Notes on PRE-SET PUSHDOWN AUTOMATA

Jan van Leeuwen

Department of Computer Science
State University of New York at Buffalo
Amherst, New York 14226

Department of Mathematics
University of Utrecht
Utrecht, The Netherlands

ABSTRACT. Motivated by practical implementation-methods for recursive program-schemata we will define and study presetting techniques for push-down automata. The main results will characterize the languages of preset pda's in terms of types of iterated substitution languages. In particular when conditions of "locally finiteness" and of "finite returning" are imposed we get a feasible machine-model for a class of developmental languages. The accepted family extends to the smallest AFL enclosing it when we drop the condition of locally finiteness. At the same time this family will be the smallest such full AFL. If all conditions are removed, preset pda's exactly represent the family of iterated regular substitution languages, a sub-family of the indexed languages. Deterministic preset pda's are also studied, and the language-family they define is shown to be closed under complementation, generalizing a classical result.

> "Any theory ... formulates an ideal average which abolishes all exceptions at either end of the scale and replaces them by an abstract mean."
>
> in C. G. Jung: The Undiscovered Self.

---

* This work has in part been supported by the Center for Mathematical Methods in the Social, Biological, and Health Sciences (SUNY at Buffalo), by NSF grant GJ 998, and by NATO grant 574.

1. INTRODUCTION

Eventually we like to introduce what might be called: "developmental systems - a programmer's point of view", but we will not immediately emphasize it here. We rather follow the original approach which led to such implications and start with analyzing some automaton-theoretic concepts.

Push-down automata represent the execution-mechanism of parameter-less recursion. They were successfully used in context-free language theory, occasionally in the theory of (monadic) program-schemata, and the deterministic version has been extensively analyzed in parsing.

There has been a twofold motivation for presetting the amount of storage in machine-models with a stack-like external memory. First of all in most implementations there is a definite series of locations allocated as a push-down register, whenever it is required. Secondly familiar types of (single variable) recursion can most efficiently be simulated when the stack is implicitly used as a counter at the same time. In the latter type of application the machine will basicly recognize/execute instruction-sequences which appear on the input-tape, a not very common but certainly useful interpretation of input.

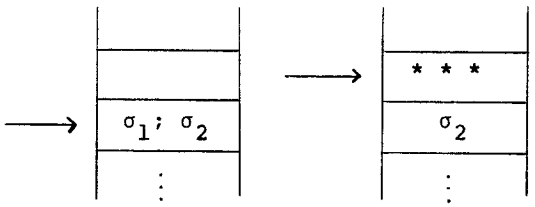
By presetting a push-down automaton we mean that at the very beginning of a computation a certain stack-square is allocated as the maximum location (or "highest" point) to which the stack may grow during that computation.

We wish to remark that if this were all, the accepted languages would still be context-free, but not all computations would terminate because the allocated space might be insufficient. The new feature is to let an overflow-indicator (or "interrupt") actively influence the computation.

Therefore the machine-model will have two transition-functions: δ (to be used whenever the stack is not maximal) and δ_{top} (to be

used when the stack reached its preset maximum), both having well-known formats.

There is one more practical concept never mentioned for ordinary pda's but relevant when the stack is going to be used as implicit parameter-value. It is the concept of an "empty" location, but this can only be argumented with more information about how we look at what can occupy locations. In the "squares" on the stack are pieces of program (or rather, pointers to subroutines).



When σ_1 is a recursive call, it will be removed from the current (top-most) location and the procedure-body called for will be pushed in the next one. When we return and find that σ_2 is a recursive call as well, an "empty" location is created and no garbage-collectioning should happen as it would improperly destroy the counting. Empty locations in the body of the stack are filled with ϕ.

2. PRESET PUSHDOWN AUTOMATA

We give a formal description of the general model.

Definition. A preset pda is an 8-tuple $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, \delta_{top}, q_0, Z_0, F \rangle$ with Q (states), Σ (inputs), Γ (stack-symbols), q_0 (initial state), Z_0 (initial stack-contents), F (final states) as usual, and for all $p \in Q$, $a \in \Sigma \cup \{\lambda\}$, $\gamma \in \Gamma$ $\delta(p, a, \gamma)$ a finite set of instructions of the form (q, ϕ), (q, A), $(q, \phi A)$, (q, AB) $(A, B \in \Gamma)$ and, similarly, $\delta_{top}(p, a, \gamma)$ a finite set of instructions of the form (q, ϕ), (q, A).

There can be more general definitions that allow for writing longer words per move, but it can be shown (see [55]) that there is no gain in power. Observe that this version is close to the program-cruncher we motivated it with, and δ_{top} for instance never writes beyond the permissible limit. The given version is a normal form.

Here is an example of a preset-pda, with notions of acceptability defined (as usual) by empty store and final state.

Example. A preset pda for $L = \{a^{n^2} \mid n \geq 1\}$ would operate as follows. Say the stack is preset at k, with Z_0 as bottom-marker. With instructions $\delta(\text{state}, a, Z_0) = \{(\text{state}, Z_0Z)\}$, $\delta(\text{state}, a, Z) = \{(\text{state}, \phi Z)\}$ it will make k-1 moves on input a and push a Z to the top. It reverses with $\delta_{top}(\text{state}, a, Z) = \{(\text{state}, \phi)\}$ and returns to the (first) Z_0 it finds downwards on λ-input. Then it lifts Z_0 two locations higher (on λ-input) and repeats the same cycle as before over and over again. When, lifting Z_0, it would have passed the preset limit it stops in a non-accepting state, otherwise it goes on until at last Z_0 is lifted into the maximal location and sweeps down on $\delta_{top}(\text{state}, a, Z_0) = \{(\text{final state}, \phi)\}$.

At a successful termination $k + (k - 2) + \ldots + 1 = (\frac{k+1}{2})^2$ (an integer) moves on input a were made, and the machine will accept all and exactly all the squares.

Instantaneous descriptions of a preset pda are of the form (state, remaining input, push-down contents, n), where n is the presetting relevant and constant for the particular computation.

Definition. A language L is called a preset pda-language if and only if a preset pda $\mathcal{U}$ as above exists such that

$L = \{x \in \Sigma^* \mid \exists_{q \in F}(q_0, x, Z_0, n) \overset{*}{\vdash} (q, \lambda, \phi, n)$ for at least one $n \geq 1\}$.

In the definition it is emphasized that more than one presetting might actually be appropriate for acceptance.

As usual for non-deterministic machines there may be both accepting and rejecting computations for an input-string, but it will still be counted in the language.

3. THE LANGUAGES OF PRESET PDA'S

Here we will develop a generative (as opposed to analytic) description of preset pda's in terms of their languages. The general theory rightaway leads into the algebraic F-iteration grammars developed in [57] and made more explicit in [103]. Most general results proved for various parallel rewriting systems follow from a few theorems in this area ([57]). Here we only give a few of the abstract concepts that were developed and found useful.

A family of languages will be defined as usual, but in addition we always assume closure under isomorphism.

Definition. Let F be a family of languages. The hyper-algebraic extension of F consists of all languages of the form $\bigcup_{k \geq 0} \tau^k(\$) \cap \Sigma^*$, where τ is an F-substitution and Σ an alphabet.*)

When F is a quasoid (i.e. containing C^* and closed under $\cap R$ and finite substitution), the hyper-algebraic extension becomes an AFL closed under iterated F-substitution. Several algebraic

*) Footnote. In later generalizations hyper-algebraic extensions were defined differently and more widely!

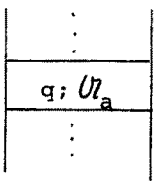
results were obtained.

The characterization theorem we will prove for preset pda languages is an interesting analogue of a classically known theorem for context-free languages (which can be described as the algebraic extension of the family of regular languages*)).

THEOREM. The family of preset pda languages is the hyper-algebraic extension of the regular languages.

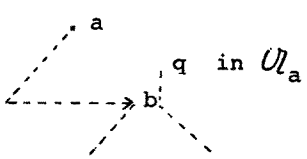
Proof.

Let τ be a regular substitution, $\mathcal{A}_a = \langle Q_a, \Sigma, \delta_a, q_a, F_a \rangle$ a finite automaton for $\tau(a)$. $\mathcal{A}_a$ may have λ-transitions, and we will in fact for simplicity assume that always $q_a \notin F_a$. Non-empty stack-locations will have a contents as shown below.



where $q \in Q_a$, representing that in generating a member of $\tau(a)$ we got as far as q in the finite automaton for it.

The idea is not to generate the word from $\tau(a)$ immediately as a whole, but symbol by symbol and each time a next symbol is generated to use it to expand its τ-image to the preset top first before proceeding with the next symbols. Thus we generate $\tau(S)$ in strictly leftmost manner and use the finite automata as a finite code for the (possibly) arbitrarily long strings we can rewrite individual symbols with.



*) Footnote. See van Leeuwen, J., "A generalization of Parikh's theorem in formal language theory", Tech. Rep. 71, Dept. of Comp. Sciences, SUNY, Buffalo, 1974.

Here is how the instructions look like. We only give their basic schemes. First realize that expansion to the present top is entirely on λ-input, only δ_{top} checks input. Therefore

$\delta(\text{state}, \lambda, [q; \mathcal{U}_a]) = \{(\text{state}, [r; \mathcal{U}_a]) \mid r \in \delta_a(q, \lambda)\} \cup$

$\{(\text{state}, \phi) \mid \exists_{r \in F_a}\ r \in \delta_a(q, \lambda)\} \cup$

$\{(\text{state}, [r; \mathcal{U}_a][q_b; \mathcal{U}_b]) \mid b \in \Sigma \text{ and } r \in \delta_a(q, b)\} \cup$

$\{(\text{state}, \phi[q_b; \mathcal{U}_b]) \mid b \in \Sigma \text{ and } \exists_{r \in F_a}\ r \in \delta_a(q,b)\}.$

Note that we may or may not stop in a final state when generating a word of $\tau(a)$. When the preset maximum (which actually stands for how far we iterate) is reached, then $\delta_{top}(\text{state}, b, [q; \mathcal{U}_a]) =$

$\{(\text{state}, [r; \mathcal{U}_a]) \mid r \in \delta_a(q, b)\} \cup$

$\{(\text{state}, \phi) \mid \exists_{r \in F_a}\ r \in \delta_a(q, b)\}$

and the obvious rules on λ-input for traversing possible λ-transitions. Successful termination now is actually on empty store but as standard we may non-deterministically send the machine in a final state at the same time.

The formal proof of the reverse is tedious but follows the same lines as the restricted case worked out in detail in [55]. Here we give the basic type of constructs.

Let $\mathcal{U} = \langle Q, \Sigma, \Gamma, \delta, \delta_{top}, q_0, Z_0, F\rangle$ be a preset pda recognizing L. The iterated (regular) substitution τ for L will have basic symbols of the form

$[p, A, B, q]$ $(p, q \in Q, A \in \Gamma, B \in \Gamma \cup \{\phi\})$

with the following semantics.

$[p, A, B, q]$ $(B \in \Gamma)$: when in state p with A under the pointer, (after a local move) it will write B and move upwards, later returning in state q (which will then continue on the B)

$[p, A, \phi, q]$: similarly, but (after a local move) it will empty the location and move upwards, later

sweeping down in state q.

Thus "variables" will represent recursions. The "terminal" symbols will actually be coded as $\bar{a}$, and later finish with the well-known cycle $\bar{a} \to a \to \not\in$ (where $\not\in \to \not\in$ and $\not\in$ outside the terminal range) to enforce synchronous termination. There is a problem there when subsequent expansions and δ_{top}-instructions are on λ-input, which in the iteration would be an untimely termination of that branch. However, the hyper-algebraic extension of regular languages is closed under arbitrary homomorphisms (from a more general result in [57]), and we may as well let the machine read \$ instead of λ all the time and erase it later from the language.

Replacement rules become

$[p, A, B, q] \to \{\bar{w}_1[p_1, A_1, B_1, q_1]\bar{w}_2 \ldots \bar{w}_k[p_k, A_k, \phi, q]\}$ |
starting in state p with A on the stack $\mathcal{U}$ will do a (local) computation on $\bar{w}_1$ input and recur in state p_1 writing BA_1 on the stack, then all possible strings w_i for local computation from q_{i-1} on B_{i-1} leading to p_i with A_i on the stack (note the final return state)} $\cup$

$\{\bar{w}_1[p_1, A_1, B_1, q_1]\bar{w}_2 \ldots \bar{w}_k[p_k, A_k, B_k, q_k]\bar{w}_{k+1}$ |
similarly, but now w_{k+1} also inducing a local computation from q_k on B_k emptying the location and returning in state q} $\cup$

$\{\bar{w}_1[p_1, A_1, q] \mid w_1$ inducing a local computation from p on A eventually leading to state p_1 and stacking $BA_1\}$.

It is straightforward that the (possibly infinite) set of strings by which [p, A, B, q] may be rewritten is a <u>regular</u> language. The construction goes through for $B = \phi$ as well. Note that by previous assumption now all $\bar{w}_i$'s are $\neq \lambda$.

The symbols [p, A, q] represent when δ_{top} has to be used.

The rules are

$[p, A, q] \rightarrow \{w \mid$ (unbarred this time) w induces a local computation with δ_{top} from p and A finally emptying the location and returning in state q (w is $\neq \lambda$ by assumption)$\}$

With appropriate start rules (a presetting 1 is to be treated separately, but there is closure under union) the generated iteration-language is exactly L.

It follows that preset pda languages form an AFL (even a full AFL). Later we will see when or when not you get an AFL with the restricted preset pda-models.

COROLLARY. Preset pda languages are indexed, and (hence) strictly included in the context-sensitive languages. (see [56])

4. DETERMINISTIC PRESET PDA'S

As opposed to classical machine-models this time it is not immediately clear when a preset pda should be called deterministic. Obviously δ and δ_{top} have to yield applicable instructions unambiguously and should be chosen as for deterministic pda's, i.e., with regards to the present modifications. But what about presetting? It can be shown that as far as computational power is concerned, the stack in preset pda's need never grow more than linear in the length of input strings and that would reasonably limit choices when we were to preset the stack functionally in the input (in global theory it may be different). We will argue that another form of "determinism" is more useful here, but illustrate it with an example first.

Example. A preset pda for $L = \{ww^R \mid w \in \Sigma^*\}$ would operate

as follows. It copies input in the stack until the presetting is reached, then with δ_{top} it changes mode and removes symbols from the stack while checking against input.

It is important to observe that for any word $ww^R \in L$ there can be only one presetting of the given machine which would lead to acceptance. In fact, the machine for $\{a^{n^2} \mid n \geq 1\}$ designed before had the same feature and was structurally strong enough to "enforce" the appropriate presetting of the stack.

Definition. L is called a weakly deterministic preset pda language if there is a preset pda $\mathcal{U}$ with deterministic δ and δ_{top} accepting L such that $\forall_{w \in L} \exists_{!\ n} (q_0, w, Z_0, n) \overset{*}{\vdash} (q, \lambda, \gamma, n)$ for some $q \in F$, with the preset maximum location reached at least once.

All deterministic context-free languages are weakly deterministic preset pda, but we get many more.

The next result is a generalization of a classical theorem on deterministic pda's.

THEOREM. The family of weakly deterministic preset pda languages is closed under complementation.

The main step in the proof is to eliminate non-terminating computations on λ-input at run-time (there can be no pre-calculation as is the classical proof for ordinary deterministic pda's). Tables stored in all locations to record past symbol-state configurations at that location are used to check for periodicities.

5. LOCALLY FINITENESS AND FINITE RETURNING

Returning to the original motivation, preset pda's as formalized so far may not yet be what computer scientists would call "practical". Further restrictions that are necessary were extensively discussed in [55], and basically relate to the original idea that in the stack-locations procedure-bodies would be stored. Implementation leads to bounds on size and, in slightly informal terminology, we would like preset pda's to have the following properties:

locally finiteness - a fixed bound on the length of local computations, i.e., with non-moving pointer.

finite returning - a fixed bound on how many recursions there can be from a base-location.

Here is the relation to developmental systems.

THEOREM. The family of languages defined by locally finite preset pda's which have the finite return property coincides with the family CR0L (or E0L).

The proof is in [55] but follows the same lines as the general result in section 3.

This representation for CR0L-languages is very powerful, and of interest also from a schematologist's point of view.

Finite returning imposed alone would permit machines to do arbitrary long calculations on the same location.

THEOREM. (Representation theorem) L can be accepted with a preset pda that has the finite return property if and only if there is a λ-free regular substitution τ and a language $L' \in$ CR0L such that $L = \tau(L')$.

The virtue of this theorem, that it directly uses the machine-representation and the previous result, contrasts with more abstract approaches.

In particular, the following results can now be derived very easily. They were independently also given in [7] in an entirely different approach.

THEOREM. The least AFL enclosing CROL at the same time is the least such full AFL.

THEOREM. (Representation theorem) L belongs to the least full AFL enclosing CROL if and only if $L = \tau(L')$ for some $L' \in$ CROL and λ-free regular substitution τ.

The results were found through the machine-approach.