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

Kamal Aboul-Hosn and Dexter Kozen

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501, USA
{kamal,kozen}@cs.cornell.edu

Abstract. KAT-ML is 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. We describe the main features of the system and illustrate its use with some examples.

1 Introduction

Kleene algebra with tests (KAT), introduced in [13], is an equational system for program verification that combines Kleene algebra (KA) 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, 3–6, 13, 15]. The system subsumes Hoare logic and is deductively complete for partial correctness over relational models [14].

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* [7, 16]. 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

$$p_1 = q_1 \wedge \cdots \wedge p_n = q_n \rightarrow p = q, \qquad (1)$$

where the atomic symbols are implicitly universally quantified. Typically, the premises $p_i = q_i$ are basic assumptions regarding the interaction of atomic programs and tests, and the conclusion $p = q$ represents the equivalence of an optimized and unoptimized program, a partial correctness assertion, or the equivalence of an annotated and unannotated 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 rule, which is much more limited. In addition, this style of reasoning allows a clean separation between first-order interpreted reasoning to justify the premises

$p_1 = q_1 \wedge \cdots \wedge p_n = q_n$ and purely propositional reasoning to establish that the conclusion $p = q$ follows from the premises.

We have implemented an interactive theorem prover KAT-ML for Kleene algebra with tests. The system is designed to reflect the natural style of reasoning with KAT that one finds in the literature. In this paper we describe the main features of the system and illustrate its use with some examples.

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 will have 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 an independently verifiable proof object in the form of a λ-term. The proof term of an unproven theorem has free task variables corresponding to the undischarged tasks. The system can import and export proofs in XML format.

We have used KAT-ML to verify 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 [14], a verification problem involving a Windows device driver [2], and an intricate scheme equivalence problem [1].

The system is implemented in Standard ML and is easy to install and use. Source code and executable images for various platforms can be downloaded from [10]. Several tutorial examples are also provided with the distribution.

The *PSPACE* decision procedure for the equational theory has been implemented by Cohen [4–6]. Cohen's approach is to try to reduce a Horn formula to an equivalent equation, then apply the *PSPACE* decision procedure to automatically verify the resulting equation. This reduction is possible in many cases, but not always. Moreover, the decision procedure does not produce an independently verifiable proof object.

2 Preliminary Definitions

2.1 Kleene Algebra

Kleene algebra (KA) is the algebra of regular expressions [11, 8]. The axiomatization used here is from [12]. A *Kleene algebra* is an algebraic structure $(K, +, \cdot, {}^*, 0, 1)$ that satisfies the following axioms:

$$(p+q)+r = p+(q+r) \qquad (2) \qquad\qquad (pq)r = p(qr) \qquad (3)$$
$$p+q = q+p \qquad (4) \qquad\qquad p1 = 1p = p \qquad (5)$$
$$p+0 = p+p = p \qquad (6) \qquad\qquad 0p = p0 = 0 \qquad (7)$$
$$p(q+r) = pq+pr \qquad (8) \qquad\qquad (p+q)r = pr+qr \qquad (9)$$
$$1+pp^* \le p^* \qquad (10) \qquad\qquad q+pr \le r \rightarrow p^*q \le r \qquad (11)$$
$$1+p^*p \le p^* \qquad (12) \qquad\qquad q+rp \le r \rightarrow qp^* \le r \qquad (13)$$

This a universal Horn axiomatization. Axioms (2)–(9) say that K is an *idempotent semiring* under $+, \cdot, 0, 1$. The adjective *idempotent* refers to (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 $p \leq q \overset{\text{def}}{\iff} p + q = q$.

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 $pq^*r = \sup_n pq^nr$. This is equivalent to infinitely many equations

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

and the infinitary Horn formula

$$(\bigwedge_{n\geq 0} pq^nr \leq s) \to pq^*r \leq s. \tag{15}$$

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 [12] 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 [12] for a more thorough introduction.

2.2 Kleene Algebra with Tests

A *Kleene algebra with tests* (KAT) [13] 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

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

The **while** program constructs are encoded as in propositional Dynamic Logic [9]:

$$p\,;\,q \stackrel{\text{def}}{=} pq$$

$$\textbf{if } B \textbf{ then } p \textbf{ else } q \stackrel{\text{def}}{=} Bp + \overline{B}q$$

$$\textbf{while } B \textbf{ do } p \stackrel{\text{def}}{=} (Bp)^*\overline{B}.$$

The Hoare partial correctness assertion $\{B\}\,p\,\{C\}$ is expressed as an equation $Bp\overline{C} = 0$, or equivalently, $Bp = BpC$. All Hoare rules are derivable in KAT; indeed, KAT is deductively complete for relationally valid propositional Hoare-style rules involving partial correctness assertions [14] (propositional Hoare logic is not).

The following simple example illustrates how equational reasoning with Horn formulas proceeds in KAT. To illustrate the use of our system, we will give a mechanical derivation of this lemma in Section 3.4.

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

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

Proof. We prove separately the four Horn formulas (i) → (ii), (i) → (iii), (ii) → (i), and (iii) → (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 distributivity 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) → (i) and (iii) → (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 $CC = C$.

See [13, 14, 17] for a more detailed introduction to KAT.

3 Description of the System

KAT-ML is an interactive theorem prover for Kleene algebra with tests. It is written in Standard ML. The system has a command-line interface that works on any platform and a graphical user interface that works on any UNIX-based operating system. 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.

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.

3.1 Representation of Proofs

KAT-ML represents a proof as a λ-term abstracted over the individual variables $p, q, \ldots$ and test variables $B, C, \ldots$ that appear in the theorem and proof variables $P_0, P_1, \ldots$ for the premises. If the proof is not complete, the proof term will also contain free task variables $T_0, T_1, \ldots$ for the undischarged tasks. All proof terms are well-typed, and the type is the theorem, according to the Curry-Howard isomorphism [18]. The theorem and its proof can be reconstructed from the proof term.

For instance, consider a universal Horn formula

$$\forall x_1 \ldots \forall x_m \;\; \varphi_1 \to \varphi_2 \to \cdots \to \varphi_n \to \psi,$$

where $\varphi_1, \ldots, \varphi_n$ are the premises, ψ is the conclusion, and $x_1, \ldots, x_m$ 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

$$\lambda x_1 \ldots \lambda x_m . \lambda P_1 \ldots \lambda P_n . (T P_1 \cdots P_n),$$

where T is a free variable of type $\varphi_1 \to \varphi_2 \to \cdots \to \varphi_n \to \psi$ 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 φ as a lemma in the proof of another theorem ψ is equivalent to substituting the proof term of φ for a free task variable in the proof term of ψ. The proof of φ need not be complete for this to happen; any undischarged tasks of φ become undischarged tasks of ψ.

3.2 Citation

The system allows two forms of citation, *focused* and *unfocused*. 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*.

In unfocused citation, the conclusion of the cited theorem is unified with the conclusion of the current task, giving a substitution of terms for the individual variables. 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

```
T6: p < r, q < r, r;r < r |- 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 $\leq$). The proof term at this point is

```
\p,q,r.\P0,P1,P2.(T6 (P0,P1,P2))
```

(in the display, \ represents λ). Here `T6` is a task variable representing a function that returns a proof of $pq + qp \leq r$ when given proofs P_0, P_1, P_2 for the three premises.

To prove $pq + qp \leq r$, it suffices to prove $pq \leq r$ and $qp \leq r$ separately. Thus an appropriate citation at this point would be the lemma

```
sup: x < z -> y < z -> x + y < z
```

The conclusion of `sup`, namely $x + y \leq z$, is unified 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 two new tasks

```
T7: p < r, q < r, r;r < r |- p;q < r
T8: p < r, q < r, r;r < r |- q;p < r
```

This operation is reflected in the proof term as follows:

```
\p,q,r.\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 that returns a proof of $pq + qp \leq r$ when `sup` is provided with proofs of its premises, which are the incomplete proofs `T7(P0,P1,P2)` and `T8(P0,P1,P2)` of $pq \leq r$ and $qp \leq r$, respectively. The arguments of `T7` and `T8` are the proofs `P0,P1,P2` of the premises of the original task `T6`.

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 unify 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 options and applies the one selected.

For example, suppose that the current task is

```
T0: p;q = 0 |- (p + q)* < q*;p*
```

The axiom

```
*R: x;z + y < z -> x*;y < z
```

is a good one to apply. However, the system will not allow the citation yet, since there is nothing to unify with y. If the task were

```
T1: p;q = 0 |- (p + q)*;1 < q*;p*
```

then y would unify with 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 |- (p + q)* < q*;p*
               --------
```

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

```
T1: p;q = 0 |- (p + q)*;1 < q*;p*
               ----------
```

Many other commands exist to facilitate the proving of theorems. The `cut` 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 $\varphi_1, \ldots, \varphi_n \vdash \psi$, the command `cut` σ yields the two new tasks

$$\varphi_1, \ldots, \varphi_n, \sigma \vdash \psi \qquad \varphi_1, \ldots, \varphi_n \vdash \sigma.$$

For a list of other commands, see the README file in the KAT-ML distribution [10].

3.3 Heuristics

KAT-ML has a simple set of heuristics to aid in proving theorems. The heuristics can automatically perform unfocused citation with premises or theorems in the library that have no premises (such as reflexivity) that unify with the current task.

The system also provides a list of suggested citations from the library, both focused and unfocused, that unify with the current task and focus. Currently, the system does not attempt to order the suggestions, but only provides a list of all possible citations. Eventually, the system will attempt to order the list of suggested citations according to some learned priority determined by usage statistics.

3.4 An Extended Example

The following is an example of the system in use. It is the proof of the first and last Horn formulas in Lemma 1. 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 [10].

```
>pub C p = C -> C p + ~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

>heuristics theorem on

no tasks

>heuristics prem on

no tasks

>pub p = ~C p + C -> C p = C
L1: p = ~C;p + C -> C;p = C  (1 task)

current task:
T2: p = ~C;p + C |- C;p = C
```

```
>proof
\C,p.\P1.(T2 P1)

current task:
T2: p = ~C;p + C |- C;p = C

>focus

current task:
T2: p = ~C;p + C |- C;p = C

C;p = C
---

>r

current task:
T2: p = ~C;p + C |- C;p = C

C;p = C
      -

>cite id+L r
cite id+L

current task:
T3: p = ~C;p + C |- C;p = 0 + C

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

>d

current task:
T3: p = ~C;p + C |- C;p = 0 + C

C;p = 0 + C
      -

>cite annihL r
cite annihL
x=? p

current task:
T4: p = ~C;p + C |- C;p = 0;p + C

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

>d

current task:
T4: p = ~C;p + C |- C;p = 0;p + C

C;p = 0;p + C
      -

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

current task:
T5: p = ~C;p + C |- C;p = C;~C;p + C

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

```
>u r

current task:
T5: p = ~C;p + C |- C;p = C;~C;p + C

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

>cite idemp. r
cite idemp.

current task:
T6: p = ~C;p + C |- C;p = C;~C;p + C;C

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

>u

current task:
T6: 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:
T7: p = ~C;p + C |- C;p = C;(~C;p + C)

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

>unfocus

current task:
T7: 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.\P1.(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] P1))))))

no tasks

>
```

4 Conclusions and Future Work

We have described an interactive theorem prover for Kleene algebra with tests (KAT). 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 and reason under those assumptions to derive the equivalence of more complicated programs. The system performs low-level reasoning tasks and bookkeeping and facilitates sharing of lemmas, but it is up to the user to develop the main proof strategies.

Our current focus is to extend the system with first-order constructs, including arrays. Here atomic programs are assignments $x := t$, where x is a program variable and t a first-order term ranging over a domain of computation of a particular first-order signature. There are only a few extra equational axioms needed for most schematic (uninterpreted) first-order reasoning and a single rule for introducing properties of the domain of computation [1, 3]. The first-order axioms are typically used to establish the correctness of premises; once this is done, reasoning reverts to the purely propositional level. A short-term goal is to implement enough first-order infrastructure to support the mechanical derivation of various proofs in first-order KAT appearing in the literature [1, 3].

Acknowledgments

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 *Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN 2001)*, volume 2057 of *Lecture Notes in Computer Science*, pages 103–122. Springer-Verlag, May 2001.
3. 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.
4. Ernie Cohen. Lazy caching in Kleene algebra. `http://citeseer.nj.nec.com/22581.html`.
5. Ernie Cohen. Hypotheses in Kleene algebra. Technical Report TM-ARH-023814, Bellcore, 1993. `http://citeseer.nj.nec.com/1688.html`.
6. Ernie Cohen. Using Kleene algebra to reason about concurrency control. Technical report, Telcordia, Morristown, N.J., 1994.
7. 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.
8. John Horton Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

9. Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979.
10. `http://www.cs.cornell.edu/kozen/KAT-ML.zip`.
11. 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.
12. Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Infor. and Comput.*, 110(2):366–390, May 1994.
13. Dexter Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.
14. Dexter Kozen. On Hoare logic and Kleene algebra with tests. *Trans. Computational Logic*, 1(1):60–76, July 2000.
15. 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.
16. 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.
17. Dexter Kozen and Jerzy Tiuryn. Substructural logic and partial correctness. *Trans. Computational Logic*, 4(3):355–378, July 2003.
18. Morten Heine Sørensen and Pawel Urzyczyn. Lectures on the Curry–Howard isomorphism. Available as DIKU Rapport 98/14, 1998.