Nonlocal Flow of Control and Kleene Algebra with Tests

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

Abstract

Kleene algebra with tests (KAT) *is an equational system for program verification that combines Kleene algebra* (KA)*, or the algebra of regular expressions, with Boolean algebra. It can model basic programming and verification constructs such as conditional tests, while loops, and Hoare triples, thus providing a relatively simple equational approach to program equivalence and partial correctness. In this paper we show how* KAT *can be used to give a rigorous equational treatment of control constructs involving nonlocal transfer of control such as unconditional jumps, loop statements with multi-level breaks, and exception handlers. We develop a compositional semantics and a complete equational axiomatization. The approach has some novel technical features, including a treatment of multi-level* `break` *statements that is reminiscent of de Bruijn indices in the variable-free lambda calculus. We illustrate the use of the system by giving a purely calculational proof that every deterministic flowchart is equivalent to a* `loop` *program with multi-level breaks.*

1 Introduction

Kleene algebra with tests (KAT) is an equational system for program verification that combines Kleene algebra (KA), or the algebra of regular expressions, with Boolean algebra. It can model basic programming and verification constructs such as conditional tests, while loops, and Hoare triples, thus providing a relatively simple equational approach to program equivalence and partial correctness. It has been applied successfully in numerous low-level verification tasks involving communication protocols, source-to-source program transformations, concurrency control, compiler optimization, and dataflow analysis [1, 3, 6, 7, 15, 16, 18, 21, 23]. The formalism allows a clean separation between first-order reasoning to establish basic premises, which may take properties of the domain of computation into account, and program manipulation, which is chiefly propositional.

There is an extensive model theory with various classes of language, relational, trace, and matrix models. All these classes share the same equational theory, and KAT is deductively complete for this theory. The relational models are of particular interest in programming language semantics. KAT subsumes propositional Hoare logic (PHL) and is deductively complete for all relationally valid Hoare-style rules, whereas PHL is not [19]. The equational theory is *PSPACE*-complete, the same as KA and PHL [8].

There is a coalgebraic theory of KAT based on a generalization of classical automata theory to include Booleans [20, 22]. The generalized automata are known as *automata with tests* or *automata on guarded strings*. An ordinary automaton with ε-transitions is an automaton with tests over the two-element Boolean algebra 2. This theory provides a more rigorous algebraic foundation for classical program schematology that allows simpler and more rigorous equivalence and inequivalence proofs [1, 25]. Informal combinatorial arguments and surgery on graph models are replaced with purely calculational equational proofs that are more amenable to automation.

Although KAT is particularly well suited for reasoning about `while` programs (programs constructed from atomic actions and tests with sequential composition, conditionals, and while loops), there is a misconception that it is less well suited for dealing with more general control constructs involving nonlocal transfer of control such as `goto` statements, `loop` statements with multi-level `break`s, and exception handlers. These constructs are typically handled with higher-order methods involving continuation passing.

In this paper, we attempt to dispel this misconception. We present a compositional semantics for `goto`, `loop`, and multi-level `break` statements and a complete equational axiomatization. The approach has some novel technical features, including an interesting treatment of the `break` n statement that is reminiscent of de Bruijn indices in the variable-free lambda calculus [10]. We illustrate the use of the system by giving a purely calculational proof of the classical folklore result that every deterministic flowchart is equivalent to a `loop` program with multi-level breaks.

2 Definitions

2.1 Kleene Algebra

Kleene algebra (KA) is the algebra of regular expressions [9, 13]. The axiomatization here is from [17]. A *Kleene algebra* is a structure $(K, +, \cdot, ^*, 0, 1)$ such that K is an idempotent semiring under $+$, $\cdot$, 0, and 1 and satisfies the axioms

$$\begin{array}{ll} 1 + pp^* \le p^* & q + pr \le r \Rightarrow p^*q \le r \\ 1 + p^*p \le p^* & q + rp \le r \Rightarrow qp^* \le r \end{array}$$

for *. There is a natural partial order $p \le q \overset{\text{def}}{\iff} p+q = q$.

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 [11].

The completeness result of [17] says that the algebra of regular sets of strings over a finite alphabet Σ is the free Kleene algebra on generators Σ. The axioms are also complete for the equational theory of relational models.

2.2 Kleene Algebra with Tests

A *Kleene algebra with tests* (KAT) [18] consists of a Kleene algebra K with an embedded Boolean algebra B such that the semiring structure on B is a subalgebra of the semiring structure on K. Elements of B are called *tests*. The Boolean negation operator is defined only on tests.

Like KA, KAT has language and relational models and is deductively complete over these interpretations [24]. The chief language-theoretic models are the regular sets of *guarded strings* over alphabets Σ and T of primitive action and test symbols, respectively (see Section 2.3). This is the free KAT on generators Σ, T. The set of guarded strings represented by a KAT expression e is denoted $\mathsf{GS}(e)$.

KAT can code elementary programming constructs and Hoare partial correctness assertions and subsumes propositional Hoare logic (PHL). It is deductively complete over relational models, whereas PHL is not. Moreover, KAT is no more difficult to decide, as PHL, KA, and KAT are all *PSPACE*-complete.

For KAT expressions e, e', we write $e \le e'$ or $e = e'$ if the relation holds in the free KAT on generators Σ, T; that is, if it is a consequence of the axioms of KAT.

See [17, 18, 19] for a more detailed introduction.

2.3 Guarded Strings

Guarded strings were introduced in [12]. Let Σ be a finite set of *action symbols* and T a finite set of *test symbols* disjoint from Σ. The symbols T generate a free Boolean algebra B; elements of B are called *tests*. An *atom* is a minimal nonzero element of B. The set of atoms is denoted At. The elements of At can be regarded either as conjunctions of literals of T (elements of T or their negations) or as truth assignments to T. We write $p, q, p_0, \ldots$ for elements of Σ and $\alpha, \beta, \alpha_0, \ldots$ for elements of At. A *guarded string* is a finite alternating sequence of atoms and actions, beginning and ending with an atom; that is, an element of $(\mathsf{At}\cdot\Sigma)^*\cdot\mathsf{At}$. We will also refer to infinite guarded strings, which are members of $(\mathsf{At}\cdot\Sigma)^\omega$, but will always qualify with the adjective "infinite" when doing so.

2.4 Automata with Tests

Automata with tests, also known as automata on guarded strings, were introduced in [20]. They are the automata-theoretic counterpart to Kleene algebra with tests (KAT). In the formalism of [20], they have two types of transitions, *action transitions* and *test transitions*, and operate over guarded strings. An ordinary automaton with ε-transitions is just an automaton with tests over the two-element Boolean algebra 2. Many of the constructions of ordinary finite-state automata, such as determinization and state minimization, generalize readily to automata with tests. In particular, there is a version of Kleene's theorem showing that these automata are equivalent in expressive power to expressions in the language of KAT.

Deterministic flowcharts correspond to a limited class of automata called *strictly deterministic* [25]. Intuitively, a strictly deterministic automaton operates by starting in its start state and scanning a sequence of atoms, which we can view as provided by an external agent. For each atom in succession, the automaton responds deterministically either by emitting an action symbol and moving to a new state, or by simply halting, according to its transition function.

Formally, a *strictly deterministic automaton* over Σ and T is a tuple

$$M \quad = \quad (Q,\, \delta,\, \mathsf{start}),$$

where Q is a finite set of *states*, $\mathsf{start} \in Q$ is the *start state*, and δ is a *transition function*

$$\delta : Q \ \to\ \mathsf{At} \ \to\ (\Sigma \times Q) + \{\mathsf{halt}\},$$

where $+$ denotes disjoint (marked) union. The element halt is not a state, but a universal constant used by an automaton to signal halting.

Given a state s and an infinite sequence of atoms σ, there is a unique finite or infinite guarded string $\mathsf{gs}(s)(\sigma)$ obtained by running the automaton starting in state s. Formally, the map

$$\mathsf{gs} : Q \ \to\ \mathsf{At}^\omega \ \to\ (\mathsf{At}\cdot\Sigma)^* \cdot \mathsf{At} \ +\ (\mathsf{At}\cdot\Sigma)^\omega$$

is defined coinductively as follows:

$$\mathsf{gs}(s)(\alpha\,\sigma) \quad \stackrel{\text{def}}{=} \quad \begin{cases} \alpha \cdot p \cdot \mathsf{gs}(t)(\sigma), & \text{if } \delta(s)(\alpha) = (p,t), \\ \alpha, & \text{if } \delta(s)(\alpha) = \mathsf{halt}. \end{cases}$$

This determines $\mathsf{gs}(s)(\sigma)$ uniquely for all $s \in Q$ and $\sigma \in A^\omega$.

The set of (finite) guarded strings represented by the automaton M is

$$\mathsf{GS}(M) \quad \stackrel{\text{def}}{=} \quad \{\mathsf{gs}(\mathsf{start})(\sigma) \mid \sigma \in \mathsf{At}^\omega\} \cap (\mathsf{At}\cdot\Sigma)^* \cdot \mathsf{At}.$$

Two automata are considered equivalent if they represent the same set of finite guarded strings.

2.5 Programming Constructs

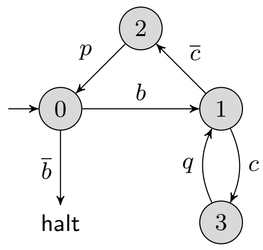
Deterministic `while` programs are formed inductively from sequential composition $(p\,;q)$, conditional test (`if` b `then` p `else` q), and while loops (`while` b `do` p), where b is a test and p, q are programs. We also include instructions `skip` (do nothing) and `fail` (looping or abnormal termination), although these constructs are redundant, being semantically equivalent to `while false do` p and `while true do skip`, respectively. We do not include a halt instruction; a program terminates normally by falling off the end.

Every `while` program can be converted to an equivalent strictly deterministic automaton. One first converts the programs to a KAT term using the standard translation

$$\begin{aligned} p\,;q &= pq & \texttt{skip} &= 1 \\ \texttt{if } b \texttt{ then } p \texttt{ else } q &= bp + \bar{b}q & \texttt{fail} &= 0 \\ \texttt{while } b \texttt{ do } p &= (bp)^*\bar{b}, \end{aligned}$$

then applies Kleene's theorem for KAT to yield an automaton with tests, which can then be converted to the form of Section 2.4. This construction is given in [25]. An example is shown in Fig. 1. In the strictly deterministic automaton shown in that figure, an edge from s to t labeled αp denotes the transition $\delta(s)(\alpha) = (p, t)$. The converse is false: there is a strictly determinis-

```
while b do {
   while c do q;
   p;
}
```



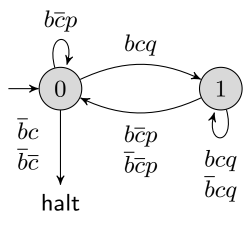


Figure 1: *A* `while` *program and its corresponding deterministic flowchart and strictly deterministic automaton*

tic automaton equivalent to no `while` program [2, 14] (see also [25], in which a three-state counterexample is given).

In addition to the usual while program constructs, we consider the following constructs:

$$\begin{array}{ll} \texttt{loop}\ p & \texttt{goto}\ \ell \\ \texttt{break}\ n,\ n \geq 1 & \ell : p \end{array}$$

Intuitively, the loop construct causes repeated execution of its body p. If p terminates normally, then control is transferred back to the beginning of p. This construct can be used in conjunction with the nonlocal breaks break n, $n \geq 1$. Intuitively, a break n instruction transfers control to the location immediately following the nth loop within whose scope the instruction occurs, counting from the innermost. The statement break is short for break 1. We also consider the unconditional jump goto ℓ, which transfers control to a labeled program $\ell : p$ if it exists, or fails if not.

The numbers n in the instructions break n are called *de Bruijn indices* in recognition of their similarity to the construct of the same name in the variable-free lambda calculus [10]. The index 0 denotes the continuation that corresponds to "falling off the end of the program" and is also considered a de Bruijn index. The de Bruijn indices along with the statement labels ℓ (which are assumed to be disjoint from the de Bruijn indices) are collectively called *continuation labels*.

3 Semantics

Programs p are interpreted by a translation to KAT expressions. This is a two-step process. In the first step, we identify a family of expressions $R_\alpha(p)$, one for each continuation label α. Intuitively, $R_0(p)$ is a KAT expression representing all valid halting computations that do not encounter a break or goto instruction. The expression $R_n(p)$ for $n \geq 1$ represents the set of computations leading to a break n (or more accurately, to any statement that has the same effect as a break n occurring at the outermost level). The expression $R_\ell(p)$ represents the set of computations leading to goto ℓ. The definitions are compositional but interdependent, thus must be defined by mutual induction.

In the second step, we define expressions $R_{\ell\alpha}(p)$ for each statement label ℓ and continuation label α. This represents the set of computations in p leading to the continuation α starting at the label ℓ. These pieces are then composed using a matrix construction to give a single expression denoting the set of valid computations of the program.

For convenience, define

$$s(\alpha) = \begin{cases} \ell, & \text{if } \alpha = \ell, \\ n+1, & \text{if } \alpha = n. \end{cases}$$

For a test or primitive action p, let

$$R_\alpha(p) \overset{\text{def}}{=} \begin{cases} p, & \text{if } \alpha = 0, \\ 0, & \text{otherwise.} \end{cases}$$

For the nonlocal atomic constructs,

$$R_\alpha(\texttt{break}\ n) \overset{\text{def}}{=} \begin{cases} 1, & \text{if } \alpha = n, \\ 0, & \text{otherwise,} \end{cases}$$

$$R_\alpha(\texttt{goto}\ \ell) \overset{\text{def}}{=} \begin{cases} 1, & \text{if } \alpha = \ell, \\ 0, & \text{otherwise.} \end{cases}$$

For nonatomic programs,

$$\begin{aligned} R_\alpha(p+q) &\overset{\text{def}}{=} R_\alpha(p) + R_\alpha(q) \\ R_\alpha(pq) &\overset{\text{def}}{=} \begin{cases} R_0(p) \cdot R_0(q), & \text{if } \alpha = 0, \\ R_\alpha(p) + R_0(p) \cdot R_\alpha(q), & \text{otherwise,} \end{cases} \\ R_\alpha(p^*) &\overset{\text{def}}{=} R_0(p)^* \cdot R_\alpha(p) + R_\alpha(1) \\ &= \begin{cases} R_0(p)^*, & \text{if } \alpha = 0, \\ R_0(p)^* \cdot R_\alpha(p), & \text{otherwise,} \end{cases} \\ R_\alpha(\texttt{loop}\ p) &\overset{\text{def}}{=} R_{s(\alpha)}(p^*) \quad = \quad R_0(p)^* \cdot R_{s(\alpha)}(p). \end{aligned}$$

Define $p \equiv q$ if $R_\alpha(p) = R_\alpha(q)$ for all continuation labels α. Since the meaning of programs will depend only on the R_α, two $\equiv$-equivalent programs can be substituted for each other.

Theorem 3.1 *The relation $\equiv$ is a congruence with respect to the* KAT *operators and satisfies all the axioms of* KAT *except $p \cdot 0 = 0$.*

Proof. All cases are straightforward except perhaps the axiom $q + rp \leq r \Rightarrow qp^* \leq r$, which we argue explicitly. Assuming

$$R_\alpha(q+rp) \subseteq R_\alpha(r) \tag{1}$$

for all α, we wish to show that

$$R_\alpha(qp^*) \subseteq R_\alpha(r) \tag{2}$$

for all α. This is true for $\alpha = 0$ because the interpretation R_0 is a homomorphism. For $\alpha \neq 0$,

$$\begin{aligned} R_\alpha(qp^*) &= R_\alpha(q) + R_0(q)R_0(p)^*R_\alpha(p) \\ &= R_\alpha(q) + R_0(qp^*)R_\alpha(p) \\ &\subseteq R_\alpha(q) + R_0(r)R_\alpha(p) && (3) \\ &\subseteq R_\alpha(q) + R_\alpha(r) + R_0(r)R_\alpha(p) \\ &= R_\alpha(q+rp) \\ &\subseteq R_\alpha(r), && (4) \end{aligned}$$

the inclusion (3) from (2) for $\alpha = 0$ and the inclusion (4) from (1).

The axiom $p \cdot 0 = 0$ is not satisfied, because $R_1(0) = 0$ but $R_1(\texttt{break} \cdot 0) = 1$. $\Box$

Example 3.2 Let us show that

$$\texttt{while } b \texttt{ do } p \quad \equiv \quad \texttt{loop if } b \texttt{ then } p \texttt{ else break}$$

provided $R_n(p) = 0$ for all $n \geq 1$. Note that this is false without the proviso: for $p = \texttt{break}$, the left-hand side is equivalent to `if` b `then break` and the right-hand side is equivalent to `skip`. Thus `break` cannot be used to break out of a `while` loop.

The left-hand side reduces to $(bp)^*\bar{b}$, and it is not hard to show that

$$R_\alpha((bp)^*\bar{b}) \quad = \quad \begin{cases} R_0(bp)^*R_0(\bar{b}), & \text{if } \alpha = 0, \\ R_0(bp)^*R_0(b)R_\alpha(p), & \text{if } \alpha \neq 0. \end{cases}$$

For the right-hand side,

$$\begin{aligned} & R_\alpha(\texttt{loop if } b \texttt{ then } p \texttt{ else break}) \\ &= R_{s(\alpha)}((\texttt{if } b \texttt{ then } p \texttt{ else break})^*) \\ &= R_{s(\alpha)}((bp + \bar{b}\,\texttt{break})^*) \\ &= R_0(bp + \bar{b}\,\texttt{break})^* R_{s(\alpha)}(bp + \bar{b}\,\texttt{break}) \\ &= R_0(bp)^*R_0(b)R_{s(\alpha)}(p) \\ &\quad + R_0(bp)^*R_0(\bar{b})R_{s(\alpha)}(\texttt{break}) \\ &= \begin{cases} R_0(bp)^*R_0(b)R_1(p) & \\ \quad + R_0(bp)^*R_0(\bar{b})R_1(\texttt{break}), & \text{if } \alpha = 0, \\ R_0(bp)^*R_0(b)R_{s(\alpha)}(p), & \text{if } \alpha \neq 0, \end{cases} \\ &= \begin{cases} R_0(bp)^*R_0(\bar{b}), & \text{if } \alpha = 0, \\ R_0(bp)^*R_0(b)R_\alpha(p), & \text{if } \alpha \neq 0, \end{cases} \end{aligned}$$

since $R_1(p) = 0$ and $R_{s(\alpha)}(p) = R_\alpha(p)$ for $\alpha \neq 0$ under the assumption. $\Box$

For the second part of the construction, let α be a continuation label and ℓ a statement label. Define the expressions $R_{\ell\alpha}(p)$ inductively as follows. Intuitively, $R_{\ell\alpha}(p)$ represents the computations of p starting at the label ℓ and leading to the continuation α.

$$\begin{aligned} R_{\ell\alpha}(\ell : p) &\overset{\text{def}}{=} R_\alpha(p) + R_{\ell\alpha}(p) \\ R_{\ell\alpha}(\ell' : p) &\overset{\text{def}}{=} R_{\ell\alpha}(p), \ \ell' \neq \ell \\ R_{\ell\alpha}(p + q) &\overset{\text{def}}{=} R_{\ell\alpha}(p) + R_{\ell\alpha}(q) \\ R_{\ell\alpha}(pq) &\overset{\text{def}}{=} R_{\ell\alpha}(p) + R_{\ell 0}(p) \cdot R_\alpha(q) + R_{\ell\alpha}(q) \\ R_{\ell\alpha}(p^*) &\overset{\text{def}}{=} R_{\ell\alpha}(p) + R_{\ell 0}(p) \cdot R_0(p)^* \cdot R_\alpha(p) \\ R_{\ell\alpha}(p) &\overset{\text{def}}{=} 0, \ p \text{ a test, primitive action,} \\ &\qquad \texttt{goto } \ell', \text{ or } \texttt{break } n. \end{aligned}$$

For `loop`, we define

$$\begin{aligned} & R_{\ell\alpha}(\texttt{loop } p) \\ &\overset{\text{def}}{=} R_{\ell,s(\alpha)}(p^*) \\ &= R_{\ell,s(\alpha)}(p) + R_{\ell 0}(p) \cdot R_0(p)^* \cdot R_{s(\alpha)}(p) \\ &= R_{\ell,s(\alpha)}(p) + R_{\ell 0}(p) \cdot R_\alpha(\texttt{loop } p). \end{aligned}$$

The statement labels ℓ need not have a unique occurrence in the program. If ℓ should have more than one occurrence, the interpretation is the natural nondeterministic one.

The KAT expressions $R_{\alpha\beta}(p)$ can be assembled into a square matrix $R(p)$ indexed by the continuation labels appearing in p; thus $R(p)_{\alpha\beta} = R_{\alpha\beta}(p)$. In addition, we include an extra row and column with index s such that $R(p)_{s\alpha} = R_\alpha(p)$. Matrix entries not otherwise specified (e.g., $R(p)_{1,0}$ or $R(p)_{\ell s}$) are 0. One can form $R(p)^*$ by the usual construction of the asterate of a square matrix over a KAT. The entry $R(p)^*_{s0}$ is a KAT expression denoting the set of all halting computations of the program. This is the least fixpoint of a system of linear inequalities involving the $R_{\alpha\beta}(p)$.

4 Basic Identities

In this section we develop some basic consequences of the definitions of Section 3. These will be used in the results of Sections 5 and 6.

First we prove some identities involving de Bruijn indices. Let I_m be inductively defined functions on programs that behave as homomorphisms with respect to the KAT operators and are defined on the other constructs as follows:

$$\begin{aligned} I_m(\texttt{goto } \ell) &= \texttt{goto } \ell \\ I_m(\texttt{break } n) &= \begin{cases} \texttt{break } n, & \text{if } n < m, \\ \texttt{break } n+1, & \text{if } n \geq m \end{cases} \\ I_m(\texttt{loop } p) &= \texttt{loop } (I_{m+1}(p)). \end{aligned}$$

Note that the I_m are identity homomorphisms on pure KAT expressions.

The function I_1 is the one we are interested in, and the other I_m are defined for auxiliary purposes. Intuitively, I_1 says to increment all the free de Bruijn indices by 1.

For example,

$$\begin{aligned}
&I_1(\texttt{loop (loop (loop (break 3))))}\\
&\quad = \texttt{loop (loop (loop (}I_4(\texttt{break 3))))}\\
&\quad = \texttt{loop (loop (loop (break 3)))}\\
&I_1(\texttt{loop (loop (break 3)))}\\
&\quad = \texttt{loop (loop (}I_3(\texttt{break 3)))}\\
&\quad = \texttt{loop (loop (break 4))}.
\end{aligned}$$

In the first expression, the `break 3` is bound, so it is not incremented, but in the second, it is free, so it is incremented. In general, an occurrence of a de Bruijn index n is bound if it occurs in the scope of at least n nested `loop` statements, otherwise it is free.

Lemma 4.1 *For all $m \geq 1$ and $n \geq 0$,*

$$\begin{aligned}
R_n(I_m(p)) &= \begin{cases} R_n(p), & \text{if } n < m,\\ 0, & \text{if } n = m,\\ R_{n-1}(p), & \text{if } n > m,\end{cases}\\
R_\ell(I_m(p)) &= R_\ell(p).
\end{aligned}$$

Proof. We prove this by induction on the structure of p. For the case $n = 0$, we only need to show

$$R_0(I_m(p)) \quad = \quad R_0(p). \tag{5}$$

For p a test, atomic action, or `goto` ℓ, $I_m(p) = p$, therefore (5) holds. For `break` k,

$$\begin{aligned}
R_0(I_m(\texttt{break } k)) &= \begin{cases} R_0(\texttt{break } k), & \text{if } k < m,\\ R_0(\texttt{break } k+1), & \text{if } k \geq m\end{cases}\\
&= 0\\
&= R_0(\texttt{break } k).
\end{aligned}$$

For expressions formed from the operators $+$, $\cdot$, and *, (5) follows from the induction hypothesis on smaller expressions and the fact that both I_m and R_0 are homomorphisms with respect to these operators. Finally, for `loop` p,

$$\begin{aligned}
R_0(I_m(\texttt{loop } p)) &= R_0(\texttt{loop } I_{m+1}(p))\\
&= R_1(I_{m+1}(p)^*)\\
&= R_1(I_{m+1}(p^*))\\
&= R_1(p^*)\\
&= R_0(\texttt{loop } p).
\end{aligned}$$

Now assume $n \geq 1$. As before, $R_n(I_m(p)) = R_n(p)$ for p a test, atomic action, or `goto` ℓ. For `break` k,

$$\begin{aligned}
&R_n(I_m(\texttt{break } k))\\
&\quad = \begin{cases} R_n(\texttt{break } k), & \text{if } k < m,\\ R_n(\texttt{break } k+1), & \text{if } k \geq m\end{cases}\\
&\quad = \begin{cases} 1, & \text{if } n = k < m \text{ or } n = k+1 \geq m+1,\\ 0, & \text{otherwise}\end{cases}\\
&\quad = \begin{cases} R_n(\texttt{break } k), & \text{if } n < m,\\ 0, & \text{if } n = m,\\ R_{n-1}(\texttt{break } k), & \text{if } n > m.\end{cases}
\end{aligned}$$

The case for the operator $+$ follows from the linearity of I_m and R_n. For $\cdot$, *, and `loop`,

$$\begin{aligned}
&R_n(I_m(pq))\\
&\quad = R_n(I_m(p) \cdot I_m(q))\\
&\quad = R_n(I_m(p)) + R_0(I_m(p)) \cdot R_n(I_m(q))\\
&\quad = \begin{cases} R_n(p) + R_0(p) \cdot R_n(q), & \text{if } n < m,\\ 0 + R_0(p) \cdot 0, & \text{if } n = m,\\ R_{n-1}(p) + R_0(p) \cdot R_{n-1}(q), & \text{if } n > m\end{cases}\\
&\quad = \begin{cases} R_n(pq), & \text{if } n < m,\\ 0, & \text{if } n = m,\\ R_{n-1}(pq), & \text{if } n > m,\end{cases}
\end{aligned}$$

$$\begin{aligned}
R_n(I_m(p^*)) &= R_n(I_m(p)^*)\\
&= R_0(I_m(p))^* \cdot R_n(I_m(p))\\
&= \begin{cases} R_0(p)^* \cdot R_n(p), & \text{if } n < m,\\ R_0(p)^* \cdot 0, & \text{if } n = m,\\ R_0(p)^* \cdot R_{n-1}(p), & \text{if } n > m\end{cases}\\
&= \begin{cases} R_n(p^*), & \text{if } n < m,\\ 0, & \text{if } n = m,\\ R_{n-1}(p^*), & \text{if } n > m,\end{cases}
\end{aligned}$$

$$\begin{aligned}
R_n(I_m(\texttt{loop } p)) &= R_n(\texttt{loop } I_{m+1}(p))\\
&= R_{n+1}(I_{m+1}(p)^*)\\
&= R_{n+1}(I_{m+1}(p^*))\\
&= \begin{cases} R_{n+1}(p^*), & \text{if } n < m,\\ 0, & \text{if } n = m,\\ R_n(p^*), & \text{if } n > m\end{cases}\\
&= \begin{cases} R_n(\texttt{loop } p), & \text{if } n < m,\\ 0, & \text{if } n = m,\\ R_{n-1}(\texttt{loop } p) & \text{if } n > m.\end{cases}
\end{aligned}$$

It remains to show that $R_\ell(I_m(p)) = R_\ell(p)$. This holds for tests, atomic actions, and `goto` instructions as above. For `break` k,

$$\begin{aligned} R_\ell(I_m(\texttt{break } k)) &= \begin{cases} R_\ell(\texttt{break } k), & \text{if } k < m, \\ R_\ell(\texttt{break } k+1), & \text{if } k \geq m \end{cases} \\ &= 0 \\ &= R_\ell(\texttt{break } k). \end{aligned}$$

The case for the operator $+$ follows from the linearity of I_m and R_ℓ. For $\cdot$, *, and `loop`,

$$\begin{aligned} R_\ell(I_m(pq)) &= R_\ell(I_m(p) \cdot I_m(q)) \\ &= R_\ell(I_m(p)) + R_0(I_m(p)) \cdot R_\ell(I_m(q)) \\ &= R_\ell(p) + R_0(p) \cdot R_\ell(q) \\ &= R_\ell(pq), \end{aligned}$$

$$\begin{aligned} R_\ell(I_m(p^*)) &= R_\ell(I_m(p)^*) \\ &= R_0(I_m(p))^* \cdot R_\ell(I_m(p)) \\ &= R_0(p)^* \cdot R_\ell(p) \\ &= R_\ell(p^*), \end{aligned}$$

$$\begin{aligned} R_\ell(I_m(\texttt{loop } p)) &= R_\ell(\texttt{loop } I_{m+1}(p)) \\ &= R_\ell(I_{m+1}(p)^*) \\ &= R_\ell(I_{m+1}(p^*)) \\ &= R_\ell(p^*) \\ &= R_\ell(\texttt{loop } p). \end{aligned}$$

□

Corollary 4.2 *For all $m \geq 1$ and $n \geq 0$,*

$$R_\beta(\texttt{loop } p) = R_{s(\beta)}(\texttt{loop } (I_1(p))).$$

Proof. By the lemma,

$$R_{s^2(\beta)}(I_1(p)) = R_{s(\beta)}(p) \qquad R_0(I_1(p)) = R_0(p),$$

thus

$$\begin{aligned} R_\beta(\texttt{loop } p) &= R_{s(\beta)}(p^*) \\ &= R_0(p)^* \cdot R_{s(\beta)}(p) \\ &= R_0(I_1(p))^* \cdot R_{s^2(\beta)}(I_1(p)) \\ &= R_{s^2(\beta)}(I_1(p)^*) \\ &= R_{s(\beta)}(\texttt{loop } (I_1(p))). \end{aligned}$$

□

Define $J_m = I_m \circ I_{m-1} \circ I_{m-2} \circ \cdots \circ I_1$.

Lemma 4.3 *For all $m \geq 1$ and $n \geq 0$,*

$$\begin{aligned} R_n(J_m(p) \cdot \texttt{break } m) &= \begin{cases} R_{n-m}(p), & \textit{if } n \geq m, \\ 0, & \textit{if } n < m \end{cases} \\ R_\ell(J_m(p) \cdot \texttt{break } m) &= R_\ell(p). \end{aligned}$$

In particular,

$$R_{s(\alpha)}(I_1(p) \cdot \texttt{break}) = R_\alpha(p).$$

Proof. From Lemma 4.1 it follows inductively that for all $m \geq 1$ and $n \geq 0$,

$$\begin{aligned} R_n(J_m(p)) &= \begin{cases} R_0(p), & \text{if } n = 0, \\ 0, & \text{if } 0 < n \leq m, \\ R_{n-m}(p), & \text{if } n > m, \end{cases} \\ R_\ell(J_m(p)) &= R_\ell(p). \end{aligned}$$

For $n = 0$,

$$\begin{aligned} R_0(J_m(p) \cdot \texttt{break } m) &= R_0(J_m(p)) \cdot R_0(\texttt{break } m) \\ &= R_0(p) \cdot 0 \\ &= 0. \end{aligned}$$

For $n > 0$,

$$\begin{aligned} &R_n(J_m(p) \cdot \texttt{break } m) \\ &= R_n(J_m(p)) + R_0(J_m(p)) \cdot R_n(\texttt{break } m) \\ &= R_n(J_m(p)) + R_0(p) \cdot R_n(\texttt{break } m) \\ &= \begin{cases} 0 + R_0(p) \cdot 0, & \text{if } 1 \leq n < m, \\ 0 + R_0(p) \cdot 1, & \text{if } n = m, \\ R_{n-m}(p) + R_0(p) \cdot 0, & \text{if } n > m \end{cases} \\ &= \begin{cases} 0, & \text{if } 1 \leq n < m, \\ R_{n-m}(p), & \text{if } n \geq m. \end{cases} \end{aligned}$$

For ℓ,

$$\begin{aligned} &R_\ell(J_m(p) \cdot \texttt{break } m) \\ &= R_\ell(J_m(p)) + R_0(J_m(p)) \cdot R_\ell(\texttt{break } m) \\ &= R_\ell(p) + R_0(p) \cdot 0 \\ &= R_\ell(p). \end{aligned}$$

□

Lemma 4.4 `loop` $(I_1(p) \cdot \texttt{break}) \equiv p$.

Proof. For any α,

$$\begin{aligned} &R_\alpha(\texttt{loop } (I_1(p) \cdot \texttt{break})) \\ &= R_{s(\alpha)}((I_1(p) \cdot \texttt{break})^*) \\ &= R_{s(\alpha)}(1 + I_1(p) \cdot \texttt{break}) \\ &= R_{s(\alpha)}(1) + R_{s(\alpha)}(I_1(p) \cdot \texttt{break}) \\ &= R_\alpha(p). \end{aligned}$$

□

Lemma 4.5 *For any q,*

$$\begin{aligned} &R_n(\texttt{loop } (J_2(p) \cdot \texttt{break } 2) \cdot q) \\ &= \begin{cases} 0, & \textit{if } n = 0, \\ R_{n-1}(p), & \textit{if } n > 0 \end{cases} \end{aligned}$$

$$R_\ell(\texttt{loop }(J_2(p)\cdot\texttt{break }2)\cdot q) \quad = \quad R_\ell(p).$$

In particular,

$$R_{s(\alpha)}(\texttt{loop }(J_2(p)\cdot\texttt{break }2)\cdot q) \quad = \quad R_\alpha(p).$$

Proof.

$$\begin{aligned}
&R_0(\texttt{loop }(J_2(p)\cdot\texttt{break }2)\cdot q)\\
&\quad= R_0(\texttt{loop }(J_2(p)\cdot\texttt{break }2))\cdot R_0(q)\\
&\quad= R_0(J_2(p)\cdot\texttt{break }2)^*\cdot R_1(J_2(p)\cdot\texttt{break }2)\cdot R_0(q)\\
&\quad= 0^*\cdot 0\cdot R_0(q)\\
&\quad= 0,
\end{aligned}$$

and for any α,

$$\begin{aligned}
&R_{s(\alpha)}(\texttt{loop }(J_2(p)\cdot\texttt{break }2)\cdot q)\\
&\quad= R_{s(\alpha)}(\texttt{loop }(J_2(p)\cdot\texttt{break }2))\\
&\qquad\quad + R_0(\texttt{loop }(J_2(p)\cdot\texttt{break }2))\cdot R_n(q)\\
&\quad= R_{s(s(\alpha))}((J_2(p)\cdot\texttt{break }2)^*)\\
&\qquad\quad + R_1((J_2(p)\cdot\texttt{break }2)^*)\cdot R_n(q)\\
&\quad= R_{s(s(\alpha))}(1+J_2(p)\cdot\texttt{break }2)\\
&\qquad\quad + R_1(1+J_2(p)\cdot\texttt{break }2)\cdot R_n(q)\\
&\quad= R_{s(s(\alpha))}(J_2(p)\cdot\texttt{break }2)+0\cdot R_n(q)\\
&\quad= R_{s(s(\alpha))}(J_2(p)\cdot\texttt{break }2)\\
&\quad= R_\alpha(p).
\end{aligned}$$

□

Lemma 4.6 *For any q,*

$$\texttt{loop }(\texttt{loop }(J_2(p)\cdot\texttt{break }2)\cdot q) \quad \equiv \quad p.$$

Proof.

$$\begin{aligned}
&R_\alpha(\texttt{loop }(\texttt{loop }(J_2(p)\cdot\texttt{break }2)\cdot q))\\
&\quad= R_0(\texttt{loop }(J_2(p)\cdot\texttt{break }2)\cdot q)^*\\
&\qquad\quad \cdot R_{s(\alpha)}(\texttt{loop }(J_2(p)\cdot\texttt{break }2)\cdot q)\\
&\quad= 0^*\cdot R_\alpha(p)\\
&\quad= R_\alpha(p).
\end{aligned}$$

□

Let $K_{\ell,m}$, $m \geq 1$, be a family of substitution operators that act as the identity on all tests and atomic actions except `goto` ℓ, and act as a KAT homomorphism on all operators except `loop`. For the remaining two constructs,

$$\begin{aligned}
K_{\ell,m}(\texttt{goto }\ell) &= \texttt{break }m\\
K_{\ell,m}(\texttt{loop }p) &= \texttt{loop }(K_{\ell,m+1}(p))
\end{aligned}$$

That is, $K_{\ell,m}$ substitutes `break` $m+k$ for each occurrence of `goto` ℓ of `loop`-depth k.

Lemma 4.7

$$\texttt{loop }(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\cdot\texttt{goto }\ell) \quad \equiv \quad p.$$

Proof. By Lemma 4.6, it suffices to show

$$\begin{aligned}
&\texttt{loop }(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\cdot\texttt{goto }\ell)\\
&\quad\equiv \texttt{loop }(\texttt{loop }(J_2(p)\cdot\texttt{break }2)).
\end{aligned}$$

Consequently, it suffices to show

$$\begin{aligned}
&\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\cdot\texttt{goto }\ell\\
&\quad\equiv \texttt{loop }(J_2(p)\cdot\texttt{break }2).
\end{aligned}$$

We need to show

$$\begin{aligned}
&R_\alpha(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\cdot\texttt{goto }\ell)\\
&\quad= R_\alpha(\texttt{loop }(J_2(p)\cdot\texttt{break }2)).
\end{aligned}$$

We have already shown in Lemma 4.5 that

$$\begin{aligned}
R_0(\texttt{loop }(J_2(p)\cdot\texttt{break }2)) &= 0\\
R_{s(\alpha)}(\texttt{loop }(J_2(p)\cdot\texttt{break }2)) &= R_\alpha(p).
\end{aligned}$$

For $n=0$,

$$\begin{aligned}
&R_0(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\cdot\texttt{goto }\ell)\\
&\quad= R_0(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2))\cdot R_0(\texttt{goto }\ell)\\
&\quad= R_0(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2))\cdot 0\\
&\quad= 0.
\end{aligned}$$

For $\alpha \neq \ell$,

$$\begin{aligned}
&R_{s(\alpha)}(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\cdot\texttt{goto }\ell)\\
&\quad= R_{s(\alpha)}(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2))\\
&\qquad\quad + R_0(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2))\cdot R_{s(\alpha)}(\texttt{goto }\ell)\\
&\quad= R_{s(s(\alpha))}(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\\
&\qquad\quad + R_1(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\cdot 0\\
&\quad= R_{s(s(\alpha))}(K_{\ell,1}(J_2(p)\cdot\texttt{break }2))\\
&\quad= R_{s(s(\alpha))}(J_2(p)\cdot\texttt{break }2)\\
&\quad= R_\alpha(p).
\end{aligned}$$

Finally, for ℓ,

$$\begin{aligned}
&R_\ell(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\cdot\texttt{goto }\ell)\\
&\quad= R_\ell(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2))\\
&\qquad\quad + R_0(\texttt{loop }(K_{\ell,1}(J_2(p))\cdot\texttt{break }2))\cdot R_\ell(\texttt{goto }\ell)\\
&\quad= R_\ell(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\\
&\qquad\quad + R_1(K_{\ell,1}(J_2(p))\cdot\texttt{break }2)\cdot 1\\
&\quad= R_\ell(K_{\ell,1}(J_2(p)\cdot\texttt{break }2))\\
&\qquad\quad + R_1(K_{\ell,1}(J_2(p)\cdot\texttt{break }2))\\
&\quad= 0+R_1(J_2(p)\cdot\texttt{break }2)+R_\ell(J_2(p)\cdot\texttt{break }2)\\
&\quad= R_\ell(J_2(p)\cdot\texttt{break }2)\\
&\quad= R_\ell(p).
\end{aligned}$$

□

4.1 Lifting Labels

In general, programs might have labels embedded deeply in them. In this section we perform some transformations to move labels to the outermost level. We show that for each label ℓ and program p, there is a program p_ℓ such that for all β, $R_{\ell\beta}(p) = R_\beta(p_\ell)$. Intuitively, jumping into the middle of p at label ℓ is the same as running p_ℓ from the beginning. Moreover, if p is deterministic, then so is p_ℓ.

Let p be a program and ℓ a label. Define

$$\begin{aligned}
p_\ell &\overset{\text{def}}{=} 0, \text{ for } p \text{ a test, atomic program, } \mathtt{break}\ n, \text{ or } \mathtt{goto}\ \ell',\\
(p+q)_\ell &\overset{\text{def}}{=} p_\ell + q_\ell,\\
(pq)_\ell &\overset{\text{def}}{=} p_\ell q + q_\ell,\\
(p^*)_\ell &\overset{\text{def}}{=} p_\ell p^*,\\
(\ell' : p)_\ell &\overset{\text{def}}{=} \begin{cases} p + p_\ell, & \text{if } \ell = \ell',\\ p_\ell, & \text{if } \ell \neq \ell',\end{cases}\\
(\mathtt{loop}\ p)_\ell &\overset{\text{def}}{=} \mathtt{loop}\ (p_\ell \cdot \mathtt{loop}\ (I_1(p))).
\end{aligned}$$

Theorem 4.8 *For all ℓ, β, and p,*

$$R_{\ell\beta}(p) \quad = \quad R_\beta(p_\ell).$$

Proof. The arguments for tests, atomic programs, $\mathtt{break}\ n$, $\mathtt{goto}\ \ell'$, $p+q$, and $\ell' : p$ are straightforward.

For pq and p^*,

$$\begin{aligned}
&R_{\ell\beta}(pq)\\
&= \begin{cases} R_{\ell\beta}(p) + R_{\ell 0}(p)\cdot R_\beta(q) + R_{\ell\beta}(q), & \text{if } \beta \neq 0,\\ R_{\ell 0}(p)\cdot R_0(q) + R_{\ell 0}(q), & \text{if } \beta = 0\end{cases}\\
&= \begin{cases} R_\beta(p_\ell) + R_0(p_\ell)\cdot R_\beta(q) + R_\beta(q_\ell), & \text{if } \beta \neq 0,\\ R_0(p_\ell)\cdot R_0(q) + R_0(q_\ell), & \text{if } \beta = 0\end{cases}\\
&= \begin{cases} R_\beta(p_\ell q) + R_\beta(q_\ell), & \text{if } \beta \neq 0,\\ R_0(p_\ell q) + R_0(q_\ell), & \text{if } \beta = 0\end{cases}\\
&= R_\beta(p_\ell q + q_\ell)\\
&= R_\beta((pq)_\ell),
\end{aligned}$$

$$\begin{aligned}
&R_{\ell\beta}(p^*)\\
&= \begin{cases} R_{\ell\beta}(p) + R_{\ell 0}(p)\cdot R_0(p)^*\cdot R_\beta(p), & \text{if } \beta \neq 0,\\ R_{\ell 0}(p)\cdot R_0(p)^*, & \text{if } \beta = 0\end{cases}\\
&= \begin{cases} R_\beta(p_\ell) + R_0(p_\ell)\cdot R_\beta(p^*), & \text{if } \beta \neq 0,\\ R_0(p_\ell)\cdot R_0(p^*), & \text{if } \beta = 0\end{cases}\\
&= R_\beta(p_\ell p^*)\\
&= R_\beta((p^*)_\ell).
\end{aligned}$$

Finally, for $\mathtt{loop}\ p$, first observe that if $R_0(q) = 0$, then

$$\begin{aligned}
R_\beta(q^*) &= R_\beta(1) + R_0(q)^*\cdot R_\beta(q)\\
&= R_\beta(1) + R_\beta(q)\\
&= \begin{cases} 1, & \text{if } \beta = 0,\\ R_\beta(q), & \text{if } \beta \neq 0.\end{cases}
\end{aligned}$$

We will apply this with $q = p_\ell \cdot \mathtt{loop}\ (I_1(p))$, observing that

$$\begin{aligned}
R_0(\mathtt{loop}\ (I_1(p))) &= R_1(I_1(p)^*)\\
&= R_0(I_1(p))^*\cdot R_1(I_1(p)) \quad = \quad 0,
\end{aligned}$$

hence

$$\begin{aligned}
&R_0(p_\ell \cdot \mathtt{loop}\ (I_1(p)))\\
&\quad = R_0(p_\ell)\cdot R_0(\mathtt{loop}\ (I_1(p))) \quad = \quad 0.
\end{aligned}$$

$$\begin{aligned}
&R_{\ell\beta}(\mathtt{loop}\ p)\\
&= R_{\ell,s(\beta)}(p^*)\\
&= R_{\ell,s(\beta)}(p) + R_{\ell 0}(p)\cdot R_0(p)^*\cdot R_{s(\beta)}(p)\\
&= R_{s(\beta)}(p_\ell) + R_0(p_\ell)\cdot R_{s(\beta)}(p^*)\\
&= R_{s(\beta)}(p_\ell) + R_0(p_\ell)\cdot R_\beta(\mathtt{loop}\ p)\\
&= R_{s(\beta)}(p_\ell) + R_0(p_\ell)\cdot R_{s(\beta)}(\mathtt{loop}\ (I_1(p)))\\
&= R_{s(\beta)}(p_\ell \cdot \mathtt{loop}\ (I_1(p)))\\
&= R_{s(\beta)}((p_\ell \cdot \mathtt{loop}\ (I_1(p)))^*)\\
&= R_\beta(\mathtt{loop}\ (p_\ell \cdot \mathtt{loop}\ (I_1(p)))).
\end{aligned}$$

□

Lemma 4.9 *If p contains no occurrence of ℓ as a label, that is, if p contains no subprogram of the form $\ell : q$, then $p_\ell \equiv 0$.*

Proof. This can be shown by induction on the structure of the program. All cases are quite obvious except perhaps for the case $\mathtt{loop}\ p$. For this case, it suffices to show that if $p_\ell \equiv 0$, then $(\mathtt{loop}\ p)_\ell \equiv 0$.

$$\begin{aligned}
(\mathtt{loop}\ p)_\ell &= \mathtt{loop}\ (p_\ell \cdot \mathtt{loop}\ (I_1(p)))\\
&\equiv \mathtt{loop}\ (0 \cdot \mathtt{loop}\ (I_1(p)))\\
&\equiv \mathtt{loop}\ 0,
\end{aligned}$$

and for any β,

$$R_\beta(\mathtt{loop}\ 0) \quad = \quad R_{s(\beta)}(0^*) \quad = \quad R_{s(\beta)}(1) \quad = \quad 0.$$

□

Note that p_ℓ contains no occurrence of ℓ as a label, since in no case in the inductive definition of p_ℓ does ℓ occur on the right-hand side; thus $(p_\ell)_\ell \equiv 0$.

Lemma 4.10 *If p is deterministic, then so is p_ℓ.*

Proof. This is true for p a test, atomic program, `break` n, or `goto` ℓ', since $p_\ell = 0$, since 0 is deterministic. The argument for `loop` p is also clear by the form of the right-hand side and the induction hypothesis.

For the case `while` b `do` p, using the encoding $(bp)^*\bar{b}$ and unwinding the definitions, one can show that

$$(\texttt{while}\ b\ \texttt{do}\ p)_\ell \quad = \quad p_\ell \cdot \texttt{while}\ b\ \texttt{do}\ p.$$

This is deterministic by the induction hypothesis.

For the case `if` b `then` p `else` q, using the encoding $bp + \bar{b}q$ and unwinding the definitions, one can show that

$$(\texttt{if}\ b\ \texttt{then}\ p\ \texttt{else}\ q)_\ell \quad = \quad p_\ell + q_\ell.$$

By determinacy, at most one of p or q contains ℓ as a label. By Lemma 4.9, at least one of p_ℓ or q_ℓ is equivalent to 0, so the entire program is equivalent to the other. Since p and q are deterministic, both p_ℓ and q_ℓ are deterministic by the induction hypothesis.

For the case pq, we have $(pq)_\ell = p_\ell q + q_\ell$. By the same argument as the previous case, the entire program is equivalent to either $p_\ell q$ or q_ℓ, both of which are deterministic by the induction hypothesis.

Finally, for $\ell : p$, we have

$$(\ell : p)_\ell \quad = \quad p + p_\ell \quad \equiv \quad p,$$

since by determinacy, p_ℓ contains no occurrence of ℓ as a label, therefore $p_\ell \equiv 0$. $\square$

4.2 Modules

For convenience, we extend the language with the following construct. If $p_1, \ldots, p_n$ are programs, then so is $(p_1, \ldots, p_n)$. The first program p_1 in the sequence is the *main program* and the $p_1, \ldots, p_n$ are called *modules*. For all α and ℓ, define

$$\begin{aligned} R_\alpha(p_1, \ldots, p_n) &\overset{\text{def}}{=} R_\alpha(p_1), \\ R_{\ell\alpha}(p_1, \ldots, p_n) &\overset{\text{def}}{=} R_{\ell\alpha}(p_1) + \cdots + R_{\ell\alpha}(p_n). \end{aligned}$$

Thus the entry point of $(p_1, \ldots, p_n)$ is the entry point of p_1, but other modules may be accessible by unconditional jumps `goto` ℓ. If the program is deterministic, so that ℓ occurs in at most one module p_i, then $R_{\ell\alpha}(p_1, \ldots, p_n) = R_{\ell\alpha}(p_i)$. A program of the form $(p_1, \ldots, p_n)$ is called *modular*.

Lemma 4.11

$$\begin{aligned} &(p_1, \ldots, p_n) \\ &\quad \equiv \ \texttt{loop}\ (I_1(p_1); \texttt{break}; \cdots; I_1(p_n); \texttt{break}). \end{aligned}$$

This allows the construct $(p_1, \ldots, p_n)$ to be eliminated if desired. We omit the proof, since this fact is not needed in the remaining sections.

5 Loops Programs are Sufficient

In this section we use our calculus to give a formal equational proof that every propositional deterministic flowchart is equivalent to a `loop` program with multi-level breaks but without unconditional jumps. This is a folklore result that has been known since at least the early 1970s [14, 26], but to our knowledge has never been treated with this level of rigor. A somewhat different conversion was given in [25] in the same formalism, but without proof.

We must actually start not with a flowchart or automaton, but with an equivalent program expression, since our calculus works only with expressions, not with automata. However, there is a straightforward construction to convert a given strictly deterministic automaton as described in Section 2.4 to a modular program with unconditional jumps, which we sketch without proof.

Given a strictly deterministic automaton M, construct a modular program with one module for each state. The module corresponding to state ℓ has label ℓ. The module corresponding to the start state occurs first in the modular program. For each state ℓ, let $\alpha_1, \ldots, \alpha_m$ be all atoms such that $\delta(\ell)(\alpha_i) = (p_i, \ell_i)$ and let $\alpha_{m+1}, \ldots, \alpha_n$ be all atoms such that $\delta(\ell)(\alpha_i) = \mathsf{halt}$. The module corresponding to ℓ is

```
$\ell$: if $\alpha_1$ then ($p_1$ ; goto $\ell_1$)
    else if $\alpha_2$ then ($p_2$ ; goto $\ell_2$)
    else if $\alpha_3$ then ($p_3$ ; goto $\ell_3$)
    ⋮
    else if $\alpha_m$ then ($p_m$ ; goto $\ell_m$)
    else skip
```

One can show without much difficulty that

$$R_{\ell\beta} \quad = \quad \begin{cases} \alpha_i p_i, & \text{if } \beta = \ell_i,\ 1 \le i \le m, \\ \alpha_{m+1} + \cdots + \alpha_n, & \text{if } \beta = 0, \\ 0, & \text{if } \beta > 0. \end{cases}$$

It follows inductively that

$$\mathsf{GS}(R^*_{\ell 0}) \quad = \quad \{\mathsf{gs}(\ell)(\sigma) \mid \sigma \in \mathsf{At}^\omega\} \cap (\mathsf{At} \cdot \Sigma)^* \cdot \mathsf{At},$$

thus

$$\begin{aligned}\mathsf{GS}(R^*_{s0}) &= \{\mathsf{gs}(\mathsf{start})(\sigma) \mid \sigma \in \mathsf{At}^\omega\} \cap (\mathsf{At}\cdot\Sigma)^* \cdot \mathsf{At} \\ &= \mathsf{GS}(M).\end{aligned}$$

Although our calculus cannot be used for this part of the construction, a formal semantic proof can be given along the lines sketched above.

Now assume we are given a deterministic program containing any of the programming constructs defined in Section 2.5, including `while` loops, unconditional jumps, `loop` instructions, or multilevel breaks.

Lemma 5.1 *Consider two deterministic modular programs with ℓth modules*

$$\ell : \texttt{loop}\ (q\,;\texttt{goto}\ \ell) \qquad\qquad \ell : \texttt{loop}\ q,$$

respectively, where $R_\ell(q) = 0$. The two programs are otherwise identical. Let F and G be the matrices corresponding to these two programs, respectively, as described in Section 3. Let α, β be two continuation labels, $\beta \neq \ell$. Then $F^*_{\alpha\beta} = G^*_{\alpha\beta}$. In particular, $F^*_{s0} = G^*_{s0}$, therefore the two programs are equivalent.

Proof. Decompose the matrices F and G as

$$F = \begin{bmatrix} F_{\ell\ell} & F' \\ C & D \end{bmatrix} \qquad G = \begin{bmatrix} G_{\ell\ell} & G' \\ C & D \end{bmatrix}$$

where D is the square matrix obtained by deleting the ℓth row and column of F or G (C and D are the same in each case, by assumption). If $\alpha \neq \ell$, then

$$\begin{aligned} F^*_{\alpha\beta} &= (D + CF^*_{\ell\ell}F')^*_{\alpha\beta} \\ G^*_{\alpha\beta} &= (D + CG^*_{\ell\ell}G')^*_{\alpha\beta}, \end{aligned}$$

and

$$\begin{aligned} F^*_{\ell\beta} &= (F^*_{\ell\ell}F'(D + CF^*_{\ell\ell}F')^*)_{\ell\beta} \\ G^*_{\ell\beta} &= (G^*_{\ell\ell}G'(D + CG^*_{\ell\ell}G')^*)_{\ell\beta}. \end{aligned}$$

In either case, it suffices to show

$$F^*_{\ell\ell}F' = G^*_{\ell\ell}G',$$

or in other words,

$$F^*_{\ell\ell}F_{\ell\beta} = G^*_{\ell\ell}G_{\ell\beta} \qquad (6)$$

for all $\beta \neq \ell$. But

$$\begin{aligned} F_{\ell\ell} &= R_\ell(\texttt{loop}\ (q\,;\texttt{goto}\ \ell)) = R_0(q) \\ F_{\ell\beta} &= R_\beta(\texttt{loop}\ (q\,;\texttt{goto}\ \ell)) = R_{s(\beta)}(q) \\ G_{\ell\ell} &= R_\ell(\texttt{loop}\ q) = 0 \\ G_{\ell\beta} &= R_\beta(\texttt{loop}\ q) = R_0(q)^*R_{s(\beta)}(q), \end{aligned}$$

so both sides of (6) are $R_0(q)^*R_{s(\beta)}(q)$. □

Lemma 5.2 *Consider a deterministic modular program with distinct modules*

$$\texttt{loop}\ (q\,;\texttt{goto}\ \ell) \qquad \ell : \texttt{loop}\ (r\,;\texttt{break}), \qquad (7)$$

where $R_\ell(q) = R_\ell(r) = 0$. Consider another program that is identical to the first, except with the left-hand module of (7) replaced by

$$\texttt{loop}\ (q\,;r\,;\texttt{break}). \qquad (8)$$

Let F and G be the matrices corresponding to these two programs, respectively. Let α, β be two continuation labels, $\beta \neq \ell$. Then $F^*_{\alpha\beta} = G^*_{\alpha\beta}$. In particular, $F^*_{s0} = G^*_{s0}$, therefore the two programs are equivalent.

Proof. By a matrix decomposition argument similar to the one in the proof of Lemma 5.1, it suffices to show that

$$F_{\alpha\beta} + F_{\alpha\ell}F_{\ell\beta} = G_{\alpha\beta}. \qquad (9)$$

By Theorem 4.8, we can assume without loss of generality that α is the label of the left-hand module of (7) in F and the module (8) of G, which is either s or an explicit label. Then

$$\begin{aligned} F_{\alpha\beta} &= R_\beta(\texttt{loop}\ (q\,;\texttt{goto}\ \ell)) = R_{s(\beta)}(q) \\ F_{\alpha\ell} &= R_\ell(\texttt{loop}\ (q\,;\texttt{goto}\ \ell)) = R_0(q) \\ F_{\ell\beta} &= R_\beta(\texttt{loop}\ (r\,;\texttt{break})) \\ &= R_{s(\beta)}(r) + R_0(r)R_{s(\beta)}(\texttt{break}) \\ G_{\alpha\beta} &= R_\beta(\texttt{loop}\ (q\,;r\,;\texttt{break})) \\ &= R_{s(\beta)}(q) + R_0(q)R_{s(\beta)}(r) \\ &\quad + R_0(q)R_0(r)R_{s(\beta)}(\texttt{break}), \end{aligned}$$

from which (9) follows. □

Theorem 5.3 *Every program is equivalent to a `loop` program with multilevel breaks but without unconditional jumps.*

Proof. Starting from an arbitrary program p with unconditional jumps, first use Theorem 4.8 to rewrite the program as a modular program $(p_1, \ldots, p_n)$, where p_1 is p (without the initial label if it exists) and $p_2, \ldots, p_n$ are all programs of the form $\ell : p_\ell$ obtained from the lifting construction of Section 4.1 for all labels ℓ occurring in the program. By Theorem 4.8, $(p_1, \ldots, p_n)$ and p have the same matrix.

Now consider the last module in the list, and say it has label ℓ. By Lemma 4.7, it can be written as

$$\ell : \texttt{loop}\ (s\,;\texttt{goto}\ \ell),$$

where $R_\ell(s) = 0$. By Lemma 5.1, this can be replaced by ℓ : `loop` s. Using Lemma 4.4, this module can be rewritten as

$$\ell : \texttt{loop}\ (r\ ;\texttt{break}),$$

where $R_\ell(r) = 0$. Now for every other module besides this one, use Lemma 4.7 to rewrite it in the form

$$\texttt{loop}\ (q\ ;\texttt{goto}\ \ell),$$

where $R_\ell(q) = 0$, then Lemma 5.2 to rewrite it in the form

$$\texttt{loop}\ (q\ ; r\ ;\texttt{break}).$$

At this point there are no longer any occurrences of `goto` ℓ in the program, so the column of the matrix corresponding to ℓ is 0. The row and column corresponding to ℓ can thus be deleted without changing the semantics of the program; the resulting matrix is that of the modular program with the last module deleted.

Continuing in this fashion, we can delete all modules except the first. We are left with a `loop` program with multilevel breaks and no unconditional jumps. □

6 Completeness

The following theorem is a straightforward consequence of the completeness of KAT, but still bears mentioning:

Theorem 6.1 *The calculus presented in this paper is sufficient to prove all valid identities between* `loop` *programs with multilevel breaks.*

Proof. The inductive definitions of the `loop` and `break` n constructs allow any program p to be reduced to a matrix $R(p)$ of KAT expressions. The meaning of p is defined to be the same as a particular entry of the asterate of that matrix, namely $R(p)^*_{s0}$. Since KAT is complete for the equational theory of the guarded string model, it can prove the equivalence of two such translations if indeed the two programs represent the same set of guarded strings. □

7 Conclusion and Open Problems

We have shown how to handle programming constructs involving nonlocal flow of control such as `goto` and multilevel `break` instructions in a simple propositional equational system. This fits well with the vision that simple equational reasoning suffices to handle a large class of basic program analysis tasks.

Some interesting open problems present themselves. Can one formulate a simple coalgebraic treatment of nonlocal flow of control involving a definition of the Brzozowski derivative [4] for the nonlocal control flow constructs? If so, it may be possible to given even simpler constructions and proofs. To what extent is it possible to extend the method to reason in the presence of commutativity conditions and other basic premises? And finally, it is clear that in principle, equivalence proofs be extracted automatically by reduction to KAT [5, 27], but can this process be streamlined by treating the nonlocal control flow constructs directly?

Acknowledgements

Thanks to Maksim Orlovich and Dustin Tseng for valuable conversations. This work was supported by NSF grant CCF-0635028.

References

[1] A. Angus and D. Kozen. Kleene algebra with tests and program schematology. Technical Report TR2001-1844, Computer Science Department, Cornell University, July 2001.

[2] E. Ashcroft and Z. Manna. The translation of goto programs into while programs. In C. Freiman, J. Griffith, and J. Rosenfeld, editors, *Proceedings of IFIP Congress 71*, volume 1, pages 250–255. North-Holland, 1972.

[3] A. Barth and D. Kozen. Equational verification of cache blocking in LU decomposition using Kleene algebra with tests. Technical Report TR2002-1865, Computer Science Department, Cornell University, June 2002.

[4] J. A. Brzozowski. Derivatives of regular expressions. *J. Assoc. Comput. Mach.*, 11:481–494, 1964.

[5] H. Chen and R. Pucella. A coalgebraic approach to Kleene algebra with tests. *Electronic Notes in Theoretical Computer Science*, 82(1), 2003.

[6] E. Cohen. Hypotheses in Kleene algebra. Technical Report TM-ARH-023814, Bellcore, 1993. `http://citeseer.nj.nec.com/1688.html`.

[7] E. Cohen. Lazy caching in Kleene algebra, 1994. `http://citeseer.nj.nec.com/22581.html`.

[8] E. Cohen, D. Kozen, and F. Smith. The complexity of Kleene algebra with tests. Technical Report TR96-1598, Computer Science Department, Cornell University, July 1996.

[9] J. H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, London, 1971.

[10] N. G. de Bruijn. Lambda calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the Church–Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[11] K. Iwano and K. Steiglitz. A semiring on convex polygons and zero-sum cycle problems. *SIAM J. Comput.*, 19(5):883–901, 1990.

[12] D. M. Kaplan. Regular expressions and the equivalence of programs. *J. Comput. Syst. Sci.*, 3:361–386, 1969.

[13] S. 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.

[14] S. R. Kosaraju. Analysis of structured programs. In *Proc. 5th ACM Symp. Theory of Computing (STOC'73)*, pages 240–252, New York, NY, USA, 1973. ACM.

[15] Ł. Kot and D. Kozen. Second-order abstract interpretation via Kleene algebra. Technical Report TR2004-1971, Computer Science Department, Cornell University, December 2004.

[16] Ł. Kot and D. Kozen. Kleene algebra and bytecode verification. In F. Spoto, editor, *Proc. 1st Workshop Bytecode Semantics, Verification, Analysis, and Transformation (Bytecode'05)*, pages 201–215, April 2005.

[17] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Infor. and Comput.*, 110(2):366–390, May 1994.

[18] D. Kozen. Kleene algebra with tests. *Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997.

[19] D. Kozen. On Hoare logic and Kleene algebra with tests. *Trans. Computational Logic*, 1(1):60–76, July 2000.

[20] D. Kozen. Automata on guarded strings and applications. *Matématica Contemporânea*, 24:117–139, 2003.

[21] D. Kozen. Kleene algebras with tests and the static analysis of programs. Technical Report TR2003-1915, Computer Science Department, Cornell University, November 2003.

[22] D. Kozen. On the coalgebraic theory of Kleene algebra with tests. Technical Report `http://hdl.handle.net/1813/10173`, Computing and Information Science, Cornell University, March 2008.

[23] D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In J. Lloyd, V. Dahl, U. Furbach, M. Kerber, K.-K. Lau, C. Palamidessi, L. M. Pereira, Y. Sagiv, and P. 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.

[24] D. Kozen and F. 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.

[25] D. Kozen and W.-L. D. Tseng. The Böhm-Jacopini theorem is false, propositionally. Technical Report `http://hdl.handle.net/1813/9478`, Computing and Information Science, Cornell University, January 2008. Proc. 9th Int. Conf. Mathematics of Program Construction (MPC'08), July 2008, to appear.

[26] W. Peterson, T. Kasami, and N. Tokura. On the capabilities of while, repeat, and exit statements. *Comm. Assoc. Comput. Mach.*, 16(8):503–512, 1973.

[27] J. Worthington. Automatic proof generation in kleene algebra. In R. Berghammer, B. Möller, and G. Struth, editors, *10th Int. Conf. Relational Methods in Computer Science (RelMiCS10) and 5th Int. Conf. Applications of Kleene Algebra (AKA5)*, volume 4988 of *Lect. Notes in Computer Science*, pages 382–396. Springer-Verlag, 2008.