Efficient Support for Common Relations in Lightweight Formal Reasoning Systems

Andrei Lapets David House

Abstract

In work that involves mathematical rigor, there are numerous benefits to adopting a representation of models and arguments that can be supplied to a formal reasoning or verification system: 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 [Lap09a], 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. This work expands one aspect of the earlier work by considering more extensively an essential capability for any formal reasoning system whose design is oriented around simulating the natural context: native support for a collection of mathematical relations that deal with common constructs in arithmetic and set theory. We provide a formal definition for a context of relations that can be used to both validate and assist formal reasoning activities. We provide a proof that any algorithm that implements this formal structure faithfully will necessary converge. Finally, we consider the efficiency of an implementation of this formal structure that leverages modular implementations of well-known data structures: balanced search trees and transitive closures of hypergraphs.

1 Introduction

In research areas involving mathematical rigor, as well as in mathematical instruction, there exist many benefits to adopting and using a formal reasoning system. These include reusability, evaluation of examples, and automated verification of formal arguments. Such systems can offer anything from detection of basic errors, such as unbound variables and type mismatches, to full confidence in an argument because it is verifiably constructed using the fundamental principles of a mathematical logic. There exist a variety of such systems, and many of these have been surveyed and compared along a variety of dimensions [Wie03]. However, to date, broad accessibility and quality interface design has not been a priority in the design of formal reasoning systems that can provide these benefits. On the contrary, a researcher hoping to enjoy the benefits of formal verification is presented with a variety of obstacles, both superficial and fundamental. Consequently, many instructors and 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, and in only a few mathematics and computer science courses are such systems employed for presenting material or authoring solutions to assignments.

To address the aforementioned issues, we propose a strategy for improving accessibility and useability of formal verification systems that involves adopting the principles 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 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 [ACP01], the Scunak mathematical assistant system [Bro06], the ForTheL language and SAD proof assistant [VLPA08], the EPGY Theorem-Proving Environment [MRS01], the ΩMEGA proof verifier [SBF+02], and in the work of Sieg and Cittadini [SC05]. One author, commenting on designing an interface for representing proofs, opines that "we seem to be stuck at the assembly language level" [Wen99]. 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 [Jac02].

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 names, references, or indexes but also structure). In explicitly recognizing the possibility that a natural context can contain distinct, inconsistent collections of experiences (such as propositions that are valid in different logical systems), we are in part inspired by the notion of a *cognitive context* found in literature on formal ontologies [PB08].

The notion of a natural context 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 Ontologies and Formal Reasoning Systems

One crucial characteristic of our notion of a natural context is support for retrieval of definitions and propositions *by structure* from an extensive library. Most of the facts or rules being applied in a typical proof found in the literature are *not explicitly referenced*. This is a characteristic of formal reasoning practices that the designers of the Scunak system [Bro06] call "[retrievability] ... by content rather than by name." Likewise, the designers of the `Tutch` system posit that an "explicit reference [to an inference rule] to [humans] interrupts rather than supports the flow of reasoning" [ACP01]. Relevant work on retrieval by structure has even been done with relation to Haskell. This includes the development of search tools that allow users to retrieve and browse expressions

within a context by their type [Kat06], and the introduction of an online search tool called Hoogle for exploring the Haskell libraries [Mit08]. We believe that any effective formal reasoning system should support similar capabilities within the context of mathematical reasoning. Such features could even lead to support for reasoning by analogy. As observed by others working in this area [ACP01], this would be beyond the current state of the art.

1.3 Finite Relation Context

In this work we define, construct, and implement efficiently what we consider to be an essential component for any system that supports formal reasoning by simulating natural contexts. This component has several parts:

(1) a library of formulas, including:

   (a) a collection of relations on formal terms and expressions,

   (b) a collection of formulas (drawn from a restricted class) involving these relations,

   (c) an interface for extending, organizing, and exploring the library;

(2) a data structure for keeping track of all the relations that apply to all the formulas a user introduces and employs while reasoning formally, and a corresponding efficient inference algorithm.

The data structure described in item (2) is not meant to facilitate formal reasoning on its own. It is an integrated component of the simulated natural context within a complete reasoning tool, a prototype of which is described in earlier work [Lap09a].

This work considers the efficiency of our implementation of this component within a system designed to simulate a natural context. We adapt existing data structures (in particular, red-black trees and hypergraphs) and base our implementation on modular libraries that support these data structures. We consider the efficiency of the libraries, as well as that of the algorithms that utilize them, because a responsive user interface is essential for effectively simulating a natural context.

2 Relation Contexts

2.1 Definitions

For any set S, define $P(S)$ to be the power set of S and S^* to be $\cup_{i=0}^{\infty} S^i$. Let $\mathcal{E}$ be the space of all syntactically well-formed (not necessarily closed) logical expressions. For the purposes of this discussion, we assume expressions can contain predicates, operators, and logical operators (including implication, conjunction, and universal quantification, which we denote by $\Rightarrow$, $\wedge$, and $\forall$, respectively). For a full and extensive presentation of the syntax of expressions, we refer the reader to previous work [Lap09a]. Let $\mathcal{R}$ be the set of all predicate and relation symbols. In particular, it is the case that

$$\{\dot{+}, \dot{-}, \dot{:}, \dot{/}, \dot{\cup}, \dot{\cap}, \dot{\times}, \dot{\rightarrow}\} \subset \mathcal{R},$$

but $\mathcal{R}$ also contains all possible user-introduced symbols for operators, relations, and predicates. Let $\mathsf{arity} : \mathcal{R} \rightarrow \mathbb{N}$ be a function that returns the arity of the relation to which each symbol corresponds. As a convention, we will refer using $E \subset \mathcal{E}$ to a *finite* subset of $\mathcal{E}$, and we will refer using $R \subset \mathcal{R}$ to a *finite* subset of $\mathcal{R}$.

Definition 2.1. For a finite $E \subset \mathcal{E}$, let $\mathcal{Q}_E$ be the set of *equivalence contexts* over E, where for all $Q \in \mathcal{Q}_E$ it is always the case that $Q \subset P(E)$.

Definition 2.2. For a finite $R \subset \mathcal{R}$, a finite $E \subset \mathcal{E}$, and $Q \in \mathcal{Q}_E$ let $\mathcal{C}_{Q,R} = P(R \times Q^*)$ be the collection of *relation contexts* over Q and R. Each $C \in \mathcal{C}_{Q,R}$ is a set of vectors. For each vector $\overline{c} \in C$ there exists $r \in R$ such that $\overline{c} \in \{r\} \times Q^{\mathsf{arity}(r)}$. Note that for each $r \in R$, there exists $C_r \subseteq C$ such that $C_r \subseteq \{r\} \times Q^{\mathsf{arity}(r)}$.

Note that for any finite $R \subset \mathcal{R}$, finite $E \subset \mathcal{E}$, and $Q \in \mathcal{Q}_E$ it is possible to represent any individual $C \in \mathcal{C}_{Q,R}$ as a hypergraph in which Q is the set of nodes and each entry in C_r for every $r \in R$ is an edge, labelled r, connecting $\mathsf{arity}(r)$ nodes. Because the range of the function arity is $\mathbb{N}$, such a graph could contain some number of distinct edges that link no nodes. However, this number of edges is bounded because R is finite.

Lemma 2.3. *For any finite $R \subset \mathcal{R}$, finite $E \subset \mathcal{E}$, and $Q \in \mathcal{Q}_E$, it is the case that $\mathcal{C}_{Q,R}$ is of finite size, and for every $C \in \mathcal{C}_{Q,R}$, C is of finite size.*

Proof. Because E is finite, $P(E)$ is finite and so every $Q \subseteq P(E)$ is finite. This means that for any $r \in R$, $Q^{\mathsf{arity}(r)}$ is finite. Because R is finite, there exists a finite number of finite sets $Q^{\mathsf{arity}(r)}$ where $r \in R$. Thus,

$$\bigcup_{r \in R} \Big(\{r\} \times Q^{\mathsf{arity}(r)} \Big) \quad \supseteq \quad C$$

is a finite union of finite sets, so every C is finite. Furthermore, this implies that

$$\mathcal{C}_{Q,R} = P\Big(\bigcup_{r \in R} \Big(\{r\} \times Q^{\mathsf{arity}(r)} \Big) \Big)$$

is finite. □

2.2 Amenable Formulas

For any set S, let $\overline{s}$ be any vector of elements drawn from S^*. We sometimes abuse notation by using $\overline{s}$ to refer to the *set* of elements in the same vector; this is always clear from the context. If $\overline{x}$ and $\overline{y}$ are vectors, then we say that $\overline{x} \subset \overline{y}$ if and only if every variable contained in $\overline{x}$ is also found in $\overline{y}$. Let $\mathcal{X}$ represent the space of all variables.

Definition 2.4. For any finite R, E, Q, we call a formula $e \in \mathcal{E}$ *amenable* if it is of the form

$$\forall \overline{x}, r_1(\overline{u}_1) \wedge \ldots \wedge r_n(\overline{u}_n) \Rightarrow r_{n+1}(\overline{u}_{n+1})$$

where for all i, $r_i \in R$ and $\overline{u}_i \in (\overline{x} \cup Q)^*$. That is, if $u \in \overline{u}_i$, then u may be an equivalence class in Q or a variable in $\overline{x}$.

Note that an amenable formula is a specific kind of Horn formula. Let $\mathcal{A}_{Q,R} \subset \mathcal{E}$ be the set of all amenable formulas under a given Q and R.

For any $\overline{u} \in (\mathcal{X} \cup Q)^*$, let $\overline{u}[\overline{q}/\overline{x}]$ denote the result of a substitution that replaces any instances $u_i \in \overline{u}$ that are in $\overline{x}$ with a corresponding $q \in \overline{q}$, and leaves any instances of $u_i \in \overline{u}$ that are in Q untouched. Let $\equiv$ represent equality over C in any relation context $C \in \mathcal{C}_{Q,R}$. We relate entries in some $C \in \mathcal{C}_{Q,R}$ to formulas according to the following definition.

Definition 2.5. For any $C \in \mathcal{C}_{Q,R}$ and $\overline{x} \in \mathcal{X}^*$, for any $\overline{c} \in C^*$ we say that $\overline{c} = (c_1, \ldots, c_n)$ *matches with respect to* $\overline{x}$ the formula

$$r_1(\overline{u}_1) \wedge \ldots \wedge r_n(\overline{u}_n)$$

if for all i, $\overline{u}_i \subset \overline{x} \cup Q$ and there exists $\overline{c} \in (C_{r_1} \times \ldots \times C_{r_n})$ and $\overline{q} \in Q^*$ such that for all i, $c_i \equiv (r_i, \overline{u}_i[\overline{q}/\overline{x}])$.

We can now define the closure of a relation context.

Definition 2.6. For any $C \in \mathcal{C}_{Q,R}$ and $A \subset \mathcal{A}_{Q,R}$ define $\mathsf{transit}(C, A)$ to be a relation context $C' \in \mathcal{C}_{Q,R}$ where $C \subset C'$ and in which it is the case that for every $a \in A$ of the form

$$\forall \overline{x}, r_1(\overline{u}_1) \wedge \ldots \wedge r_n(\overline{u}_n) \Rightarrow r_{n+1}(\overline{u}_{n+1}),$$

if a collection of edges $\overline{c} \subset C$ matches with respect to $\overline{x}$ the formula $r_1(\overline{u}_1) \wedge \ldots \wedge r_n(\overline{u}_n)$ then $(r_{n+1}, \overline{u}_{n+1}[\overline{c}/\overline{x}]) \in C'$.

Definition 2.7. For any $C \in \mathcal{C}_{Q,R}$ and $A \subset \mathcal{A}_{Q,R}$ define,

$$\begin{aligned} C^{(0)} &= C \\ C^{(i+1)} &= \mathsf{transit}(C^{(i)}, A), \end{aligned}$$

and let

$$\mathsf{closure}(C, A) = \bigcup_{i \in \mathbb{N}} C^{(i)}.$$

We are interested in implementing an efficient algorithm that computes $\mathsf{closure}(C, A)$, and we discuss such an implementation in Section 3. However, we first argue that there necessarily exists an algorithm for computing $\mathsf{closure}(C, A)$ that terminates.

Theorem 2.8. *For any $C \in \mathcal{C}_{Q,R}$ and $A \subset \mathcal{A}_{Q,R}$, $\mathsf{closure}(C, A)$ is of finite size.*

Proof. We know by Lemma 2.3 that any $C \in \mathcal{C}_{Q,R}$ is finite, and there are finitely many such sets. This means that there is an upper bound $b \in \mathbb{N}$ on the size of any set $C \in \mathcal{C}_{Q,R}$.

We know that for any $C \in \mathcal{C}_{Q,R}$, it is the case that $C^{(0)} \in \mathcal{C}_{Q,R}$, so it is finite. For any $C^{(i)} \in \mathcal{C}_{Q,R}$, for any i, if any $\overline{c} \subset C^{(i)}$ matches the left-hand side of any formula $a \in A$, then the substitution $[\overline{q}/\overline{x}]$ cannot introduce any new equivalence classes, so all equivalence classes within $\overline{u}_{n+1}[\overline{q}/\overline{x}]$ must already be drawn from Q. Thus, $C^{(i+1)} \in \mathcal{C}_{Q,R}$, so $C^{(i+1)}$ is also finite.

Furthermore, for any i, one of two cases must hold. If $C^{(i+1)} = C^{(i)}$, then $C^{(j)} = C^{(i)}$ for all $j \geq i$, so $\mathsf{closure}(C, A) = C^{(i)}$, so $\mathsf{closure}(C, A)$ is finite. If $C^{(i+1)} \neq C^{(i)}$, then $C^{(i+1)} \supset C^{(i)}$, which means

$$b - |C^{(i+1)}| < b - |C^{(i)}|.$$

Thus, a single application of $\mathsf{transit}$ must either have no effect, or increase the size of the set. The number of such possible increases is bounded by the finite value $b \in \mathbb{N}$, so $\mathsf{closure}(C, A)$ is finite. □

3 Implementation and Efficiency

Our implementation of a data structure for relation contexts drawn from $\mathcal{C}_{Q,R}$ utilizes two auxiliary data structures: one for the set Q and another for the set C. Both data structures are represented as finite association maps. The set Q is represented as a finite map $M_e : E \to Q$ and the set $C \in \mathcal{C}_{Q,R}$ is represented as a finite map $M_r : R \to \{C_r \mid r \in R\}$.

In our implementation, we represent association maps using purely functional red-black trees. Our implementation is a variant of a classic example of a pure functional implementation of this data structure [Oka99]. The data structure is polymorphic in both the type of the indices and the type of the range values to which indices are mapped (this is essential for our application, as described further below). Like the classic example, our structure supports only insertion of new associations between indices and values, and an update of the value associated with an existing index. There is no support for removal, and such a capability is not required for our application.

3.1 Hypergraph Transitive Closure Algorithm Efficiency

The details of the full verification algorithm within our formal reasoning system are presented in earlier work [Lap09a]. In this report, we focus on a specific scenario relevant to our feature.

Throughout the process of verification of a formal argument, the formal reasoning system maintains a simulated natural context that includes some E, R, Q, and $C \in \mathcal{C}_{Q,R}$. At each point in this process, the system may encounter an additional assumption in the form of an expression introduced by the user. If this expression can be converted to some $(r, \overline{e}) \in R \times E^*$, it is then necessary to determine the equivalence classes corresponding to $\overline{e}$. By using the association map M_e that is implemented using a data structure that supports insertion and retrieval operations of a logarithmic complexity, it is possible to both adjust M_e to some M'_e (if new expressions were introduced) and obtain the list of equivalence classes $\overline{q}$ in time $O(|\overline{e}| \log |Q|)$. Next, it is necessary to compute $C' = \mathsf{closure}(C \cup (r, \overline{e}), A)$. Our algorithm accomplishes this by repeatedly applying the transit operation until it detects convergence.

3.1.1 Upper Bound

Suppose that A is finite, and all the conjunction lists on the left-hand sides of the formulas in A are at most of length n. Furthermore, assume that $k = \max\{\mathsf{arity}(r) \mid r \in R\}$. Note that this also implies that $|C| \leq |Q|^{k+1}$. In Section 3.1.2, we treat n and k as constants from the perspective of a complexity analysis, as they will typically be bounded above by a small integer within any real-world implementation.

In a single computation of transit, for a given amenable formula a containing n subexpressions in its premise, at most n subsets of C must be obtained (corresponding to various subsets C_r), and this takes time $O(n \log |R|)$, which is negligible. It is then necessary to check if any $\overline{c} \in C^n$ matches the premises of a, and this takes time $O(|C|^n)$, or $O(|Q|^{(k+1)n})$. Thus must be repeated for each $a \in A$. In the worst case, each transit computation contributes only one of a possible $|Q|^{k+1}$ new elements, and all the distinct elements are generated in the closure computation before it converges. This results in a worst-case complexity of

$$O(|A| \cdot |Q|^{(k+1)(n+1)}).$$

3.1.2 Practical Considerations

While the worst-case complexity of the closure computation is characterized by a polynomial of a potentially high degree, the computation's complexity under practical conditions is substantially lower. We characterize practical conditions in two ways: by noting constant upper bounds on parameters within the complexity formula, and by assuming a topological restrictions on the hypergraph representing a relation context.

For the collection of about 100 amenable formulas A within our current implementation of the formal reasoning system, the average number of premises in a formula's conjunction list is bounded above by 3. Similarly, the average arity of relations in R is also bounded above by 3. The $|Q|^{k+1}$ factor in our worst-case complexity formula represents a scenario in which each computation of transit adds exactly one new element to the relation context. However, a much more common scenario is one in which this operation adds a collection of elements to the relation context, and only a few more transit computations are necessary to reach convergence. One possible hypothesis is to assume that about $\log(|Q|^{k+1}) = (k+1)\log|Q|$ computations of transit are necessary to reach convergence. Adopting this assumptions and substituting n and k for our upper bounds on their averages, we obtain a complexity of

$$O(|A| \cdot |Q|^4 \cdot 4 \log |Q|).$$

This new complexity formula is less problematic. During the verification of a typical example proof (such as our proof that $\sqrt{2}$ is irrational, presented in an earlier report [Lap09a]), $|Q|$ does not exceed 100, which means our complexity formula's most significant term, $|Q|^4$, does not exceed 10^8. This number of operations can take place in under 4 seconds on a modern 3.0GHz processor with 1GB of RAM.

3.2 Management Interface for Amenable Formula Database

While our definitions and algorithms treat as homogenous the collection of amenable formulas used to compute the transitive closure of the hypergraph, the formulas themselves can be drawn from a diverse variety of disciplines and possibly even multiple logical systems. It is essential that a formal reasoning system that implements the feature we describe in this report can provide an interface that both reveals this collection of formulas, and allows this collection to be extended and categorized. To this end, we have developed a separate representation of amenable formulas (and, in fact, all syntactically well-formed formulas) that is compatible with common implementations of relational databases (such as the SQL family of servers).

We have integrated our formal reasoning system with the MediaWiki content management system [Wik] online,[1] allowing users to verify their formal arguments as they are writing or editing them simply by clicking a button on the edit page. This online application also includes an instance of a relational database containing a collection of propositions involving common mathematical concepts. This database is used directly by the integrated formal reasoning system when it verifies articles. There exists a simple web form that allows a designer to submit new formulas and a user to browse and search for formulas by constant, operator, or predicate. While a formula is associated with particular disciplines in part by the constants, operators, and predicates that it contains, the database includes a simplistic tagging mechanism to better accommodate categorization of formulas. Extending this interface is one of the objectives of future work.

[1] Found at `http://www.safre.org`.

4 Related Work

In earlier work we have demonstrated how a formal reasoning system that simulates a natural context (including the ability to utilize a large collection of facts about common mathematical concepts) can provide lightweight verification of formal arguments within the context of novel research [LK09]. We have also observed the value of native support for an equivalence relation on expressions within a formal reasoning system that was used in a classroom setting to reason about purely functional programs [Lap09b].

The work described in this report represents an effort to join two areas of research that are currently disparate: the assembly and study of general-purpose ontologies, and the development of formal reasoning tools (especially lightweight tools) for mathematics and related fields. There exists a great deal of work in both areas. We review some work dealing with ontologies that inspires our efforts in Section 4.1. We review related work on usability and accessible design of formal reasoning systems in Section 4.2.

4.1 Formal Ontologies and Commonsense Reasoning

Even if one considers a small collection of mathematical concepts, a practicing mathematician is familiar with a great number and variety of propositions that describe relationships between the concepts in this collection. If one attempted to organize these propositions into a hierarchical library, one could find it difficult to decide how these propositions should be grouped, and it is arguable that little would be gained from this exercise. We believe that it is more important to simply assemble a large collection of pertinent propositions. If such a collection is extensive, is indexed by structure and other relevant characteristics, and is accessible to users through a flexible interface, it may be more valuable than an organized library of a more limited scope.

Our views are inspired by work in the subdiscipline of artificial intelligence that deals with the assembly and application of ontologies, and our efforts in designing lightweight formal reasoning systems are in part motivated by a desire to apply this work. Some work in this area deals with applications of semi-formal and formal knowledge databases in the development of software that can assist humans in everyday tasks. One example is the Cyc Project, an effort to assemble "[a] system that rests on a large, general-purpose knowledge base" that "can potentially manage tasks that require world knowledge, or 'common sense' - the knowledge that every person assumes his neighbors also possess" [PML+06]. Instead of working with a database of commonsense propositions about real-world concepts, our work involves the assembly and practical utilization of a database of propositions involving common mathematical concepts.

A large knowledge repository is of limited use without an effective interface for management and exploration, and we attempt to begin addressing this need with our management interface, intended for use by an expert designer or administrator of a formal reasoning system. Our interface is inspired in part by the online search tool Hoogle, designed to allow users to explore the Haskell libraries [Mit08]. It is also influenced by work being done as part of the Open Mind Common Sense project, some of which [Chk05, CG05, SBL04] aims to collect a large database of commonsense propositions [LS04a, LS04b] about the real world from users. Users submit propositions using a simple online submission form, and this database is exposed through an interface that processes user queries. While we currently believe that such an interface within the context of our work should be used only by experts and system designers, we also think that such an interface is an effective tool for accumulating over time and from many experts a large collection of propositions about any

domain, including the one we are considering.

Arguably, our work does not yet involve a database of *computational* methods (algorithms) that can be utilized through an interface, but more complex logical propositions can be used to represent a process or algorithm. In this regard, our work shares its underlying philosophy with that of the WolframAlpha "answer engine" (as opposed to *search* engine) [Wol], which allows users to interact through a natural language interface with a collection of databases that contain both facts and algorithms. However, we do not believe that natural language is necessarily an effective interface for a database of propositions or computational methods if it does not provide a means for specifying the context of a query. In particular, even a human capable of communicating in a natural language cannot answer queries when the queries are posed out of context.

Our work can also be viewed as a variant of type inference in which the usual priorities have been rearranged. Traditional type systems are designed to have terse descriptions and definitions, and this allows proofs about the properties of such systems to be concise. By contrast, we prioritize the assembly of a large collection of inference rules that can contribute to the flexibility of a formal reasoning system, and we offer only lightweight assurances about the system's soundness and consistency.

4.2 Formal Verification Systems

Several formal reasoning systems exist whose development was motivated by observations and goals similar to those that motivated the development of our system, described in further detail in earlier work [Lap09a]. Our system can be viewed in part as further development and integration of the various ideas incorporated into those systems, and furthermore as an example of which priorities and features of those systems we believe should be emphasized (and which are disadvantageous and may need to be removed or modified) if we hope to improve the practical usefulness and accessibility of verification systems.

4.2.1 Accessibility and a Lightweight Approach

In scenarios in which there are established conventions for notation, it is important that these conventions be exploited to the greatest extent possible. Our observations are shared by the designers of Scunak [Bro06], who refer to the need for "naturality" in a system's concrete representation. The designers of the ForTheL language and SAD proof assistant [VLPA08] share our motivations and suggest similar principles to ours. In providing a familiar syntax and supporting verification of formal manipulations without explicit reference to them, the design of our system is similar to Scunak [Bro06]. However, it is worth noting that we disagree with the claim made by the designers of Scunak that proofs of fundamental concepts, such as algebraic distribution laws, are a good touchstone for measuring the success of proof verification systems. On the contrary, focusing on such proofs distracts from efforts directed towards recognizing more common reasoning activities involving both explicit and implicit interactions between many common mathematical concepts. For verification systems used at the college level and beyond, distributivity is one of many natural, implicit manipulations that should be supported without requiring any user effort. We believe that the feature described in this report brings us one step closer to this goal.

The Alloy [Jac02] system supports the construction of static models using first order logic while only providing partial confidence in correctness. This modelling language allows a user to formally define relations and specify a set of constraints on those relations. The system can then perform

a search for counterexamples to these constraints within a finite space. In this regard, the feature we describe in this report is similar to Alloy. However, our system deals with defining and inferring relations over a finite subset of highly structured mathematical expressions rather than potentially infinite spaces of atoms subject to constraints. Furthermore, the purpose of our system is to provide a mechanism to support forward reasoning rather than to analyze a static description of a model. Alloy is intentionally designed to allow partial and incomplete specifications. This lightweight approach is demonstrably useful and inspired the lightweight aspects of the feature described in this report, as well as that of the entire formal reasoning system of which it is a component.

4.2.2 Native Support

The feature described in this report allows the formal reasoning system employing it to provide native support for algebraic manipulations involving common mathematical operators and relations, and this feature is recognized by the authors of the `Tutch` system [ACP01] as essential to making further progress in the design of effective formal reasoning systems. While providing native support has drawbacks when it comes to guaranteeing consistency and makes it more difficult to show a system is fully trustworthy, it can also greatly improve the usability of a formal reasoning system because the system designer has greater flexibility in removing any obstacles a user might face.

The feature described in this report allows our formal reasoning system to track common properties of variables and expressions. For example, for an expression within an equation over real numbers, it is possible to track whether each subexpression is integral, positive, non-zero, and so forth. This capability is essential for verifying common algebraic manipulations, and the EPGY environment [MRS01] provides a similar capability. The EPGY Theorem-Proving Environment is designed for teaching algebra within a classroom setting at the levels of middle and high school. Its design provides extensive support for algebraic manipulations and computations, but no direct support for a lightweight approach. Manipulations are strictly enforced, and authoring arguments requires the use of an interactive interface.

5 Conclusions and Future Work

We have defined a simple and straightforward feature for supporting simple propositions about common mathematical concepts within a formal reasoning system capable of lightweight verification of mathematical arguments. Our definitions facilitate an implementation of this feature that uses two well-known data structures, and this in turn ensures that the feature can be implemented efficiently using well-understood techniques.

There exist multiple directions for further work. The interface for managing and exploring supported propositions can be extended with additional search and analysis capabilities. In particular, it would be useful to be able to analyze, or even visualize, relationships and dependencies between propositions. It would also be useful to have the ability to label and categorize the supported proposition based on certain criteria (such as the logical systems under which they are valid). The ability to search for formulas using structural similarity, as well as to automatically index relationships between analogous formulas, would also be of value.

Further extensions to the context definition are also possible. In particular, it should be possible to extend support to slightly more complex propositions, particularly those involving at least one existential quantifier, or even those containing higher-order predicates. Ensuring that the defined algorithms converge under such a scheme would make for an interesting challenge. Finally, it should

be possible to further improve the efficiency of the implementation. This might be accomplished by developing a more specialized data structure for representing association maps, such as a structure that employs caching in a manner well-suited to our application.

References

[ACP01] A. Abel, B. Chang, and F. Pfenning. Human-readable machine-verifiable proofs for teaching constructive logic, 2001.

[Bro06] Chad E. Brown. Verifying and Invalidating Textbook Proofs using Scunak. In *Mathematical Knowledge Management, MKM 2006*, pages 110–123, Wokingham, England, 2006.

[CG05] Timothy Chklovski and Yolanda 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.

[Chk05] Timothy Chklovski. Towards managing knowledge collection from volunteer contributors. Proceedings of AAAI Spring Symposium on Knowledge Collection from Volunteer Contributors (KCVC05)., Stanford, CA, 2005.

[Jac02] Daniel Jackson. Alloy: a lightweight object modelling notation. *Software Engineering and Methodology*, 11(2):256–290, 2002.

[Kat06] Susumu Katayama. Library for systematic search for expressions. In *AIC'06: Proceedings of the 6th WSEAS International Conference on Applied Informatics and Communications*, pages 381–387, Stevens Point, Wisconsin, USA, 2006. World Scientific and Engineering Academy and Society (WSEAS).

[Lap09a] Andrei Lapets. Improving the accessibility of lightweight formal verification systems. Technical Report BUCS-TR-2009-015, Computer Science Department, Boston University, April 30 2009.

[Lap09b] Andrei Lapets. Lightweight Formal Verification in Classroom Instruction of Reasoning about Functional Code. Technical report, CS Dept., Boston University, November 2009.

[LK09] Andrei Lapets and Assaf Kfoury. Verification with Natural Contexts: Soundness of Safe Compositional Network Sketches. Technical Report BUCS-TR-2009-030, CS Dept., Boston University, October 1 2009.

[LS04a] H. Liu and P. Singh. Commonsense reasoning in and over natural language. In M. Negoitaand 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.

[LS04b] H. Liu and P. Singh. Conceptnet — a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, 2004.

[Mit08] Neil Mitchell. Hoogle overview. *The Monad.Reader*, (12):27–35, November 2008.

[MRS01] David McMath, Marianna Rozenfeld, and Richard 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.

[Oka99] Chris Okasaki. Red-black trees in a functional setting. *J. Funct. Program.*, 9(4):471–477, 1999.

[PB08] Christiana Panayiotou and Brandon 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.

[PML+06] Kathy Panton, Cynthia Matuszek, Douglas Lenat, Dave Schneider, Michael Witbrock, Nick Siegel, and Blake Shepard. Common Sense Reasoning – From Cyc to Intelligent Assistant. In Yang Cai and Julio Abascal, editors, *Ambient Intelligence in Everyday Life*, volume 3864 of *LNAI*, pages 1–31. Springer, 2006.

[SBF+02] Jörg H. Siekmann, Christoph Benzmüller, Armin Fiedler, Andreas Meier, and Martin Pollet. Proof Development with OMEGA: sqrt(2) Is Irrational. In *LPAR*, pages 367–387, 2002.

[SBL04] P. Singh, B. Barry, and H. Liu. Teaching machines about everyday life. *BT Technology Journal*, 22(4):227–240, 2004.

[SC05] Wilfried Sieg and Saverio Cittadini. Normal Natural Deduction Proofs (in Non-classical Logics). In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning*, volume 2605 of *Lecture Notes in Computer Science*, pages 169–191. Springer, 2005.

[VLPA08] Konstantin Verchinine, Alexander Lyaletski, Andrei Paskevich, and Anatoly 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.

[Wen99] Markus 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.

[Wie03] Freek 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.

[Wik] Wikimedia Foundation, Inc. MediaWiki. http://www.mediawiki.org/.

[Wol] Wolfram Alpha L.L.C. WolframAlpha computational knowledge engine. http://www.wolframalpha.com/about.html.