Logic and Verification Seminar
On Regular Tree Model Checking

Haitm Mohammed Arhoumah Mahmoud

Technische Universität Kaiserslautern

Abstract

Regular model checking often considers parameterized systems with linear topologies, in this paper however we consider parameterized systems with tree topologies. since tree topologies can be more suitable for systems which use recursion as underlying scheme of computation. We make use of the already known results from regular model checking using strings and generalize it to trees. States are now being represented by trees over a finite alphabet, transition relation by regular relation on trees. We will introduce the tree automata and use them to recognize regular tree languages and regular tree relation, using the tree automata we will compute the transitive closure of the transition relation. We will then introduce the notion of *well-behaved transducer* and prove the termination of the method when considering such transducer.

Introduction

Testing can only show the presence of errors, not their absence.
— Edsger W. Dijkstra (1930 – 2002)

In this section we want to motivate regular *tree* model checking and summarize already known results from regular model checking.
In regular *tree* model checking we introduce regular tree languages as a symbolic representation of state spaces and regular tree relations, given as finite-state tree transducers. the tree transducers are a symbolic representation of the transition relation.
Generally Regular model checking is used to verify various classes of protocols with linear or ring-formed topologies, nevertheless there are several classes of systems for which regular model checking isn't too powerful for. there are many reasons for that i.e the topology of the system is not linear.
Regardless of which type of regular model checking we are using the huge problem stays the same, which is that the state space is not bounded, meaning a *naive* standard iteration based symbolic reachability algorithm doesn't ensure termination.

The paper we are summarizing fouces on how to compute the transitive closure for a large class of actions. we start from the tree transducer corresponding to an action, we compute then the symbolic transducer corresponding to the transitive closure. Then we introduce the *well-behaved-transducer* and ensure termination.

Tree Automata

In order for us to define the tree transducer when need to first introduce the tree automata, and some related preliminaries
A *tree automaton* over a ranked alphabet Σ is a tuple $A = (Q, F, \delta)$ where Q is finite set of states, $F \subseteq Q$ is a set of final states and δ is the transition relation, represented by a set of rules of the form $(q_1,...,q_p) \xrightarrow{f}$ q where $f \in \Sigma$ and q_1,...,q_p,q $\in Q$.

Conventions:

i) Throughout this paper we assume Q to be finite.
ii) Furthermore we let the tree automata A process from the leaves to the root. the semantic of transition rule of the form $(q_1,...,q_p) \xrightarrow{f}$ q is the following: if the children of a node n are already annotated from left to right with $q_1, ...q_n$ and if $\lambda(n) = f$ then the node n can be annotated by q. we also make the convention that $\xrightarrow{f}$ q (with empty left-hand-side) means that a leaf labeled with $f \in \Sigma_p$ can be annotated by q.
a *run* r of A on a tree T = (S,λ) *in* T(Σ) is a mapping from S to Q such that for each node n $\in$ T with children $n_1...n_k$: $(r(n_1)...r(n_k)) \xrightarrow{\lambda(n)} r(n) \in \delta$
By $A \Rightarrow_A q$ we denote that r is a run of A on T
An m-ary relation on the alphabet Σ is a set of tuples of the form $(w_1, ..., w_n)$, where $w_1, ..., w_n \in \Sigma^*$ and $|w_1| = \cdots = |w_n|$
we notice that a language K over Σ^m characterizes an m-ary relation $[K]$ on Σ in the sense that $(w_1, \ldots, w_n) \in [K]$ if and only if $(w_1, \times \cdots \times w_n) \in K$. A relation R is said to be regular if R = [K] for some regular language K
We use the tree automaton to define relations, namely an automaton A over $\Sigma^{\bullet}(m)$ characterizes an m-ary relation on T(Σ) namely the Relation R = [L(A)], similar to the definition over strings, A relation R is said to be regular if there is a tree automate A with R = [L(A)]. notice that we often use R instead of R(A)
We are now ready to define the tree transducer: Let D be a tree automaton over $\Sigma^{\bullet}(2)$ we call D a tree transducer over Σ.

Example (tree Automata): Using the tree Automata we just defined we can construct such an Automata B such that B recognizes the tree language which is the set of true boolean expressions over Σ

$B = (Q, F, \delta)$ over $\Sigma = \{0,\ 1,\ \text{and},\ \text{or}\}$ (with $\rho(and) = \rho(or) = 2$ and $\rho(1) = \rho(0) = 0$), $Q = \{q_0, q_1\}$,$F = \{q_1\}$ and δ defined as:

$$\begin{array}{ccc}
\xrightarrow{0} q_0 & \xrightarrow{1} q_1 & \\
(q_0, q_0) \xrightarrow{or} q_0 & (q_0, q_1) \xrightarrow{or} q_1 & \\
(q_0, q_0) \xrightarrow{and} q_0 & (q_1, q_0) \xrightarrow{or} q_1 & (q_0, q_1) \xrightarrow{and} q_0 \\
(q_1, q_0) \xrightarrow{and} q_0 & (q_1, q_1) \xrightarrow{and} q_1 & (q_1, q_1) \xrightarrow{or} q_1
\end{array}$$

Notice that B accepts only true boolean expressions over Σ

Example (tree Transducer): Let $B = (Q, F, \delta)$ over Σ ={0,1,or,and,not,x,y} (with $\rho(and) = \rho(or) = 2, \rho(not) = 1$ and $\rho(1) = \rho(0) = 0$), $Q = \{q_0, q_1\}$,$F = \{q_1\}$
Consider now the negation operation that take an expression over Σ and returns its negation
($\delta-$ Sketch) The Transducer would replace
• every 0 by 1
• every or by and
• every x by not(x)
(and vice versa)

Symbolic Transducers

In pursuance of computing the transitive closure we need to define the notion of symbolic transducer.
Informally *symbolic* transducers are compact representations of history transducer, a *history* transducer is an infinite tree transducer of a given tree transducer D. the name *history* reflects that the transducer encodes the histories of all derivations.

Definition

Given a tree transducer $D = (Q, F, \delta)$ over a ranked alphabet Σ the history transducer H for D is an (infinite) transducer (Q_H, F_H, δ_H) such that $Q_H = Q^*$ and $F_H = F^*$
δ_H is the set of rules of the form $(w_1, \dots, w_p) \xrightarrow{f, f'} w$ such that there is a k $\geq$ 0 where for all rules the following holds:
i) $|w_1| = \cdots = |w_p| = |w| = k$
ii) $\exists f_1, f_2, \dots f_{k+1}$ with $f = f_1$ and $f' = f_{k+1}$, and $(w_1(i), \dots, w_p(i) \xrightarrow{f, f_{i+1}} w(i)$ belongs to δ $\forall i \leq k)$

by definition of the history transducer H we derive the following lemma which states that H characterizes the reflexive transitive (presumably the most important lemma of the paper)
Lemma *For a transducer D and its history transducer H, we have $R(H) = (R(D))^*$*

Definition

Given a tree transducer $D = (Q, F, \delta)$ over a ranked alphabet Σ We define the *symbolic tree transducer S* for D to be transducer (Q_S, F_S, δ_S) where Q_S is a set of regular expressions over Q,F_S is a set of regular expressions over Q, F_S is a set of regular expressions over Q and δ_S contains a set of rules each of the form $(\phi_1, \ldots, \phi_p) \xrightarrow{f,f'} \phi$
Lemma Let $\phi_1, \ldots, \phi_p$ be regular expressions and f, f' Symbols we define $(\phi_1, \ldots, \phi_p \xrightarrow{f,f'})$ to be the set $\{w | \exists\ w_1 \in \phi_1 \ldots \exists w_p \in \phi_p, (w_1 \ldots, w_p) \xrightarrow{f,f'} w \in \delta_H\}$
for regular expressions $\phi_1, \ldots, \phi_p$ and symbol f, f', the set $\phi_1, \ldots, \phi_p \xrightarrow{f,f'}$ is effectively regular
For the following algorithms this lemma ensures the computation of δ consistently computable

Algorithm 1: Algorithm for computing symbolic transducer

```
    Input  : Tree Transducer$D = (Q, F, \delta)$
    Output: Symbolic Transducer $S = (Q_S, F_s, \delta_S)$
1 $Q_S = \emptyset, F_S = \emptyset, \delta_S = \emptyset$
2 repeat
3   for each $p, f, f' \in \Sigma_p$ and $\phi_1, \ldots, \phi_p \in Q_s$ do
4       $\phi := (\phi_1, \ldots, \phi_p) \xrightarrow{f,f'}$
5       $Q_S := Q_S \cup \{\phi\}$
6       $\delta_S := \delta_s \cup \{(\phi_1, \ldots, \phi_p)\} \xrightarrow{f,f'} \phi\}$
7 until no new states or rules can be added to $Q_S$ and $\delta_S$
8 $F_S := \phi \in Q_S | (\phi \cap F^*) \neq \emptyset$
```

Lemma *consider a transducer D and its corresponding history transducer H and symbolic transducer S. For trees T and T'*
1) if $(T \times T') \Rightarrow_S \phi$ then $(T \times T') \Rightarrow_H w \quad \forall w \in \phi$
2) if $(T \times T') \Rightarrow_H w$ then $(T \times T') \Rightarrow_S \phi$ for some $\phi | w \in \phi$
Corollary *R(S)=R(H)*
Using this corollary we get the following theorem
Theorem *For a transducer D and its corresponding symbolic transducer S we have R(S) =* $(R(D))^*$

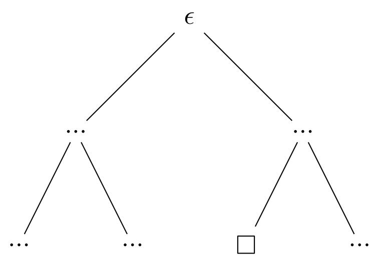
Notice if we let *algorithm 1* run, we can't ensure termination, since there are infinitely many regular expressions over the set of states of D.

Saturation

In this section we want to provide some conditions under which our *Algorithm* will always terminate. for that we need some preliminaries namely *idempotent states*
Intuitively idempotent states denote states for which the transducer only accepts context corresponding to a copy operation on nodes

Definition

A *context* over Σ is a tree (S_C, λ_C) over $\Sigma \cup \{\square\}$ such that there is exactly one $n_c \in S_C$ with $\lambda(n_c) = \square$
Informally a context is tree with a single "hole " at one of its leaves (see following Figure)
For a context $C = \{S_C, \lambda_C\}$ and a tree $T = (S, \lambda)$ we define C[T] to be the tree (S_1, λ_1), where
i) $S_1 = S_C \cup \{n_c \cdot n | n_c \in Sc \text{ and } \lambda_C(n_c) = \square and n \in S\}$
ii) for each n $\in S_c$ with n $\neq n_c$ we have $\lambda_1(n) = \lambda_C(n)$.
iii) for each $n_1 = n_c \cdot n$ with $n \in S$ we have $\lambda_1(n_1) = \lambda(n)$
the operation we just defined represents a substitution, where we replace the hole in C by T



For an automaton $A = (Q, F, S)$ we define the *suffix* of a state $q \in Q$ as follows

$$suff(q) = \{C : Context | C(q) \Rightarrow_A F\}$$

a state is said to be *idempotent* if and only if suff(q) contains only copying contexts, a context is said to copying if it correspondes to a copy operation on all its node

Definition

For a transducer $D = (Q, F, \delta)$ its history transducer $H = (Q_H, F_H, \delta_H)$ and its symbolic transducer $S = (Q_S, F_S, \delta_S)$ Consider $W \subseteq Q_H$ and $X \subseteq Q$. We define the saturation of W by X denoted $\lceil W \rceil_X$ as the smallest set W' containing W and closed under the following two rules for each $q \in X$
1) if $w_1 \cdot w_2 \in W'$ then $w_1 \cdot q \cdot w_2 \in W'$
2) if $w_1 \cdot q \cdot q \cdot w_2 \in W'$ then $w_1 \cdots q \cdot w_2 \in W'$
We let $Q_{idm} \subseteq Q$ to be the set of all idempotent states in Q
We observe that the saturation operation defines an equivalence relation on set of of states H
and therefore (by lemma 2) also for the states Q_S
So now if we merge all equivalent states in S together, we could achieve a much better termination condition, which is to terminate as soon as there are no new states added. this intuition is very helpful as we will soon see
Theorem R(S) = R(S_{SAT})
we will know introduce a series of lemmas, which will lead us to the MyHill-Nerode theorem
Lemma 5 Let $w_1, w_2 \in Q_H$ and let $q \in Q$be an idempotent state.
1) $suff(w_1 \cdot q \cdot w_2) \subseteq suff(w_1 \cdot w_2)$
2) $suff(w_1 \cdot q \cdot w_2) \subseteq suff(w_1 \cdot q \cdot qw_2)$
Lemma 6 For a $W_1, W_2 \subseteq Q_H$ if $\lceil W_1 \rceil = \lceil W_2 \rceil$ then $suff(W_1) = suff(W_2)$
Lemma 7 if $\exists i \leq p : (\phi_1, \ldots, \phi_i, \ldots, \phi_p) \xrightarrow{f,f'} \phi \in \delta_S$ and $suff(\phi_i) = suff(\phi_i')$ then $\exists \phi'$ such that $suff(\phi) = suff(\phi')$ and $(\phi_1, \ldots, \phi_i', \ldots, \phi_p) \xrightarrow{f,f'} \phi' \in \delta_S$
Now we observe that our equivalence relation is congruence, therefore we can apply the extension of the MyHill-Nerode theorem to trees

Termination

To make a good use of the above results we need to ensure that we only generate a finite number of equivalence classes.
In this last section we want to introduce the last class of transducer, for which termination is guaranteed.
These transducers have a set of states which can partitioned into three parts:
1) states whose prefixes only perform copy operations
2) states which which are *idempotent*
3) states which perform changes, for these states we require that they satisfy the finite local depth property

Definition

Let $Q_1 \subseteq Q$ For a regular relation R(D) and $k \in \mathbb{N}$ we say that R has *local depth* k with respect to Q if the following for R holds:
1) For Two Tree $T = (S, \lambda)$ and $T' = (S, \lambda')$ and $(T, T') \in R^m$ Then there

are tree $T_i = (S, \lambda_i)$ for $i : 0 \leq i \leq m$ such that $T_0 = T, T_m = T'$ related by accepting runs $T_i \cdot T_{i+1} \Rightarrow_D F$
2) $\forall n \in S$, there are at most k different j with $r_j(n) \in Q$

Definition

We call A transducer $D = (Q, F, \delta)$ *well-behaved* if Q contains:
1) a single copying prefix state q_{cpy}
2) a single idempotent state q_{idm}
3) and R(D) has a finite local depth with respect to $Q \setminus \{q_{cpy}, q_{idm}\}$
Lemma 8 *for any regular expression* ϕ generated by Algorithm 1, if $w^l \cdot qcpy \cdot w^r \in \phi$ then $w^l \cdot z \cdot w^r \in \phi$ for any $z \in q_{cpy}^*$
Informally the lemma means that a state q_{cpy} can only appear in the form q_{cpy}^* after generating the regular expression by algorithm 1
The next two lemmas will help us to further restrict the form of the regular expression
Lemma 9 Let ϕ be regular expression generated by algorithm 1, if $\phi \subseteq \{q_{cpy}, q_{idm}\}^*$ then $\lceil \phi \rceil_{\{q_{idm}\}}$ is the union of one or more of the following regular expressions:

$$\begin{array}{rrr} q_{idm}^* & q_{idm}^+ & (q_{cpy} + q_{idm})^* \\ q_{idm}(q_{cpy} + q_{idm})^* & (q_{cpy} + q_{idm})^* q_{idm} & q_{idm}(q_{cpy} + q_{idm})^* q_{idm} \\ (q_{cpy} + q_{idm})^* q_{idm} (q_{cpy} + q_{idm})^* & & \end{array}$$

Lemma 10 Let D be *well-behaved-transducer* with local depth k, Algorithm 1 needs only to consider regular expressions of the form $\phi_0 \cdot q_1 \cdot \phi_1 \cdot q_2 \cdots \phi_{n-1} \cdot q_n \cdot \phi_n$ with n $\leq$ k each $\lceil \phi_i \rceil$ is the *union of one or more of the seven regular expressions* as describe above in lemma 9, note that $q_i \notin \{q_{cpy}, q_{idm}\}$
With that we can now ensure termination for the algorithm 1
Theorem 3 *Algorithm 1 terminate for any well behaved transducer*

References

• Regular Tree Model Checking Parosh Aziz Abdulla, Bengt Jonsson, Pritha Mahata, and Julien d'Orso (CAV'02)