

KAT-ML: An Interactive Theorem Prover for Kleene Algebra with Tests

Kamal Aboul-Hosn Dexter Kozen
kamal@cs.cornell.edu kozen@cs.cornell.edu

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501, USA

Abstract

We describe KAT-ML, an implementation of an interactive theorem prover for Kleene algebra with tests (KAT). The system is designed to reflect the natural style of reasoning with KAT that one finds in the literature. One can also use the system to reason about properties of simple imperative programs using schematic KAT (SKAT). We explain how the system works and illustrate its use with some examples, including an extensive scheme equivalence proof.

1 Introduction

Kleene algebra with tests (KAT), introduced in [21], is an equational system for program verification that combines Kleene algebra (KA), the algebra of regular expressions, with Boolean algebra. KAT has been applied successfully in various low-level verification tasks involving communication protocols, basic safety analysis, source-to-source program transformation, concurrency control, compiler optimization, and dataflow analysis [1, 4, 7, 6, 8, 21, 25]. This system subsumes Hoare logic and is deductively complete for partial correctness over relational models [22].

Much attention has focused on the equational theory of KA and KAT. The axioms of KAT are known to be deductively complete for the equational theory of language-theoretic and relational models, and validity is decidable in *PSPACE* [9, 27]. But because of the practical importance of premises, it is the universal Horn theory that is of more interest; that is, the set of valid sentences of the form

$$p_1 = q_1 \wedge \cdots \wedge p_n = q_n \rightarrow p = q, \tag{1}$$

where the atomic symbols are implicitly universally quantified. Typically, the premises $p_i = q_i$ are assumptions regarding the interaction of atomic programs

and tests, and the conclusion $p = q$ represents the equivalence of an optimized and unoptimized program or of an unannotated and annotated program. The necessary premises are obtained by inspection of the program and their validity may depend on properties of the domain of computation, but they are usually quite simple and easy to verify by inspection, since they typically only involve atomic programs and tests. Once the premises are established, the proof of (1) is purely propositional. This ability to introduce premises as needed is one of the features that makes KAT so versatile. By comparison, Hoare logic has only the assignment axiom for introducing non-propositional structure, which is significantly more limited. In addition, this style of reasoning allows a clean separation between first-order interpreted reasoning to justify the premises $p_i = q_i$ and purely propositional reasoning to establish that the conclusion $p = q$ follows from the premises.

The *PSPACE* decision procedure for the equational theory has been implemented by Cohen [7, 6, 8]. Cohen’s approach is to try to reduce a Horn formula to an equation, then apply the *PSPACE* decision procedure to verify the resulting equation automatically. However, this reduction is not always possible.

KAT can also be used to reason about flowchart schemes in an algebraic framework. A *flowchart scheme* is a vertex-labeled graph that represents an uninterpreted program. This version of KAT, called *schematic KAT* (SKAT), was introduced in [1]. The semantics of SKAT coincides with the semantics of flowchart schemes over a ranked alphabet Σ . A translation to SKAT from a flowchart scheme is possible by considering the scheme to be a *schematic automaton*, a generalization of automata on guarded strings [24]. The equivalence of schematic automata and SKAT expressions, as well as the soundness of the method for scheme equivalence, are proven in [1].

Our system, KAT-ML, allows the user to develop a proof interactively in a natural human style, keeping track of the details of the proof. An unproven theorem has a number of outstanding *tasks* in the form of unproven Horn formulas. The initial task is the theorem itself. The user applies axioms and lemmas to simplify the tasks, which may introduce new (presumably simpler) tasks. When all tasks are discharged, the proof is complete.

As the user applies proof rules, the system constructs a representation of the proof in the form of a λ -term. The proof term of an unproven theorem has free task variables corresponding to the undischarged tasks. The completed proof can be verified and exported to \LaTeX . The system is based on the publish/cite system described in [26].

KAT-ML also has the capability of reasoning at the schematic level. One can input simple imperative programs, translate them to KAT, and then use propositional rules and theorems and schematic axioms to reason about the programs. The formal proof maintained in the system can be regarded as verification of the code’s behavior. Other extensions of KAT such as von Wright’s refinement algebra [36] or Kleene algebra with domain of Desharnais et al. [12] could be supported in the system with few changes.

In this paper we describe KAT-ML and give examples of its use. We have verified formally several known results in the literature, some of which had

previously been verified only by hand, including the KAT translation of the Hoare partial correctness rules [22], a verification problem involving a Windows device driver [2], and an intricate scheme equivalence problem [1]. The last is provided in this paper as an extended example of the system’s capabilities.

The system is implemented in Standard ML and is easy to install and use. Source code and executable images for various platforms are available. Several tutorial examples are also provided. The distribution is available from the KAT-ML website [18].

2 Preliminary Definitions

2.1 Kleene Algebra

Kleene algebra (KA) is the algebra of regular expressions [19, 10]. The axiomatization used here is from [20]. A *Kleene algebra* is an algebraic structure $(K, +, \cdot, *, 0, 1)$ that satisfies the following axioms:

$$\begin{array}{llll}
(p+q)+r = p+(q+r) & (2) & (pq)r = p(qr) & (3) \\
p+q = q+p & (4) & p1 = 1p = p & (5) \\
p+0 = p+p = p & (6) & 0p = p0 = 0 & (7) \\
p(q+r) = pq+pr & (8) & (p+q)r = pr+qr & (9) \\
1+pp^* \leq p^* & (10) & q+pr \leq r \rightarrow p^*q \leq r & (11) \\
1+p^*p \leq p^* & (12) & q+rp \leq r \rightarrow qp^* \leq r & (13)
\end{array}$$

This is a universal Horn axiomatization. We use pq to represent $p \cdot q$. Axioms (2)–(9) say that K is an *idempotent semiring* under $+, \cdot, 0, 1$. The adjective *idempotent* refers to the axiom $p+p=p$ (6). Axioms (10)–(13) say that p^*q is the \leq -least solution to $q+px \leq x$ and qp^* is the \leq -least solution to $q+xp \leq x$, where \leq refers to the natural partial order on K defined by $p \leq q \stackrel{\text{def}}{\iff} p+q=q$.

Standard models include the family of regular sets over a finite alphabet, the family of binary relations on a set, and the family of $n \times n$ matrices over another Kleene algebra. Other more unusual interpretations include the $\min, +$ algebra, also known as the *tropical semiring*, used in shortest path algorithms, and models consisting of convex polyhedra used in computational geometry.

There are several alternative axiomatizations in the literature, most of them infinitary. For example, a Kleene algebra is called *star-continuous* if it satisfies the infinitary property $pq^*r = \sup_n pq^n r$. This is equivalent to infinitely many equations

$$pq^n r \leq pq^* r, \quad n \geq 0 \tag{14}$$

and the infinitary Horn formula

$$\left(\bigwedge_{n \geq 0} pq^n r \leq s \right) \rightarrow pq^* r \leq s. \tag{15}$$

All natural models are star-continuous. However, this axiom is much stronger than the finitary Horn axiomatization given above and would be more difficult

to implement, since it would require meta-rules to handle the induction needed to establish (14) and (15).

The completeness result of [20] says that all true identities between regular expressions interpreted as regular sets of strings are derivable from the axioms. In other words, the algebra of regular sets of strings over the finite alphabet P is the free Kleene algebra on generators P . The axioms are also complete for the equational theory of relational models.

See [20] for a more thorough introduction.

2.2 Kleene Algebra with Tests

A *Kleene algebra with tests* (KAT) [21] is just a Kleene algebra with an embedded Boolean subalgebra. That is, it is a two-sorted structure $(K, B, +, \cdot, *, \bar{}, 0, 1)$ such that

- $(K, +, \cdot, *, 0, 1)$ is a Kleene algebra,
- $(B, +, \cdot, \bar{}, 0, 1)$ is a Boolean algebra, and
- $B \subseteq K$.

Elements of B are called *tests*. The Boolean complementation operator $\bar{}$ is defined only on tests. In KAT-ML, variables beginning with an upper-case character denote tests, and those beginning with a lower-case character denote arbitrary Kleene elements.

The axioms of Boolean algebra are purely equational. In addition to the Kleene algebra axioms above, tests satisfy the equations

$$\begin{array}{ll}
 BC & = CB & BB & = B \\
 B + CD & = (B + C)(B + D) & B + 1 & = 1 \\
 \overline{B + C} & = \overline{B} \overline{C} & \overline{BC} & = \overline{B} + \overline{C} \\
 B + \overline{B} & = 1 & B\overline{B} & = 0 \\
 \overline{\overline{B}} & = B & &
 \end{array}$$

The **while** program constructs are encoded as in propositional Dynamic Logic [13]:

$$\begin{array}{l}
 p; q \stackrel{\text{def}}{=} pq \\
 \text{if } B \text{ then } p \text{ else } q \stackrel{\text{def}}{=} Bp + \overline{B}q \\
 \text{while } B \text{ do } p \stackrel{\text{def}}{=} (Bp)^*\overline{B}.
 \end{array}$$

The Hoare partial correctness assertion $\{B\} p \{C\}$ is expressed as an inequality $Bp \leq pC$, or equivalently as an equation $Bp\overline{C} = 0$ or $Bp = BpC$. Intuitively, $Bp\overline{C} = 0$ says that there is no execution of p for which the input state satisfies the precondition B and the output state satisfies the postcondition \overline{C} , and $Bp = BpC$ says that the test C is always redundant after the execution of p under precondition B . The usual Hoare rules translate to universal Horn

formulas of KAT. Under this translation, all Hoare rules are derivable in KAT; indeed, KAT is deductively complete for relationally valid propositional Hoare-style rules involving partial correctness assertions [22], whereas propositional Hoare logic is not.

The following simple example illustrates how equational reasoning with Horn formulas proceeds in KAT. To illustrate the use of KAT-ML, we will give a mechanical derivation of this lemma in Section 3.5.

Lemma 1 *The following equations are equivalent in KAT:*

- (i) $Cp = C$
- (ii) $Cp + \overline{C} = 1$
- (iii) $p = \overline{C}p + C$.

Proof 1 *We prove separately the four Horn formulas (i) \rightarrow (ii), (i) \rightarrow (iii), (ii) \rightarrow (i), and (iii) \rightarrow (i).*

For the first, assume that (i) holds. Replace Cp by C on the left-hand side of (ii) and use the Boolean algebra axiom $C + \overline{C} = 1$.

For the second, assume again that (i) holds. Replace the second occurrence of C on the right-hand side of (iii) by Cp and use the distributive law $\overline{C}p + Cp = (\overline{C} + C)p$, the Boolean algebra axiom $\overline{C} + C = 1$, and the multiplicative identity axiom $1p = p$.

Finally, for (ii) \rightarrow (i) and (iii) \rightarrow (i), multiply both sides of (ii) or (iii) on the left by C and use distributivity and the Boolean algebra axioms $C\overline{C} = 0$ and $CC = C$ as well as (6) and (7).

See [21, 22, 28] for a more detailed introduction to KAT.

2.3 Schematic KAT

Schematic KAT (SKAT) is a specialization of KAT involving an augmented syntax to handle first-order constructs and restricted semantic actions whose intended semantics coincides with the semantics of first-order flowchart schemes over a ranked alphabet Σ [1]. Atomic actions are assignment operations $x := t$, where x is a variable and t is a Σ -term.

Five identities are paramount in proofs using SKAT:

$$x := s; y := t = y := t[x/s]; x := s \quad (y \notin FV(s)) \quad (16)$$

$$x := s; y := t = x := s; y := t[x/s] \quad (x \notin FV(s)) \quad (17)$$

$$x := s; x := t = x := t[x/s] \quad (18)$$

$$\varphi[x/t]; x := t = x := t; \varphi \quad (19)$$

$$x := x = 1 \quad (20)$$

where x and y are distinct variables and $FV(s)$ is the set of variables occurring in s in (16) and (17). The notation $s[x/t]$ denotes the result of substituting t for

all occurrences of x in s . As special cases of (16) and (19), we have

$$x := s; y := t = y := t; x := s \quad (y \notin FV(s), x \notin FV(t)) \quad (21)$$

$$\varphi; x := t = x := t; \varphi \quad (x \notin FV(\varphi)) \quad (22)$$

$$s = t; x := s = s = t; x := t \quad (23)$$

3 Description of the System

3.1 Rationale for Independent Implementation

We might have implemented KAT in the context of an existing general-purpose automated deduction system such as NuPRL, Isabelle, or Coq. In fact, Isabelle has already been used to reason about Kleene algebra by several researchers. Struth formalizes Church-Rosser proofs in Kleene algebra and checks them using Isabelle [35, 34]. Kahl also works in Isabelle to create theories that could be used to reason about Kleene algebras [17]. He uses the Isar (Intelligible Semi-Automated Reasoning) language [31, 37, 5] and locales [3] to create and display proofs for Kleene algebra and heterogeneous relational algebras. Other proof assistants such as PCP (Point and Click Proofs) [16] emphasize human interaction in proof creation over automation. The PCP system is designed with Javascript to run in a web browser. It facilitates the manual creation of proofs in several algebraic theories, including KA. The system is geared specifically towards web-based presentations of proofs in algebra courses, but does not provide any facility for proof reuse.

We initially considered implementing KAT in the context of NuPRL and MetaPRL [15] and expended considerable effort in this direction. However, we discovered that some aspects of these more complex and general systems make them less desirable for our purposes. Because of their complexity, they tend to have steep learning curves that make them impractical for novice users who just want to experiment with KAT by proving a few theorems. Our experience with NuPRL indicated that installing and learning the system require a level of effort that is prohibitive for all but the most determined user, and are difficult without expert assistance. Moreover, encoding KAT requires the translation of the primitive KAT constructs into the (quite different) primitive NuPRL constructs, a task requiring considerable design effort and orthogonal to our main interest. We were interested in providing a lighter-weight tool that would appeal to naive users, allowing them to quickly understand the system and begin proving theorems immediately. Indeed, an early version of KAT-ML was used successfully by students in an undergraduate course on automata theory to understand and manipulate regular expressions.

Furthermore, systems such as MetaPRL are meant to be general tools for reasoning in several different logics. Because of this generality, it is difficult to take advantage of the structure of a specialized logic such as KAT in the internal data representation. For example, in KAT we know that addition and multiplication are associative, and we can draw advantage from this fact in the form of more efficient data structures for the representation of terms. In

systems such as NuPRL, associativity is not built in, but must be programmed as axioms. Thus proofs contain many citations of associativity to rebalance terms, contributing to their complexity. Similarly, because KAT only deals with universal formulas, most of the infrastructure for quantifier manipulation can remain implicit.

For a theorem prover whose goal is to automate as many of steps as possible, these are not serious issues, but if the goal is to faithfully reflect the equational reasoning style specific to KAT used by humans, they are an undesirable distraction.

3.2 Overview of KAT-ML

KAT-ML is an interactive theorem prover for Kleene algebra with tests. It is written in Standard ML and is available for several platforms. The system has a command-line interface and a graphical user interface. A user can create and manage libraries of KAT theorems that can be proved and cited by name in later proofs. A few standard libraries containing the axioms of KAT and commonly used lemmas are provided. The system is freely available for downloading from the project website [18].

KAT-ML maintains a library of proofs that can be used easily, even by novices. We have used KAT-ML to verify several proofs in the literature, all of which are explained in detail in the distribution and on the KAT-ML website. KAT-ML has been used by others, including the author of [32], who installed, learned, and used the system to prove a theorem for his paper in only a few hours.

At the core of the KAT theorem prover are the commands *publish* and *cite*. Publication is a mechanism for making previous constructions available in an abbreviated form. Citation incorporates previously constructed objects in a proof without having to reconstruct them. All other commands relate to these two in some way. In contrast to other systems, in which these operations are typically implemented at the system level, in KAT-ML they are considered part of the underlying proof theory.

3.3 Representation of Proofs

KAT-ML is a constructive logic in which a theorem is regarded as a type and a proof of that theorem as an object of that type, according to the Curry–Howard Isomorphism [33]. Proofs are represented as λ -terms abstracted over variables p, q, \dots and B, C, \dots ranging over individual elements and tests, respectively, and variables P_0, P_1, \dots ranging over proofs. If the proof is not complete, the proof term also contains free task variables T_0, T_1, \dots for the undischarged tasks. The theorem and its proof can be reconstructed from the proof term.

For instance, consider a theorem

$$\forall x_1 \dots \forall x_m \quad \varphi_1 \rightarrow \varphi_2 \rightarrow \dots \rightarrow \varphi_n \rightarrow \psi,$$

where $\varphi_1, \dots, \varphi_n$ are the premises, ψ is the conclusion, and x_1, \dots, x_m are all of the individual variables that appear in the φ_i or ψ . Viewed as a type, this theorem would be realized by a proof term representing a function that takes an arbitrary substitution for the variables x_i and proofs of the premises φ_j and returns a proof of the conclusion ψ . Initially, the proof is represented as the λ -term

$$\lambda x_1 \dots \lambda x_m. \lambda P_1 \dots \lambda P_n. (TP_1 \dots P_n),$$

where T is a free variable of type $\varphi_1 \rightarrow \varphi_2 \rightarrow \dots \rightarrow \varphi_n \rightarrow \psi$ representing the main task. Publishing the theorem results in the creation of this initial proof term. As proof rules are applied, the proof term is expanded accordingly. Citing a theorem α as a lemma in the proof of another theorem β is equivalent to substituting the proof term of α for a free task variable in the proof term of β . The proof of α need not be complete for this to happen; any undischarged tasks of α become undischarged tasks of β .

3.4 Citation

Citations are applied to the current task. One may cite a published theorem with the command *cite* or a premise of the current task with the command *use*.

The system allows two forms of citation, *focused* and *unfocused*. In unfocused citation, the conclusion of the cited theorem is matched with the conclusion of the current task, giving a substitution of terms for the individual and test variables of the cited theorem. This substitution is then applied to the premises of the cited theorem, and the current task is replaced with several new (presumably simpler) tasks, one for each premise of the cited theorem. Each specialized premise of the cited theorem must now be proved under the premises of the original task.

For example, suppose the current task is

$$\text{T6: } p < r, q < r, r;r < r \mid - p;q + q;p < r$$

indicating that one must prove the conclusion $pq + qp \leq r$ under the three premises $p \leq r, q \leq r$, and $rr \leq r$ (in the display, the symbol $<$ denotes less-than-or-equal-to \leq and $;$ denotes sequential composition). The proof term at this point is

$$\backslash p, q, r. \backslash P0, P1, P2. (\text{T6 } (P0, P1, P2)) \tag{24}$$

(in the display, \backslash represents λ). This means that T6 should return a proof of $pq + qp \leq r$, given proofs P0, P1, and P2 for the three premises.

An appropriate citation at this point would be the lemma

$$\text{sup: } x < z \rightarrow y < z \rightarrow x + y < z$$

The conclusion of *sup*, namely $x + y \leq z$, is matched with the conclusion of the task T6, giving the substitution $x = pq, y = qp, z = r$. This substitution is then applied to the premises of *sup*, and the old task T6 is replaced by the new tasks

T7: $p < r, q < r, r;r < r \mid - p;q < r$
T8: $p < r, q < r, r;r < r \mid - q;p < r$

This operation is reflected in the proof term as follows:

$\backslash p,q,r.\backslash P0,P1,P2.(sup [x=p;q y=q;p z=r] (T7 (P0,P1,P2),$
T8 (P0,P1,P2)))

This new proof term is a function of the same type as (24), but its body has been expanded to reflect the application of the lemma `sup`. The free task variables T7 and T8 represent the remaining undischarged tasks.

A premise can be cited with the command `use` just when the conclusion is identical to that premise, in which case the corresponding task variable is replaced with the proof variable of the cited premise.

Focused citation is used to implement the proof rule of substitution of equals for equals. In focused citation, a subterm of the conclusion of the current task is specified; this subterm is called the *focus*. The system provides a set of navigation commands to allow the user to focus on any subterm. When there is a current focus, any citation will attempt to match either the left- or the right-hand side of the conclusion of the cited theorem with the focus, then replace it with the specialized other side. As with unfocused citation, new tasks are introduced for the premises of the cited theorem. A corresponding substitution is also made in the proof term. In the event that multiple substitutions are possible, the system prompts the user with the available options and applies the one selected.

For example, suppose that the current task is

T0: $p;q = 0 \mid - (p + q)* < q*;p*$

The axiom

R: $x;z + y < z \rightarrow x;y < z$

would be a good one to cite. However, the system will not allow the citation yet, since there is nothing to match y . If the task were

T1: $p;q = 0 \mid - (p + q)*;1 < q*;p*$

then y would match 1. We can make this change by focusing on the left-hand side of the conclusion of T0 and citing the axiom

id.R: $x;1 = x$

Focusing on the desired subterm gives

T0: $p;q = 0 \mid - \underline{(p + q)*} < q*;p*$

where the focus is underlined. Now citing `id.R` matches the right-hand side with the focus and replaces it with the specialized left-hand side of `id.R`, yielding

$$T1: p;q = 0 \mid - (p + q)*;1 < q*;p*$$

At this point we can apply ***R**.

Another useful rule is the *cut* rule. This rule adds a new premise σ to the list of premises of the current task and adds a second task to prove σ under the original premises. Starting from the task $\varphi_1, \dots, \varphi_n \vdash \psi$, the command `cut σ` yields the new tasks

$$\begin{array}{l} \varphi_1, \dots, \varphi_n, \sigma \vdash \psi \\ \varphi_1, \dots, \varphi_n \vdash \sigma. \end{array}$$

3.5 An Extended Example

The following is an example of the system in use. It illustrates the interactive development of the implications (i)→(ii) and (iii)→(i) in the proof of Lemma 1. In the display, \sim represents Boolean negation. The proof demonstrates basic publication and citation, focus, and navigation. For more examples of varying complexity, see the Examples directory in the KAT-ML distribution [18]. The command-line interface is used here instead of the graphical user interface for ease of reading.

```

>pub C p = C -> C p + ~C = 1          C + ~C = 1
L0: C;p = C -> C;p + ~C = 1          -
(1 task)

current task:
T0: C;p = C | - C;p + ~C = 1

>proof
\C,p.\P0.(T0 P0)

current task:
T0: C;p = C | - C;p + ~C = 1

>focus

current task:
T0: C;p = C | - C;p + ~C = 1

C;p + ~C = 1
-----

>down

current task:
T0: C;p = C | - C;p + ~C = 1

C;p + ~C = 1
---

>use A0 1
cite A0

current task:
T1: C;p = C | - C + ~C = 1

C + ~C = 1
-

>unfocus

current task:
T1: C;p = C | - C + ~C = 1

>cite compl+
cite compl+
task completed

no tasks

>proof
\C,p.\P0.(subst [0,0,1] (C;p + ~C = 1)
  L P0 (compl+ [B=C]))

no tasks

>pub p = ~C p + C -> C p = C
L3: p = ~C;p + C -> C;p = C (1 task)

current task:
T15: p = ~C;p + C | - C;p = C

>proof
\C,p.\P3.(T15 P3)

current task:
T15: p = ~C;p + C | - C;p = C

>focus

current task:
T15: p = ~C;p + C | - C;p = C

```

```

C;p = C
---
>r

current task:
T15: p = ~C;p + C |- C;p = C

C;p = C
-

>cite id+L r
cite id+L

current task:
T16: p = ~C;p + C |- C;p = 0 + C

C;p = 0 + C
-----

>d

current task:
T16: p = ~C;p + C |- C;p = 0 + C

C;p = 0 + C
-

>cite annihL r
cite annihL
x=? p

current task:
T17: p = ~C;p + C |- C;p = 0;p + C

C;p = 0;p + C
---

>d

current task:
T17: p = ~C;p + C |- C;p = 0;p + C

C;p = 0;p + C
-

>cite compl. r
cite compl.
B=? C

current task:
T18: p = ~C;p + C |- C;p = C;~C;p + C

C;p = C;~C;p + C
-----

>u r

current task:
T18: p = ~C;p + C |- C;p = C;~C;p + C

C;p = C;~C;p + C
-

>cite idemp. r
cite idemp.

current task:
T19: p = ~C;p + C |- C;p = C;~C;p + C;C

C;p = C;~C;p + C;C
---

>u

current task:
T19: p = ~C;p + C |- C;p = C;~C;p + C;C

C;p = C;~C;p + C;C
-----

>cite distrL r
cite distrL

current task:
T20: p = ~C;p + C |- C;p = C;(~C;p + C)

C;p = C;(~C;p + C)
-----

>unfocus

current task:
T20: p = ~C;p + C |- C;p = C;(~C;p + C)

>cite cong.L
cite cong.L
cite A0
task completed

no tasks

>proof
\C,p.\P3.(subst [1,1] (C;p = C) R
(id+L [x=C])
(subst [1,0,1] (C;p = 0 + C) R
(annihL [x=p]) (subst [1,0,0,1]
(C;p = 0;p + C) R
(compl. [B=C]) (subst [1,1,1]
(C;p = C;~C;p + C) R
(idemp. [B=C]) (subst [1,1]
(C;p = C;~C;p + C;C) R
(distrL [x=C y=~C;p z=C])
(cong.L [x=C y=p z=~C;p + C] P3))))))

no tasks

```

3.6 Heuristics and Reductions

KAT-ML has a set of simple heuristics to aid in proving theorems. It is true that a *PSPACE* decision procedure exists for the equational theory of KAT, including the ability to reduce some Horn formulas to equations, which we

could have used to perform more steps automatically. However, its usefulness is limited. Only certain forms of premises can be reduced to equations. In fact, the Horn theory of star-continuous Kleene algebras and relational Kleene algebras is Π_1^1 -complete [23, 14]. Even limited to premises of the form $ab = ba$, which express the commutativity of primitive operations and occur frequently in program equivalence proofs [6], these theories are undecidable. In general, the decidability of (not necessarily star-continuous) Kleene algebra with Horn formulas containing premises of this form is unknown. We decided to focus our attention on more practical heuristics for KAT-ML.

The heuristics can automatically perform unfocused citation with premises or theorems in the library that have no premises (such as reflexivity) that match the current task. The system also provides a list of suggested citations from the library, both focused and unfocused, that match the current task and focus. Currently, the system does not attempt to order the suggestions, but only provides a list of possible citations.

In addition, KAT-ML has a more complex heuristic system called *reductions*. Reductions are sequences of citations of theorems and premises and focus motion carried out by the system. Reductions are derived from MetaPRL tactics for KAT [15]. A user can create new reductions, store them, and apply them manually or automatically. A reduction is enabled if it can be applied to the current task at the current focus.

The most basic reduction command either cites a theorem or moves the focus. The former is of the form *theorem side*, where *theorem* is the name of a theorem in the library and *side* is *l* or *r*, indicating which side should be used in the matching for a focused citation. The command *move direction* shifts focus left, right, up, or down, when *direction* is *l*, *r*, *u*, or *d*, respectively. The keyword *premises*, which is enabled if any of the premises of the current task can be used, is also a basic reduction.

Reductions can be combined as follows:

$red_1 + red_2$ is enabled if either red_1 or red_2 is enabled
 $red_1 red_2$ is enabled if red_1 is enabled, and after applying red_1 ,
 red_2 is enabled
 $(red)^*$ is always enabled; it applies red as many times as possible.

There are several other special reductions for testing the result of other reductions without actually performing them. These reductions do not change the state of the current task.

$fails [red]$ is true if red is not enabled
 $succeeds [red]$ is true if red is enabled
 $match [term]$ is true if the current focus matches the KAT term $term$.

With the addition of 0 and 1, it is not hard to verify that the language of reductions itself satisfies the axioms of KAT. The reductions *match* and *succeeds* are Boolean terms and *fails* has the same effect as the negation operator. In the system preferences, it is possible to limit the length of time the system tries

to apply reductions or specifically limit the number of times a *-reduction is applied to avoid circularities or nonterminating computations. The user has the ability to create and manage reductions and their application with the command `reduce`.

Reductions are meant to encapsulate common sequences of citations and changes of focus that would otherwise be done manually. For example, a standard sequence of citations in KAT uses premises and Boolean commutativity to move a Boolean term in one direction in a sequence of terms as far as possible, then eliminate it with idempotence. One could specify this reduction as

```
((commut. l + premises);move r)*;idemp. l
```

If the current task were

```
T6: A;b = b;A, A;c = c;A |- A;b;c;D;A = b;c;D;A
```

and the current focus were on `A;b`, the user could use the above reduction sequence to automatically get the new task

```
T7: A;b = b;A, A;c = c;A |- b;c;D;A = b;c;D;A
```

which can be completed with reflexivity of equality.

While our heuristics are not as extensive as the tactics present in several existing theorem provers, their simplicity allows them to be created and applied quickly and easily. As argued by Delahaye [11], complex tactic languages separated from the underlying logic of a theorem prover often complicate systems for both the designer and the user.

3.7 Proof Output and Verification

Once a proof is complete, the system can export it in XML format. There is a separate postprocessor that translates the XML file to \LaTeX source, which produces human-readable output. The exported proof correctly numbers and references assumptions and tasks and prints every step in the proof. With minimal alteration, one could incorporate the proof in a paper. Examples will be given later.

KAT-ML has a built-in verifier. It checks each step of the proof to make sure that it is valid and that there are no circularities in the library. The verifier also exists as a stand-alone program. One could use it to create a central repository of theorems, uploaded by users and verified by the system so that others could download and use them. We have created and tested a prototype of such a system. It is available on the KAT-ML website.

3.8 SKAT in KAT-ML

The KAT theorem prover has the ability to parse simple imperative programming language constructs and translate programs into propositional KAT. One may then cite the schematic axioms (16)–(23) to create and use premises automatically based on schematic properties. The schematic axioms are used only to

establish premises used at the propositional level, where most of the reasoning is done.

The syntax for the imperative language is:

$$\begin{aligned}
A & ::= N \mid S \mid A + A \mid A - A \mid A * A \mid A / A \mid A \% A \mid S(L) \mid (A) \\
B & ::= \text{true} \mid \text{false} \mid A = A \mid A <= A \mid A >= A \mid A > A \mid A < A \mid !B \\
& \quad \mid B \ \&\& \ B \mid B \ || \ B \mid (B) \\
C & ::= S := A \mid \$B \mid \text{if } (B) \text{ then } \{C\} \text{ else } \{C\} \mid \text{while } (B) \text{ do } \{C\} \mid C; C
\end{aligned}$$

Here A , B , and C denote arithmetic expressions, Boolean expressions, and imperative commands, respectively. N , S , and L correspond to the natural numbers, strings, and lists of arithmetic expressions, respectively. The arithmetic operations are addition (+), subtraction (−), multiplication (*), division (/), and mod (%). $S(L)$ represents a function call with a list of arithmetic arguments. For Booleans, we have standard comparisons for arithmetic expressions (=, <=, >=, <, >) and the Boolean operators negation (!), conjunction (&&), and disjunction (||). The operator $\$B$ allows one to execute a Boolean expression as an imperative command or guard. Booleans are programs in KAT, which is very important in the creation of proofs. The $\$$ is used only to resolve an ambiguity in the grammar. A program is a statement C .

In the system, all commands related to first-order terms are managed in the first-order terms window. One can create a new theorem based on programs entered by the user, with any necessary premises. Upon publication of a theorem, KAT-ML maintains a translation table for the user, mapping KAT primitive propositions to assignments and Boolean tests. Once published, the user can create the proof using any of the applicable propositional axioms and theorems, as well as the schematic first-order axioms.

If a schematic axiom is cited, the system translates the necessary terms back into the first-order equivalents, matches them with the axiom, checks necessary preconditions (such as $x \notin FV(s)$), and then replaces the terms with new terms. If necessary, KAT-ML makes propositions out of newly created first-order terms and adds them to the translation table. If the system cannot determine one of the expressions needed in the matching, it prompts the user to fill one in.

The citation of a first-order axiom (16)–(20) is a shortcut for steps normally done manually. The system creates a new premise φ and performs a cut, thereby creating two new tasks, one for the original conclusion with φ as an additional premise and one for proving the conclusion φ under the original premises. The system immediately proves the latter by replacing all occurrences of it in the proof by an application of the first-order axiom.

For example, consider the program $x := 5 ; z := x + 7$. Assume that the system already has these assignments translated to propositional terms such that a represents $x := 5$ and b represents $z := x + 7$. We wish to apply the schematic axiom (16) to the term ab by matching ab with the left-hand side of (16). KAT-ML looks up a and b to find the first-order terms they represent. Next, it attempts to match the terms with $x := s; y := t$. It succeeds, matching x with x , s with 5, y with z , and t with $x + 7$. The system then checks any

necessary preconditions, in this case that x and z are distinct and that z is not a free variable of 5 . These conditions are true, so the system creates a new term and makes propositional substitutions.

Now KAT-ML creates a new first-order term representing the right-hand side of the axiom with the appropriate substitutions made, giving $z := 5 + 7 ; x := 5$. The system creates a new primitive proposition c for $z := 5 + 7$ and translates the new program to the propositional term ca . Now the system performs a cut on the equation $ab = ca$. The first of the two new tasks created is $ab = ca$. It is replaced in the proof term by a special construct including the name of the first-order axiom used and the substitution, thus completing that task. In the other task, ab is replaced with ca using the new premise.

Sometimes first-order unification does not give a unique substitution. Consider trying to replace the assignment $x := 2 + 5$ with $x := 2 ; x := x + 5$ using (18). The system can match x with x and $t[x/s]$ with $2 + 5$, but could choose from infinitely many possibilities for s . Consequently, the system asks the user to input the desired value for s , which is 2 in this case.

As a longer example, consider the following proof from [30]. We wish to prove the following two programs equivalent:

$$\begin{array}{ll} y := x; & x := 2 * x; \\ y := 2 * y; & y := 2 * y; \\ x := 2 * x & y := x \end{array}$$

By hand, the proof requires two citations of (18) and one citation of (16). When we type the programs into KAT-ML, the system creates new propositions a, b , and c , corresponding to $y := x$, $y := 2 * y$, and $x := 2 * x$, respectively. The proof using the system is in Figure 1. The command-line interface is used for ease of reading and movement within the equation is suppressed.

We first focus on ab , which is $y := x ; y := 2 * y$. We then cite (18), matching with the left-hand side. This matches x with y , s with x , and t with $2 * y$. After the substitution, the right-hand side becomes $y := 2 * x$, for which the system creates a new term d and uses the appropriate newly created assumption $ab = d$. Next, we move the focus to dc and cite (16), matching with the right side. As a result, we get the new assumption $ca = dc$, which is used to replace the focused term. Finally, we want to replace a with ba , so we focus on it and cite (18). In this case, the system matches x with y and $t[x/s]$ with x . However, the system cannot find a unique substitution for s , so it asks the user to specify it. We want s to be $2 * y$. Finally, we cite reflexivity of equality to complete the proof. Note how the proof term represents the citation of the schematic axioms as a substitution specifying the name of the axiom and the propositional term that represents each statement in the axiom.

The L^AT_EX output generated by the system for this theorem is in Figure 2. Here S1 and S3 refer to axiom (16) and (18), respectively.

```

current task:
T1: -----
    a;b;c = c;b;a

a;b;c = c;b;a
>focus
no premises

current task:
T1: -----
    a;b;c = c;b;a

a;b;c = c;b;a
---
>cite S3 l
cite S3
cite A0
no premises

current task:
T4: -----
    d;c = c;b;a

d;c = c;b;a
---
>cite S1 r
cite S1
cite A0
no premises

current task:
T7: -----
    c;a = c;b;a

c;a = c;b;a
-

>cite S3 r
cite S3
s?2 * y
cite A0
no premises

current task:
T10: -----
    c;b;a = c;b;a

c;b;a = c;b;a
---
>unfocus
no premises

current task:
T10: -----
    c;b;a = c;b;a

c;b;a = c;b;a
>cite ref=
cite ref=
task completed

no tasks
>proof
\b,a,c,d.(subst [0,0,2] (a;b;c = c;b;a) L
(S3 [x := s=a x := t=b x :=t[x/s]=d])
(subst [0,1] (d;c = c;b;a) R
(S1 [y := t[x/s]=d x := s=c
x := s=c y := t=a])
(subst [0,1,1] (c;a = c;b;a) R
(S3 [x := t[x/s]=a x := s=b x := t=a])
(ref= [x=c;b;a])))

no tasks

```

Figure 1: Proof steps for theorem from [30]

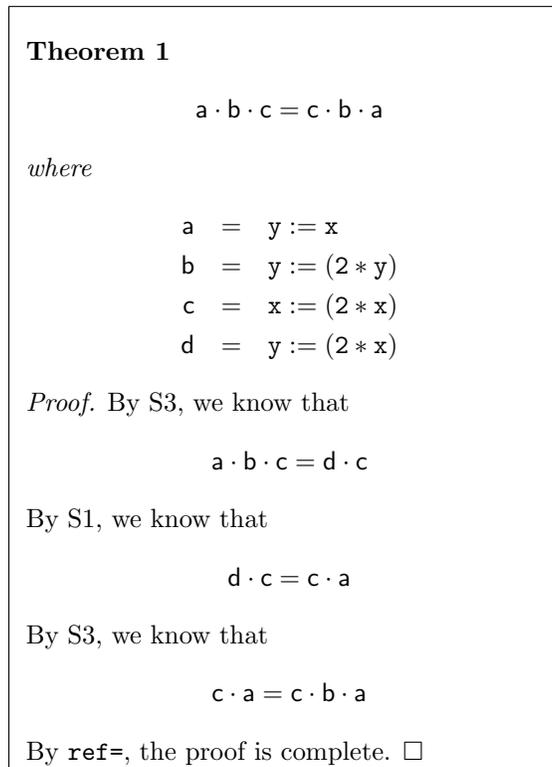


Figure 2: Generated L^AT_EX output

3.9 A Schematic Example

Paterson presents the problem of proving the equivalence of the schemes in Figures 3 and 4. Manna proves the equivalence of the schemes by manipulating the structures of the graphs themselves [29]. Presented in [1] is a proof of the equivalence of the two schemes using the axioms of SKAT and algebraic reasoning. With KAT-ML, the citation of all of the SKAT axioms, with all variable substitutions, is handled by the system.

Without the first-order axioms, it is still possible to prove the equivalence of these schemes. However, it requires that all of the citations of first-order axioms be determined in advance and added as premises to the theorem. The proof was completed successfully without the use of schematic axioms, with a total of 46 premises created manually.

While the proof is correct, it does not explain the origin of the premises. This would be desirable if the proof were distributed and independently verified. With the first-order level of reasoning, the system creates a special substitution in the proof term to indicate that a first-order axiom was cited.

When using the first-order capabilities, we need only five premises, corre-

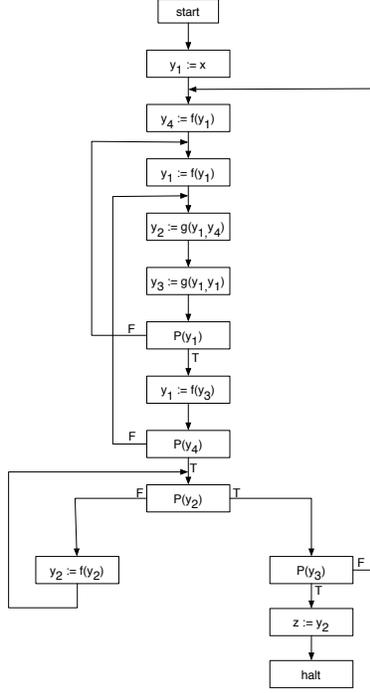


Figure 3: Scheme S_{6A}

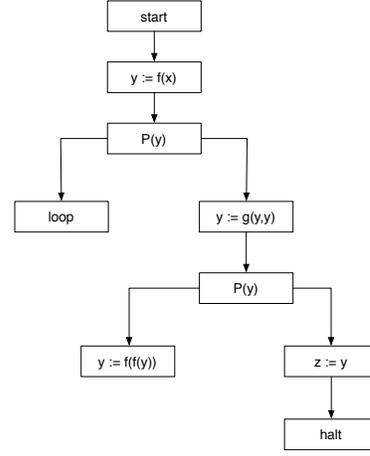


Figure 4: Scheme S_{6E}

sponding to the citation of specific lemmas proven in [1]. Once entered and translated by KAT-ML, the theorem we must prove is in Figure 5.

$$\begin{aligned}
 & I; n; r; (\overline{C}; H; p; s)*; C; i = I; r; (\overline{C}; H; s)*; C; i & (25) \\
 & b; c; d; e; f; (\overline{B}; d; e; f + B; g; \overline{E}; e; f + B; g; E; (\overline{C}; h)*; C; \overline{D}; c; d; e; f)*; B; g; E; \\
 & (\overline{C}; h)*; C; D; i = b; c; d; e; f; (\overline{B}; d; e; f + B; g; \overline{E}; e; f + B; g; E; \\
 & (\overline{C}; h)*; C; \overline{D}; c; d; e; f)*; B; E; (\overline{C}; h)*; C; D; i & (26) \\
 & b; (c; d; t; f; B; g; (\overline{C}; h)*; C; \overline{D})*; c; d; t; f; (\overline{C}; h)*; B; C; D; i = \\
 & b; (d; t; f; B; g; (\overline{C}; h)*; C; \overline{D})*; d; t; f; (\overline{C}; h)*; B; C; D; i & (27) \\
 & n; (B; t; f; \overline{C}; p; (\overline{C}; h)*; C)*; t; f; C; (\overline{C}; h)*; B; C; i = \\
 & n; (B; t; \overline{C}; p; (\overline{C}; h)*; C)*; C; (\overline{C}; h)*; B; C; i & (28) \\
 & o; C; u; (\overline{C}; q; C; u)*; C; i = j; F; k; (\overline{F}; l; F; k)*; F; m & (29)
 \end{aligned}$$

$$\begin{aligned}
 & b; c; d; e; f; (\overline{B}; d; e; f)*; B; g; ((\overline{E} + E; (\overline{C}; h)*; C; \overline{D}; c; d); e; f; (\overline{B}; d; e; f)*; B; g)*; \\
 & E; (\overline{C}; h)*; C; D; i = j; F; k; (\overline{F}; l; F; k)*; F; m
 \end{aligned}$$

Figure 5: Scheme equivalence theorem

The statement of the theorem is not meant to be read directly. The user enters a program at the first-order level. A translation table created by the system, shown for this example in Table 1, can be used to interpret terms. The

| | | | |
|-------|----------------------|-------|----------------------------------|
| B : | $P(y_1) = 1$ | h : | $y_2 := f(y_2)$ |
| C : | $P(y_2) = 1$ | i : | $z := y_2$ |
| D : | $P(y_3) = 1$ | j : | $y := f(x)$ |
| E : | $P(y_4) = 1$ | k : | $y := g(y, y)$ |
| F : | $P(y) = 1$ | l : | $y := f(f(y))$ |
| G : | $P(f(y_1)) = 1$ | m : | $z := y$ |
| H : | $P(f(f(y_2))) = 1$ | n : | $y_1 := f(x)$ |
| I : | $P(f(x)) = 1$ | o : | $y_2 := f(x)$ |
| b : | $y_1 := x$ | p : | $y_1 := f(f(y_2))$ |
| c : | $y_4 := f(y_1)$ | q : | $y_2 := f(f(y_2))$ |
| d : | $y_1 := f(y_1)$ | r : | $y_2 := g(f(x), f(x))$ |
| e : | $y_2 := g(y_1, y_4)$ | s : | $y_2 := g(f(f(y_2)), f(f(y_2)))$ |
| f : | $y_3 := g(y_1, y_1)$ | t : | $y_2 := g(y_1, y_1)$ |
| g : | $y_1 := f(y_3)$ | u : | $y_2 := g(y_2, y_2)$ |

Table 1: Translation table for scheme proof

translation from automata to KAT expressions applies a generalized version of Kleene’s theorem, as described in [1]. The premises (25)–(29) represent lemmas concerning variable elimination and renaming. These lemmas use properties of homomorphisms of KAT expressions, which cannot be handled by the system.

The proof proceeds exactly as in the original paper [1]. We highlight some of the advantageous uses of the system here.

One task that comes up frequently is of the form $a = a(A \leftrightarrow B)$, which says that A and B are equivalent after executing a . For instance, in the scheme equivalence problem above, we need to prove that $tf = tf(C \leftrightarrow D)$, where

$$\begin{aligned}
 t & \text{ is } y_2 := g(y_1, y_1), \\
 f & \text{ is } y_3 := g(y_1, y_1), \\
 C & \text{ is } P(y_2) = 1, \text{ and} \\
 D & \text{ is } P(y_3) = 1.
 \end{aligned}$$

We represent $C \leftrightarrow D$ as $(CD + \overline{C} \overline{D})$. The proof steps are in Figure 6. Changes in focus have been suppressed.

The proof proceeds by using (19) and the laws of Boolean algebra. After citing distributivity, we use (19) to commute C and f , which is possible because $y_3 \notin FV(P(y_2) = 1)$. However, when we apply the axiom to tC , x matches y_2 , which is a free variable in the Boolean test. Therefore, y_2 is replaced by t , which is $g(y_1, y_1)$, creating the new test $P(g(y_1, y_1)) = 1$, represented by the new term K . The other citations of (19) are similar to these two.

Once we have the Booleans on the left-hand side of each sequence, we use Boolean axioms to get the right-hand side of the equality to match the left-hand side, then cite reflexivity. The proof term (Figure 7) reflects our sequence of

| | |
|---------------------------|-------------------------|
| t;f = t;f;(C;D + ~C;~D) | |
| t;f = t;f;(C;D + ~C;~D) | cite S7 |
| ----- | cite A5 |
| cite distrL | t;f = K;t;f + ~K;t;f;~D |
| | ---- |
| t;f = t;f;C;D + t;f;~C;~D | cite S7 |
| --- | cite A6 |
| cite S7 | t;f = K;t;f + ~K;t;~K;f |
| cite A0 | ---- |
| t;f = t;C;f;D + t;f;~C;~D | cite S7 |
| --- | cite A7 |
| cite S7 | t;f = K;t;f + ~K;~K;t;f |
| cite A1 | ----- |
| t;f = K;t;f;D + t;f;~C;~D | cite idemp. |
| --- | t;f = K;t;f + ~K;t;f |
| cite S7 | ----- |
| cite A2 | cite distrR |
| t;f = K;t;K;f + t;f;~C;~D | t;f = (K + ~K);t;f |
| --- | ----- |
| cite S7 | cite compl+ |
| cite A3 | t;f = 1;t;f |
| t;f = K;K;t;f + t;f;~C;~D | ---- |
| --- | cite id.L |
| cite idemp. | t;f = t;f |
| t;f = K;t;f + t;f;~C;~D | --- |
| ---- | t;f = t;f |
| cite S7 | cite ref= |
| cite A4 | t;f = t;f |
| t;f = K;t;f + t;~C;f;~D | task completed |
| ---- | |

Figure 6: Proof steps for $tf = tf(C \leftrightarrow D)$

citations.

When doing the proof manually, it is easy to conclude that $tfC = tfD$, which is the actual step used in the full proof. However, formalizing this equality requires an additional cut and citations of distributivity and some rules related to Booleans.

The complete proof includes more than 50 proven tasks. When exported to \LaTeX , the proof is 41 pages, compared to the 9 pages of the original, hand-constructed proof. The increased size is not unreasonable, given that it is a completely formal, mechanically developed and verified proof of one of Manna's

```

\t,f,C,D,K.(subst [1,1] (t;f = t;f;(C;D + ~C;~D)) L (distrL [x=t;f y=C;D z=~C;~D])
(subst [1,0,1,2] (t;f = t;f;C;D + t;f;~C;~D) R (S7 [x := t=f &phi[x//t]=C x := t=f])
(subst [1,0,0,2] (t;f = t;C;f;D + t;f;~C;~D) R (S7 [x := t=t &phi[x//t]=K x := t=t])
(subst [1,0,2,2] (t;f = K;t;f;D + t;f;~C;~D) R (S7 [x := t=f &phi[x//t]=K x := t=f])
(subst [1,0,1,2] (t;f = K;t;K;f + t;f;~C;~D) R (S7 [x := t=t &phi[x//t]=K x := t=t])
(subst [1,0,0,2] (t;f = K;K;t;f + t;f;~C;~D) L (idemp. [B=K])
(subst [1,1,1,2] (t;f = K;t;f + t;f;~C;~D) R (S7 [x := t=f &phi[x//t]=~C x := t=f])
(subst [1,1,0,2] (t;f = K;t;f + t;~C;f;~D) R (S7 [x := t=t &phi[x//t]=~K x := t=t])
(subst [1,1,2,2] (t;f = K;t;f + ~K;t;f;~D) R (S7 [x := t=f &phi[x//t]=~K x := t=f])
(subst [1,1,1,2] (t;f = K;t;f + ~K;t;~K;f) R (S7 [x := t=t &phi[x//t]=~K x := t=t])
(subst [1,1,0,2] (t;f = K;t;f + ~K;~K;t;f) L (idemp. [B=~K])
(subst [1,1] (t;f = K;t;f + ~K;t;f) R (distrR [x=K y=~K z=t;f])
(subst [1,0,1] (t;f = (K + ~K);t;f) L (compl+ [B=K])
(subst [1,1] (t;f = 1;t;f) L (id.L [x=t;f]) (ref= [x=t;f])))

```

Figure 7: Proof term for $tf = tf(C \leftrightarrow D)$

most difficult examples.

4 Conclusions

We have described an interactive theorem prover for Kleene algebra with tests (KAT). The system provides an intuitive interface with simple commands that allow a user to learn the system quickly. We feel that the most interesting part of this work is not the particular data structures or algorithms we have chosen—these are fairly standard—but rather the design of the mode of interaction between the user and the system.

Our main goal was not to automate as much of the reasoning process as possible, but rather to provide support to the user for developing proofs in a natural human style, similar to proofs in KAT found in the literature. KAT is naturally equational, and equational reasoning pervades every aspect of reasoning with KAT. Our system is true to that style. The user can introduce self-evident equational premises describing the interaction of atomic programs and tests using SKAT and reason under those assumptions to derive the equivalence of more complicated programs. The system performs low-level reasoning and bookkeeping tasks and facilitates sharing of theorems using a proof-theoretic library mechanism, but it is up to the user to develop the main proof strategies. Ultimately, KAT-ML could provide a user-friendly and mathematically sound apparatus for code analysis and verification.

Acknowledgments

We are indebted to Nikita Kuznetsov for his work on the heuristics and reduction system. This work was supported in part by NSF grant CCR-0105586 and ONR Grant N00014-01-1-0968. The views and conclusions contained herein

are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of these organizations or the US Government.

References

- [1] Allegra Angus and Dexter Kozen. Kleene algebra with tests and program schematology. Technical Report 2001-1844, Computer Science Department, Cornell University, July 2001.
- [2] Thomas Ball and Sriram K. Rajamani. Automatically validating temporal safety properties of interfaces. In *Proc. 8th Int. SPIN Workshop on Model Checking of Software (SPIN 2001)*, volume 2057 of *Lect. Notes in Comput. Sci.*, pages 103–122. Springer-Verlag, May 2001.
- [3] Clemens Ballarin. Locales and locale expressions in Isabelle/Isar. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *TYPES*, volume 3085 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2003.
- [4] Adam Barth and Dexter Kozen. Equational verification of cache blocking in LU decomposition using Kleene algebra with tests. Technical Report 2002-1865, Computer Science Department, Cornell University, June 2002.
- [5] Gertrud Bauer and Markus Wenzel. Calculational reasoning revisited—An Isabelle/Isar experience. In Richard J. Boulton and Paul B. Jackson, editors, *Proc. 14th Int. Conf. Theorem Proving in Higher Order Logics (TPHOLs'01)*, volume 2152 of *Lect. Notes in Comput. Sci.* Springer, 2001.
- [6] Ernie Cohen. Hypotheses in Kleene algebra. Technical Report TM-ARH-023814, Bellcore, 1993. <http://citeseer.nj.nec.com/1688.html>.
- [7] Ernie Cohen. Lazy caching in Kleene algebra, 1994. Manuscript. <http://citeseer.nj.nec.com/22581.html>.
- [8] Ernie Cohen. Using Kleene algebra to reason about concurrency control. Technical report, Telcordia, Morristown, N.J., 1994.
- [9] Ernie Cohen, Dexter Kozen, and Frederick Smith. The complexity of Kleene algebra with tests. Technical Report 96-1598, Computer Science Department, Cornell University, July 1996.
- [10] John Horton Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.
- [11] David Delahaye. A tactic language for the system Coq. In Michel Parigot and Andrei Voronkov, editors, *Logics for Programming, Artificial Intelligence, and Reasoning (LPAR'00)*, volume 1955 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2000.

- [12] J. Desharnais, B. Moller, and G. Struth. Kleene algebra with domain. Technical Report 2003-07, Universität Augsburg, Institut für Informatik, June 2003.
- [13] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.
- [14] Chris Hardin and Dexter Kozen. On the complexity of the Horn theory of REL. Technical Report 2003-1896, Computer Science Department, Cornell University, May 2003.
- [15] Jason Hickey, Aleksey Nogin, Robert L. Constable, Brian E. Aydemir, Eli Barzilay, Yegor Bryukhov, Richard Eaton, Adam Granicz, Alexei Kopylov, Christoph Kreitz, Vladimir N. Krupski, Lori Lorigo, Stephan Schmitt, Carl Witty, and Xin Yu. MetaPRL—A modular logical environment. In David Basin and Burkhart Wolff, editors, *Proc. 16th Int. Conf. Theorem Proving in Higher Order Logics (TPHOLs 2003)*, volume 2758 of *LNCS*, pages 287–303. Springer-Verlag, 2003.
- [16] Peter Jipsen. PCP: Point and click proofs, 2001.
<http://www1.chapman.edu/~jipsen/PCP/PCPhome.html>.
- [17] Wolfram Kahl. Calculational relation-algebraic proofs in Isabelle/Isar. In Rudolf Berghammer, Bernhard Möller, and Georg Struth, editors, *Proc. Int. Conf. Relational Methods in Computer Science (RelMiCS'03)*, volume 3051 of *Lecture Notes in Computer Science*, pages 178–190. Springer, 2003.
- [18] <http://www.cs.cornell.edu/projects/kat/>.
- [19] Stephen C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, N.J., 1956.
- [20] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Infor. and Comput.*, 110(2):366–390, May 1994.
- [21] Dexter Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.
- [22] Dexter Kozen. On Hoare logic and Kleene algebra with tests. *Trans. Computational Logic*, 1(1):60–76, July 2000.
- [23] Dexter Kozen. On the complexity of reasoning in Kleene algebra. *Information and Computation*, 179:152–162, 2002.
- [24] Dexter Kozen. Automata on guarded strings and applications. *Matematica Contemporânea*, 24:117–139, 2003.

- [25] Dexter Kozen and Maria-Cristina Patron. Certification of compiler optimizations using Kleene algebra with tests. In John Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luis Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proc. 1st Int. Conf. Computational Logic (CL2000)*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 568–582, London, July 2000. Springer-Verlag.
- [26] Dexter Kozen and Ganesh Ramanarayanan. Publication/citation: A proof-theoretic approach to mathematical knowledge management. Technical Report 2005-1985, Computer Science Department, Cornell University, March 2005.
- [27] Dexter Kozen and Frederick Smith. Kleene algebra with tests: Completeness and decidability. In D. van Dalen and M. Bezem, editors, *Proc. 10th Int. Workshop Computer Science Logic (CSL'96)*, volume 1258 of *Lecture Notes in Computer Science*, pages 244–259, Utrecht, The Netherlands, September 1996. Springer-Verlag.
- [28] Dexter Kozen and Jerzy Tiuryn. Substructural logic and partial correctness. *Trans. Computational Logic*, 4(3):355–378, July 2003.
- [29] Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, 1974.
- [30] Nikolay Mateev, Vijay Menon, and Keshav Pingali. Fractal symbolic analysis. In *Proc. 15th Int. Conf. Supercomputing (ICS'01)*, pages 38–49, New York, 2001. ACM Press.
- [31] Tobias Nipkow. Structured proofs in Isar/HOL. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs (TYPES 2002)*, volume 2646 of *LNCS*, pages 259–278. Springer, 2003.
- [32] Riccardo Pucella. On partially additive Kleene algebras. In *Proc. 8th Int. Conf. Relational Methods in Computer Science (ReMiCS 8)*, February 2005.
- [33] Morten Heine Sørensen and Pawel Urzyczyn. Lectures on the Curry–Howard isomorphism. Technical Report DIKU Rapport 98/14, Department of Computer Science (DIKU), University of Copenhagen, 1998.
- [34] Georg Struth. Isabelle specification and proofs of Church-Rosser theorems, 2001.
<http://www.informatik.uni-augsburg.de/~struth/papers/isabelle>.
- [35] Georg Struth. Calculating Church-Rosser proofs in Kleene algebra. In *Proc. 6th Int. Conf. Relational Methods in Computer Science (ReMiCS'01)*, pages 276–290, London, 2002. Springer-Verlag.
- [36] Joakim von Wright. From Kleene algebra to refinement algebra. In Eerke A. Boiten and Bernhard Möller, editors, *Proc. Conf. Mathematics of Program Construction (MPC'02)*, volume 2386 of *Lect. Notes in Comput. Sci.*, pages 233–262. Springer, July 2002.

- [37] Markus M. Wenzel. *Isabelle/Isar: A Versatile Environment for Human-Readable Formal Proof Documents*. PhD thesis, Institut für Informatik, TU München, 2002.