Verification with Natural Contexts: Soundness of Safe Compositional Network Sketches

Andrei Lapets Assaf Kfoury

October 1, 2009

Abstract

In research areas involving mathematical rigor, there are numerous benefits to adopting a formal representation of models and arguments: reusability, automatic evaluation of examples, and verification of consistency and correctness. However, accessibility has not been a priority in the design of formal verification tools that can provide these benefits. In earlier work [30] we attempt to address this broad problem by proposing several specific design criteria organized around the notion of a *natural context*: the sphere of awareness a working human user maintains of the relevant constructs, arguments, experiences, and background materials necessary to accomplish the task at hand. In this report we evaluate our proposed design criteria by utilizing within the context of novel research a formal reasoning system that is designed according to these criteria. In particular, we consider how the design and capabilities of the formal reasoning system that we employ influence, aid, or hinder our ability to accomplish a formal reasoning task – the assembly of a machine-verifiable proof pertaining to the NetSketch formalism.

NetSketch is a tool for the specification of constrained-flow applications and the certification of desirable safety properties imposed thereon. NetSketch is conceived to assist system integrators in two types of activities: modeling and design. It provides capabilities for compositional analysis based on a strongly-typed domain-specific language (DSL) for describing and reasoning about constrained-flow networks and invariants that need to be enforced thereupon. In a companion paper [13] we overview NetSketch, highlight its salient features, and illustrate how it could be used in actual applications. In this paper, we define using a machine-readable syntax major parts of the formal system underlying the operation of NetSketch, along with its semantics and a corresponding notion of validity. We then provide a proof of soundness for the formalism that can be partially verified using a lightweight formal reasoning system that simulates natural contexts. A traditional presentation of these definitions and arguments can be found in the full report on the NetSketch formalism [12].

1 Introduction and Motivation

In research that requires mathematical rigor there exist many benefits to adopting a formal representation. These include reusability, automatic evaluation of examples, and the opportunity to employ automated verification. There exist a variety of such systems, and many of these have been surveyed and compared along a variety of dimensions [54]. However, to date, broad accessibility and quality interface design has not been a priority in the design of such systems. On the contrary, researchers hoping to enjoy the benefits of formal verification are presented with a variety of obstacles, both superficial and fundamental. Consequently, many researchers today choose to ignore such systems. In the literature in most domains of computer science and mathematics there are only isolated attempts to include machine-verified proofs of novel research results.

To address the aforementioned issues, we have proposed in earlier work [30] principles for the design of automated verifiers and assistants for formal reasoning. We argue that by following these principles it is possible to assemble formal reasoning systems that have better accessibility and useability. These principles require the adoption of what we consider *lightweight* design. In this context, lightweight design is not exclusively a passive principle that consists of loosening restrictions. A system that is lightweight from the user's perspective might also provide robust and natural interfaces for search and interaction, might give

This work is supported in part by NSF awards CNS-0952145, CCF-0820138, CSR-0720604, and EFRI-0735974.

the user many degrees of freedom in deciding how precisely or extensively arguments are verified, and might actively anticipate and tighten what the user is trying to do using appropriate heuristics. This strategy is based on our own understanding and experience as well as the observations that motivated other recent projects with similar goals. In particular, the need for natural interfaces (both superficial and functional) and the value of appropriate heuristics have been recognized to varying degrees by the designers of the `Tutch` proof checker [2], the Scunak mathematical assistant system [16], the ForTheL language and SAD proof assistant [52], the EPGY Theorem-Proving Environment [37], the ΩMEGA proof verifier [46], and in the work of Sieg and Cittadini [45]. One author, commenting on designing an interface for representing proofs, opines that "we seem to be stuck at the assembly language level" [53]. Both the need for a natural interface as well as the value of a lightweight design are also recognized by some in the model checking community, such as the creators of the Alloy modelling language [29].

1.1 Natural Contexts in Lightweight Formal Verification

We focus our efforts by considering the notion of a *natural context*: the sphere of awareness a working human user maintains of the relevant constructs, arguments, experiences, and background materials necessary to accomplish the task at hand. The relevant characteristics of a natural context in such a scenario are (1) its size (encompassing a wide array of experiences, potentially inconsistent or unrelated), and (2) a flexible and powerful interface that allows easy exploration, querying, and short-term adjustments (using not just reference or index but also structure). In explicitly recognizing the possibility that a natural context can contain distinct, inconsistent collections of experiences (such as different logical systems), we are in part inspired by the notion of a *cognitive context* found in literature on formal ontologies [39].

This notion can be a useful guide within the realm of formal reasoning. We hypothesize that a lightweight verification system can be successful if it can maintain a highly rich and relevant context for the user, and furthermore, if it allows the user to interact with this context in a familiar and efficient manner. A concrete example of a natural context familiar to those working in programming language design is a data structure that maintains type signatures and constraints. Its interface consists of the programming language syntax, the means for producing status or error messages, the type inference algorithm, and any constraint solver supporting that algorithm.

1.2 Utilization for Novel Research: the NetSketch Formalism

In this work, we employ an implementation of a formal reasoning system[1] designed to simulate a natural context in accomplishing a novel research task that requires formal rigor: assembling an argument that the NetSketch formalism is sound. In related work [13, 12], we present more extensively the background and motivation underlying NetSketch and its underlying formalism, and we refer the reader interested in learning more about this formalism to these materials. Here, we present only a brief introduction to NetSketch.

1.2.1 Constrained-Flow Networks

Many large-scale, safety-critical systems can be viewed as interconnections of subsystems, or modules, each of which is a producer, consumer, or regulator of *flows*. These flows are characterized by a set of variables and a set of constraints thereof, reflecting *inherent* or *assumed* properties or rules governing how the modules operate (and what constitutes safe operation). Our notion of flow encompasses streams of physical entities (*e.g.*, vehicles on a road, fluid in a pipe), data objects (*e.g.*, sensor network packets or video frames), or consumable resources (*e.g.*, electric energy or compute cycles).

Many complex, large-scale systems can be viewed at least in part as *constrained-flow networks*, and current practices in the assembly of such systems involve the integration of various subsystems into a whole by "system integrators" who may not possess expertise or knowledge of the internals of the subsystems on

[1] We employ a formal reasoning system implemented in Haskell; source code and a demonstration version of the system, integrated with an online content management system, can be found at `http://www.safre.org`.

which they rely. NetSketch [13] is a tool that assists system integrators in two types of activities: modelling and design. As a modelling tool, NetSketch enables the abstraction of an existing (flow network) system while retaining sufficient information about it to carry out future analysis of safety properties. As a design tool, NetSketch enables the exploration of alternative safe designs as well as the identification of minimal requirements for missing subsystems in partial designs. Formal analysis is at the heart of both of the above modelling and design activities.

1.2.2 The NetSketch Formalism

Support for safety analysis in design and/or development tools such as NetSketch is based on sound formalisms that are not specific to (and do not require expertise in) particular domains. NetSketch enables the composition of exact analyses of small subsystems by adopting a constrained-flow network formalism that exposes the tradeoffs between exactness of analysis and scalability of analysis. This is done using a strongly-typed domain-specific language (DSL) for describing and reasoning about constrained-flow networks at various levels of "sketchiness" (precision) along with invariants that need to be enforced thereupon. We are interested in using a formal reasoning system to formally define NetSketch's DSL and to prove its soundness relative to appropriately-defined notions of validity.

There are a few essential concepts that constitute our formalism for compositional analysis of problems involving constrained-flow networks. The formalism provides a language for defining *networks* – graphs consisting of nodes and edges. Networks are constructed out of many instances of defined *modules* (small network components). These are also graphs but are typically of a size that is sufficiently small for a complete or exhaustive analysis. The formalism provides means for defining such modules and assembling them into networks. If an analysis of the network is desired, it may be possible to take advantage of this modularity by analyzing the components individually in a more precise manner, and then composing the results to analyze the entire network. Analyses are represented using a language of *constraints*. If the engineer considers flows across a network of modules, the relevant *parameters* describing this flow (e.g., the number of open lanes on a highway, or the throughput of a network link) can be mathematically constrained for each instance of a module. These constraints can model both limitations of modules and problems the engineer must solve.

Thus, the formalism consists of two intertwined languages: a language for describing networks composed of modules, and a language for describing constraints governing flows across the network components. In order to ensure that our system works correctly "under the hood", it is necessary to define a precise *semantics* for these languages, along with a rigorous notion of what it means for an analysis of a network to be "correct". Only once these are defined is it possible to provide a guarantee that the system is indeed safe to use.

1.3 Format of Presentation and Overview

All formal definitions and arguments within the text of this report that are parsed by the formal reasoning system are presented within a framed box. An example is included below.

> Assert for any $x, y, z \in \mathbb{R}$, if $x > y$ and $y > z$ then $x > z$.

It is worth noting that the raw LaTeX source from which this report was compiled can be supplied directly to the formal reasoning system we employed (including all the LaTeX markup and informal exposition).

In Sections 2 and 3 we introduce the concepts that underlie our formalism. In Section 3, we use our formal reasoning system to outline a semantics for constraint sets and network flows, and to define a notion of validity in terms of which the soundness of the framework can be defined. In Section 4 we present the typed DSL for network sketches. In Section 5 we assemble a proof of the soundness of the NetSketch formalism with respect to its semantics. This proof can be partially verified using the automated lightweight verification capabilities of our formal reasoning system. Finally, we review related work in Section 6 and present our conclusions in Section 7.

2 Modules: Untyped and Typed

We introduce several preliminary notions. For more detailed definitions, we refer readers to the full technical report on the formalism [12].

Definition 1 (*Syntax of Constraints*)**.** Let $\mathcal{X} = \{x_0, x_1, x_2, \ldots\}$ be an infinite set of *parameters*. The set of *constraints over* $\mathbb{N}$ *and* $\mathcal{X}$ can be defined in extended BNF style, where we use metavariables n and x to range over $\mathbb{N}$ and $\mathcal{X}$, respectively:

$$\begin{aligned} e \in \textsc{Exp} &::= n \mid x \mid e_1 * e_2 \mid e_1 + e_2 \mid e_1 - e_2 \mid \ldots \\ c \in \textsc{Const} &::= e_1 = e_2 \mid e_1 < e_2 \mid e_1 \leqslant e_2 \mid \ldots \end{aligned}$$

We include in Const at least equalities and orderings of expressions. Possible extensions of Const include conditional constraints, negated constraints, time-dependent constraints, and others. Within our formalism, constraints in Const are part of a *given* flow network abstraction and may be arbitrarily complex; constraints to be inferred or checked against the given constraints are from some restriction of Const, such as *linear constraints*:

$$\begin{aligned} e \in \textsc{LinExp} &::= n \mid x \mid n * x \mid e_1 + e_2 \\ c \in \textsc{LinConst} &::= e_1 = e_2 \mid e_1 < e_2 \mid e_1 \leqslant e_2 \end{aligned}$$

We are adopting a lightweight approach in our machine-assisted verification of the formalism, so we do not explicitly model the structure of the set of constraints. Instead, we introduce abstract constants and predicates related to constraint sets.

> Introduce the constant Const. Assume for any C, C is a constraint set iff $C \subseteq$ Const. Assume for any C, if C is a constraint set then C is a set.
>
> Introduce the constant $\mathcal{X}$. Assume for any P, P is a parameter set iff $P \subseteq \mathcal{X}$. Assume for any P, if P is a parameter set then P is a set.
>
> Introduce the constant parameters. Assume for any C, if $C \subseteq$ Const then $\mathsf{parameters}(C) \subseteq \mathcal{X}$.

□

Definition 2 (*Untyped Modules*)**.** A *module* is a network component that is amenable to exhaustive analysis. We specify an untyped module A by a four-tuple $(A, I, O, \mathcal{C})$.

> Assume for any A, I, O, C,
> (A, I, O, C) is an untyped module
> iff
> $I \subseteq \mathsf{parameters}(C)$,
> $O \subseteq \mathsf{parameters}(C)$,
> $O \cap I = \emptyset$, and
> $C \subseteq$ Const.

□

Definition 3 (*Typed Modules*)**.** For a module $(A, I, O, \mathcal{C})$ as specified in Definition 2, a *typed specification* or *typing* for $(A, I, O, \mathcal{C})$ is the untyped module specification paired with a constraint set C' from some restricted space of constraints (as described in Definition 1).

> Introduce the set operator $:$.
>
> Assume for any A, I, O, C, C',
> $(A, I, O, C) : C'$ is a typed module
> iff
> (A, I, O, C) is an untyped module,
> $C' \subseteq$ CONST, and
> $\mathsf{parameters}(C') \subseteq I \cup O$.

A *typing judgment* $(A, I, O, \mathcal{C}) : C'$ may or may not be valid. The validity of judgments presumes a formal definition of the semantics of modules, which we introduce in Section 3. □

3 Semantics of Network Typings

A *network typing*, as defined precisely in Section 4 further below, is specified by an expression of the form $(M, I, O, \mathcal{C}) : C$ where $(M, I, O, \mathcal{C})$ is an untyped network and C is a finite set of constraints such that $\mathsf{parameters}(C) \subseteq I \cup O$. In this section, we introduce the definition and semantics of constraints.

Definition 4 (*Satisfaction of Constraints*)**.** Let $\mathcal{Y} \subseteq \mathcal{X}$ be a subset of the parameter space. Let V be a *valuation for* $\mathcal{Y}$, *i.e.*, V is a map from $\mathcal{Y}$ to $\mathbb{N}$. Suppose all expressions and constraints are written over parameters in $\mathcal{Y}$. We use "$\models$" to denote the satisfaction relation.

> Introduce the set operator $\models$.

The interpretation of an expression relative to V is defined by induction over $e \in$ EXP:

$$V(e) = \begin{cases} n & \text{if } e = n, \\ V(x) & \text{if } e = x \in \mathcal{Y}, \\ p & \text{if } e = e_1 * e_2 \ \&\ p = V(e_1) * V(e_2), \\ q & \text{if } e = e_1 + e_2 \ \&\ q = V(e_1) + V(e_2), \\ r & \text{if } e = e_1 - e_2 \ \&\ r = V(e_1) - V(e_2), \end{cases}$$

Satisfaction of a constraint by V is defined by cases over $c \in$ CONST:

$$\begin{aligned} V &\models e_1 = e_2 \quad \text{iff} \quad V(e_1) = V(e_2) \\ V &\models e_1 < e_2 \quad \text{iff} \quad V(e_1) < V(e_2) \\ V &\models e_1 \leqslant e_2 \quad \text{iff} \quad V(e_1) \leqslant V(e_2) \end{aligned}$$

Satisfaction of a set of constraints relative to V is defined in the natural way:

$$V \models \{c_1, \ldots, c_p\} \quad \text{iff} \quad V \models c_1 \text{ and } \ldots \text{ and } V \models c_p$$

□

We introduce several useful operators involving parameters, constraints, and network typings.

Definition 5 (*Comparison of Constraint Sets*)**.** We define a precise way of relating constraint sets to one another in terms of constraint semantics.

> Introduce the set operators $\Rrightarrow, \Lleftarrow\Rrightarrow$.
>
> Assume for any C, C',
> if $C \subseteq \textsc{Const}$, $C' \subseteq \textsc{Const}$ then
> $C \Rrightarrow C'$
> iff
> for all $V \in \mathcal{X} \to \mathbb{N}$, $V \models C$ implies $V \models C'$.
>
> Assume for any C, C', $C \Lleftarrow\Rrightarrow C'$ iff $C \Rrightarrow C'$ and $C' \Rrightarrow C$.

Naturally, any set implies all subsets of itself.

> Assume for any C, C', $C' \subseteq C$ implies $C \Rrightarrow C'$.

□

Definition 6 (*Closures of Constraint Sets*)**.** For a finite constraint set C, its *closure* is the set of all constraints implied by C.

> Introduce the constant $\mathsf{closure}$. Assume for any C, $\mathsf{closure}(C) = \{c | c \in \textsc{Const}, C \Rrightarrow \{c\}\}$.
> Assume for any C, if $C \subseteq \textsc{Const}$ then $\mathsf{closure}(C) \subseteq \textsc{Const}$.

Naturally, a set is a subset of its own closure, and the closure of a set contains the closure of each of its subsets.

> Assume for any C, $C \subseteq \mathsf{closure}(C)$.
> Assume for any C, C', $C \subseteq C'$ implies $\mathsf{closure}(C) \subseteq \mathsf{closure}(C')$.

If the parameters of two constraint sets are disjoint, then the $\mathsf{closure}$ operation preserves the union operation on these two constraint sets.

> Assume for all C, C',
> if $\mathsf{parameters}(C) \cap \mathsf{parameters}(C') = \emptyset$ then $\mathsf{closure}(C \cup C') = \mathsf{closure}(C) \cup \mathsf{closure}(C')$.

Furthermore, the parameters of the closures of any such two constraint sets are also disjoint.

> Assume for any C, C', $\mathsf{parameters}(C) \cap \mathsf{parameters}(C') = \emptyset$
> implies $\mathsf{parameters}(\mathsf{closure}(C)) \cap \mathsf{parameters}(\mathsf{closure}(C')) = \emptyset$.

□

Definition 7 (*Restrictions on Constraint Sets*)**.** In subsequent definitions, we require the ability to restrict a constraint set using particular parameter sets.

> Introduce set operators $\restriction, \downharpoonright$.
> Assume for any C, P,
> $C \subseteq \textsc{Const}$ and $P \subseteq \mathcal{X}$
> implies that
> $C \restriction P = \{c | c \in C, \mathsf{parameters}(c) \subseteq P\}$,
> $C \downharpoonright P = \{c | c \in C, \mathsf{parameters}(c) \cap P \neq \emptyset\}$.

Thus, $C \restriction P$ is the subset of C in which only parameters from P occur, and $C \downharpoonright P$ is the subset of C in which every constraint has an occurrence of a parameter from P. We present some facts about these operators.

> Assume for any C, P, $C \restriction P \subseteq C$.
> Assume for any C, P, $C \downharpoonright P \subseteq C$.
> Assume for any C, C', S, S', if $C \subseteq C'$ and $S \subseteq S'$ then $C' \restriction S' \Rrightarrow C \restriction S$.
> Assume for any C, C', S, S', if $C \subseteq C'$ and $S \subseteq S'$ then $C' \downharpoonright S' \Rrightarrow C \downharpoonright S$.
> Assume for any C, C', P, P',
> $\mathsf{parameters}(C) \cap \mathsf{parameters}(C') = \emptyset$
> implies that
> $(C \downharpoonright P) \cup (C' \downharpoonright P') = (C \cup C') \downharpoonright (P \cup P')$.

□

Definition 8 (*Pre- and Post-conditions for Network Sketches*)**.** Let $((M, I, O, \mathcal{C}) : C)$ be a typed network sketch. Recall that $\mathsf{parameters}(C) \subseteq I \cup O$. We partition $\mathsf{closure}(C)$ into two subsets.

> Introduce the constants $\mathsf{pre}, \mathsf{post}$. Assume for any $M, I, O, \mathcal{C}, C$,
> $\mathsf{pre}((M, I, O, \mathcal{C}), C) = \mathsf{closure}(C) \restriction I$,
> $\mathsf{post}((M, I, O, \mathcal{C}), C) = \mathsf{closure}(C) \downharpoonright O = \mathsf{closure}(C) - \mathsf{pre}((M, I, O, \mathcal{C}), C)$.

Note that while the parameters of $\mathsf{pre}(C)$ are all in I, the parameters of $\mathsf{post}(C)$ are not necessarily all in O, because some constraints in C may contain both input and output parameters. For both operators, the constraint set C for any network sketch implies the constraints in the pre and post constraint sets.

> Assume for all $M, I, O, \mathcal{C}, C$, $C \Rrightarrow \mathsf{pre}((M, I, O, \mathcal{C}), C)$.
> Assume for all $M, I, O, \mathcal{C}, C$, $C \Rrightarrow \mathsf{post}((M, I, O, \mathcal{C}), C)$.

□

In the full report describing our formalism [12], we introduce two different semantics, corresponding to what we call "weak satisfaction" and "strong satisfaction" of typing judgements. Both semantics are meaningful, corresponding to whether or not network nodes act as "autonomous systems", *i.e.*, whether or not each node coordinates its action with its neighbors or according to instructions from a network administrator. The definitions of "weak satisfaction" and "strong satisfaction" are very similar except that the first involves an *existential quantification* and the second a *universal quantification*. In this report, we introduce only the strong variant of satisfaction.

Definition 9 (*Satisfaction*)**.** Let $V : I \to \mathbb{N}$ be a fixed valuation of the input parameters of a network $(M, I, O, \mathcal{C})$.

Assume for any $M, I, O, \mathcal{C}, C$, for any V,
$V \models ((M, I, O, \mathcal{C}) : C)$
iff
$V \models \mathsf{pre}((M, I, O, \mathcal{C}), C)$ implies that
for all V',
$V \subseteq V'$ and for all $K \in \mathcal{C}$, $V' \models K$
implies that
$V' \models \mathsf{post}((M, I, O, \mathcal{C}), C)$.

There are several useful facts about the $\models$ relation.

Assume for any V, V', C, if $V \subseteq V'$ and $V \models C$ then $V' \models C$.
Assume for any V, C, C', if $C \Rrightarrow C'$ and $V \models C$ then $V \models C'$.
Assume for any V, C, C', $V \models C$ and $V \models C'$ iff $V \models C \cup C'$.

□

Definition 10 (*Validity of Typings*)**.** Informally, a typing is *strongly valid* iff, for every network flow satisfying $\mathsf{pre}((M, I, O, \mathcal{C}), C)$, and ***for every way*** of channelling the flow through $(M, I, O, \mathcal{C})$ that is consistent with its internal constraints $\mathcal{C}$, $\mathsf{post}((M, I, O, \mathcal{C}), C)$ is satisfied.

Assume for any $M, I, O, \mathcal{C}, C$,
$((M, I, O, \mathcal{C}) : C)$ is strongly valid
iff
for all $V \in I \to \mathbb{N}$,
$V \models ((M, I, O, \mathcal{C}) : C)$.

□

4 Typed Network Sketches

We define a specification language to assemble modules together, also allowing for the presence of network holes. This is a strongly-typed *d*omain-*s*pecific *l*anguage (DSL). For the sake of concision, we present only the typed version of the DSL and refer readers interested in seeing separate presentations of both the untyped and typed DSL to the full technical report presenting our formalism [12].

A network sketch is written as $(M, I, O, \mathcal{C})$, where I and O are the sets of input and output parameters, and $\mathcal{C}$ is a finite set of finite constraint sets. M is *not* a name but an expression built up from: (1) module names and (2) the constructors **conn**, **loop**, and **hole**. A "network hole" can be viewed as a place-holders with some associated attributes. In this presentation, each network hole $\mathbf{hole}(X, \{M_1, \ldots, M_n\})$ explicitly contains a label X and a set of network sketches $\{M_1, \ldots, M_n\}$. Any of the network sketches in $\{M_1, \ldots, M_n\}$ can be interchangeably "placed" into this hole, depending on changing conditions of operation in the network as a whole. This treatment of network holes differs from the treatment in the technical report describing the formalism [12], but only superficially.[2]

[2]In the full report's treatment [12], a **let**-bound, labelled hole can only appear *once* within its scope. Our presentation

Definition 11 (*Syntax of Raw Network Sketches*)**.** We introduce the raw syntax in extended BNF style. The formal expressions written according to the following BNF are said to be "raw" because they do not specify how the internal constraints of a network sketch are assembled together from those of its subcomponents. This is what the rules in Section 4.1 do precisely.

Introduce the constants **conn**, **loop**, **hole**.

$$
\begin{array}{lll}
A, B & \in \textsc{ModuleNames} & \\
X, Y, Z & \in \textsc{HoleNames} & \\
\theta & \in \mathcal{X} \to \mathcal{X} & \\
M, N & \in \textsc{RawSketches} & ::= \\
& & A \\
& & \mid \mathbf{conn}(\theta, M, N) \\
& & \mid \mathbf{loop}(\theta, M) \\
& & \mid \mathbf{hole}(X, \{M_1, \ldots, M_n\})
\end{array}
$$

As a convention, we use upper-case letters to refer to modules and networks – from the early alphabet (A, B) for modules and from the middle alphabet (M, N) for networks. Also note carefully that M and N are *metavariables*, ranging over expressions in RawSketches; they do not appear as formal symbols in such expressions written in full. By contrast, A and B are *names* of modules and can occur as formal symbols in expressions of RawSketches. We assume that each occurrence of the same module or the same hole in a raw sketch has its own private set of names. This invariant can be ensured using isomorphic renaming. For a slightly more detailed discussion of how this can be accomplished, we refer readers to the full technical report presenting the formalism [12]. □

4.1 Type Inference Rules

When networks are connected (i.e. composed) with themselves or one another, maps are introduced that relate the parameters that represent the output links of a network to parameters that represent the input links of a network. By convention, these maps are represented using lowercase Greek letters (θ, ϕ, ψ). We also introduce a function $\mathsf{constraints}$ that transforms a map θ into a corresponding set of constraints $\{x = \theta(x) | x \in \mathsf{dom}(x)\}$. Because we are adopting a lightweight verification approach in which we do not explicitly define the space Const, we do not explicitly define $\mathsf{constraints}$.

Introduce the constant $\mathsf{constraints}$. Assume for any $\theta \in \mathcal{X} \to \mathcal{X}$, $\mathsf{constraints}(\theta)$ is a constraint set.

Each network $(M, I, O, \mathcal{C})$ has a collection of constraint sets $\mathcal{C}$ as a component, and assembling multiple modules using some of these rules requires that a complex operation on these collections be performed. We summarize these operations in the definitions presented below. We refer readers to the full report describing our formalism [12] for a more detailed discussion of the reasoning behind these operations.

"collapses" the **let** syntax within network sketches in the other report by placing directly within a hole's syntax the set of network sketches bound to that hole. It also becomes necessary to assume that all hole labels are unique, but this can easily achieved through renaming. This change in the formalism presentation is motivated by a desire to simplify the formal definitions and arguments by removing the need to maintain an explicit environment.

Introduce the constants $\mathbf{cn}, \mathbf{lp}, \mathbf{hl}$.

Assume for any $\mathcal{C}, \mathcal{C}', \theta$,
$\quad \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}') = \{C \cup C' \cup \mathsf{constraints}(\theta) | C \in \mathcal{C}, C' \in \mathcal{C}'\}$.

Assume for any $\mathcal{C}, \theta$,
$\quad \mathbf{lp}(\theta, \mathcal{C}) = \{C \cup \mathsf{constraints}(\theta) | C \in \mathcal{C}\}$.

Assume for any $n, I, O, I', O', \mathcal{C}$,
$\quad \mathbf{hl}(n, I, O, I', O', \mathcal{C}) =$
$\qquad \{C_i \cup \mathsf{constraints}(\phi) \cup \mathsf{constraints}(\psi) |$
$\qquad\quad i \in \{0, \ldots, n\}, \phi \in I' \to I_i, \psi \in O_i \to O', C \in \mathcal{C}\}$.

We introduce a few properties of these operations with respect to satisfaction.

Assume for any $V, \mathcal{C}, \theta$,
$\qquad$ for all $K \in \mathbf{lp}(\theta, \mathcal{C})$, $V \models K$
$\quad$ implies that
$\qquad$ for all $K' \in \mathcal{C}$, $V \models K'$.

Assume for any $V, \mathcal{C}, \mathcal{C}', \theta$,
$\qquad$ for all $K \in \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}')$, $V \models K$
$\quad$ implies that
$\qquad$ for all $K' \in \mathcal{C}$, $V \models K'$ and for all $K'' \in \mathcal{C}'$, $V \models K''$.

Our inference rules are a means by which to inductively construct valid typing judgments. We indicate that a typing judgment is valid by prefixing it with the $\vdash$ (turnstile) symbol.

Introduce the constant $\vdash$.

There are 5 inference rules: [Module], [Connect], [Loop], [Hole], and [Weaken]. We begin with the base case, the inference rule [Module].

Assume for any $A, I, O, \mathcal{C}, C, C'$,
$\qquad (A, I, O, C) : C'$ is a typed module, and
$\qquad C \Rrightarrow C'$
$\quad$ implies that
$\qquad \vdash (A, I, O, \{C\}) : C'$.

The rule [Connect] takes two network sketches, M and N, and returns a network sketch $\mathbf{conn}(\theta, M, N)$ in which some of the output parameters in M are identified with some of the input parameters in N according to what θ prescribes. We present the inference rule [Connect].

Assume for any $M, I, O, \mathcal{C}, C, N, I', O', \mathcal{C}', C', \theta$, for all I'', O'', C'',
$\vdash (M, I, O, \mathcal{C}) : C$,
$\vdash (N, I', O', \mathcal{C}') : C'$,
$\theta \in O \to I'$, θ is injective,
$I'' = I \cup (I' - \mathsf{ran}(\theta))$,
$O'' = O' \cup (O - \mathsf{dom}(\theta))$,
$C'' = C \cup C' \restriction I'' \cup O''$, and
$\mathsf{post}((M, I, O, \mathcal{C}), C) \Rrightarrow \mathsf{pre}((N, I', O', \mathcal{C}'), C') \restriction \mathsf{ran}(\theta)$
implies that
$\vdash (\mathbf{conn}(\theta, M, N), I'', O'', \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}')) : C''$.

Rule [Loop] takes one network sketch, M, and returns a new network sketch $\mathbf{loop}(\theta, M)$ in which some of the output parameters in M are identified with some of the input parameters in M according to θ. We present the inference rule [Loop].

Assume for any $M, I, O, \mathcal{C}, C, \theta$, for all I', O', C',
$\vdash (M, I, O, \mathcal{C}) : C$,
$\theta \in O \to I$, θ is injective, and
$I' = I - \mathsf{ran}(\theta)$,
$O' = O - \mathsf{dom}(\theta)$,
$C' = C \restriction I' \cup O'$, and
$\mathsf{pre}((\mathbf{loop}(\theta, M), I', O', \mathbf{lp}(\theta, \mathcal{C})), C') \Rrightarrow \mathsf{pre}((M, I, O, \mathcal{C}), C)$
implies that
$\vdash (\mathbf{loop}(\theta, M), I', O', \mathbf{lp}(\theta, \mathcal{C})) : C'$.

The rule [Hole] is a little more complicated than the preceding rules. Each of the networks M_i that can be placed into the hole must be typed, and the collection of constraint sets governing the hole (as defined by **hl** further above) must take into consideration, for every network M_i, every possible permutation of the domains of the maps ϕ and ψ that connect M_i to the parameters of the hole. Furthermore, the hole's type must be equivalent to the type of each of the networks M_i under each possible permutation of the connecting maps. Finally, the overall type C' of the hole itself must make it possible to satisfy the definition of validity, so the side condition $\mathsf{pre}((M', I', O', \mathcal{C}'), C') \Rrightarrow \mathsf{post}((M', I', O', \mathcal{C}'), C')$ is stipulated. We present the inference rule [Hole].

Assume for any $n, X, M, I, O, \mathcal{C}, C$, for all $M', I', O', \mathcal{C}', C'$,
for all $i \in \{0, \ldots, n\}$,
$\vdash (M_i, I_i, O_i, \mathcal{C}_i) : C_i$,
for all $\phi \in I' \to I_i, \psi \in O_i \to O'$,
$C' \cup \mathsf{constraints}(\phi) \cup \mathsf{constraints}(\psi) \Lleftrightarrow C_i$,
$M' = \mathbf{hole}(X, \{M_i | i \in \{0, \ldots, n\}\})$,
$\mathcal{C}' = \mathbf{hl}(n, I, O, I', O', \mathcal{C})$,
$\mathsf{pre}((M', I', O', \mathcal{C}'), C') \Rrightarrow \mathsf{post}((M', I', O', \mathcal{C}'), C')$
implies that
$\vdash (M', I', O', \mathcal{C}') : C'$.

In the premises of each of the rules, we introduced crucial side conditions expressing a relationship that must be satisfied by the derived types. These include any premises which are not typing judgments. Some of these may appear restrictive at first, but the rule [Weaken] allows for the adjustment of derived types and

constraints (in particular, their weakening) in order to satisfy the side conditions. We present the inference rule [WEAKEN].

Assume for any $M, I, O, \mathcal{C}, C, C'$,
$\vdash (M, I, O, \mathcal{C}) : C$,
$\mathsf{pre}((M, I, O, \mathcal{C}), C') \Rrightarrow \mathsf{pre}((M, I, O, \mathcal{C}), C)$, and
$\mathsf{post}((M, I, O, \mathcal{C}), C) \Rrightarrow \mathsf{post}((M, I, O, \mathcal{C}), C')$
implies that
$\vdash (M, I, O, \mathcal{C}) : C'$.

5 Soundness

The inference rules for typed network sketches presented in Section 4 are sound with respect to both strong and weak versions of validity. We present a machine-verifiable argument that the inference rules are sound with respect to the strong version of validity, but the proof for the weak version is almost identical. In the full technical report describing this formalist [12], we note within the proof any differences that arise between the proofs for the two kinds of validity.

5.1 Supporting Lemmas

The proof requires a few simple lemmas involving the operators introduced and defined in previous sections. Because we are only interested in performing a lightweight automated verification of our proof, and because we are fairly confident that these simple lemmas are valid, we do not provide proofs for these lemmas. Our confidence in the correctness of our proof of soundness can be improved further by providing machine-verifiable proofs for these lemmas.

Assume for any C, P, P', if $P \subseteq P'$ then $C \restriction P' \restriction P = C \restriction P$.
Assume for any C, P, P', if $P \subseteq P'$ then $\mathsf{closure}(C \restriction P') \restriction P \Rrightarrow \mathsf{closure}(C) \restriction P$.
Assume for any V, C, P, P', if $V \models C \downarrow P$ and $V \models C \restriction P'$ then $V \models C \restriction P' \cup P$.

5.2 Proof of Soundness

The claim is stated formally in Theorem 12. The theorem is proven by an inductive argument for which there exists one base case.

Theorem 12 (Soundness)**.** *If* $\vdash (N, I, O, \mathcal{C}) : C$ *can be derived by the inference rules then for any* V*,* $V \models (N, I, O, \mathcal{C}) : C$*.*

Proof. The theorem holds by induction over the structure of the derivation $\vdash (N, I, O, \mathcal{C}) : C$. Proposition 13 is the base case, and Propositions 14, 15, 16, and 17 cover the four possible inductive cases. □

5.3 Base Case

Modules are the basis of our inductive proof. While it is possible to construct a module A with constraints C for which no V exists that can satisfy C, our definitions of satisfaction and validity handle this by requiring post constraints to be satisfied only when both C and the pre constraints are satisfied. Thus, any module with unreasonably restrictive constraints is trivially valid. Under both these and more routine circumstances, the premises for the inference rule [MODULE] ensure that all typed modules trivially satisfy our theorem.

Proposition 13 (Module)**.** *If we have by the inference rule* [MODULE] *that* $\vdash (A, I, O, \mathcal{C}) : C$ *then it is the case that* $V \models (A, I, O, \mathcal{C}) : C$*.*

Proof. The argument relies on our condition for constraints within the inference rule [MODULE].

Assert for any A, I, O, C, C',
  $(A, I, O, C), C'$ is a typed module and
  $C \Rrightarrow C'$
 implies that
  for any V, if $V \models \mathsf{pre}((A, I, O, \{C\}), C')$ then
   for all V' if $V \subseteq V'$ and for all $K \in \{C\}$, $V' \models K$ then
    $V' \models C$,
    $C' \Rrightarrow \mathsf{post}((A, I, O, \{C\}), C')$, and
    $V' \models \mathsf{post}((A, I, O, \{C\}), C')$.

□

5.4 Inductive Cases

Proposition 14 (Connect)**.** *If* $V \models (M, I, O, \mathcal{C}) : C$*,* $V \models (N, I', O', \mathcal{C}') : C'$*, and we have by the inference rule* [CONNECT] *that*

$$\vdash (\mathbf{conn}(\theta, M, N), I'', O'', \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}')) : C''$$

then it is the case that $V \models (\mathbf{conn}(\theta, M, N), I'', O'', \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}')) : C''$*.*

Proof. The argument is lengthy and relies in part on the fact that the parameters of the constraint sets for the two networks M and N are disjoint.

Assert for any $M, I, O, \mathcal{C}, C, N, I', O', \mathcal{C}', C', \theta$, for all I'', O'', C'',
  I, O are sets, I', O' are sets, C is a constraint set, C' is a constraint set,
  $\mathsf{parameters}(C) \cap \mathsf{parameters}(C') = \emptyset$,
  $\vdash (M, I, O, \mathcal{C}) : C$,
  $\vdash (N, I', O', \mathcal{C}') : C'$,
  $\theta \in O \to I'$, θ is injective,
  $I'' = I \cup (I' - \mathsf{ran}(\theta))$,
  $O'' = O' \cup (O - \mathsf{dom}(\theta))$,
  $C'' = C \cup C' \restriction I'' \cup O''$, and
  $\mathsf{post}((M, I, O, \mathcal{C}), C) \Rrightarrow \mathsf{pre}((N, I', O', \mathcal{C}'), C') \downharpoonright \mathsf{ran}(\theta)$
 implies that

for any V,
if $V \models ((M, I, O, \mathcal{C}) : C)$ and $V \models ((N, I', O', \mathcal{C}') : C')$ then
if $V \models \mathsf{pre}((\mathbf{conn}(\theta, M, N), I'', O'', \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}')), C'')$ then

$\mathsf{pre}((\mathbf{conn}(\theta, M, N), I'', O'', \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}')), C'') = \mathsf{closure}(C'') \restriction I'',$

$V \models \mathsf{closure}(C'') \restriction I'',$
$V \models \mathsf{closure}(C \cup C' \restriction I'' \cup O'') \restriction I'',$

$\mathsf{closure}(C \cup C' \restriction I'' \cup O'') \restriction I'' \Rrightarrow \mathsf{closure}(C \cup C') \restriction I'',$

$V \models \mathsf{closure}(C \cup C') \restriction I'',$

$\mathsf{closure}(C \cup C') \restriction I'' \Rrightarrow \mathsf{closure}(C) \restriction I,$

$V \models \mathsf{closure}(C) \restriction I,$

$\mathsf{pre}((M, I, O, \mathcal{C}), C) = \mathsf{closure}(C) \restriction I,$

$V \models \mathsf{pre}((M, I, O, \mathcal{C}), C),$

for all V', if $V \subseteq V'$ and for all $K' \in \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}')$, $V' \models K'$ then
for all $K' \in \mathcal{C}$, $V' \models K'$,
$V' \models \mathsf{post}((M, I, O, \mathcal{C}), C),$
$V' \models \mathsf{pre}((N, I', O', \mathcal{C}'), C') \downarrow \mathsf{ran}(\theta),$

$\mathsf{pre}((N, I', O', \mathcal{C}'), C') = \mathsf{closure}(C') \restriction I',$

$V' \models \mathsf{closure}(C') \restriction I' \downarrow \mathsf{ran}(\theta),$

$\mathsf{closure}(C') \restriction I'' \Rrightarrow \mathsf{closure}(C') \restriction (I' - \mathsf{ran}(\theta)),$
$\mathsf{closure}(C \cup C') \restriction I'' \Rrightarrow \mathsf{closure}(C') \restriction I'',$

$V' \models \mathsf{closure}(C') \restriction (I' - \mathsf{ran}(\theta)),$

$I' - \mathsf{ran}(\theta) \subseteq I',$
$\mathsf{closure}(C') \restriction I' \restriction (I' - \mathsf{ran}(\theta)) = \mathsf{closure}(C') \restriction (I' - \mathsf{ran}(\theta)),$

$V' \models \mathsf{closure}(C') \restriction I' \restriction (I' - \mathsf{ran}(\theta)),$

$(I' - \mathsf{ran}(\theta)) \cup \mathsf{ran}(\theta) = I',$

$V' \models \mathsf{closure}(C') \restriction I' \restriction (I' - \mathsf{ran}(\theta)) \cup \mathsf{ran}(\theta),$
$V' \models \mathsf{closure}(C') \restriction I' \restriction I',$
$V' \models \mathsf{closure}(C') \restriction I',$
$V' \models \mathsf{pre}((N, I', O', \mathcal{C}'), C'),$
$V' \models (N, I', O', \mathcal{C}') : C',$

for all V'', if $V' \subseteq V''$ then
for all $K'' \in \mathcal{C}'$, $V' \models K''$ and $V'' \models K''$, for all $K' \in \mathcal{C}'$, $V'' \models K'$,
$V'' \models \mathsf{post}((M, I, O, \mathcal{C}), C)$,
$V'' \models \mathsf{post}((N, I', O', \mathcal{C}'), C')$,

$\mathsf{post}((M, I, O, \mathcal{C}), C) = \mathsf{closure}(C) \lfloor O$,
$\mathsf{post}((N, I', O', \mathcal{C}'), C') = \mathsf{closure}(C') \lfloor O'$,

$V'' \models \mathsf{closure}(C) \lfloor O$,
$V'' \models \mathsf{closure}(C') \lfloor O'$,
$V'' \models \mathsf{closure}(C) \lfloor O \cup \mathsf{closure}(C') \lfloor O'$,

$\mathsf{closure}(C) \lfloor O \cup \mathsf{closure}(C') \lfloor O' = \mathsf{closure}(C) \cup \mathsf{closure}(C') \lfloor O \cup O'$,

$V'' \models \mathsf{closure}(C) \cup \mathsf{closure}(C') \lfloor O \cup O'$,

$\mathsf{closure}(C \cup C') = \mathsf{closure}(C) \cup \mathsf{closure}(C')$,

$V'' \models \mathsf{closure}(C \cup C') \lfloor O \cup O'$,

$C \cup C' \subseteq \mathsf{closure}(C \cup C')$,
$C \cup C' \upharpoonright I'' \cup O'' \subseteq C \cup C'$,
$O'' \subseteq O \cup O'$, $C'' \subseteq C \cup C'$,
$\mathsf{closure}(C'') \subseteq \mathsf{closure}(C \cup C')$,
$\mathsf{closure}(C \cup C') \lfloor O \cup O' \Rrightarrow \mathsf{closure}(C'') \lfloor O''$,
$\mathsf{post}((\mathbf{conn}(\theta, M, N), I'', O'', \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}')), C'') = \mathsf{closure}(C'') \lfloor O''$,

$V'' \models \mathsf{post}((\mathbf{conn}(\theta, M, N), I'', O'', \mathbf{cn}(\theta, \mathcal{C}, \mathcal{C}')), C'')$.

□

Proposition 15 (Loop). *If* $V \models (M, I, O, \mathcal{C}) : C$ *and we have by the inference rule* [Loop] *that*

$$\vdash (\mathbf{loop}(\theta, M), I', O', \mathbf{lp}(\theta, \mathcal{C})) : C'$$

then it is the case that $V \models (\mathbf{loop}(\theta, M), I', O', \mathbf{lp}(\theta, \mathcal{C})) : C'$.

Proof. The proof takes advantage of the side conditions in the inference rule.

Assert for any $M, I, O, \mathcal{C}, C, \theta$, for all I', O', C',
M, I, O, C, I', O', C' are sets,
$\vdash (M, I, O, \mathcal{C}) : C$,
$\theta \in O \to I$, θ is injective, and
$I' = I - \mathsf{ran}(\theta)$,
$O' = O - \mathsf{dom}(\theta)$,
$C' = C \upharpoonright I' \cup O'$, and
$\mathsf{pre}((\mathbf{loop}(\theta, M), I', O', \mathbf{lp}(\theta, \mathcal{C})), C') \Rrightarrow \mathsf{pre}((M, I, O, \mathcal{C}), C)$
implies that

for any V,
if $V \models ((M, I, O, \mathcal{C}) : C)$ then
if $V \models \mathsf{pre}((\mathbf{loop}(\theta, M), I', O', \mathbf{lp}(\theta, \mathcal{C})), C')$ then
$V \models \mathsf{pre}((M, I, O, \mathcal{C}), C)$,
for all V',
if $V \subseteq V'$ and for all $K' \in \mathbf{lp}(\theta, \mathcal{C})$, $V' \models K'$ then
for all $K \in \mathcal{C}$, $V' \models K$,
$V' \models \mathsf{post}((M, I, O, \mathcal{C}), C)$,

$\mathsf{post}((M, I, O, \mathcal{C}), C) = \mathsf{closure}(C) \lfloor O$,

$V' \models \mathsf{closure}(C) \lfloor O$,

$O - \mathsf{dom}(\theta) \subseteq O$,
$C \restriction I' \cup O' \subseteq C$,
$\mathsf{closure}(C) \lfloor O \Rrightarrow \mathsf{closure}(C') \lfloor O'$,
$\mathsf{post}((\mathbf{loop}(\theta, M), I', O', \mathbf{lp}(\theta, \mathcal{C})), C') = \mathsf{closure}(C') \lfloor O'$,

$V' \models \mathsf{closure}(C') \lfloor O'$,
$V' \models \mathsf{post}((\mathbf{loop}(\theta, M), I', O', \mathbf{lp}(\theta, \mathcal{C})), C')$.

□

Proposition 16 (Hole)**.** *If $V \models (M_i, I_i, O_i, \mathcal{C}_i) : C_i$ for $i \in \{1, \ldots, n\}$ where M, I, O, and $\mathcal{C}$ are vectors, and we have by the inference rule* [Hole] *that*

$$\vdash (\mathbf{hole}(X, \{M_1, \ldots, M_n\}), I', O', \mathcal{C}') : C'$$

then it is the case that $V \models (\mathbf{hole}(X, \{M_1, \ldots, M_n\}), I', O', \mathcal{C}') : C'$.

Proof. The proof is a straightforward application of the side conditions in the rule [Hole].

Assert for any $n, X, M, I, O, \mathcal{C}, C$, for all $M', I', O', C', \mathcal{C}'$,
for all $i \in \{0, \ldots, n\}$,
$\vdash (M_i, I_i, O_i, \mathcal{C}_i) : C_i$,
for all $\phi \in I' \to I_i, \psi \in O_i \to O'$,
$C' \cup \mathsf{constraints}(\phi) \cup \mathsf{constraints}(\psi) \Lleftarrow\Rrightarrow C_i$,
$M' = \mathbf{hole}(X, \{M_i | i \in \{0, \ldots, n\}\})$,
$\mathcal{C}' = \mathbf{hl}(n, I, O, I', O', \mathcal{C})$,
$\mathsf{pre}((M', I', O', \mathcal{C}'), C') \Rrightarrow \mathsf{post}((M', I', O', \mathcal{C}'), C')$
implies that
for any V, if $V \models (M', I', O', \mathcal{C}') : C'$ then
$V \models \mathsf{pre}((M', I', O', \mathcal{C}'), C')$
implies that
$V \models \mathsf{post}((M', I', O', \mathcal{C}'), C')$,
for all V',
if $V \subseteq V'$ and for all $K \in \mathbf{hl}(n, I, O, I', O', \mathcal{C})$, $V' \models K$ then
$V' \models \mathsf{post}((M', I', O', \mathcal{C}'), C')$.

□

Proposition 17 (Weaken)**.** *If* $V \models (M, I, O, \mathcal{C}) : C$ *and we have by the inference rule* [Weaken] *that* $\vdash (M, I, O, \mathcal{C}) : C'$ *then it is the case that* $V \models (M, I, O, \mathcal{C}) : C'$*.*

Proof. The argument is straightforward.

Assert for any $M, I, O, \mathcal{C}, C, C'$,
$(M, I, O, \mathcal{C}) : C$,
$\mathsf{pre}((M, I, O, \mathcal{C}), C') \Rrightarrow \mathsf{pre}((M, I, O, \mathcal{C}), C)$, and
$\mathsf{post}((M, I, O, \mathcal{C}), C) \Rrightarrow \mathsf{post}((M, I, O, \mathcal{C}), C')$
implies that
for any V,
if $V \models ((M, I, O, \mathcal{C}) : C)$ then
$V \models \mathsf{pre}((M, I, O, \mathcal{C}), C')$
implies that
$V \models \mathsf{pre}((M, I, O, \mathcal{C}), C)$ and
for all V',
if $V \subseteq V'$ and for all $K \in \mathcal{C}$, $V' \models K$ then
$V' \models \mathsf{post}((M, I, O, \mathcal{C}), C)$,
$V' \models \mathsf{post}((M, I, O, \mathcal{C}), C')$.

□

6 Related Work

The work described in this report represents an effort to apply to novel research a formal reasoning system that is designed according to principles that emphasize usability and lightweight verification approaches. However, the DSL and underlying formalism, modelled using a formal reasoning system in this report, is itself a language designed according to such principles. It differs most significantly in that it is intended for use in specific domains that can be modelled as constrained-flow networks. As a result, there are two distinct bodies of related work. We briefly review in Section 6.1 work that deals with the design of accessible and lightweight formal reasoning systems, as well as the application of such systems in modelling formalisms. We review in Section 6.2 related work on other formalisms that can be applied in domains which can be modelled as constrained-flow networks.

6.1 Usability and Application of Formal Reasoning Systems

This report illustrates the usefulness of several characteristics of a formal reasoning system that is designed to simulate a natural context. The interface of the system is familiar, straightforward, and requires no explicit reference of facts as they are applied within a formal argument. These features are recognized as important within several efforts and projects that have similar goals [2, 16, 52, 37, 46, 45]. Furthermore, the system takes advantage of an extensive library of definitions and propositions dealing with common mathematical concepts and provides native support for some of these concepts. This is inspired by work in a subdiscipline of artificial intelligence that deals with the assembly and application of ontologies. Particular examples include the Cyc Project [40] and Open Mind Common Sense [17, 18, 47, 31, 32].

There exist few examples of applications of lightweight formal reasoning systems within *novel* research. Some examples include applications within cryptography [6, 8], security in computation [4, 7, 9, 5], and economic mechanism design [48].

6.2 Formalisms for Modelling Constrained-flow Networks

Our formalism for reasoning about constrained-flow networks was inspired by and based upon formalisms for reasoning about programs developed over the decades within the programming languages community. While our work focuses in particular on networks and constraints on flows, there is much relevant work in the community addressing the general problem of reasoning about distributed programs. However, most previously proposed systems for reasoning in general about the behavior of distributed programs (Process algebra [10], Petri nets [42], Π-calculus [38], finite-state models [34, 35, 36], and model checking [26, 27]) rely upon the retention of details about the *internals* of a system's components in assessing their interactions with one another. While this affords these systems great expressive power, that expressiveness necessarily carries with it a burden of complexity. Such an approach is inherently not modular in its analysis: the details maintained in a representation or model of a component are not easily introduced or removed. Furthermore, these specifications are often wedded to particular methodologies and thus do not have the *generality* necessary to allow multiple kinds of analysis, making it difficult to reason about how systems specified using different methodologies interact.

Discovering and enforcing bounds on execution of program fragments is a well-established problem in computing [55], and our notion of types (*i.e.*, linear constraints) for networks can be viewed as a generalization of type systems expressing upper bounds on program execution times. Existing work on this problem includes the aiT tool (described in [49], and elsewhere), which uses control-flow analysis and abstract interpretation to provide static analysis capabilities for determining worst and best case execution time bounds. Other works, belonging to what have been called Dependent Type Systems, provide capabilities for estimating an upper bound on execution time and memory requirements via a formal type system that has been annotated with size bounds on data types. These include (but are not limited to) Static Dependent Costs [43], Sized Type Systems [28], and Sized Time Systems [33]. Many other Dependent Type Systems directly target resource bounding for the real-time embedded community (*e.g.*, the current incarnation of the Sized Time System [23], Mobile Resource Guarantees for Smart Devices [3]).

More generally, there has been a large interest in applying custom type systems to domain specific languages (which peaked in the late nineties, *e.g.*, the USENIX Conference on Domain-Specific Languages (DSL) in 1997 and 1999). Later type systems have been used to bound other resources such as expected heap space usage (*e.g.*, [25], [3]). The support for constructing, modelling, inferring, and visualizing networks and properties of network constraints provided by our work is similar to the capabilities provided by modelling and checking tools such as Alloy [29].

One of the essential activities our formalism aims to support is reasoning about and finding solution ranges for sets of constraints that describe properties of a network. In its most general form, this is known as the *constraint satisfaction problem* [51] and is widely studied [50]. One variant of the constraint satisfaction problem relevant to our work involves only linear constraints. Finding solutions respecting collections of linear constraints is a classic problem that has been considered in a large variety of work over the decades. There exists a great deal of established material [44], including many documented algorithms [20, Ch. 29], and many analyses of practical considerations [22]. However, most approaches consider a homogenous set of constraints of a particular class.

The work in this paper extends and generalizes our earlier work in Traffic (*Typed Representation and Analysis of Flows For Interoperability Checks* [11]), and complements our earlier work in Chain (*Canonical Homomorphic Abstraction of Infinite Network protocol compositions* [15]).

While our formalism supports the specification and verification of desirable global properties and has a rigorous foundation, it remains ultimately lightweight. By "lightweight" we mean to contrast our work to the heavy-going formal approaches – accessible to a narrow community of experts – which are permeating much of current research on formal methods and the foundations of programming languages (such as the work on automated proof assistants [41, 24, 19, 21], or the work on polymorphic and higher-order type systems [1], or the work on calculi for distributing computing [14]). In doing so, our goal is to ensure that the constructions presented to users are the *minimum* that they might need to accomplish their task.

7 Conclusions

We have utilized a formal reasoning system to define and reason about a novel compositional formalism underlying a typed domain-specific language. This formalism can be used to model and assemble networks, and to reason about and analyze constraints on flows through these networks. We showed that this formalism is sound with respect to its semantics by providing a machine-verified proof of its correctness.

Simultaneously, we were able to demonstrate some of the advantages of using a lightweight formal reasoning system that simulates a natural context. We were able to define the semantics for our formalism in a machine-readable representation that is also highly accessible to humans. We were able to use LaTeX syntax and to introduce user-defined infix operators thanks to the support provided by the formal reasoning system's flexible interface and parser. We were also able to take full advantage of the system's native support for concepts in set theory by employing, without explicit references, laws that govern the relationships between common operations in set theory. Despite the fact that only lightweight verification was employed, the formal assembly process led to the discovery of a few minor errors, and to the simplification of a few side conditions and definitions. The lightweight approach was actually beneficial in allowing us to easily move around verified chunks of an argument without concern for context, something that would be difficult to do when using an interactive theorem proving environment. The lightweight approach also allowed us to introduce and utilize a few lemmas without an explicit proof. We can further improve our confidence in the correctness of our formal argument by providing machine-verifiable proofs of these simple lemmas.

References

[1] *Proceedings of the 8th International Conference on Typed Lambda Calculi and Applications*, Paris, France, June 2007.

[2] A. Abel, B. Chang, and F. Pfenning. Human-readable machine-verifiable proofs for teaching constructive logic, 2001.

[3] D. Aspinall, S. Gilmore, M. Hofmann, D. Sannella, and I. Stark. Mobile resource guarantees for smart devices. In *Construction and Analysis of Safe, Secure and Interoperable Smart Devices: Proceedings of the International Workshop CASSIS 2004*, number 3362 in Lecture Notes in Computer Science, pages 1–26. Springer-Verlag, 2005.

[4] M. Backes, C. Hritcu, and M. Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proceedings of 21st IEEE Computer Security Foundations Symposium (CSF)*, June 2008.

[5] M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proceedings of 20th International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686. Springer, February 2003.

[6] M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound im- plementations using composition and formally verified bisimulation, 2002.

[7] M. Backes, M. Maffei, and D. Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Proceedings of 29th IEEE Symposium on Security and Privacy*, May 2008.

[8] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations, 2003.

[9] M. Backes and M. Schunter. From absence of certain vulnerabilities towards security proofs - pushing the limits of formal verification. In *Proceedings of the 10th ACM Workshop on New Security Paradigms (NSPW)*, pages 67–74, August 2003.

[10] J. Baeten and W. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[11] A. Bestavros, A. Bradley, A. Kfoury, and I. Matta. Typed Abstraction of Complex Network Compositions. In *Proceedings of the 13th IEEE International Conference on Network Protocols (ICNP'05)*, Boston, MA, November 2005.

[12] A. Bestavros, A. Kfoury, A. Lapets, and M. Ocean. Safe Compositional Network Sketches: The Formal Framework. Technical Report BUCS-TR-2009-029, CS Dept., Boston University, October 1 2009.

[13] A. Bestavros, A. Kfoury, A. Lapets, and M. Ocean. Safe Compositional Network Sketches: Tool and Use Cases. Technical Report BUCS-TR-2009-028, CS Dept., Boston University, September 29 2009.

[14] G. Boudol. The π-calculus in direct style. In *97: 24th*, pages 228–241, 1997.

[15] A. Bradley, A. Bestavros, and A. Kfoury. Systematic Verification of Safety Properties of Arbitrary Network Protocol Compositions Using CHAIN. In *Proceedings of ICNP'03: The 11th IEEE International Conference on Network Protocols*, Atlanta, GA, November 2003.

[16] C. E. Brown. Verifying and Invalidating Textbook Proofs using Scunak. In *Mathematical Knowledge Management, MKM 2006*, pages 110–123, Wokingham, England, 2006.

[17] T. Chklovski. Towards managing knowledge collection from volunteer contributors. Proceedings of AAAI Spring Symposium on Knowledge Collection from Volunteer Contributors (KCVC05)., Stanford, CA, 2005.

[18] T. Chklovski and Y. Gil. Improving the design of intelligent acquisition interfaces for collecting world knowledge from web contributors. In *K-CAP '05: Proceedings of the 3rd international conference on Knowledge capture*, pages 35–42, New York, NY, USA, 2005. ACM.

[19] A. Ciaffaglione. *Certified reasoning on Real Numbers and Objects in Co-inductive Type Theory*. PhD thesis, Dipartimento di Matematica e Informatica Università di Udine, Italy, 2003. available as outline.

[20] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Electrical Engineering and Computer Scienece Series. The MIT Press, McGraw-Hill Book Company, 1990.

[21] K. Crary and S. Sarkar. Foundational certified code in a metalogical framework. In *Nineteenth International Conference on Automated Deduction*, Miami, Florida, 2003.

[22] R. Fletcher. *Practical methods of optimization; (2nd ed.)*. Wiley-Interscience, New York, NY, USA, 1987.

[23] K. Hammond, C. Ferdinand, and R. Heckmann. Towards formally verifiable resource bounds for real-time embedded systems. *SIGBED Rev.*, 3(4):27–36, 2006.

[24] H. Herbelin. A λ-calculus structure isomorphic to Gentzen-style sequent calculus structure. In *"Proc. Conf. Computer Science Logic"*, volume 933, pages 61–75. Springer-Verlag, 1994.

[25] M. Hofmann and S. Jost. Static prediction of heap space usage for first-order functional programs. In *POPL03*, pages 185–197. ACM Press, 2003.

[26] G. J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):1–17, May 1997.

[27] G. J. Holzmann and M. H. Smith. A practical method for verifying event-driven software. In *Proc. ICSE99*, pages 597–607, Los Angeles, CA, May 1999.

[28] J. Hughes, L. Pareto, and A. Sabry. Proving the correctness of reactive systems using sized types. In *ACM PoPL*, pages 410–423, 1996.

[29] D. Jackson. Alloy: a lightweight object modelling notation. *Software Engineering and Methodology*, 11(2):256–290, 2002.

[30] A. Lapets. Improving the accessibility of lightweight formal verification systems. Technical Report BUCS-TR-2009-015, Computer Science Department, Boston University, April 30 2009.

[31] H. Liu and P. Singh. Commonsense reasoning in and over natural language. In M. N. R. J. Howlett and L. C. Jain, editors, *Proc. of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES-2004) - Wellingtonand New Zealand*. Springer-Verlag, Septemberand 22-24 2004.

[32] H. Liu and P. Singh. Conceptnet — a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, 2004.

[33] H.-W. Loidl and K. Hammond. A sized time system for a parallel functional language. In *Proceedings of the Glasgow Workshop on Functional Programming*, Ullapool, Scotland, July 1996.

[34] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3)(3):219–246, Sept. 1989.

[35] N. Lynch and F. Vaandrager. Forward and backward simulations – part I: Untimed systems. *Information and Computation*, 121(2):214–233, Sept. 1995.

[36] N. Lynch and F. Vaandrager. Forward and backward simulations – part II: Timing-based systems. *Information and Computation*, 128(1):1–25, July 1996.

[37] D. McMath, M. Rozenfeld, and R. Sommer. A Computer Environment for Writing Ordinary Mathematical Proofs. In *LPAR '01: Proceedings of the Artificial Intelligence on Logic for Programming*, pages 507–516, London, UK, 2001. Springer-Verlag.

[38] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes (Part I and II). *Information and Computation*, (100):1–77, 1992.

[39] C. Panayiotou and B. Bennett. Cognitive context and arguments from ontologies for learning. In *Proceeding of the 2008 conference on Formal Ontology in Information Systems*, pages 65–78, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[40] K. Panton, C. Matuszek, D. Lenat, D. Schneider, M. Witbrock, N. Siegel, and B. Shepard. Common Sense Reasoning – From Cyc to Intelligent Assistant. In Y. Cai and J. Abascal, editors, *Ambient Intelligence in Everyday Life*, volume 3864 of *LNAI*, pages 1–31. Springer, 2006.

[41] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume LNCS 828. Springer-Verlag, 1994.

[42] C. A. Petri. *Communication with Automata*. PhD thesis, Univ. Bonn, 1966.

[43] B. Reistad and D. K. Gifford. Static dependent costs for estimating execution time. In *LISP and Functional Programming*, pages 65–78, 1994.

[44] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & sons, 1998.

[45] W. Sieg and S. Cittadini. Normal Natural Deduction Proofs (in Non-classical Logics). In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning*, volume 2605 of *Lecture Notes in Computer Science*, pages 169–191. Springer, 2005.

[46] J. H. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Proof Development with OMEGA: sqrt(2) Is Irrational. In *LPAR*, pages 367–387, 2002.

[47] P. Singh, B. Barry, and H. Liu. Teaching machines about everyday life. *BT Technology Journal*, 22(4):227–240, 2004.

[48] E. M. Tadjouddine and F. Guerin. Verifying dominant strategy equilibria in auctions. In H.-D. Burkhard, G. Lindemann, R. Verbrugge, and L. Z. Varga, editors, *CEEMAS*, volume 4696 of *Lecture Notes in Computer Science*, pages 288–297. Springer, 2007.

[49] H. Theiling, C. Ferdinand, and R. Wilhelm. Fast and precise wcet prediction by separated cache and path analyses. *Real-Time Syst.*, 18(2-3):157–179, 2000.

[50] E. Tsang. A glimpse of constraint satisfaction. *Artif. Intell. Rev.*, 13(3):215–227, 1999.

[51] E. P. K. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego, 1993.

[52] K. Verchinine, A. Lyaletski, A. Paskevich, and A. Anisimov. On Correctness of Mathematical Texts from a Logical and Practical Point of View. In *Proceedings of the 9th AISC international conference, the 15th Calculemas symposium, and the 7th international MKM conference on Intelligent Computer Mathematics*, pages 583–598, Berlin, Heidelberg, 2008. Springer-Verlag.

[53] M. Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In *TPHOLs '99: Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics*, pages 167–184, London, UK, 1999. Springer-Verlag.

[54] F. Wiedijk. Comparing mathematical provers. In *MKM '03: Proceedings of the Second International Conference on Mathematical Knowledge Management*, pages 188–202, London, UK, 2003. Springer-Verlag.

[55] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *Trans. on Embedded Computing Sys.*, 7(3):1–53, 2008.