http://logitext.mit.edu/logitext.fcgi/tutorial

[10] Maksym Petkus. *Why and How zk-SNARK Works: Definitive Explanation.* arXiv:1906.07221. https://arxiv.org/pdf/1906.07221.pdf