UNDERGRADUATE PROJECT REPORT

Saturation Algorithms for Ordered Tree Pushdown Systems

Sarthak Garg
Under the guidance of Professor Anil Seth
November 24, 2016

1 ABSTRACT

Pushdown systems, due to the availability of a stack, can model the control flow of programs in languages such as C and Java. Bouajjani et al. first gave a saturation algorithm for deciding the reachability problem in pushdown systems. After that through the years, many attempts have been made to design saturation based algorithms for different forms of push down systems and for solving harder problems than reachability. We explore and analyse one such extensions:

- Ordered Tree-Pushdown Systems [2] by Lorenzo Clemente, Pawel Parys, Sylvain Salvati, Igor Walukiewicz, where they define a new class of Pushdown systems which subsumes several other classes

This whole report is an attempt to provide a summary and my interpretation of the above paper

2 PRELIMINARIES

2.1 PUSHDOWN SYSTEMS

A pushdown system is a triple of (P, T, δ). P is a set of control locations, T is the set of stack alphabet and δ is the set of transitions of the pushdown system. Formally δ is a function from $P \times T$ to $P \times T^*$. A transition $(p, \gamma) \rightarrow (q, \sigma) \in \delta$ if upon having the control state as p and the top of the stack symbol as γ, the system can to a transition to a control state q and replace the top of the stack by a stack word σ. From the presence of this transition, we can say that the state of the system $(p,\gamma.\omega)$ is a predecessor of the state $(q,\sigma.\omega)$ where ω is any stack word. Let this predecessor function from states to states be defined by pre(s). Now define the transitive and reflexive closure of $pre(s)$ as $pre^*(s)$. The reachability problem in pushdown systems is to compute the set $pre^*(s)$ for any given state s.

2.2 Alternating Multi Automaton

It can be seen by inspection that for even simple pushdown systems the set of configurations in $pre^*(s)$ can be infinite. To represent this infinite set of states, we need some kind of a finite structure. Alternating multi automaton provides a good choice for this by representing regular sets of configurations of a pushdown system. For a pushdown system (P,T,δ), an alternating multi automaton is a five tuple (Q,Σ,Δ,I,F). Q is a set of states of the automaton. I is the set of initial states of the automaton which comprise exactly of the control state set P of the pushdown system $(I \subset Q)$. F is the set of accepting states. Σ is the set of alphabets of the automaton which is equal to the stack alphabet of the pushdown system. Δ is a function from $(Q \times \Sigma)$ to the power set of Q. $(q,a,Q' \subseteq Q) \in \Delta$ if there is a transition $q \xrightarrow[a]{} Q'$ in the automaton. We can see the alternations by looking at how runs are defined for multi automatons.
$q \xrightarrow[a\omega]{} Q_1 \ldots Q_n$ if $\exists$ a transition $q \xrightarrow[a]{} \{q_1 \ldots q_n\}$ and $q_k \xrightarrow[\omega]{} Q_k \forall k$ such that $1 \leq k \leq n$
A configuration (p,ω) is accepted by the multi automaton iff $p \xrightarrow[\omega]{} Q' \subseteq F$

2.3 Saturation algorithm for computing Pre*

Bouajjani, Esparza and Maler in their seminal paper [1] showed that pre^* preserves regularity, i.e. if S be a set of configurations accepted by an alternating multi automaton A, then an alternating multi automaton A^* recognizing the set of configurations $pre^*(S)$ can be constructed. We start the algorithm by taking the automaton A and at each step adding new transitions from the initial states to some set of states. Since no new states are added, the algorithm is bound to terminate in exponential number of steps.
The correctness of saturation algorithms is proved by the technique of valuation soundness and valuation completeness.

3 Extending saturation algorithm for Ordered Tree Pushdown System

Clemente, Parys, Salvati and Walukiewicz introduce an extension of pushdown systems: Ordered tree pushdown systems (OTPS) [2]. They then formulate a saturation based algorithm for computing reachability for OTPS. OTPS are of interest because of their great expressive power. They can simulate Ordered multi pushdown systems, Annotated higher order pushdown systems, and Krivine machine. We present their saturation algorithm and it's analysis here.

Algorithm 1 Saturation for Reachability

```
procedure SATURATION
    Input: Pushdown system B and set of configuration S
    $A \leftarrow$ Alternating multi automaton recognizing S
    $done \leftarrow false$
    while $done = false$ do
        $done \leftarrow true$
        for every transition $(j,\gamma) \rightarrow (k,\omega)$ in B do
            for every subset of states Q' in A such that $k \xrightarrow[\omega]{} Q'$ do
                if The transition $j \xrightarrow[\gamma]{} Q'$ is absent then
                    $done = false$
                    Add transition $j \xrightarrow[\gamma]{} Q'$ to A
                end if
            end for
        end for
    end while
    Return $A$
end procedure
```

3.1 Ordered tree

A node of a tree is a word over positive integers, i.e. $u \in N^*$. A predecessor, successor relation can then be defined over nodes as follows:
u pred v iff u is a prefix of v
u succ v iff v is a prefix of u
We can define a tree domain $D \subseteq N^*$ such that D is prefix closed and if $u.(i+1) \in D$ then $u.i \in D$ for $i \in N$. This condition just states if the $(i+1)^{th}$ child of u exists then i^{th} must exist. Next we define a ranked alphabet as the tuple $(\Sigma, rank)$ where Σ is the set of alphabet and rank is a function from Σ to N. The ranking function describes the arity or the number of children of every alphabet. This ranked alphabet can be extended to the tuple $(\Sigma, rank, ord)$ where ord is a function from Σ to the set of whole numbers. Once we have an ordered alphabet in place, we can define a tree as a labelling of the nodes in the domain with the ordered alphabet. Formally tree is a function t from the tree domain D to Σ of the ranked alphabet, such that
$t(u) = a \in \Sigma$ and number of successors of $u = n = rank(a)$
$t^{-1}(a) = \{u \in D$ such that $t(u) = a\}$
The order of a tree t is the order of it's root alphabet, i.e. $ord(t) = ord(t(\epsilon))$

3.2 Rewrite rules and Substitution

Currently trees are mapping from the nodes to a ranked alphabet. This allows for fixed trees. However to denote a family of trees, which have a common structure at the top, we extend the alphabet to include some variables $V = \{x, y \dots\}$ with some order and a rank = 0, i.e. we restrict the placement of variables at the leaves of the tree.

To instantiate a tree from such family of trees, a substitution σ can be defined. σ is a mapping from each variable to a tree, such that $ord(x) = ord(\sigma(x))$ for every variable x. Now the tree $t\sigma$ is a tree obtained by replacing every variable in the leaf nodes by it's corresponding substitution tree.
In case of a normal pushdown system, in each transition we replace the 'top' alphabet of the stack with some word. In case of tree pushdown systems, we are thus motivated to replace the top portion of a tree with some other top portion. This implies that each transition can be expressed as a 'rewrite rule' which is nothing but a tuple of trees (l, r) enriched with variables.
To restrict the possible rewrite rules, the concept of linearity and groundness are introduced as follows:
A tree t is linear if each variable occurs in t atmost once.
For a pair of trees l and r, l is r-ground if l and r do not share any variables.

3.3 Alternating Tree Automaton

The alternating tree automaton is a modification of the alternating automaton discussed in the preliminaries. The stack alphabet is replaced by the ranked alphabet Σ. Also instead of containing transitions of the form $q \underset{a}{\rightarrow} Q'$, the tree automaton contains the transitions of the form $p \underset{a}{\rightarrow} P_1 P_2 \ldots P_n$ where $n = rank(a)$. We impose restrictions of order on the transition. For any 2 transitions $p \underset{a}{\rightarrow} \ldots$ and $p \underset{b}{\rightarrow} \ldots$, $ord(p) := ord(a) = ord(b)$. The second restriction is that on any run, for all states q reaching a variable x, $ord(q) = ord(x)$.
The set of trees on which a run exists, are said to be accepted by the alternating tree automaton.

3.4 Ordered Tree Pushdown Systems

OTPS is a generalization of pushdown system, where the pushdown is a tree instead of a stack word. An intuition of the transitions in such a system was given in the section of 'Rewrite rules and substitutions'. Formally an OTPS is a tuple (n, Σ, Δ, P). The ranked tree alphabet Σ contains order at most n. The set P comprises of the control locations (similiar to the standard pushdown system) and Δ is the set of rewrite rules (l, r) of the following 2 forms as per [2]:

- (shallow rule) $a(u_1, u_2 \ldots u_m) \rightarrow r$. Here each u_i is r-ground or a variable
- (deep rule) $a(u_1, u_2 \ldots u_k b(v_1, v_2 \ldots v_m), u_{k+1} \ldots u_m) \rightarrow r$. Here all the placeholders in the left hand side are r-ground or variables. Moreover for all i such that $ord(u_i) \leq ord(b)$, u_i is r-ground.

3.5 Saturation Algorithm for AOTPS

A saturation based algorithm is used to compute reachability. Firstly to deal with r ground subtrees, we introduce states of the form p^v for every subtree present in the trees of the rewrite rules. We consider 2 cases as per [2]:

- Handling shallow rules: For a rule $(p, l) \rightarrow (q, r)$
  Let $l = a(u_1, u_2 \ldots u_m)$. We compute the sets of states $P_1 \ldots P_m$. If u_i is r-ground, $P_i = p^{u_i}$.

If u_i is a variable (say x), then consider a run from q on the tree r and collect all the states that reach x (Let this set be R). Set $P_i = R$
Now add a transition $p \underset{a}{\rightarrow} P_1 \dots P_m$

- Handling deep rules: For a rule $(p, l) \rightarrow (q, r)$
  Let $l = a(u_1, u_2 \dots u_k b(v_1, v_2 \dots v_m), u_{k+1} \dots u_m)$. Without loss of generality we can assume that u_i is r-ground for all $i \leq k$. If this is not so, then a shallow rule permuting the order of the subtrees can be introduced.
  We can not just do away with adding extra transitions, since we have a lookahead, for which an extra state must be created. Proceeding similiar to the case of shallow rule
  Attempt 1: We can compute the sets P_i corresponding to u_i and S_i corresponding to v_i. Add the following 2 transitions:
  $p \underset{a}{\rightarrow} P_1 \dots P_k Q^b P_{k+1} \dots P_m$ and
  $Q^b \underset{a}{\rightarrow} S_1 \dots S_{m'}$
  In this case, the number of new states added (of the form Q^b) are finite and thus the algorithm is bound to terminate. However this approach is erroneous and violates soundness. This is illustrated in by the following example.
  Let $l = a(b(x)y)$. It is possible that in a particular run tree t of r, $t^{-1}(x) = q_0$ and $t^{-1}(y) = q_1$ and in another run tree s of r, $s^{-1}(x) = q_1$ and $s^{-1}(y) = q_0$. Corresponding to these 2 run trees, if the transitions are added as specified above, then we can generate a run tree k from state p on l such that $k^{-1}(x) = q_0$ and $k^{-1}(y) = q_0$. Hence soundness is violated.
  Attempt 2: To overcome this, we need to 'save' the context comprising of the rewrite rule and the states collected in the variables of $u_{k+1} \dots u_k$ inside the intermediate state Q^b. We thus use $g(P_{k+1} \dots P_m)$ (where g is the rewrite rule) in place of Q^b. The ordering condition in the deep rules ensure that the number of such new intermediate states introduced remains bounded.

References

[1] O. M. Ahmed Bouajjani, Javier Esparza. Reachability Analysis of Pushdown Automata: Application to Model-Checking. `http://www.lsv.ens-cachan.fr/~demri/BEM97.pdf`.

[2] S. S. I. W. Lorenzo Clemente, Pawel Parys. Ordered Tree-Pushdown Systems. `https://arxiv.org/pdf/1510.03278v1.pdf`.

[3] C. O. M.Hague. Saturation Method for the Modal Mu-Calculus. `https://www.cs.ox.ac.uk/files/2798/fullmu.pdf`.