Universiteit Leiden
Opleiding Informatica

Context Free Guarded Languages

A system for determining Guarded Strings

Name: Jelco Burger Studentnr: 1110136

Date: 19/08/2015

1st supervisor: Marcello Bonsangue
2nd supervisor: Walter Kosters

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Guarded languages are sets of guarded strings, which are derived from KAT (Kleene Algebra with Tests). KAT is an algebraic system used for program verification, making use of a combined alphabet as it merges Boolean algebra with the traditional Kleene algebra (KA). As KAT is a relatively fresh concept in language theory and computer science, many possibilities have yet to be explored. In this paper, a context-free system is presented that makes it possible to determine the language of any given KAT expression and identify whether a guarded string can be generated by a certain KAT expression. Since a guarded string can resemble programming code, a system for generating such strings can prove useful for model checking. We can check whether or not the guarded string is part of a language and thus, whether it is syntactically correct in the environment it is in. We will also take a look at encoding recursive programs with KAT expressions and guarded strings.

Contents

1 Introduction

This thesis on the subject of Guarded Languages is written as part of the bachelor of Computer Science at Leiden University and was supervised by Marcello Bonsangue. The topci of Guarded languages has recently been growing in popularity in language theory (90's). These languages composed by a combined alphabet can accomplish anything a regular language already could, but the incorporated Boolean alphabet creates a wider range of possibilities. With the added potential of conditional statements it is now possible to mimic programming syntax with Kleene algebras with tests (KAT). KAT has been useful to model while programs. In this thesis we extend KAT to model context free behaviour, so that programs with full recursion can be modelled too. Being able to systematically compose the language for any given KAT expression allows for the possibility to involve computers to algorithmicly check whether a guarded string is part of a given guarded language by applying $Hoare\ logic$ [**?**]. This is particularly useful as guarded strings can resemble basic computer programs as the base of most program correctness tools lies in Hoare logic. In Section 2 back ground information will be provided, in Section 3 the system for determining guarded strings in a context free manner will be presented. Then we will look into recursion and guarded strings in Section 4 and we end by drawing a conclusion and discuss the future possibilities in Section 5.

2 Background

In this section we will introduce guarded strings and clarify why a set of these can not be linked to a Kleene algebra (KA), the algebra concerning regular expressions. Because of the lack of possibilities that Kleene Algebra provides us with, we will take a look at Kleene Algebra with Tests (KAT), and find out how we can tailor KAT expressions to manufacture our method that defines a system that generates sets of guarded strings.

2.1 Kleene Algebra

A Kleene algebra $KA = (K, +, \cdot, ^*, 0, 1)$ satisfies the following axioms

$$\begin{aligned}
(r_1 + r_2) + r_3 &= r_1 + (r_2 + r_3) & \qquad r_1(r_2 + r_3) &= r_1r_2 + r_1r_3 \\
r_1 + r_2 &= r_2 + r_1 & (r_1 + r_2)r_3 &= r_1r_3 + r_2r_3 \\
r + 0 &= r + r = r & 1 + rr^* &\leq r^* \\
r_1(r_2r_3) &= (r_1r_2)r_3 & 1 + r^*r &\leq r^* \\
1r &= r1 = r & r_1 + r_2r_3 \leq r_3 &\rightarrow r_2^*r_1 \leq r_3 \\
0r &= r0 = 0 & r_1 + r_2r_3 \leq r_2 &\rightarrow r_1r_3^* \leq r_2
\end{aligned}$$

where r_1, r_2 and r_3 denote regular expressions and $r_1 \leq r_2$ iff $r_1 + r_2 = r_2$. Furthermore, 1 denotes the empty word λ and 0 is used when referring to the empty set $\emptyset$. The set of regular expressions over a non-empty alphabet Σ is defined as follows:

$$r := 0 \mid 1 \mid p \in \Sigma \mid (r_1 + r_2) \mid (r_1 \cdot r_2) \mid r^*$$

also referred to as R_Σ which composes a Kleene algebra considering the algebraic structure $(R_\Sigma, +, \cdot, ^*, 0, 1)$. A word over Σ is a finite sequence of elements of Σ. Σ^* is defined to be the set of all words over Σ. A language over an alphabet Σ is a subset of Σ^*. The language

corresponding L to r is defined by:

$$L(0) = \emptyset \qquad L(r_1 + r_2) = L(r_1) \cup L(r_2)$$
$$L(1) = 1 \qquad L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$$
$$L(p) = p \text{ where } p \in \Sigma \qquad L(r^*) = L(r)^*$$

The above axioms can be used to assemble a complete proof system for determining equivalence in regular expressions [6].

2.2 Kleene Algebra with Tests

A Kleene algebra with tests (KAT) is a KA with a Boolean subalgebra $K = (K, B, +, \cdot, ^*, 0, 1, ^-)$ incorporated, where $^-$ is a unary operator which denotes the negation of a Boolean expression and is defined solely on B. KAT is an equational system, combining KA with Boolean algebra, applied for the verification of programs [4]. The following holds:

- $(K, +, \cdot, ^*, 0, 1)$ is a Kleene algebra.
- $(B, +, \cdot, ^-, 0, 1)$ is a Boolean algebra.
- $(B, +, \cdot, 0, 1)$ is a subalgebra of $(K, +, \cdot, ^-, 0, 1)$.

KAT satisfies the Kleene algebra axioms along with the axioms for a Boolean algebra. Now, define $\Sigma = \{p_1, p_2, \ldots, p_n\}$ to be a set of $action$ symbols with $n \geq 1$ and define $T = \{t_1, t_2, \ldots, t_m\}$ to be a set of $test$ symbols with $m \geq 1$. KAT expressions can be defined by the following grammar:

$$b \in Bexp := 0 \mid 1 \mid t \in T \mid \bar{b} \mid b_1 + b_2 \mid b_1 \cdot b_2$$
$$e \in Exp := p \in \Sigma \mid b \in Bexp \mid e_1^* \mid e_1 + e_2 \mid e_1 \cdot e_2$$

Here, the alphabet is the combined action and testalphabet $\Sigma \cup$ T and $b_1, b_2 \in Bexp$ and $e_1, e_2 \in Exp$. Similar to KA we will omit the concatenation operator for convenience. A language corresponding to a KAT expression is called a $guarded\ language$ which is a set of $guarded\ strings$. A truth assignment to T is composed by all the elements of T, where each element is either $true$ or $false$, for instance: $t_1\overline{t_2}\overline{t_3}t_4$ is a truth assignment for a Boolean alphabet of 4 elements. We will refer to such a truth assignment as being an $atom$. Let the set of all $atoms$ be At, which has size $2^{|T|}$. A $guarded\ string$ is composed by alternating elements from At with elements from Σ, starting and ending with an element from At. The minimum size of a guarded string is one atom $\in$ At. Thus, the $guarded\ language$ over some Σ and T is $GL = (\text{At}\cdot\Sigma)^* \cdot \text{At}$. Two guarded strings x, y can be concatenated into $x \cdot y$ only if the last element (Boolean atom) of x equals the first element of y, hereby deleting one of the redundant occurences of the shared atom. If last(x) $\neq$ first(y) then the concatenation xy does not exist. For guarded languages $X, Y \subseteq GL$ we have that $X \diamond Y$ is defined to be the set of all strings xy for which $x \in X$ and $y \in Y$. The language corresponding to a KAT expression is inductively defined by:

$$GL(p) = \{\alpha_1 p \alpha_2 \mid \alpha_1, \alpha_2 \in At\} \quad p \in \Sigma \qquad GL(e_1 e_2) = GL(e_1) \diamond GL(e_2)$$
$$GL(b) = \{\alpha \in At \mid \alpha \vdash b\} \quad b \in Bexp \qquad GL(e^*) = \cup_{n \geq 0} GL(e)^n$$
$$GL(e_1 + e_2) = GL(e_1) \cup GL(e_2)$$

Two KAT expressions e_1 and e_2 are considered to be $equivalent$ [1] iff the languages they denote are equal. Thus, $GL(e_1) = GL(e_2) \iff e_1 = e_2$.
Because of the conditional statements in KAT, more languages can be created than with a KA. All languages that can be created by a KA are a subset of all possible guarded languages. To mimic a regular expression with KAT, let the test alphabet T be empty or only use Boolean statements which will always evaluate to true, like tautologies, so the language over Σ and T will be $GL = (\{\} \cdot \Sigma)^* \cdot \{\}$ which essentially is the same as regular languages over Σ^*. Thus, Regular expressions $\subseteq$ KAT expressions.

3 Context Free Guarded Strings

We will present a system for determining context free guarded strings in this section and also give a concrete example of how the rules of the system are applied. We first note that the set of all the languages that the traditional Context Free Language accepts is a subset of GL, as we can take an empty alphabet for T which will leave us with a language over Σ^*. Our system will not be a traditional 4-tuple we are used to from context free grammars, but a 5-tuple $H = (V, \Sigma, T, P, S)$ due to the combined alphabet. We will also make an extension on the definition of KAT expressions Exp to E_Exp and introduce two types of rules. The system H consists of:

- V is a set of $variables$ or $procedures$.
- Σ is the alphabet of action symbols of the language generated by G.
- T is the test alphabet containing Boolean elements.
- P is the set of procedures, P is a function of V to E_Exp. Thus, $P : V \to E\_Exp$. We use $p \rightsquigarrow e$ for P(p) = e, with p being the $name$ of the procedure and e being the $body$.
- S is the $startsymbol$, the initial procedure, $S \in V$.

We modify the grammar defined in Section 2.2 that generates KAT expression to:

$$b \in Bexp := 0 \mid 1 \mid t \in T \mid \bar{b} \mid b_1 + b_2 \mid b_1 \cdot b_2$$
$$e \in E\_Exp := a \in \Sigma \mid b \in Bexp \mid p \in V \mid e_1 + e_2 \mid e_1 \cdot e_2$$

By adding procedure names to the set of variables. We distinguish between two categories of rules in our system :

Termination rules are of the form $e \downarrow \alpha$ where $e \in E\_Exp$ is an $expression$ and $\alpha \in$ At represents the current valuation of the elements of T. The formula $e \downarrow \alpha$ indicates that e terminates with truth valuation α.
Transition rules are of the form $e \xrightarrow{\langle \alpha, q \rangle} e'$ where $\alpha \in$ At is the $label$, the $atom$ which expression e starts with. The action succeeding α is $q \in \Sigma$ and e' is the tail part of e, the string minus the first two elements.
The start symbol S now goes to the KAT-expression representing the guarded language, for instance $S \rightsquigarrow b_1\overline{b_2}q + \overline{b_1}b_2 1$.

We define the following rules:

The set of rules for termination:

1. $\dfrac{\alpha \vdash b}{b \downarrow \alpha} \quad \alpha \in \mathsf{At}$
2. $\dfrac{e \downarrow \alpha}{p \downarrow \alpha} \quad p \rightsquigarrow e$
3. $\dfrac{e_1 \downarrow \alpha}{e_1 + e_2 \downarrow \alpha}$
4. $\dfrac{e_2 \downarrow \alpha}{e_1 + e_2 \downarrow \alpha}$
5. $\dfrac{e_1 \downarrow \alpha \quad e_2 \downarrow \alpha}{e_1 e_2 \downarrow \alpha}$

The set of transition rules:

1. $q \xrightarrow{\langle \alpha, q \rangle} 1 \quad q \in \Sigma$
2. $\dfrac{e \xrightarrow{\langle \alpha, q \rangle} e'}{p \xrightarrow{\langle \alpha, q \rangle} e'} \quad p \rightsquigarrow e$
3. $\dfrac{e_1 \xrightarrow{\langle \alpha, q \rangle} e'}{e_1 + e_2 \xrightarrow{\langle \alpha, q \rangle} e'}$
4. $\dfrac{e_2 \xrightarrow{\langle \alpha, q \rangle} e'}{e_1 + e_2 \xrightarrow{\langle \alpha, q \rangle} e'}$
5. $\dfrac{e_1 \xrightarrow{\langle \alpha, q \rangle} e'}{e_1 e_2 \xrightarrow{\langle \alpha, q \rangle} e' e_2}$
6. $\dfrac{e_1 \downarrow \alpha \quad e_2 \xrightarrow{\langle \alpha, q \rangle} \alpha}{e_1 e_2 \xrightarrow{\langle \alpha, q \rangle} e'}$

Transition rules create pairs of $\langle \alpha, q \rangle$ while consuming the KAT expression from left to right. A derivation of a string with this system should always end with a termination rule on the last element. We distinguish between termination and transition rules, because the last test is never succeeded by any action as a guarded string x is defined as $x \in (\mathsf{At} \cdot \Sigma)^* \cdot \mathsf{At}$. Therefore we can not keep creating pairs of tests and actions, thus terminating with the last test. We define the guarded language GL corresponding to a KAT expression e with aid of system H by:

$$GL(e) = \{\langle \alpha, q \rangle \cdot w \mid e \xrightarrow{\langle \alpha, q \rangle} e', w \in GL(e')\} \cup \{\alpha \mid e \downarrow \alpha\}$$

This way we can recursively determine the guarded language of a KAT expression by defining the language of e' until the last element is reached or establish that the language is empty if we can not continue by any rule at some point.

3.1 Regular Languages as Guarded Languages

In regular languages there is only one alphabet $\Sigma = \{a_0, a_1, \ldots, a_m\}$ and we already stated that the set of regular languages is a subset of the set of guarded languages. Therefore we must be able to map a string $x = q_0 q_1 \ldots q_n \mid q_i \in \Sigma, \ \ 0 \leq i \leq n$ of a regular language to strings of a corresponding guarded language. For the corresponding guarded language we take the same alphabet Σ and an arbitrary test alphabet T. A string y of the guarded language will be of the

form: $y = \langle \alpha_0, q_0 \rangle \langle \alpha_1, q_1 \rangle \dots \langle \alpha_n, q_n \rangle \alpha_{n+1}$. By taking $\alpha_i \in \text{At}$ with $0 \leq i \leq n+1$ we basically say that the tests of the string can be satisfied by any atom. This results in every q_i being executed and yielding strings with different atoms between the actions. The consequence of this is that multiple strings can map to a single string x of a regular language, where the same actions take place for all of these guarded strings as the one string in the regular language they all map to.

3.2 Example

For a simple example we look at the KAT expression $e = q_1 b_1$ with $\text{T} = \{b_1, b_2\}$ and $\Sigma = \{q_1\}$, and determine the language accepted by the expression. We start with the starting symbol, $S \rightsquigarrow q_1 b_1$. The first rule we use is transition rule 2 for the start symbol:

$$\frac{q_1 b_1 \xrightarrow{\langle ?, ? \rangle} ?}{S \xrightarrow{\langle ?, ? \rangle} ?}$$

Where this leads to is unclear for now as we first need to determine what step we can do on the body of S. Therefore we look at transition rule 5 as $q_1 b_1$ is composed of two expressions. The bottom part of rule 5 must correspond with the upper part of rule 2 previously applied. This way we can build up a tree until we reach a termination rule or transition rule 1. We can also get stuck if there are no possible rules to apply, meaning no steps can be done and the language is in that case empty. Applying rule 5 yields:

$$\frac{\dfrac{q_1 \xrightarrow{\langle ?, ? \rangle} ?}{q_1 b_1 \xrightarrow{\langle ?, ? \rangle} ?}}{S \xrightarrow{\langle ?, ? \rangle} ?}$$

Using transition rule 1 we determine that q_1 leads to a 1, e.g., a $skip$. This logically makes sense since there is no test preceding q_1, making every $\alpha \in \text{At}$ an option for this step. We can now fill in the blanks:

$$\frac{\dfrac{q_1 \xrightarrow{\langle \alpha, q_1 \rangle} 1}{q_1 b_1 \xrightarrow{\langle \alpha, q_1 \rangle} 1 b_1}}{S \xrightarrow{\langle \alpha, q_1 \rangle} 1 b_1}$$

This is one step which can be performed in four different ways as α can be any atom. We can start establishing the guarded language of e now:

$$GL(S) = \{\langle \alpha, q_1 \rangle \cdot w \mid w \in GL(1 b_1)\} \cup \{\alpha \mid q_1 b_1 \downarrow \alpha\}$$

We note that we can only apply termination rule 5 for the right part of the union and then get stuck, because there is no rule for $q_1 \downarrow \alpha$. This results in the right part evaluating to the empty set $\emptyset$.
We still need to determine the guarded language for $GL(1 b_1)$. We apply termination rule 5 which can be done in two different ways:

$$\frac{1 \downarrow b_1b_2 \quad b_1 \downarrow b_1b_2}{1b_1 \downarrow b_1b_2} \qquad \frac{1 \downarrow b_1\overline{b_2} \quad b_1 \downarrow b_1\overline{b_2}}{1b_1 \downarrow b_1\overline{b_2}}$$

This leads us to $GL(1b_1) = \{b_1b_2, b_1\overline{b_2}\}$. By substituting $GL(1b_1)$ in the previous expression for $GL(S)$, we have sufficient information to determine the guarded language for e. Indeed, we have the four ways we can make a step $\langle \alpha, q_1 \rangle$ from S concatenated with the two elements of $GL(1b_1)$:

$$GL(S) = \{\langle \alpha, q_1 \rangle \cdot w \mid w \in \{b_1b_2, b_1\overline{b_2}\}\} \cup \emptyset = \\ \{\langle b_1b_2, q_1\rangle b_1b_2,\ \langle b_1b_2, q_1\rangle b_1\overline{b_2},\ \langle b_1\overline{b_2}, q_1\rangle b_1b_2,\ \langle b_1\overline{b_2},\ q_1\rangle b_1\overline{b_2},\\ \langle \overline{b_1}b_2, q_1\rangle b_1b_2,\ \langle \overline{b_1}b_2, q_1\rangle b_1\overline{b_2},\ \langle \overline{b_1}\overline{b_2}, q_1\rangle b_1b_2,\ \langle \overline{b_1}\overline{b_2}, q_1\rangle b_1\overline{b_2}\}$$

4 Recursion and Guarded Strings

As the main property of guarded strings is that they can mimic programs, we investigated whether we can use guarded strings to simulate recursive programs. By using the property of a while-loop of being recursive and the Kleene star operator in KAT expressions, we can make a KAT expression resembling a while-loop. A simple while-program is shown below:

while (b) **do**
    q;
od

This can be resembled by the KAT expression K: $(bq)^*\bar{b}$. Here $\Sigma = \{q\}$ and T $= \{b\}$ and $GL(K) = \{\bar{b},\ \langle b, q\rangle\bar{b},\ \langle b, q\rangle\langle b, q\rangle\bar{b}, \ldots\}$. The amount of times that $\langle b, q\rangle$ is copied is representing the amount of iterations of the while-loop. See [5] for more on KAT expressions and while-loops.
We can also construct KAT expressions and guarded strings for recursive functions. We consider procedure declarations $p \rightsquigarrow e$ such that p is the name of the procedure and $e \in E\_Exp$ represents the body. This expression in the body generally consists of a conditional statement followed by some (or no) action(s) and in the end the procedure p is invoked or not depending on whether the conditional statement was met. Thus, by considering the procedure declaration in our system we are able to determine the guarded strings for a recursive program such as:

proc p **do**	**proc** q **do**
a_1;	**if** t_1 **then**
call q;	call p;
a_2;	**else**
od	a_3;
	od
$p \rightsquigarrow a_1qa_2$	$q \rightsquigarrow bp + \bar{b}a_3$

Let $\Sigma = \{a_1, a_2, a_3\}$, where the elements of Σ represent the various actions in this program. Furthermore, let T $= \{t_1, t_2\}$. Now the starting symbol will lead to: $S \rightsquigarrow p$. We continue

by making KAT expressions for both of the procedures: a_1qa_2 and $bp + \bar{b}a_3$ for p and q respectively. First of all notice how the KAT expression p goes to does not commence with a Boolean atom, while the definition states that guarded strings are strings alternating between tests and actions. The Boolean at the beginning of the string is in this case 1 and is therefore omitted.
By introducing procedures $p, q \in P$, we now have to define where these new procedures goes to. Required is that p can invoke q in its body and vice versa. Consequently we define the productions: $p \rightsquigarrow a_1qa_2$ and $q \rightsquigarrow bp + \bar{b}a_3$. The guarded string representing this program depends on how the value of t_1 changes, as this is the only test element occurring. An example of a guarded string resembling this program is:
$\langle t_1\overline{t_2}, a_1\rangle\langle t_1t_2, a_1\rangle\langle\overline{t_1}\overline{t_2}, a_1\rangle\langle\overline{t_1}\overline{t_2}, a_3\rangle\langle\overline{t_1}\overline{t_2}, a_2\rangle\langle\overline{t_1}\overline{t_2}, a_2\rangle\langle\overline{t_1}\overline{t_2}, a_2\rangle\overline{t_1}\overline{t_2} \in GL(p)$
This is a string guarded string derived from the above program by our system where p and q are both invoked three times, the middle tuple represents the occurrence of q where the else-branch was reached. Even though the amount of strings representing this program is infinite, we can find any finite string using our system.

5 Conclusion and Future Work

With our system we can determine the set of context free guarded strings for any given KAT expression. Also, we can make KAT expressions for programs with full recursion and determine the guarded language corresponding to this KAT expression.
As for future work we can be look into implementing the system into a context-free parsing algorithm [2, 8]. The structural way of composing a guarded language for an expression can prove to be useful in syntax checking of programs. Proving partial correctness of (recursive) programs using Hoare logic in combination with our system is also to be explored. Another interesting thing to look into would be the correlation between the (deterministic) graphs for KAT expressions and the guarded languages corresponding to them.

References

[1] Almeida, R., Broda, S., & Moreira, N. (2012). Deciding KAT and Hoare logic with derivatives. arXiv preprint arXiv:1210.2456.

[2] Earley, J. (1970). An efficient context-free parsing algorithm. Communications of the ACM, 13(2), 94–102.

[3] Kozen, D. (1994). A completeness theorem for Kleene algebras and the algebra of regular events. Information and Computation, 110(2), 366–390.

[4] Kozen, D., & Smith, F. (1997). Kleene algebra with tests: Completeness and decidability. In Computer Science Logic (pp. 244–259). Springer.

[5] Kozen, D. (2000). On Hoare logic and Kleene algebra with tests. ACM Transactions on Computational Logic (TOCL), 1(1), 60–76.

[6] Kozen, D. (2008). Nonlocal flow of control and Kleene algebra with tests. In Logic in Computer Science, 2008. LICS'08. 23rd Annual IEEE Symposium on (pp. 105–117). IEEE.

[7] Hoare C. A. R. (1969). An axiomatic basis for computer programming. Comm. Assoc. Comput. Mach., 12:576–80

[8] Tomita, M. (1987). An efficient augmented-context-free parsing algorithm. Computational Linguistics, 13(1-2), 31–46.