Active Type-Checking and Translation

Cyrus Omar Nathan Fulton Jonathan Aldrich

School of Computer Science
Carnegie Mellon University
{comar, jonathan.aldrich}@cs.cmu.edu

Abstract

Researchers and domain experts typically describe new language-based abstractions as extensions of some existing language, but most statically-typed languages are actually *monolithic*: they do not give users the ability to specify the semantics of new core types and operators from within. This choice has led to a proliferation of mutually incompatible standalone languages, each built around a small collection of privileged constructs. An alternative to this *language-oriented* approach is to work in an extensible programming language where the compile-time behaviors determining the functionality of new core constructs are specified and implemented modularly within user libraries. Designing such a *library-oriented* extensibility mechanism for statically-typed languages that is both safe and expressive is non-trivial. This paper introduces a mechanism called active type-checking and translation (AT&T) that aims to address these challenges. By relying upon type-level computation in a novel way, AT&T admits user specification of a wide range of compile-time behaviors over a fixed grammar in a safe and modular manner. We discuss two points in the design space: (1) a simple calculus designed to distill the essential concepts and admit formal safety theorems, and (2) a fully-implemented language called Ace that we use to demonstrate the expressive power of AT&T in several real-world domains, including scientific computing, security, functional programming and object-oriented programming.

Categories and Subject Descriptors D.3.2 [*Programming Languages*]: Language Classifications—Extensible Languages; D.3.4 [*Programming Languages*]: Processors—Compilers; F.3.1 [*Logics & Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs—Specification Techniques

1. Introduction

Programming languages have historically been specified and implemented monolithically. To introduce new primitive constructs, researchers or domain experts have developed a new language or a dialect of an existing language, with the help of tools like domain-specific language frameworks and compiler generators [**?**]. Unfortunately, taking a so-called *language-oriented approach* [**?**], where different languages are used for different components of an application, can lead to problems at language boundaries: a library's external interface must only rely on constructs that can be expressed in all possible calling languages. This means that specialized invariants cannot be checked statically, decreasing reliability and performance. It also often requires that developers generate verbose and unnatural "glue" code, defeating a primary purpose of specialized languages: hiding these low-level details from end-user developers.

Extensible programming languages promise to decrease the need for new standalone languages by providing more granular, language-based support for introducing new primitive constructs (that is, constructs that cannot be adequately expressed in terms of existing syntactic forms and primitive operations.) Developers would gain the freedom to choose those constructs that are most appropriate for their application domain and development discipline. Researchers would gain the ability to distribute new constructs for evaluation by a broader development community without requiring the approval of maintainers of mainstream languages, who are naturally risk-averse and uninterested in niche domains.

A significant challenge that faces language extensibility mechanisms is in maintaining the overall safety properties of the language and compilation process in the presence of arbitrary combinations of user extensions. The mechanism must ensure that basic metatheoretic and global safety guarantees of the language cannot be weakened, that extensions are safely composable, and that type checking and compilation remains decidable. The correctness of an extension itself should be modularly verifiable, so that its users can rely on it for verifying and compiling their own code. These are the issues that we seek to address in this work.

The approach we describe, *active type-checking and compilation* (AT&T), makes use of type-level computation in a novel way. To review, in languages supporting type-level computation, the syntactic class of types is not simply declarative. Instead, it forms a programming language itself (the *type-level language*). Types themselves are one kind of value in this language, but there can be many others. To ensure the safety of type-level computations, *kinds* classify type-level terms, just as types classify expression-level terms. The simplest example of a language featuring type-level computation is Girard's System F_ω [**?**]. In F_ω, types have kind $\star$ and type-level functions have kinds like $\star \rightarrow \star$. A growing number of implemented languages now feature more sophisticated type-level languages (see Section 5). We emphasize that type-level computation occurs during compilation, rather than at run-time, because type-level terms that are used where types would normally be expected must be reduced to normal form before type-checking can proceed.

In this work, we wish to allow extensions to strengthen the static semantics of our language. Naturally, extension specifications will also need to be evaluated during compilation and manipulate representations of types. This observation suggests that the type-level language may be able to serve directly as a specification language. In this paper, we show that this is indeed the case. By introducing some new constructs at the type-level, developers can specify the semantics of operators associated with newly-introduced families of primitive types with type-level functions. The compiler front-end invokes these functions to synthesize types for and assign meanings to expressions, by translation into a *typed internal language*. Unlike conventional metaprogramming systems, these *type-level specifications* do not directly manipulate or rewrite expressions. Instead, they examine and manipulate the types of these expressions. By using a sufficiently constraining kind system and incorporating techniques from typed compilation into the type-level language directly, the global safety properties of the language and compilation process can be guaranteed. In other words, users can only *increase* the safety of the language.

We focus on extending the static semantics of a language with a fixed, though flexible, grammar. Techniques for extensible parsing have been proposed in the past (e.g. [**?**]), and we conjecture that these can be made compatible with the mechanism described in this paper with some simple modifications, but we do not discuss this further here. We also focus on *functional*, rather than declarative, specifications of language constructs. Extracting a compiler from a declarative language specification (e.g. in Twelf [**?**]) has not yet been shown practical, but we note that a future mechanism of this sort could safely target a language implementing the mechanism we discuss here.

The organization and key contributions of this paper are:

- In Section 2, we develop a core calculus, λ_{A}, and give simple examples of language features that can be expressed with it. We formalize the compiler front-end and state several lemmas that lead to useful safety theorems for the compiler and language as a whole. We show how AT&T requires that the language provide a solution to a type-level variant of Wadler's expression problem **??**.
- In Section 3, we briefly introduce the Ace programming language, which is based fundamentally on an elaboration of AT&T that supports a richer set of syntactic forms and a variant of type inference. It uses object-oriented inheritance to solve the type-level expression problem. A number of practical extensions have been written using Ace, including a complete implementation of the OpenCL type system (based on C99) as a library.
- In Section 4, we briefly describe another point in the design space, a language design we call Birdie. Birdie lifts an extension of the Gallina language, used by the Coq proof assistant, into the type level (leading to a language with dependent kinds). This additional complexity allows for full proofs of correctness for type-level specifications, and can allow proofs soundness of functional specifications against conventional inductive specifications. The expression problem is solved using a constrained formulation of open data types, rather than using object-oriented inheritance.
- In Section 5 we compare AT&T to previous work on extensible languages and compilers, metaprogramming systems and formal specification languages and conclude in Section 6 with a discussion of future work.

$$
\begin{array}{l}
\mathsf{family}\ \mathrm{NAT}[\mathtt{Unit}]\ ::\ \lambda\mathbf{self} : \mathtt{Type} \in \mathrm{NAT}.\mathsf{itype}(\mathtt{int})\ \{(\\
\quad \mathbf{z} = \lambda\mathbf{self} : \mathtt{Type}.\mathsf{const}([\![\mathsf{iterm}(0) \sim \mathbf{self}]\!]); \\
\quad \mathbf{s} = \lambda\mathbf{self} : \mathtt{Type}.\mathsf{op}(\lambda\mathbf{d1} : \mathtt{Den}. \\
\qquad \mathsf{if\ typeof}\ \mathbf{d1} \equiv \mathbf{self}\ \mathsf{then}\ [\![\mathsf{valof}\ \mathbf{d1} \sim \mathbf{self}]\!]\ \mathsf{else\ err})) : \\
\Theta\}\ \mathsf{in\ let\ nat} = \mathsf{type}[()] \in \textsc{nat}\ \mathsf{in}\ x
\end{array}
$$

Figure 1. Specifying natural numbers in λ_{A}. This term is referred to as ψ_{nat}.

2. Type-Level Specifications in λ_{A}

2.1 Example: Natural Numbers in λ_{A}

We begin with a simple calculus with no primitive notion of natural numbers, nor any more general notion of an inductive data type. We can, however, concretely specify both the static and dynamic semantics of natural numbers, including the natural recursor of Gödel's T [**?**], using type-level specifications. Let us begin in Figure 1 with the type **nat** and its constructors, **z** and **s**.

$$\psi_1 := [\mathsf{program}(\mathsf{nat}.\mathbf{s}\langle\rangle(\mathsf{nat}.\mathbf{z}\langle\rangle()))/\psi_{\text{nat}}]$$

$$\psi_1 \Longrightarrow [\![1 + 0 \sim \mathbf{nat}]\!]$$

expressions	e	$::=$	$\mathrm{x} \mid \lambda \mathrm{x}{:}\psi.e \mid e_1\ e_2 \mid \psi_{\text{type}}.\mathbf{op}\langle \psi_1, \ldots, \psi_m \rangle() \mid \psi_{\text{type}}.\mathbf{op}\langle \psi_1, \ldots, \psi_m \rangle(e_1, \ldots, e_n)$
type-level specifications	ψ	$::=$	$\mathbf{t} \mid \lambda \mathbf{t}_1{:}\kappa_1,\ \ldots\ ,\mathbf{t}_n{:}\kappa_n.\psi \mid \psi(\psi_1, \ldots, \psi_n) \mid \mathsf{if}\ \psi_0 \equiv \psi_1\ \mathsf{then}\ \psi_2\ \mathsf{else}\ \psi_3$
standard terms		$\mid$	$() \mid (\psi_1, \psi_2) \mid \mathsf{fst}\ \psi \mid \mathsf{snd}\ \psi$
type families		$\mid$	$\mathsf{family}\ \textsc{fam}[\kappa_{\text{idx}}]\ ::\ \psi_{\text{rep}}\ \{\theta : \Theta\}\ \mathsf{in}\ \psi \mid \mathsf{famcase}\ \psi\ \mathsf{of}\ \phi\ \mathsf{then}\ \psi_0\ \mathsf{else}\ \psi_1$
types		$\mid$	$\mathsf{type}[\psi_{\text{idx}}] \in \phi \mid \mathsf{idxof}\ \psi_{\text{type}} \mid \mathsf{repof}\ \psi_{\text{type}}$
operator definitions		$\mid$	$\mathsf{const}(\psi) \mid \mathsf{op}(\psi)$
denotations		$\mid$	$[\![\psi_{\text{iterm}} \sim \psi_{\text{type}}]\!] \mid \mathsf{valof}\ \psi_{\text{den}} \mid \mathsf{typeof}\ \psi_{\text{den}} \mid \mathsf{err}$
internal language		$\mid$	$\mathsf{iterm}(\mu) \mid \mathsf{itype}(\delta)$
programs		$\mid$	$\mathsf{program}(e)$
family specifications	ϕ	$::=$	$\textsc{fam} \mid \mathsf{family}\ [\kappa_{\text{idx}}]\ ::\ \psi_{\text{rep}}\ \{\theta : \Theta\}$
operator lists	θ	$::=$	$\cdot \mid \theta; \mathbf{op} = \psi$
kinds	κ	$::=$	$\kappa_1, \ldots, \kappa_n \to \kappa \mid \texttt{Unit} \mid \kappa_1 \times \kappa_2 \mid \texttt{Type} \in \eta \mid \texttt{Type} \mid \texttt{Op}_n \mid \texttt{Den}[\eta] \mid \texttt{Den} \mid \texttt{IType} \mid \texttt{ITerm} \mid \texttt{Program}$
family signatures	η	$::=$	$\texttt{family}[\kappa_{\text{idx}}]\{\Theta\}$
operator list signatures	Θ	$::=$	$\cdot \mid \Theta; \mathbf{op} : \kappa$
internal terms	μ	$::=$	$\mathrm{u} \mid \lambda \mathrm{u} :: \delta.\mu \mid \mathsf{fix}\ \mathrm{f} :: \delta.\mu \mid \mu_1\ \mu_2 \mid \mathsf{if}\ \mu_0 \equiv \mu_1\ \mathsf{then}\ \mu_2\ \mathsf{else}\ \mu_3$
		$\mid$	$n \mid \mu_1 + \mu_2 \mid (\mu_1, \mu_2) \mid \mathsf{fst}\ \mu \mid \mathsf{snd}\ \mu \mid \uparrow(\psi)$
internal types	δ	$::=$	$\delta_1 \to \delta_2 \mid \texttt{int} \mid \delta_1 \times \delta_2 \mid \uparrow(\psi)$

Figure 2. Syntax of λ_{A}. $\mathbb{Z}$ denotes integer literals and LABEL denotes any label.

$$\frac{\overbrace{\cdot \vdash \psi_{\text{prog}} : \texttt{Program}}^{\text{kind checking}} \qquad \overbrace{\Xi \vdash \psi_{\text{prog}} \curlyvee_{\text{ok}} \mathsf{program}(e)}^{\text{specification normalization}} \qquad \overbrace{\cdot \vdash e \longrightarrow [\![\mu \sim \mathsf{type}[\psi_{\text{idx}}] \in \phi]\!]}^{\text{verification \& translation}}}{\psi_{\text{prog}} \Longrightarrow [\![\mu \sim \mathsf{type}[\psi_{\text{idx}}] \in \phi]\!]}$$

Figure 3. Central compilation judgement of λ_{A}.

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf af-

$\boxed{\Sigma\;\Delta \vdash \psi : \kappa}$ $\quad \Sigma ::= \cdot \mid \Sigma, \textsc{fam} : \texttt{family}[\kappa_{\text{idx}}]\{\Theta\} \quad \Delta ::= \cdot \mid \Delta, \mathbf{t} : \kappa$

$$\frac{\mathbf{t} : \kappa \in \Delta}{\Sigma\;\Delta \vdash \mathbf{t} : \kappa}\;(\text{var}_\psi) \qquad \frac{}{\Sigma\;\Delta \vdash () : \texttt{Unit}}\;(\text{unit}_\psi) \qquad \frac{\Sigma\;\Delta \vdash \psi_1 : \kappa_1 \quad \Sigma\;\Delta \vdash \psi_2 : \kappa_2}{\Sigma\;\Delta \vdash (\psi_1, \psi_2) : \kappa_1 \times \kappa_2}\;(\times_\psi) \qquad \frac{\Sigma\;\Delta \vdash \psi : \kappa_1 \times \kappa_2}{\Sigma\;\Delta \vdash \mathsf{fst}\;\psi : \kappa_1}\;(\text{fst}_\psi)$$

$$\frac{\Sigma\;\Delta \vdash \psi : \kappa_1 \times \kappa_2}{\Sigma\;\Delta \vdash \mathsf{snd}\;\psi : \kappa_2}\;(\text{snd}_\psi) \qquad \frac{\Sigma \vdash \kappa_1 \quad \cdots \quad \Sigma \vdash \kappa_n \quad \Sigma\;\Delta, \mathbf{t}_1{:}\kappa_1,\; \ldots\;, \mathbf{t}_n{:}\kappa_n \vdash \psi : \kappa}{\Sigma\;\Delta \vdash \lambda \mathbf{t}_1{:}\kappa_1,\; \ldots\;, \mathbf{t}_n{:}\kappa_n . \psi : \kappa_1, \ldots, \kappa_n \to \kappa}\;(\lambda_\psi)$$

$$\frac{\Sigma\;\Delta \vdash \psi : \kappa_1, \ldots, \kappa_n \to \kappa \quad \Sigma\;\Delta \vdash \psi_1 : \kappa_1 \quad \cdots \quad \Sigma\;\Delta \vdash \psi_n : \kappa_n}{\Sigma\;\Delta \vdash \psi(\psi_1, \ldots, \psi_n) : \kappa}\;(\text{app}_\psi)$$

$$\frac{\Sigma\;\Delta \vdash \psi_0 : \kappa_1 \quad \Sigma\;\Delta \vdash \psi_1 : \kappa_1 \quad \Sigma\;\Delta \vdash \psi_2 : \kappa_2 \quad \Sigma\;\Delta \vdash \psi_3 : \kappa_2}{\Sigma\;\Delta \vdash \mathsf{if}\;\psi_0 \equiv \psi_1\;\mathsf{then}\;\psi_2\;\mathsf{else}\;\psi_3 : \kappa_2}\;(\text{cond}_\psi)$$

$$\frac{\Sigma, \textsc{fam} : \texttt{family}[\kappa_{\text{idx}}]\{\Theta\}\;\Delta \vdash \mathsf{family}\;[\kappa_{\text{idx}}]\;::\;\psi_{\text{rep}}\;\{\theta : \Theta\} : \texttt{family}[\kappa_{\text{idx}}]\{\Theta\} \quad \Sigma, \textsc{fam} : \texttt{family}[\kappa_{\text{idx}}]\{\Theta\}\;\Delta \vdash \psi : \kappa}{\Sigma\;\Delta \vdash \mathsf{family}\;\textsc{fam}[\kappa_{\text{idx}}]\;::\;\psi_{\text{rep}}\;\{\theta : \Theta\}\;\mathsf{in}\;\psi : \kappa}\;(\text{fambind})$$

$$\frac{\Sigma\;\Delta \vdash \psi_{\text{type}} : \texttt{Type} \quad \Sigma\;\Delta \vdash \phi : \eta \quad \Sigma\;\Delta \vdash \psi_0 : \texttt{Type} \in \eta \to \kappa \quad \Sigma\;\Delta \vdash \psi_1 : \kappa}{\Sigma\;\Delta \vdash \mathsf{famcase}\;\psi_{\text{type}}\;\mathsf{of}\;\phi\;\mathsf{then}\;\psi_0\;\mathsf{else}\;\psi_1 : \kappa}\;(\text{famcase-T})$$

$$\frac{\Sigma\;\Delta \vdash \psi_{\text{den}} : \texttt{Den} \quad \Sigma\;\Delta \vdash \phi : \eta \quad \Sigma\;\Delta \vdash \psi_0 : \texttt{Den}[\eta] \to \kappa \quad \Sigma\;\Delta \vdash \psi_1 : \kappa}{\Sigma\;\Delta \vdash \mathsf{famcase}\;\psi_{\text{den}}\;\mathsf{of}\;\phi\;\mathsf{then}\;\psi_0\;\mathsf{else}\;\psi_1 : \kappa}\;(\text{famcase-D})$$

$$\frac{\Sigma\;\Delta \vdash \phi : \texttt{family}[\kappa_{\text{idx}}]\{\Theta\} \quad \Sigma\;\Delta \vdash \psi_{\text{idx}} : \kappa_{\text{idx}}}{\Sigma\;\Delta \vdash \mathsf{type}[\psi_{\text{idx}}] \in \phi : \texttt{Type} \in \texttt{family}[\kappa_{\text{idx}}]\{\Theta\}}\;(\text{Type}_I) \qquad \frac{\Sigma\;\Delta \vdash \psi : \texttt{Type} \in \eta}{\Sigma\;\Delta \vdash \psi : \texttt{Type}}\;(\text{Type-}\subseteq)$$

$$\frac{\Sigma\;\Delta \vdash \psi : \texttt{Type} \in \texttt{family}[\kappa_{\text{idx}}]\{\Theta\}}{\Sigma\;\Delta \vdash \mathsf{idxof}\;\psi : \kappa_{\text{idx}}}\;(\text{idxof}) \qquad \frac{\Sigma\;\Delta \vdash \psi : \texttt{Type}}{\Sigma\;\Delta \vdash \mathsf{repof}\;\psi : \texttt{IType}}\;(\text{repof}) \qquad \frac{\Sigma\;\Delta \vdash \psi : \texttt{Den}}{\Sigma\;\Delta \vdash \mathsf{const}(\psi) : \texttt{Op}_0}\;(\text{const})$$

$$\frac{\Sigma\;\Delta \vdash \psi : \texttt{Den}, \overset{n}{\cdots}, \texttt{Den} \to \texttt{Den}}{\Sigma\;\Delta \vdash \mathsf{op}(\psi) : \texttt{Op}_n}\;(n\text{-op}) \qquad \frac{\Sigma\;\Delta \vdash \psi_{\text{iterm}} : \texttt{ITerm} \quad \Sigma\;\Delta \vdash \psi_{\text{type}} : \texttt{Type} \in \eta}{\Sigma\;\Delta \vdash [\![\psi_{\text{iterm}} \sim \psi_{\text{type}}]\!] : \texttt{Den}[\eta]}\;(\text{Den}_I)$$

$$\frac{\Sigma\;\Delta \vdash \psi_{\text{iterm}} : \texttt{ITerm} \quad \Sigma\;\Delta \vdash \psi_{\text{type}} : \texttt{Type}}{\Sigma\;\Delta \vdash [\![\psi_{\text{iterm}} \sim \psi_{\text{type}}]\!] : \texttt{Den}}\;(\text{Den}_I\text{-}\subseteq) \qquad \frac{\Sigma\;\Delta \vdash \psi : \texttt{Den}[\eta]}{\Sigma\;\Delta \vdash \psi : \texttt{Den}}\;(\text{Den-}\subseteq) \qquad \frac{}{\Sigma\;\Delta \vdash \mathsf{err} : \texttt{Den}}\;(\text{err})$$

$$\frac{\Sigma\;\Delta \vdash \psi : \texttt{Den}[\eta]}{\Sigma\;\Delta \vdash \mathsf{typeof}\;\psi : \texttt{Type} \in \eta}\;(\text{typeof}) \qquad \frac{\Sigma\;\Delta \vdash \psi : \texttt{Den}[\eta]}{\Sigma\;\Delta \vdash \mathsf{valof}\;\psi : \texttt{ITerm}}\;(\text{valof})$$

$$\frac{\Sigma\;\Delta\;\cdot \vdash \mu}{\Sigma\;\Delta \vdash \mathsf{iterm}(\mu) : \texttt{ITerm}}\;(\text{iterm}) \qquad \frac{\Sigma\;\Delta \vdash \delta}{\Sigma\;\Delta \vdash \mathsf{itype}(\delta) : \texttt{IType}}\;(\text{itype}) \qquad \frac{\Sigma\;\Delta\;\cdot \vdash e}{\Sigma\;\Delta \vdash \mathsf{program}(e) : \texttt{Program}}\;(\text{program})$$

$\boxed{\Sigma\;\Delta \vdash \phi : \eta}$

$$\frac{\textsc{fam} : \eta \in \Sigma}{\Sigma\;\Delta \vdash \textsc{fam} : \eta}\;(\text{var}_\phi) \qquad \frac{\Sigma \vdash \texttt{family}[\kappa_{\text{idx}}]\{\Theta\} \quad \Sigma\;\Delta \vdash \psi_{\text{rep}} : \kappa_{\text{idx}} \to \texttt{IType} \quad \Sigma\;\Delta \vdash \theta : \Theta}{\Sigma\;\Delta \vdash \mathsf{family}\;[\kappa_{\text{idx}}]\;::\;\psi_{\text{rep}}\;\{\theta : \Theta\} : \texttt{family}[\kappa_{\text{idx}}]\{\Theta\}}\;(\text{family}_I)$$

$\boxed{\Sigma\;\Delta \vdash \theta : \Theta}$

$$\frac{}{\Sigma\;\Delta \vdash \cdot : \cdot}\;(\text{no-ops}) \qquad \frac{\Sigma, \textsc{fam} : \eta\;\Delta \vdash \theta : \Theta \quad \Sigma, \textsc{fam} : \eta\;\Delta \vdash \psi : \texttt{Type} \in \eta \to \kappa}{\Sigma, \textsc{fam} : \eta\;\Delta \vdash \theta; \mathbf{id} = \psi : \Theta; \mathbf{id} : \kappa}\;(\text{op-seq})$$

Figure 4. Kind checking rules for λ_{A}. All contexts have standard substructural properties (not shown).

$\boxed{\Sigma \vdash \kappa}$ $\boxed{\Sigma \vdash \kappa \text{ simple}}$

$$\frac{\Sigma \vdash \kappa_1 \quad \cdots \quad \Sigma \vdash \kappa_n \quad \Sigma \vdash \kappa}{\Sigma \vdash \kappa_1, \ldots, \kappa_n \to \kappa}\ (\to_\kappa) \qquad \frac{\Sigma \vdash \kappa_1 \quad \Sigma \vdash \kappa_2}{\Sigma \vdash \kappa_1 \times \kappa_2}\ (\times_\kappa) \qquad \frac{\Sigma \vdash \kappa \text{ simple}}{\Sigma \vdash \kappa}\ (\text{simple})$$

$$\frac{\Sigma \vdash \kappa_1 \text{ simple} \quad \Sigma \vdash \kappa_2 \text{ simple}}{\Sigma \vdash \kappa_1 \times \kappa_2 \text{ simple}}\ (\times_\kappa\text{-simple}) \qquad \frac{}{\Sigma \vdash \texttt{Unit} \text{ simple}}\ (\text{Unit}_\kappa) \qquad \frac{}{\Sigma \vdash \texttt{Type} \text{ simple}}\ (\text{Type-}\subseteq_\kappa)$$

$$\frac{\Sigma \vdash \eta}{\Sigma \vdash \texttt{Type} \in \eta \text{ simple}}\ (\text{Type}_\kappa) \qquad \frac{}{\Sigma \vdash \texttt{Den} \text{ simple}}\ (\text{Den-}\subseteq_\kappa) \qquad \frac{\Sigma \vdash \eta}{\Sigma \vdash \texttt{Den}[\eta] \text{ simple}}\ (\text{Den}_\kappa) \qquad \frac{}{\Sigma \vdash \texttt{IType} \text{ simple}}\ (\text{IType}_\kappa)$$

$$\frac{}{\Sigma \vdash \texttt{ITerm} \text{ simple}}\ (\text{ITerm}_\kappa) \qquad \frac{}{\Sigma \vdash \texttt{Program} \text{ simple}}\ (\text{Program}_\kappa) \qquad \frac{}{\Sigma \vdash \texttt{Op}_n \text{ simple}}\ (n\text{-Op}_\kappa)$$

$\boxed{\Sigma \vdash \eta}$

$$\frac{\Sigma \vdash \kappa_{\text{idx}} \text{ simple} \quad \Sigma \vdash \Theta}{\Sigma \vdash \texttt{family}[\kappa_{\text{idx}}]\{\Theta\}}\ (\text{family}_\eta)$$

$\boxed{\Sigma \vdash \Theta}$

$$\frac{}{\Sigma \vdash \cdot}\ (\text{No-Ops}) \qquad \frac{\Sigma \vdash \Theta \quad \Sigma \vdash \kappa_1, \ldots, \kappa_m \to \texttt{Op}_n}{\Sigma \vdash \Theta; \textbf{id} : \kappa_1, \ldots, \kappa_m \to \texttt{Op}_n}\ ((m,n)\text{-Op-Seq})$$

$\boxed{\Sigma\ \Delta\ \Omega \vdash e}$ $\quad \Omega ::= \cdot \mid \Omega, \mathrm{x}$

$$\frac{\mathrm{x} \in \Omega}{\Sigma\ \Delta\ \Omega \vdash \mathrm{x}}\ (\text{var-form}_e) \qquad \frac{\Sigma\ \Delta \vdash \psi : \texttt{Type} \quad \Sigma\ \Delta\ \Omega, \mathrm{x} \vdash e}{\Sigma\ \Delta\ \Omega \vdash \lambda \mathrm{x}{:}\psi.e}\ (\lambda\text{-form}_e) \qquad \frac{\Sigma\ \Delta\ \Omega \vdash e_1 \quad \Sigma\ \Delta\ \Omega \vdash e_2}{\Sigma\ \Delta\ \Omega \vdash e_1\ e_2}\ (\text{app-form}_e)$$

$$\frac{\Sigma\ \Delta \vdash \psi_{\text{type}} : \texttt{Type} \in \texttt{family}[\kappa_{\text{idx}}]\{\Theta\} \quad \textbf{id} : \kappa_1, \ldots, \kappa_m \to \texttt{Op}_0 \in \Theta \quad \Sigma\ \Delta \vdash \psi_1 : \kappa_1 \quad \cdots \quad \Sigma\ \Delta \vdash \psi_m : \kappa_m}{\Sigma\ \Delta\ \Omega \vdash \psi_{\text{type}}.\textbf{id}\langle \psi_1, \ldots, \psi_m \rangle()}\ ((m,0)\text{-op-app-form}_e)$$

$$\frac{\begin{array}{c}\Sigma\ \Delta \vdash \psi_{\text{type}} : \texttt{Type} \in \texttt{family}[\kappa_{\text{idx}}]\{\Theta\} \quad \textbf{id} : \kappa_1, \ldots, \kappa_m \to \texttt{Op}_n \in \Theta \quad \Sigma\ \Delta \vdash \psi_1 : \kappa_1 \quad \cdots \quad \Sigma\ \Delta \vdash \psi_m : \kappa_m \\ \Sigma\ \Delta\ \Omega \vdash e_1 \quad \cdots \quad \Sigma\ \Delta\ \Omega \vdash e_n\end{array}}{\Sigma\ \Delta\ \Omega \vdash \psi_{\text{type}}.\textbf{id}\langle \psi_1, \ldots, \psi_m \rangle(e_1, \ldots, e_n)}\ ((m,n)\text{-op-app-form}_e)$$

$\boxed{\Sigma\ \Delta\ \Omega \vdash \mu}$ $\quad \Omega ::= \cdot \mid \Omega, \mathrm{u}$

$$\frac{\mathrm{u} \in \Omega}{\Sigma\ \Delta\ \Omega \vdash \mathrm{u}}\ (\text{var-form}_\mu) \qquad \frac{\Sigma\ \Delta \vdash \delta \quad \Sigma\ \Delta\ \Omega, \mathrm{u} \vdash \mu}{\Sigma\ \Delta\ \Omega \vdash \lambda \mathrm{u} :: \delta.\mu}\ (\lambda\text{-form}_\mu) \qquad \frac{\Sigma\ \Delta \vdash \delta \quad \Sigma\ \Delta\ \Omega, \mathrm{f} \vdash \mu}{\Sigma\ \Delta\ \Omega \vdash \mathsf{fix}\ \mathrm{f} :: \delta.\mu}\ (\text{fix-form}_\mu)$$

$$\frac{\Sigma\ \Delta\ \Omega \vdash \mu_1 \quad \Sigma\ \Delta\ \Omega \vdash \mu_2}{\Sigma\ \Delta\ \Omega \vdash \mu_1\ \mu_2}\ (\text{app-form}_\mu) \qquad \frac{\Sigma\ \Delta\ \Omega \vdash \mu_1 \quad \Sigma\ \Delta\ \Omega \vdash \mu_2 \quad \Sigma\ \Delta\ \Omega \vdash \mu_3 \quad \Sigma\ \Delta\ \Omega \vdash \mu_4}{\Sigma\ \Delta\ \Omega \vdash \mathsf{if}\ \mu_1 \equiv \mu_2\ \mathsf{then}\ \mu_3\ \mathsf{else}\ \mu_4}\ (\text{cond-form}_\mu)$$

$$\frac{}{\Sigma\ \Delta\ \Omega \vdash n}\ (\text{int-form}_\mu) \qquad \frac{\Sigma\ \Delta\ \Omega \vdash \mu_1 \quad \Sigma\ \Delta\ \Omega \vdash \mu_2}{\Sigma\ \Delta\ \Omega \vdash \mu_1 + \mu_2}\ (+\text{-form}_\mu) \qquad \frac{\Sigma\ \Delta\ \Omega \vdash \mu_1 \quad \Sigma\ \Delta\ \Omega \vdash \mu_2}{\Sigma\ \Delta\ \Omega \vdash (\mu_1, \mu_2)}\ (\times\text{-form}_\mu)$$

$$\frac{\Sigma\ \Delta\ \Omega \vdash \mu}{\Sigma\ \Delta\ \Omega \vdash \mathsf{fst}\ \mu}\ (\text{fst-form}_\mu) \qquad \frac{\Sigma\ \Delta\ \Omega \vdash \mu}{\Sigma\ \Delta\ \Omega \vdash \mathsf{snd}\ \mu}\ (\text{snd-form}_\mu) \qquad \frac{\Sigma\ \Delta \vdash \psi : \texttt{ITerm}}{\Sigma\ \Delta\ \Omega \vdash \uparrow(\psi)}\ (\uparrow\text{-form}_\mu)$$

$\boxed{\Sigma\ \Delta \vdash \delta}$

$$\frac{\Sigma\ \Delta \vdash \delta_1 \quad \Sigma\ \Delta \vdash \delta_2}{\Sigma\ \Delta \vdash \delta_1 \to \delta_2}\ (\to\text{-form}_\delta) \qquad \frac{}{\Sigma\ \Delta \vdash \texttt{int}}\ (\text{int-form}_\delta) \qquad \frac{\Sigma\ \Delta \vdash \delta_1 \quad \Sigma\ \Delta \vdash \delta_2}{\Sigma\ \Delta \vdash \delta_1 \times \delta_2}\ (\times\text{-form}_\delta)$$

$$\frac{\Sigma\ \Delta \vdash \psi : \texttt{IType}}{\Sigma\ \Delta \vdash \uparrow(\psi)}\ (\uparrow\text{-form}_\delta)$$

$\boxed{\Xi \vdash \psi \curlyvee_{\mathcal{E}} \psi'}$ $\boxed{\Xi \vdash \theta \curlyvee_{\mathcal{E}} \theta'}$ $\boxed{\Xi \vdash \phi \curlyvee_{\mathcal{E}} \phi'}$ $\quad \Xi ::= \cdot \mid \Xi, \mathsf{family}\ [\kappa_{\text{idx}}]\ ::\ \psi_{\text{rep}}\ \{\theta : \Theta\} \qquad \mathcal{E} ::= \text{ok} \mid\, ! \mid \mathcal{E},\mathcal{E}$

$$\frac{}{\Xi \vdash () \curlyvee_{\text{ok}} ()}\ (\text{unit}_\beta) \qquad \frac{\Xi \vdash \psi_1 \curlyvee_{\mathcal{E}_1} \psi_1' \qquad \Xi \vdash \psi_2 \curlyvee_{\mathcal{E}_2} \psi_2'}{\Xi \vdash (\psi_1, \psi_2) \curlyvee_{\mathcal{E}_1,\mathcal{E}_2} (\psi_1', \psi_2')}\ (\text{pair}_\beta) \qquad \frac{\Xi \vdash \psi \curlyvee_{\mathcal{E}} (\psi_1, \psi_2)}{\Xi \vdash \mathsf{fst}\ \psi \curlyvee_{\mathcal{E}} \psi_1}\ (\text{fst}_\beta) \qquad \frac{\Xi \vdash \psi \curlyvee_{\mathcal{E}} (\psi_1, \psi_2)}{\Xi \vdash \mathsf{snd}\ \psi \curlyvee_{\mathcal{E}} \psi_2}\ (\text{snd}_\beta)$$

$$\frac{}{\Xi \vdash \lambda \mathbf{t}_1{:}\kappa_1,\ \ldots,\mathbf{t}_n{:}\kappa_n.\psi \curlyvee_{\text{ok}} \lambda \mathbf{t}_1{:}\kappa_1,\ \ldots,\mathbf{t}_n{:}\kappa_n.\psi}\ (\lambda_\beta)$$

$$\frac{\Xi \vdash \psi_0 \curlyvee_{\mathcal{E}_0} \lambda \mathbf{t}_1{:}\kappa_1,\ \ldots,\mathbf{t}_n{:}\kappa_n.\psi \qquad \Xi \vdash \psi_1 \curlyvee_{\mathcal{E}_1} \psi_1' \qquad \cdots \qquad \Xi \vdash \psi_n \curlyvee_{\mathcal{E}_n} \psi_n' \qquad \Xi \vdash [\psi_1'/t_1,\ \ldots,\psi_n'/t_n]\psi \curlyvee_{\mathcal{E}} \psi'}{\Xi \vdash \psi_0(\psi_1,\ldots,\psi_n) \curlyvee_{\mathcal{E}_0,\mathcal{E}_1,\ldots,\mathcal{E}_n,\mathcal{E}} \psi'}\ (\text{app}_\beta)$$

$$\frac{\Xi \vdash \psi_0 \curlyvee_{\mathcal{E}_0} \psi_0' \qquad \Xi \vdash \psi_1 \curlyvee_{\mathcal{E}_1} \psi_0' \qquad \Xi \vdash \psi_2 \curlyvee_{\mathcal{E}_2} \psi_2'}{\Xi \vdash \mathsf{if}\ \psi_0 \equiv \psi_1\ \mathsf{then}\ \psi_2\ \mathsf{else}\ \psi_3 \curlyvee_{\mathcal{E}_0,\mathcal{E}_1,\mathcal{E}_2} \psi_2'}\ (\text{if}^1_\beta)$$

$$\frac{\Xi \vdash \psi_0 \curlyvee_{\mathcal{E}_0} \psi_0' \qquad \Xi \vdash \psi_1 \curlyvee_{\mathcal{E}_1} \psi_1' \qquad \psi_0' \neq \psi_1' \qquad \Xi \vdash \psi_3 \curlyvee_{\mathcal{E}_3} \psi_3'}{\Xi \vdash \mathsf{if}\ \psi_0 \equiv \psi_1\ \mathsf{then}\ \psi_2\ \mathsf{else}\ \psi_3 \curlyvee_{\mathcal{E}_0,\mathcal{E}_1,\mathcal{E}_3} \psi_3'}\ (\text{if}^2_\beta)$$

$$\frac{\Xi \vdash \mathsf{family}\ [\kappa_{\text{idx}}]\ ::\ \psi_{\text{rep}}\ \{\theta : \Theta\} \curlyvee_{\mathcal{E}_0} \phi \qquad \Xi \vdash [\phi/\text{FAM}]\psi \curlyvee_{\mathcal{E}_1} \psi'}{\Xi \vdash \mathsf{family}\ \text{FAM}[\kappa_{\text{idx}}]\ ::\ \psi_{\text{rep}}\ \{\theta : \Theta\}\ \mathsf{in}\ \psi \curlyvee_{\mathcal{E}_0,\mathcal{E}_1} \psi'}\ (\text{fambind}_\beta)$$

$$\frac{\Xi \vdash \psi_{\text{rep}} \curlyvee_{\mathcal{E}_0} {\psi_{\text{rep}}}' \qquad \Xi \vdash \theta \curlyvee_{\mathcal{E}_1} \theta'}{\Xi \vdash \mathsf{family}\ [\kappa_{\text{idx}}]\ ::\ \psi_{\text{rep}}\ \{\theta : \Theta\} \curlyvee_{\mathcal{E}_0,\mathcal{E}_1} \mathsf{family}\ [\kappa_{\text{idx}}]\ ::\ {\psi_{\text{rep}}}'\ \{\theta' : \Theta\}}\ (\text{family}_\beta)$$

$$\frac{\Xi \vdash \psi_{\text{type}} \curlyvee_{\mathcal{E}_0} \mathsf{type}[\psi_{\text{idx}}] \in \phi' \qquad \Xi \vdash \phi \curlyvee_{\mathcal{E}_1} \phi' \qquad \Xi \vdash \psi_0(\mathsf{type}[\psi_{\text{idx}}] \in \phi') \curlyvee_{\mathcal{E}_2} \psi_0'}{\Xi \vdash \mathsf{famcase}\ \psi_{\text{type}}\ \mathsf{of}\ \phi\ \mathsf{then}\ \psi_0\ \mathsf{else}\ \psi_1 \curlyvee_{\mathcal{E}_0,\mathcal{E}_1,\mathcal{E}_2} \psi_0'}\ (\text{famcase-T}^1_\beta)$$

$$\frac{\Xi \vdash \psi_{\text{type}} \curlyvee_{\mathcal{E}_0} \mathsf{type}[\psi_{\text{idx}}] \in \phi' \qquad \Xi \vdash \phi \curlyvee_{\mathcal{E}_1} \phi'' \qquad \phi' \neq \phi'' \qquad \Xi \vdash \psi_1 \curlyvee_{\mathcal{E}_2} \psi_1'}{\Xi \vdash \mathsf{famcase}\ \psi_{\text{type}}\ \mathsf{of}\ \phi\ \mathsf{then}\ \psi_0\ \mathsf{else}\ \psi_1 \curlyvee_{\mathcal{E}_0,\mathcal{E}_1,\mathcal{E}_2} \psi_1'}\ (\text{famcase-T}^2_\beta)$$

$$\frac{\Xi \vdash \psi_{\text{den}} \curlyvee_{\mathcal{E}_0} [\![\psi_{\text{iterm}} \sim \mathsf{type}[\psi_{\text{idx}}] \in \phi']\!] \qquad \Xi \vdash \phi \curlyvee_{\mathcal{E}_1} \phi' \qquad \Xi \vdash \psi_0([\![\psi_{\text{iterm}} \sim \mathsf{type}[\psi_{\text{idx}}] \in \phi']\!]) \curlyvee_{\mathcal{E}_2} \psi_0'}{\Xi \vdash \mathsf{famcase}\ \psi_{\text{den}}\ \mathsf{of}\ \phi\ \mathsf{then}\ \psi_0\ \mathsf{else}\ \psi_1 \curlyvee_{\mathcal{E}_0,\mathcal{E}_1,\mathcal{E}_2} \psi_0'}\ (\text{famcase-D}^1_\beta)$$

$$\frac{\Xi \vdash \psi_{\text{den}} \curlyvee_{\mathcal{E}_0} [\![\psi_{\text{iterm}} \sim \mathsf{type}[\psi_{\text{idx}}] \in \phi']\!] \qquad \Xi \vdash \phi \curlyvee_{\mathcal{E}_1} \phi'' \qquad \phi' \neq \phi'' \qquad \Xi \vdash \psi_1 \curlyvee_{\mathcal{E}_2} \psi_1'}{\Xi \vdash \mathsf{famcase}\ \psi_{\text{den}}\ \mathsf{of}\ \phi\ \mathsf{then}\ \psi_0\ \mathsf{else}\ \psi_1 \curlyvee_{\mathcal{E}_0,\mathcal{E}_1,\mathcal{E}_2} \psi_1'}\ (\text{famcase-D}^2_\beta)$$

$$\frac{\Xi \vdash \psi_{\text{idx}} \curlyvee_{\mathcal{E}} {\psi_{\text{idx}}}'}{\Xi \vdash \mathsf{type}[\psi_{\text{idx}}] \in \phi \curlyvee_{\mathcal{E}} \mathsf{type}[{\psi_{\text{idx}}}'] \in \mathit{fam}} \qquad \frac{\Xi \vdash \psi_{\text{type}} \curlyvee_{\mathcal{E}} \mathsf{type}[\psi_{\text{idx}}] \in \phi}{\Xi \vdash \mathsf{idxof}\ \psi_{\text{type}} \curlyvee_{\mathcal{E}} \psi_{\text{idx}}}$$

$$\frac{\Xi \vdash \psi_{\text{type}} \curlyvee_{\mathcal{E}_0} \mathsf{type}[\psi_{\text{idx}}] \in \phi \qquad \mathsf{family}\ [\kappa_{\text{idx}}]\ ::\ \psi_{\text{rep}}\ \{\theta : \Theta\} \in \Xi \qquad \Xi \vdash \psi_{\text{rep}}(\psi_{\text{idx}}) \curlyvee_{\mathcal{E}_1} \psi'}{\Xi \vdash \mathsf{repof}\ \psi_{\text{type}} \curlyvee_{\mathcal{E}_0,\mathcal{E}_1} \psi'} \qquad \frac{\Xi \vdash \psi \curlyvee_{\mathcal{E}} \psi'}{\Xi \vdash \mathsf{const}(\psi) \curlyvee_{\mathcal{E}} \mathsf{const}(\psi')}$$

$$\frac{\Xi \vdash \psi \curlyvee_{\mathcal{E}} \psi'}{\Xi \vdash \mathsf{op}(\psi) \curlyvee_{\mathcal{E}} \mathsf{op}(\psi')} \qquad \frac{\Xi \vdash \psi_{\text{iterm}} \curlyvee_{\mathcal{E}_0} {\psi_{\text{iterm}}}' \qquad \Xi \vdash \psi_{\text{type}} \curlyvee_{\mathcal{E}_1} {\psi_{\text{type}}}'}{\Xi \vdash [\![\psi_{\text{iterm}} \sim \psi_{\text{type}}]\!] \curlyvee_{\mathcal{E}_0,\mathcal{E}_1} [\![{\psi_{\text{iterm}}}' \sim {\psi_{\text{type}}}']\!]} \qquad \frac{\Xi \vdash \psi_{\text{den}} \curlyvee_{\mathcal{E}} [\![\psi_{\text{iterm}} \sim \psi_{\text{type}}]\!]}{\Xi \vdash \mathsf{valof}\ \psi_{\text{den}} \curlyvee_{\mathcal{E}} \psi_{\text{iterm}}}$$

$$\frac{\Xi \vdash \psi_{\text{den}} \curlyvee_{\mathcal{E}} [\![\psi_{\text{iterm}} \sim \psi_{\text{type}}]\!]}{\Xi \vdash \mathsf{typeof}\ \psi_{\text{den}} \curlyvee_{\mathcal{E}} \psi_{\text{type}}} \qquad \frac{}{\Xi \vdash \mathsf{err} \curlyvee_{\text{ok}} \mathsf{err}} \qquad \frac{\Xi \vdash \mu \curlyvee_{\mathcal{E}} \mu'}{\Xi \vdash \mathsf{iterm}(\mu) \curlyvee_{\mathcal{E}} \mathsf{iterm}(\mu')} \qquad \frac{\Xi \vdash \delta \curlyvee_{\mathcal{E}} \delta'}{\Xi \vdash \mathsf{itype}(\delta) \curlyvee_{\mathcal{E}} \mathsf{itype}(\delta')}$$

$$\frac{e \curlyvee_{\mathcal{E}} e'}{\Xi \vdash \mathsf{program}(e) \curlyvee_{\mathcal{E}} \mathsf{program}(e')} \qquad \frac{}{\Xi \vdash \cdot \curlyvee_{\text{ok}} \cdot} \qquad \frac{\Xi \vdash \theta \curlyvee_{\mathcal{E}_0} \theta' \qquad \Xi \vdash \psi \curlyvee_{\mathcal{E}_1} \psi'}{\Xi \vdash \theta; \mathbf{op} = \psi \curlyvee_{\mathcal{E}_0,\mathcal{E}_1} \theta'; \mathbf{op} = \psi'}$$

$\boxed{e \curlyvee_{\mathcal{E}} e'}$

$$\frac{\Xi \vdash \psi \curlyvee_{\mathcal{E}} \psi'}{\Xi \vdash \lambda \mathrm{x}{:}\psi.e \curlyvee_{\mathcal{E}} \lambda \mathrm{x}{:}\psi'.e} \qquad \frac{\Xi \vdash e_1 \curlyvee_{\mathcal{E}_1} e_1' \qquad \Xi \vdash e_2 \curlyvee_{\mathcal{E}_2} e_2'}{\Xi \vdash e_1\ e_2 \curlyvee_{\mathcal{E}_1,\mathcal{E}_2} e_1'\ e_2'}$$

$$\frac{\Xi \vdash \psi_{\text{type}} \curlyvee_{\mathcal{E}} {\psi_{\text{type}}}' \qquad \Xi \vdash \psi_1 \curlyvee_{\mathcal{E}_1^1} \psi_1' \qquad \cdots \qquad \Xi \vdash \psi_m \curlyvee_{\mathcal{E}_m^1} \psi_m' \qquad \Xi \vdash e_1 \curlyvee_{\mathcal{E}_1^2} e_1' \qquad \cdots \qquad \Xi \vdash e_n \curlyvee_{\mathcal{E}_n^2} e_n'}{\Xi \vdash \psi_{\text{type}}.\mathbf{op}\langle \psi_1,\ldots,\psi_m\rangle(e_1,\ldots,e_n) \curlyvee_{\mathcal{E},\mathcal{E}_1^1,\ldots,\mathcal{E}_m^1,\mathcal{E}_1^2,\ldots,\mathcal{E}_n^2} {\psi_{\text{type}}}'.\mathbf{op}\langle \psi_1',\ldots,\psi_m'\rangle(e_1',\ldots,e_n')}$$

$\boxed{\mu \curlyvee_{\mathcal{E}} \mu'}$

dasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf

afdasdfsdf afdasdfsdf