The Lattice of Automata Classes: On "Mild" Extensions of Pushdown Automata

Christian Wurm

Fakultät für Linguistik und Literaturwissenschaften, Universität Bielefeld
cwurm@uni-bielefeld.de

Abstract
We introduce the lattice of automata classes, which is defined through (a closure of) transition relations of automata. In particular, we are interested how this lattice precisely relates to recognizing power of automata. We scrutinize this question for a small sublattice, which can be said to be the sublattice of mild extensions of pushdown automata.

1. Introduction

We introduce the lattice of automata classes, as defined by (a closure of) classes of relations. The main question on this lattice is: how does this classification of automata relate to the classification in terms of languages recognized? We study a small sublattice, which is bounded below by the class of pushdown automata (PDA), and topped by the class of automata with computable transition relations TM, which is equivalent to the class of Turing machines in terms of languages recognized (henceforth: *language equivalent*). Our basic result is, that the join of the smallest known proper extensions of PDA known to us is already language equivalent to TM. This sublattice modulo language equivalence has thus a quite simple structure.

The two extensions we consider are firstly embedded pushdown automata (EPDA, [10]) and secondly 2 stack automata (2SA, [1]). Both classes recognize classes of mildly context-sensitive languages (see [4]), and give rise to proper infinite hierarchies within this family (see [9]). The main motivation for both kinds of automata comes from linguistics: natural languages are now generally considered to be not contained in the class of context-free languages,[1] but in a class which is only "slightly larger"; the class of mildly context-sensitive languages (or rather: grammar formalisms) has been introduced as an attempts to make this intuitive notion precise.[2]

We then present the join of EPDA and 2SA in the automata lattice; we call this class extended pushdown automata (ExPDA). This class still is a mild extension of PDA, in the sense that its memory device is a stack of stacks, on which one can put a constantly bounded amount

[1]This claim was already made by Chomsky in the 1960s, but remained controversial until the work of [7]

[2]Actually, there are various usages of the term mildly context-sensitive. Originally it was applied to classes of grammar formalisms and the classes of languages they generate; as a consequence, it can also be applied to language immediately (namely to the languages in a class of mildly context-sensitive languages), and so the terms also describes a class of languages.

of symbols in each move, and from which one can only remove the uppermost symbol of the uppermost stack in a move. We show that nonetheless ExPDA recognize any recursively enumerable language, and consequently, all its important decision problems are undecidable. This is an interesting result, as this model does not have any parallel stack manipulation. This gives us an important insight into a fragment of the lattice of automata: whereas we have some infinite ascending chains, which meet at the class of PDA, already the join of the two smallest extensions known to us is Turing equivalent.

2. Pushdown Automata and Extensions

2.1. Pushdown Automata

A pushdown automaton (PDA) P is a tuple $\langle \Sigma, \Gamma, Q, q_0, F, \delta \rangle$, where Σ is a finite input alphabet, Γ is a finite stack alphabet, Q is a finite state set, $q_0 \in Q$ is the initial state, $F \subseteq Q$ a set of accepting states, and $\delta \subseteq \Sigma \times Q \times \Gamma \times \Gamma^* \times Q$ is the transition relation, which maps an input letter (or ϵ), a state and a stack symbol onto a string of stack symbols and a new state. We write stacks as growing from left to right; so all changes of stack configurations are of the form $\dagger\vec{x}\gamma \to \dagger\vec{x}\vec{y}$, for $\gamma \in \Gamma, \vec{x}, \vec{y} \in \Gamma^*$, and $\dagger \notin \Gamma$ is used to mark the bottom of the stack. In the sequel, we will often treat stacks (and extensions of stacks) as simple strings.

A configuration C of a PDA is an element of $Q \times \dagger(\Gamma^*) \times \Sigma^*$, where for $C = \langle q_i, \vec{x}, \vec{w} \rangle$, q_i is the current state of the automaton, $\vec{x}$ is the stack content, and $\vec{w}$ is the portion of the input word which is not read yet. We assume that pushdown automata read their input words from left to right. The initial configuration has the form $(q_0, \dagger, w)$ for q_0 the initial state, $\dagger$, the bottom symbol for stacks, marks the empty stack, and $w \in \Sigma^*$ is any input. Without loss of generality, and for all automaton models we will be considering here, we will assume that PDA accept words only if after reading the entire word their stack (or extended stack) is empty and if they are not in the initial state. We therefore have to assume (still without loss of generality), that q_0 cannot be reached by any transition. That is to say then, that $F = Q - \{q_0\}$. So, P accepts $\vec{w}$ if after reading $\vec{w}$ it is in configuration $\langle q, \dagger, \epsilon \rangle$ for any $q \in F$.

Transitions between configurations are defined as follows: we write $\langle q_i, \vec{x}\gamma, a\vec{w} \rangle \vdash_a \langle q_j, \vec{x}\vec{y}, \vec{w} \rangle$, iff $(\vec{y}, q_j) \in \delta(a, q_i, \gamma)$, and often simply write $\vdash$ instead of $\vdash_a$. $\vdash^*$ (or $\vdash^*_{\vec{w}}$) is the reflexive and transitive closure of $\vdash$. So δ gives rise to a relation $\vdash$ between automata configurations. Note that once we have specified the relation between δ and $\vdash$ for an automaton class, we can use the two interchangeably. We will use this fact later on.

2.2. Embedded PDA

An **embedded pushdown automaton** (EPDA) is a PDA, which has as memory device a stack of stacks rather than a simple stack.[3] We can manipulate the uppermost stack as usual, but we can in addition push entire stacks below and above the uppermost stack. For this and the following automaton models, we will only define the transition relations, as for the rest they coincide with simple PDA.

Definition 1 *For* $q_i, q_j \in Q, a \in \Sigma \cup \epsilon, \vec{w} \in \Sigma^*, \alpha, \alpha_1, \alpha_2 \in (\dagger(\Gamma^+))^*, \ddagger, \dagger \notin \Gamma, Z \in \Gamma, \vec{\beta}, \vec{\gamma} \in \Gamma^*$, *an EPDA transition is defined as follows:*

1. $\langle q_i, \ddagger\alpha \dagger \vec{\beta}Z, a\vec{w}\rangle \vdash \langle q_j, \ddagger\alpha\alpha_1 \dagger \vec{\beta}\vec{\gamma}\alpha_2, \vec{w}\rangle$, *if* $(q_j, \alpha_1, \vec{\gamma}, \alpha_2) \in \delta(q_i, a, Z)$ *and* $\vec{\beta}\vec{\gamma} \neq \epsilon$.
2. $\langle q_i, \ddagger\alpha \dagger Z, a\vec{w}\rangle \vdash \langle q_j, \ddagger\alpha\alpha_1\alpha_2, \vec{w}\rangle$, *if* $(q_j, \alpha_1, \epsilon, \alpha_2) \in \delta(q_i, a, Z)$.

There is a natural generalization of this concept. EPDA of level 1 are simply PDA; EPDA as defined here are of level 2 (having stacks of stacks). One can generalize this to any level k, allowing stacks of stacks of stacks etc., and the resulting infinite hierarchy characterizes exactly Weir's control language hierarchy ([9]). As we will not go beyond level 2 in this paper, we do not make an explicit reference to this; by EPDA we mean: EPDA of level 2. Note that there is a new symbol $\ddagger$, to mark the bottom of the stack of stacks; the accepting configuration is thus $\langle q, \ddagger, \epsilon\rangle$ for $q \in Q - \{q_0\}$.

2.3. 2-stack Automata

A related concept are the so-called 2-stack automata (2SA, [1]). 2SA are language equivalent to EPDA of level 2.[4] A 2SA is a PDA with 2 stacks (this again can be generalized for any number k). One can push symbols on any of its stacks, but it is not allowed to pop a symbol from the second stack unless the first stack is empty: The definition is again similar to the definition of a simple PDA; all that differs is the transition relation, which is as follows:

Definition 2 *For* $q_i, q_j \in Q, a \in \Sigma \cup \epsilon, \vec{w} \in \Sigma^*, \ddagger, \dagger \notin \Gamma, Z \in \Gamma, \beta, \gamma, \alpha, \alpha_1 \in \Gamma^*$, *a 2SA transition is defined as:*

1. $\langle q_i, \ddagger\alpha \dagger \beta Z, a\vec{w}\rangle \vdash \langle q_j, \ddagger\alpha\alpha_1 \dagger \beta\gamma, \vec{w}\rangle$, *if* $(q_j, \alpha_1, \gamma) \in \delta(q_i, a, Z)$.
2. $\langle q_i, \ddagger\alpha Z \dagger \epsilon, a\vec{w}\rangle \vdash \langle q_j, \ddagger\alpha \dagger \epsilon, \vec{w}\rangle$, *if* $(q_j, \epsilon, \epsilon) \in \delta(q_i, a, Z)$.

A 2SA stack is thus a string $s \in \ddagger\Gamma^* \dagger \Gamma^*$; the accepting configuration is $\langle q, \ddagger\dagger, \epsilon\rangle$, $q \in Q - \{q_0\}$.[5]

[3] It is important to distinguish EPDA from higher-order PDA (HPDA), see [10] for a very concise description of the difference. The main difference is that the latter allow for the copying of entire stacks, whereas the former do not. See [5] for some examples of how the machine works.

[4] The question whether this also holds for higher levels, is to our knowledge not resolved yet.

[5] It is quite clear that this concept is very similar to EPDA: call the positions left to an occurrence of $\dagger$ **insertion points**. Both models have in common that we can delete a letter only on the right edge of the memory string, and that we can add something only on the right edge, or insert it in the rightmost insertion

3. The Lattice of Automata Classes

3.1. Computational Classes of Relations and Classes of Automata

We now introduce the **lattice of automata classes**. Call a **semiautomaton** an automaton without initial or accepting configurations specified. Semiautomata are characterized exhaustively by their transition relation: it specifies input alphabet, states, and possible automaton configurations.

As transition relation we do now not consider δ, but the relation of configurations it induces, which we denote by $\vdash$. We can say that a class of semiautomata $\mathcal{C}$ is uniquely specified by a class of relations $\mathcal{R}$: namely $\mathfrak{A} \in \mathcal{C}_{\mathcal{R}}$, if $\vdash_{\mathfrak{A}} \in \mathcal{R}$, where $\vdash_{\mathfrak{A}}$ is the transition relation of $\mathfrak{A}$. Note that this class of relations also specifies whether we allow for ϵ-transitions, as the letters are specified within it, and furthermore, it specifies whether the automaton is deterministic or not, in that it may be a class of functions or not.

What we are still lacking are the accepting and initial configurations. These again can be said to form a relation, which is obtained by Cartesian product: let I be the set of initial configurations, F be the set of final states configurations; then $I \times F$ is the **accepting relation**. So in order to define a class of automata, we need a class of transition relations, and a class of accepting relations. In general, one has to consider the two separately, however, in our case we can safely assume that the class of accepting relations is simply the class of transition relations.

This characterization of classes of automata in terms of classes of relations is yet unsatisfying in one regard: it might well be that two classes of relations do exactly the same job with respect to automata. Therefore, we introduce the notion of a **composite isomorphism**. We say a map i on sets is **pointwise**, if for two sets M, N, if $M \cap N \neq \emptyset$, then $i(M) \cap i(N) \neq \emptyset$.

Definition 3 *We say that $i : \Sigma^* \times \Sigma^* \to T^* \times T^*$ is a composite isomorphism, if for any set of relations $(R_i)_{i \in I}$, $R_i \subseteq \Sigma^* \times \Sigma^*$, $k \in \mathbb{N}$, we have (1) $h[R_{i_1} \circ R_{i_2} \circ \ldots \circ R_{i_k}] = i[R_{i_1}] \circ i[R_{i_2}] \circ \ldots \circ i[R_{i_k}]$, (2) i is a bijection, and (3) i is pointwise.*

Composite isomorphisms are a restricted class of isomorphisms over relation monoids, with the operation of composition, and the identity relation as neutral element, which maintain the non-emptiness of intersections of relations. Note that it already follows from the definition that $i[id] = id$, as $R \circ id = id \circ R = R$. Obviously, statements about automata classes as induced by classes of relations can be made only up to composite isomorphisms; this is taken for granted in the sequel, though we will not make it explicit for reasons of space. However, there is a further problem: even if there is no composite isomorphism, properly different classes of relations might give rise to *language equivalent* classes of automata: it is possible that we can simulate one relation with another only by introducing additional non-determinism or ϵ-transitions.

point. The only difference is that for the EPDA model, each time we insert something, we have to add at least one new insertion point. For the 2-stack model, on the other side, there is only one given insertion point, and no way to create a new one.

3.2. The Lattice of Automata Classes

Given that we identify classes of automata with classes of relations, it is now easy to define a lattice of automata in terms of union and intersection. Recall that $\mathcal{C}_{\mathcal{R}}$ is the class of automata with transition relation in $\mathcal{R}$.

Definition 4 *Given two classes $\mathcal{C}_{\mathcal{R}_1}$ and $\mathcal{C}_{\mathcal{R}_2}$, we define $\mathcal{C}_{\mathcal{R}_1} \vee \mathcal{C}_{\mathcal{R}_2} := \mathcal{C}_{\mathcal{R}_1 \cup \mathcal{R}_2}$, and $\mathcal{C}_{\mathcal{R}_1} \wedge \mathcal{C}_{\mathcal{R}_2} := \mathcal{C}_{\mathcal{R}_1 \cap \mathcal{R}_2}$.*

It is easy to see that this defines a bounded complete lattice $\langle \mathfrak{C}, \vee, \wedge, \bot, \top \rangle$, with the empty class as $\bot$, and the class of all relations as $\top$. This is still somewhat informal, as we have not specified the closure under composite isomorphism. But even if we take this into account here, this lattice still would underspecify the recognizing power of a class of automata: it might well be that different classes of relations give rise to language equivalent classes of automata.

Obviously, classes of automata are interesting (mostly) for the classes of languages they recognize. We write $L(\mathfrak{A})$ for the language an automaton $\mathfrak{A}$ recognizes. For a class of automata $\mathcal{C}$, we call $L(\mathcal{C})$ the class of languages L, such that there is an automaton $\mathfrak{A} \in \mathcal{C}$, such that $L(\mathfrak{A}) = L$. So two classes $\mathcal{C}_1, \mathcal{C}_2$ are language equivalent, in symbols, $\mathcal{C}_1 \sim \mathcal{C}_2$, if $L(\mathcal{C}_1) = L(\mathcal{C}_2)$. A question which immediately arises is the following: how complex is the lattice of automata modulo language equivalence, that is, the structure $\langle [\mathfrak{C}]_{\sim}, \vee, \wedge, \bot, \top \rangle$? We will give a partial answer to this question for a small sublattice, the lower bound of which is provided by pushdown automata, the upper bound, as we will see, by the class of computable relations. As we have said, this sublattice has gained considerable attention in the literature, mainly from a linguistic point of view.

4. The Sublattice Bounded by PDA and TM

4.1. The Lattice

What we already know is that there are several infinite ascending chains in this sublattice; we restrict ourselves to EPDA and 2SA, which are both smallest extensions of PDA, in the sense that there is no known class of automata larger than PDA and smaller than EPDA/2SA, which is not equivalent to one of the three. The meet of these different chains, that is, $EPDA \wedge 2SA$, is the class of PDA. The reason is that the †, or any other the distinguished symbol which regulates the "deep" push moves, in EPDA always has to be put into the deep stack, and in the 2SA, it must not be pushed at all. So the only push moves they have in common are those which only consist of shallow pushing.

A question which is much harder to answer is the following: what is the expressive power of $EPDA \vee 2SA$, the join of the two classes? We will answer this question here in the following way: let TM be the class of automata induced by the class of computable relations. Obviously, these are equivalent to Turing machines (this class is observed in [2]). We show that $(EPDA \vee 2SA) \sim TM$.

4.2. Extended Pushdown Automata

We now define the class $EPDA \vee 2SA$; we will call this automaton model **extended pushdown automaton** (ExPDA). An extended stack is still a stack of stacks, but we can either insert a new stack (of stacks) below the topmost stack, or we can write a simple string onto the stack below the topmost, or both. Again, for the formal definition we will restrict ourselves to level two, though it can be easily generalized. The transition relation is $\delta \subseteq \Sigma \times Q \times \Gamma \times (\dagger \cup \Gamma)^+ \times \Gamma \times Q$. Transitions are as follows:

Definition 5 *For $q_i, q_j \in Q, a \in \Sigma \cup \epsilon, \vec{w} \in \Sigma^*, \alpha, \alpha_1, \alpha_2 (\dagger \cup \Gamma)^+, Z \in \Gamma, \vec{\beta}, \vec{\gamma} \in \Gamma^*$, ExPDA transitions are defined as:*

1. *$\langle q_i, \ddagger\alpha \dagger \vec{\beta}Z, a\vec{w}\rangle \vdash \langle q_j, \ddagger\alpha\alpha_1 \dagger \vec{\beta}\vec{\gamma}\alpha_2, \vec{w}\rangle$, if $(q_j, \alpha_1, \vec{\gamma}, \alpha_2) \in \delta(q_i, a, Z)$ and $\vec{\beta}\vec{\gamma} \neq \epsilon$.*
2. *$\langle q_i, \ddagger\alpha \dagger^+ Z, a\vec{w}\rangle \vdash \langle q_j, \ddagger\alpha\alpha_1\alpha_2, \vec{w}\rangle$, if $(q_j, \alpha_1, \epsilon, \alpha_2) \in \delta(q_i, a, Z)$ and $\vec{\beta}\vec{\gamma} \neq \epsilon$.*[6]

We accept by $\langle q, \ddagger, \epsilon\rangle$, $q \in Q - \{q_0\}$. In this formulation we are slightly more general than $EPDA \vee 2SA$, because we can push a string in $(\Gamma^*)(\dagger(\dagger \cup \Gamma)^+)$ below a $\dagger$-symbol. It is however clear that we can simulate each such ExPDA transition by a 2SA transition followed by a EPDA transition, in the decomposition indicated above; and as we grant ourselves arbitrary ϵ-transitions, and are interested in language equivalence, this really makes no difference.

4.3. Conjunctive Pushdown Automata

A conjunctive pushdown automaton (CPDA) is obtained by the conjunction of two independent PDA. Independence means that the transition of one machine does not depend on the configuration of the other. Their conjunction is only relevant for the mode of acceptance: for acceptance, both stacks must be empty.

Definition 6 *A (2-)CPDA is a tuple $\langle \mathcal{P}, \mathcal{Q}\rangle$, where both $\mathcal{P}, \mathcal{Q}$ are PDA over the same input alphabet Σ. A configuration con of a CPDA is an element of $CON_{\mathcal{P}} \times CON_{\mathcal{Q}}$, where $CON_{\mathcal{P}}$ is the set of configurations of $\mathcal{P}$ and $CON_{\mathcal{Q}}$ is the set of configurations of $\mathcal{Q}$. Transitions between states are as follows, for $a \in \Sigma \cup \epsilon$: $(con_{\mathcal{P}}, con_{\mathcal{Q}}) \vdash_a (con'_{\mathcal{P}}, con'_{\mathcal{Q}})$ is a transition of a CPDA if $(con_{\mathcal{P}} \vdash_a con'_{\mathcal{P}})$ is a transition of $\mathcal{P}$ and $(con_{\mathcal{Q}} \vdash_a con'_{\mathcal{Q}})$ is a transition of $\mathcal{Q}$.*

A CPDA accepts a word $\vec{w}$ if after reading $\vec{w}$ it is in a configuration $\langle C_{\mathcal{P}}, C_{\mathcal{Q}}\rangle$, where $C_{\mathcal{P}}$ is an accepting configuration of $\mathcal{P}$ and $C_{\mathcal{Q}}$ is an accepting configuration of $\mathcal{Q}$. The following proposition is well known, and follows from a theorem of Hartmanis ([3]):

Theorem 7 *For any recursively enumerable language L, there is a CPDA C, such that $L = L(C)$.*

[6]Note that in our definition, any sequence $\dagger^+$ can arise in principle; but our treatment makes it equivalent to a single $\dagger$, when we pop.

This follows from the fact that the language of computations of any Turing machine is the intersection of two context-free languages. So all we have to do is to simulate everything but the initial input string by ϵ-transitions. So the independence condition is not restrictive in terms of languages recognized; CPDA can simulate Turing machines. So we know that $CPDA \sim TM$. We will show that $ExPDA \sim CPDA$.

5. Two Useful Normal Forms

Usually, stack manipulation consists in rewriting the uppermost (in our notation: rightmost) symbol into a possibly empty string of symbols: $\vec{y}x \to \vec{y}\vec{z}$, where $x \in \Gamma, \vec{y}, \vec{z} \in \Gamma^*$. **Stack normal form** means that the uppermost stack symbol is not rewritten, but either something is stacked on top of it, or it is popped:

Definition 8 *A PDA is in stack normal form (SNF) if its transitions are of the form:* $\langle q_i, \vec{\gamma}x, a\vec{w}\rangle \vdash \langle q_j, \vec{\gamma}x\vec{\gamma'}, q_j\rangle$, *or* $\langle q_i, \vec{\gamma}x, a\vec{w}\rangle \vdash \langle q_j, \vec{\gamma}, \vec{w}\rangle$, *for* $\vec{\gamma}, \vec{\gamma'} \in \Gamma^*$, $x \in \Gamma$, $a \in \Sigma \cup \epsilon$.

Lemma 5.1 *For every PDA $\mathcal{P}$, there is a PDA $\mathcal{P}'$ in stack normal form such that $L(\mathcal{P}) = L(\mathcal{P}')$*

We omit the simple proof. In the sequel, we will assume without loss of generality that our CPDA-components are in SNF. Furthermore, we can transform any PDA in a way that it makes transitions not for single input letters, but for pairs (or any number) of input letters. We omit the simple construction. Then, given a CPDA, we can transform its components asynchronously such that for each input letter, only a single stack is manipulated. We call this **alternating normal form** for CPDA (ANF). In the sequel, we will without loss of generality assume that our CPDA are in alternating normal form.

6. The Main Theorem

Theorem 9 *Any CPDA CP can be transformed into an ExPDA EP such that $L(CP) = L(EP)$.*

This is our main theorem; the rest of this section will be devoted to its proof.

Proof idea: we will translate each single CPDA transition into a *set* of ExPDA transitions. This resulting set of ExPDA configurations will have a certain property (we call it p-property) with respect to the CPDA transition; we make sure that this correlation holds for initial configurations, and show as inductive step that it is preserved for arbitrary CPDA transitions. The correlation roughly states that firstly, the symbols of each single CPDA-stack occur in the same order in the corresponding ExPDA stacks, and secondly, that we generate the set of *all* possible orders of elements of the two CPDA stacks in a single ExPDA stack which satisfies the first condition. This way, we can simulate two parallel (conjunctive) stacks within a (disjunctive) set of single extended stacks. Of course, we will need some auxiliary properties, which we describe in the sequel. In particular, this correlation makes sure that the CPDA accepts if and only if the ExPDA accepts.

6.1. Translation: Preliminaries

We call the CPDA CP, the ExPDA we construct EP. Let Γ be the set of stack symbols of CP. Assume without loss of generality that the set of symbols used for the first stack and the ones used for the second stack are pairwise disjunct. We thus have $\Gamma = \Gamma_1 \cup \Gamma_2$. We will denote symbols in Γ_1 with uppercase Latin letters, symbols in Γ_2 with lowercase Latin letters.

We call the stack alphabet for EP Δ, and we define it as follows: We put $\Delta_1 := \Gamma_1 \times (\Gamma_1 \cup \perp) \times \{1,2\}$, $\Delta_2 := \Gamma_2 \times (\Gamma_2 \cup \perp) \times \{1,2\}$. Put $\Delta = \Delta_1 \cup \Delta_2$. For brevity, we will say the symbols in Δ_1, Δ_2 have different **colors**. For all $\delta \in \Delta$, we will call $\pi_1(\delta)$, the first component, the **dominant symbol**. It makes up the correspondence between Γ and Δ. We use the second component to encode which is the next symbol below the current one of the same color. Therefore we need an additional symbol $\perp$ in the second component, which indicates that there is no symbol of the same color below the current symbol. The third component should indicate the color of the uppermost symbol below/left of the next $\dagger$ symbol. Note that this will not always be true, but is the intuitive motivation behind the additional components.

We form the state set of EP by Cartesian product of the state set of CP with some additional states which we will define in the sequel. We yield the transition function for states then in the manner well-known from finite automata. As the given states are thus handled easily, we suppress them in the sequel, and we only mention the new components. The given states are a component in the state transitions which will always be equal in δ_{CP} (which induces $\vdash$) and the translation in δ_{EP} (which induces $\Vdash$). We will also suppress the input associated with transitions, as it is supposed to remain invariant.

We now define the new state-components. Already by itself, they will form the Cartesian product of three components. This state set is a subset of $\Gamma_1 \times \Gamma_2 \times \{1,2\}$[7]. Intuitively speaking, they contain the following information: the first component tells us the dominant symbol of the next to top stack symbol in Δ_1; it is easy to see that we can memorize this for the case we are stacking. For the case we are popping, consider what is to follow. The second component encodes the dominant symbol of the next to top stack symbol in Δ_2. The third component tells us whether below the next $\dagger$ symbol is a symbol either in Δ_1 or Δ_2 (that is, its color). So in case we have a state (component) $(A, a, 1)$, we know the next to top symbol in Δ_1 has dominant symbol A, the next to top symbol in Δ_2 has dominant symbol a, and below the next $\dagger$ there is a symbol in Δ_1.

The reader will have noted that there is some redundancy in the information encoded in states and stack symbols. The reason is that each of them will encode the *correct* information only under certain circumstances. The most complicated part of the proof will consist in showing that, given the redundancy, we get the correct information under *all* circumstances. We will

[7]We hope the reader will not confuse this with Δ, the stack alphabet, which looks quite similar!

now partition the CP transitions into four subrelations, and show the translation for each class separately. Given a CPDA in ANF and with its stacks in SNF, we have four interesting kinds of transitions: (i) push the left stack (and leave the right one unchanged), (ii) push the right stack (and leave the left one unchanged), (iii) pop the left stack (and leave the right one unchanged), or (iv) pop the right stack (and leave the left one unchanged). There is also a fifth option: (v) leave both stacks unchanged. It is quite obvious how to translate it (the identity on CP stacks translates as the identity on the EP stack), so we will not consider this case in the sequel.

We will denote the transition relation of CP, induced by δ_{CP}, by $\vdash$, and take the standard convention to write $C_1 \vdash C_2$ instead of $(C_1, C_2) \in \vdash$. The partition we have described (informally) above gives rise to a partition of the relation $\vdash$ into $\{\vdash^{\alpha} \colon \alpha \in \{i, ii, iii, iv, v\}\}$. We will consider the four elements of the partition of $\vdash$ separately, and show how we can simulate each of them with a relation $\Vdash^{\alpha} \colon \alpha \in \{i, ii, iii, iv\}$, the union of which is the transition relation $\Vdash$ of the ExPDA EP, such that $L(EP) = L(CP)$.

We will use the following conventions: we use $\boxtimes$ as variable for one normal PDA stack. Accordingly, CPDA stacks will be of the form $(\boxtimes, \boxtimes)$. $\blacksquare$ is used as a variable for ExPDA-stacks, that is: $\blacksquare \in \ddagger(\dagger \cup \Delta)^*$. However, we generally designate the topmost part of ExPDA stacks of the form $\dagger\Delta^+$ by $\square$; that is, $\square \in \Delta^+$, and ExPDA stacks are represented as $\blacksquare\square$.

6.2. The p-property

Take two (ϵ-)homomorphisms $\phi_1 : \Delta \cup \{\dagger, \ddagger\} \to \Gamma_1 \cup \{\dagger\}$, $\phi_2 : \Delta \cup \{\dagger, \ddagger\} \to \Gamma_2 \cup \{\dagger\}$, where $\phi_i(\ddagger) = \dagger$, $\phi_i(\alpha) = \epsilon$ if $\alpha \notin \Delta_i \cup \{\ddagger\}$, (which also includes $\dagger$), and $\phi_i((\gamma_i, \gamma'_j, k)) = \gamma_i$; that is, ϕ_i maps symbols in Δ_i onto their dominant symbol.

We consider ExPDA stacks as strings, and extend ϕ_1, ϕ_2 to strings in the usual fashion. We then have for $\blacksquare\square$ a stack configuration of EP, $(\boxtimes_1, \boxtimes_2)$ a stack configuration of CP, $\phi_1(\blacksquare\square) = \boxtimes_1$, $\phi_2(\blacksquare\square) = \boxtimes_2$. We build the product map $\langle\phi_1, \phi_2\rangle$, where $\langle\phi_1, \phi_2\rangle(\blacksquare\square) = (\boxtimes_1, \boxtimes_2)$; we have thus a map from EP stack configurations to CP stack configurations. In the sequel, we will consider the inverse map $\langle\phi_1, \phi_2\rangle^{-1}$, where $\langle\phi_1, \phi_2\rangle^{-1}(\boxtimes_1, \boxtimes_2) := \{\blacksquare\square : \phi_1(\blacksquare\square) = \boxtimes_1, \phi_2(\blacksquare\square) = \boxtimes_2\}$.

Next, define a rational map χ mapping stacks (as strings) onto stacks, which (1) inserts $\dagger$ between any two adjacent symbols $\delta_1 \in \Delta_1, \delta_2 \in \Delta_2$, and maps any sequence $\alpha \in \dagger^+$ onto $\dagger$. For the rest, χ computes the identity. We will use χ mostly on sets instead of strings, where it is defined pointwise.

Given a (single!) CP stack configuration C of the form $(\boxtimes_1, \boxtimes_2)$, we say that a *set* of EP stack configurations $\mathcal{E}$ has the p-property with respect to C, if $\mathcal{E} = \chi(\langle\phi_1, \phi_2\rangle^{-1}(C))$ (note that in $\langle\phi_1, \phi_2\rangle^{-1}(C)$, we have $\dagger$ at all possible positions, as it is the maximal preimage, and $\phi_1(\dagger) = \phi_2(\dagger) = \epsilon$). We also say C and $\mathcal{E}$ are p-congruent, and write $\mathcal{E} \cong_p C$. Note that the conditions are unique, that is, for each CP-configuration C, there is a exactly one set $\mathcal{E}$ such that $C \cong_p \mathcal{E}$, as χ is a function.

6.3. The Crucial Lemma

We now present a translation of single CP transitions to sets of EP transitions. We will show inductively that this translation preserves p-congruence between the resulting stack configurations. The initial EP-configuration has the p-property wrt. the initial CP-configuration; furthermore, our translation of $\vdash$ to $\Vdash$ preserves the property from a configuration and set of configurations to its successor and set of successors, respectively.

6.3.1. The Base Case

Given that we omit input and CP state as invariant under translation, the initial stack configuration of CP is $C_1 := (\dagger, \dagger)$. The initial stack configuration of EP is $\{E_1\} = \mathcal{E}_1 := \{\ddagger\dagger\}$. It is easy to check that $\mathcal{E}_1$ has the p-property wrt C_1.

6.3.2. Induction Step

We now take care of the possible transitions. We need to consider the four cases of non-trivial transitions we described above. Recall that uppercase letters $(X, Y, Z, ...)$ are used for Γ_1, lowercase letters $(x, y, z, ...)$ for letters in Γ_2.

6.3.3. (i) $\vdash^i \to \Vdash^i$

To translate a single push transition of CP, we have to distinguish various cases, depending on the uppermost symbol of the EP-stack and current state of EP. And even then, we need a set of successors rather than a single one. Note that, as we said above, we suppress the CP states and the input word; both are understood to be invariant under the translation. But contrary to CP configurations, EP configurations are still written as pairs of stacks and states. So the following case distinction is over uppermost symbol of ExPDA stack and current ExPDA state. So, if $(\boxtimes_1, \boxtimes_2) \vdash^i (\boxtimes_1 X, \boxtimes_2)$, then for $\blacksquare\square \in \mathcal{E}$, $\square = \square'\delta$ for some $\delta \in \Delta$,

a) $\blacksquare\square'(Y, Z, j), (Y, x, j) \Vdash^i \{(\blacksquare\square \dagger (X, Y, 1), (X, x, 1)),$
$(\blacksquare\square(X, Y, j) \times (X, x, j))$: for $j \in \{1, 2\}\}$

b)i. $\blacksquare\square'(y, z, 1), (Y, y, 1) \Vdash^i \{\blacksquare\square \dagger (X, Y, 2), (X, y, 2),$
$\blacksquare \dagger (X, Y, 1)\square, (X, y, 1), \blacksquare(X, Y, 1)\square, (X, y, 1), \}$.

b)ii. $\blacksquare\square'(y, z, 2), (Y, y, 2) \Vdash^i \{(\blacksquare\square \dagger (X, Y, 2), (X, y, 2)),$
$(\blacksquare \dagger (X, Y, 2)\square'(y, z, 1), (X, y, 1))\}$.

c)i. $\blacksquare\square'(y, z, 2), (Y, y, 1) \Vdash^i \{(\blacksquare\square'(y, z, 1) \dagger (X, Y, 2), (X, y, 2)),$
$(\blacksquare \dagger (X, Y, 1)\square'(y, z, 1), (X, y, 1))$, $(\blacksquare(X, Y, 1)\square'(y, z, 1), (X, y, 1))\}$.

c)ii. $\blacksquare\square'(Y,Z,1),(Y,y,2) \Vdash^i \{(\blacksquare\square'(Y,Z,2)\dagger(X,Y,2),(X,y,2)),$
$(\blacksquare\dagger(X,Y,2)\square'(y,z,1),(X,y,1))\}$.

d)i. $\blacksquare\square'(Y,Z,2),(X,y,1) \Vdash^i \{(\blacksquare\square'(Y,Z,2)\dagger(X,Y,1),(X,x,1)),$
$(\blacksquare\square'(Y,Z,2)(X,Y,2),(X,x,2))\}$.

d)ii. $\blacksquare\square'(y,z,1),(Y,y,2) \Vdash^i \{(\blacksquare\square'(y,z,1)\dagger(X,Y,2),(X,y,2)),$
$(\blacksquare\dagger(X,Y,1)\square'(y,z,1),(X,y,1)),(\blacksquare(X,Y,1)\square'(y,z,1),(X,y,1)),\}$.

Note that the left hand sides of c) and d) are configurations which do not arise by our push moves. We will need them only later, since they might result from pop-transitions. As they form the most problematic cases, we will also explain them in more detail later on. We can see that the resulting ExPDA configurations allow us "reconstruct" the CPDA-configuration, because the EP states and/or stacks always "know" the uppermost letter of each color, and each symbol in Δ_i tells us what the next dominant letter in Δ_i is below it, by its second component. It is this latter property that we need in order to preserve the former when translating pop moves. We call the (sets of) configurations resulting from the transitions $\mathcal{E}^{\Vdash^{(i)}}$ and $C^{\vdash^{(i)}}$. The following lemma shows the first part of the induction step.

Lemma 6.1 *If $C \cong_p \mathcal{E}$, then $C^{\vdash^{(i)}} \cong_p \mathcal{E}^{\Vdash^{(i)}}$.*

Proof. 1. if $(\blacksquare\square,q) \in \mathcal{E}^{\Vdash^{(i)}}$ for some state q, then χ computes the identity on $\blacksquare\square$. We easily check that we create no multiple $\dagger$ sequences. Furthermore, for the "shallow" push moves we easily see that we always insert a $\dagger$ between symbols of different colors. To see this for the deep push moves, consider the following: the stack suffix $\square$ has only one color by assumption. We push a symbol δ deeply, only if δ has a different color from the last symbol of $\square$; and so it follows that it has a different color from the leftmost symbol in Δ of $\square$. When we insert the symbol there, there is a $\dagger$ which separates them, as the leftmost symbol of $\square$ is $\dagger$. For the left edge of the inserted symbol, we have a control mechanism: the third projection of the topmost sign, and the third projection of the state. For the configurations which emerge from pushing, which are the left hand sides of cases a) and b), it is easy that both state and stack correctly predict the same color of the next symbol below $\dagger$; and so we know whether we have to insert a $\dagger$ below, or create both possibilities. The cases c) and d), which arise from pop-moves are more complicated, and will discussed below. This does not affect the current part of the proof, as their left-hand sides do not emerge anyway.

2. If $(\blacksquare\square,q) \in \mathcal{E}^{\Vdash^{(i)}}$ for some q, then $\blacksquare\square \in \chi(\langle\phi_1,\phi_2\rangle^{-1}(C^{\vdash^i}))$. It now suffices to show that if $(\blacksquare\square,q) \in \mathcal{E}^{\Vdash^{(i)}}$ for some q, then $\phi_1(\blacksquare\square) = \boxtimes_1$, and $\phi_2(\blacksquare\square) = \boxtimes_2$ This is immediate for all transitions which only perform "shallow" push moves, as ϕ_i are string homomorphisms, and we only add a suffix letter. Regarding the "deep" push transitions: by induction hypothesis, whenever we have a change of color of the stack, there is a $\dagger$ in between. We only perform deep pushes if the top symbol is different in color from the symbol we push. Therefore, moving downward, the next symbol of the same color (as the pushed one) has to be preceded by a $\dagger$ symbol, where the deep push will be inserted. Therefore, we do not change the order of symbols of the same color, so the homomorphism condition remains satisfied.

3. If $\blacksquare\square \in \chi(\langle\phi_1,\phi_2\rangle^{-1}(C^{\vdash^i}))$, then $(\blacksquare\square, q) \in \mathcal{E}^{\Vdash^{(i)}}))$ for some q.

We now only have to check maximality of the pre-image with respect to symbols in Δ: define a *legitimate insertion point* for a symbol of color i into a stack $\blacksquare\square$ to be each position right of the rightmost symbol of color i in $\blacksquare\square$. From the premiss it follows, that for all legitimate insertion points of $\blacksquare\square$, there is an $\blacksquare\square' \in \mathcal{E}$ such that this legitimate insertion point actually *is* an insertion point, in the sense that there is the rightmost $\dagger$. This is because $\mathcal{E}$ is the maximal pre-image of $\langle\phi_1,\phi_2\rangle$ (modulo χ), which maps $\dagger$ to ϵ. We call this property $\dagger$-*completeness.* This obviously makes sure that we insert on all legitimate insertion points.

So it follows that $\chi^{-1}\{\blacksquare\square : (\blacksquare\square, q) \in \mathcal{E}^{\Vdash^i})\} \supseteq \langle\phi_1,\phi_2\rangle^{-1}(C^{\vdash^i})$, because for each legitimate insertion point, we insert the new symbol in some stack in $\mathcal{E}$, and χ^{-1} creates the maximal preimage with respect to multiple $\dagger$ symbols. But as we have said above, we have $\chi(\mathcal{E}^{\Vdash^i}) = \mathcal{E}^{\Vdash^i}$. Consequently, as χ is a map which preserves inclusion of sets, we have $\chi(\chi^{-1}\chi((\mathcal{E}^{\Vdash^i}))) = \mathcal{E}^{\Vdash^i} \supseteq \chi(\langle\phi_1,\phi_2\rangle^{-1}(C^{\vdash^i}))$. □

6.3.4. (ii) $\vdash^{(ii)} \to \Vdash^{(ii)}$

The translation from $\vdash^{(ii)}$ to $\Vdash^{(ii)}$ is completely parallel, therefore we will not discuss it.

6.3.5. (iii) $\vdash^{(iii)} \to \Vdash^{(iii)}$

We start with some auxiliary notions for the configurations which arise from our translation so far: for each *EP*-configuration resulting from the above translation, (*property 1*) the third state projection correctly tells us the color of the next symbol under the rightmost $\dagger$; and (*property 2)* in any *EP*-stack resulting from push moves, if there is a sequence $\delta_i \dagger \Delta^+\delta_j\dagger$, $\delta_i \in \Delta_i$, then $\pi_3(\delta_j) = i$. That is, the uppermost symbol of a $\dagger$-free substack will correctly indicate the color of the next symbol left of the next $\dagger$ to its left. Note that it will only be the rightmost symbol of the substack which tells us correctly for sure: because the color of the symbol left of $\dagger$ can change via a deep push, and the symbols in between cannot be manipulated any more. The rightmost will encode the correct information, because after we write a $\dagger$ to its right, we cannot push below the $\dagger$ to its left anymore, and therefore not change the color of the symbol.

Now the translation:

If $(\boxtimes_1 X, \boxtimes_2) \vdash^{iii} (\boxtimes_1, \boxtimes_2)$, then for $\blacksquare\square \in \mathcal{E}$, $\square = \square'\delta$ for some $\delta \in \Delta$,

a) $(\blacksquare\square'(X,Y,1)), (X,y,1) \Vdash^{(iii)} \{\blacksquare\square, (Y,y,1)\}$;

b) $(\blacksquare\square'(X,Y,2)), (X,y,2) \Vdash^{(iii)} \{\blacksquare\square, (Y,y,2)\}$;

c) $(\blacksquare\square'(X,Y,1)), (X,y,2) \Vdash^{(iii)} \{\blacksquare\square, (Y,y,2)\}$;

d) $(\blacksquare\square'(X,Y,2)), (X,y,1) \Vdash^{(iii)} \{\blacksquare\square, (Y,y,2)\}$.

For all other EP configurations, the transition will be undefined. After a $\Vdash^{iii}$ transition, the state still correctly encodes the uppermost dominant symbol of each color, and each letter in in Δ_i tells us what is the next dominant letter in Δ_i below is. Therefore, we always know which transition we can apply. Note that there is an asymmetry: whereas case a) and b) are completely symmetric, c) and d) are not: in c), the third projection of the state has "precedence" over the stack symbol, in d), it is the third projection of the stack symbol which has precedence. The same was the case for c) and d) in the $\Vdash^{i}$ transitions.

The reason is as follows: For the configurations resulting from a push move, the states correctly predict the color below the next † symbol. Conversely, when we perform pop moves, this is no longer the case: because if we pop a symbol right on top of the †, we also pop the †; but then the state will no longer correctly predict the next color below the next †, whereas, by property 2, the next stack symbol (which was below the †) will do so, by our above considerations. Now the problem is: *we do not know when we pop a † symbol*; our transitions are blind to this. So we know that either state or stack symbol makes the correct prediction with its third projection, but there is no simple way to know which one of the two, in case they contradict each other. That there still is a way is shown by the following:

Lemma 6.2 *Assume that uppermost symbol δ and current state q make different predictions on the color below the next †; that is, $\pi_3(\delta) \neq \pi_3(q)$. Then the uppermost stack symbol correctly predicts the color below † if and only if the stack symbol is of the color the state predicts (and so the stack symbol predicts a color different from his own). More formally: in case where $\pi_3(\delta) = i$, $\pi_3(q) = j$, $i \neq j$, the symbol below the next † has color i, if and only if $\delta \in \Delta_j$.*

Proof. *If* We can readily check that this configuration can only emerge from a pop-move, where in addition a † has been removed; because there is no other move which would yield this result. So we can presuppose that the last move was a pop-move removing a †. So if there is an uppermost symbol $\delta \in \Delta_i$, which predicts a symbol $\delta' \in \Delta_j$ below the next †, (i.e. $\pi_3(\delta) = j$), where $j \neq i$, then it always predicts it correctly, as we have explained above. This is because we can change the color under the next to top † only from being the same as top symbol to being different, not vice versa (check $\Vdash^{i}$).

Only if: assume for the uppermost stack symbol δ, current state q, $\pi_3(q) = i$, $\delta \in \Delta_j$ for $i \neq j$, and $\pi_3(\delta) \neq \pi_3(q)$. Then consequently δ predicts its own color, and it might be wrong. On the other side, the state cannot be wrong, as this is only possible if we popped a †. But if there would have been a † on top of the uppermost symbol, which was deleted by the last push move, then we would have $\delta \in \Delta_i$ for $i = \pi_3(q)$, that is, it would have been of the color predicted by the state. This is because then the symbol δ', which we popped in the last move, knew the color of δ ($\pi_3(\delta') = i$), for $\delta \in \Delta_i$ by the above considerations; and furthermore, in our pop moves, the state takes over its third projection from the symbol that is popped, and so we have $\pi_3(q) = \pi_3(\delta')$. Therefore, the δ must make the wrong prediction in its third projection. □

This means: we always can read of a configuration whether the state or the stack makes the correct prediction with its third component, in the problematic case they make different predictions. We also easily see that the two auxiliary properties we introduced above are preserved in transitions: (*Property 1*): by our transitions, if the topmost symbol is in Δ_1, and the third state projection is '2', then still the next symbol under the next † is in Δ_2. If this property was given before, our transitions preserve it. (*Property 2*) This follows from the above lemma. It might well be that the uppermost symbol of the stack makes a wrong prediction; but this is recognized and remedied as soon as we push a symbol on top, by the push transitions c)i. and c)ii. of $\Vdash^{(i)}$. So below a †, we still make the right prediction.

We now check the preservation of p-congruence:

Lemma 6.3 *If $C \cong_p \mathcal{E}$, then $C^{\vdash^{(iii)}} \cong_p \mathcal{E}^{\Vdash^{(iii)}}$.*

Proof. Again, we simply check the conditions.

1. if $(\blacksquare\square, q) \in \mathcal{E}^{\Vdash^{(iii)}}$ for some state q, then χ computes the identity on $\blacksquare\square$. This is easy to check in this case.

2. If $(\blacksquare\square, q) \in \mathcal{E}^{\Vdash^{(iii)}}$ for some q, then $\blacksquare\square \in (C^{\vdash^{iii}})$. It is clear that $\phi_1(\blacksquare\square) = \boxtimes_1$, $\phi_2(\blacksquare\square) = \boxtimes_2$ if the premise is fulfilled, as all we did was take away an element.

3. If $\blacksquare\square \in \chi(\langle\phi_1, \phi_2\rangle^{-1}(C^{\vdash^i}))$, then $(\blacksquare\square, q) \in \mathcal{E}^{\Vdash^{(i)}}$ for some q. If $\mathcal{E}$ fulfills the induction premiss with respect to C, then $\mathcal{E}^{\Vdash^{(iii)}}$ fulfills it with respect to $C^{\vdash^{(iii)}}$: for $C = (\boxtimes_1 X, \boxtimes_2)$, $C' = (\boxtimes_1, \boxtimes_2)$, $\chi(\langle\phi_1, \phi_2\rangle^{-1}(C'))$ is precisely the set of all $\{E : E(X, y, i) \in \chi(\langle\phi_1, \phi_2\rangle^{-1}(C))$ for some $y, i\}$, because both ϕ_i are string homomorphisms. Therefore, the claim follows. □

6.3.6. (iv) $\vdash^{(iv)} \rightarrow \Vdash^{(iv)}$

Translation of $\vdash^{(iv)}$ is symmetrical to the previous case and is therefore omitted.

This almost completes the proof. We still need a simple lemma. Call the accepting configurations of EP F_{EP}; the accepting configurations of CP F_{CP}. Recall also that in our translation from CP to EP, we let each transition be induced by the same input letter. We easily get the following:

Lemma 6.4 *If $\mathcal{E} \cong_p C$, then $C \in F_{CP}$ if and only if $\mathcal{E} \cap F_{EP} \neq \emptyset$.*

Proof. $\langle\phi_1, \phi_2\rangle(\blacksquare\square) = (\dagger, \dagger)$ if and only if $\blacksquare\square \in \ddagger\dagger^*$; and $\chi(\{\ddagger\dagger^*\}) = \{\ddagger\dagger, \ddagger\}$. This clearly intersects with the empty stack configuration of EP which is ‡. As both automata accept by empty stack and non-initial state, this proves the claim. □

This completes the main theorem: both classes of automata recognize the same class of languages. As we have stated above, $L(CPDA)$ is actually the class of recursively enumerable languages. So we have the following corollary.

Corollary 10 *For every recursively enumerable language L, we can construct an ExPDA EP such that $L = L(EP)$.*

From this result, we see that it must be possible to simulate a Turing machine tape with the extended stack of an ExPDA. It is not quite obvious how this can be done directly, however, our main theorem shows an indirect way of construction.

7. Conclusion

One main field of application for "mild" extensions of PDA is linguistic theory, where they are used to characterize the important class of mildly context-sensitive languages. In some regards, there is however still an unsatisfactory situation in that there is no automata-theoretic characterization for important classes of languages, as the class generated by multiple context-free grammars (or equivalent formalisms, [6]), which are among the most important ones for linguistic theory (see [5] for a survey of grammars and automata). Therefore, one should still keep looking for new extensions of PDA.[8] We have therefore taken a closer look at the sublattice of automata classes, modulo language equivalence, bounded by PDA and Turing machines. We have seen that its structure is quite simple, for the classes that are known: there are several infinite ascending chains: n-EPDA, nSA, and furthermore the hierarchy of higher order PDA, which we did not consider here. However, already the join of the two smallest extensions known to us comes with so much additional power, that the resulting class of automata is language equivalent to the class TM of Turing machines.

On the negative side, this shows that $ExPDA$, the class we introduced here as the join of EPDA and $2SA$, is not an interesting model to consider: all its important decision problems are undecidable. On the positive side, we see that the sublattice we considered is quite simple in its structure; and if we want to find new interesting classes therein, we have to be very careful to small details: on the first glimpse, ExPDA seem almost identical to EPDA. So our results might help to see the boundary regions of mild context-sensitivity more concisely (given that there are competing and rather approximate definitions), and shed some new light on the question: what is it that makes an automaton recognize a mildly context-sensitive language?

References

[1] Tilman Becker. A new automaton model for tags: 2-SA. *Computational Intelligence*, 10:422–430, 1994.

[2] Bruno Courcelle and Irène Durand. Fly-automata, their properties and applications. In Béatrice Bouchou-Markhoff, Pascal Caron, Jean-Marc Champarnaud, and Denis Maurel, editors, *CIAA*, volume 6807 of *Lecture Notes in Computer Science*, pages 264–272. Springer, 2011.

[8] Actually, there are some classes more powerful than EPDA which we have not considered here: for example thread automata, see [8].

[3] Juris Hartmanis. Context-free languages and Turing machine computations. In Jacob T. Schwartz, editor, *Mathematical Aspects of Computer Science*, Proceedings of symposia in applied mathematics 19, pages 42–51. 1967.

[4] Aravind K. Joshi, K. Vijay-Shanker, and David Weir. The convergence of Mildly Context–sensitive grammar formalisms. In Peter Sells, Stuart M. Shieber, and Thomas Wasow, editors, *Foundational Issues in Natural Language*, pages 31–81. MIT Press, Cambridge (Mass.), 1991.

[5] Laura Kallmeyer. *Parsing Beyond Context-Free Grammars.* Cognitive Technologies. Springer, 2010.

[6] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On Multiple Context–free Grammars. *Theor. Comp. Sci.*, 88:191–229, 1991.

[7] Stuart Shieber. Evidence against the context–freeness of natural languages. *Linguistics and Philosophy*, 8:333–343, 1985.

[8] Eric Villemonte de la Clergerie. Parsing MCS languages with Thread Automata. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+6),* Venezia, 2002.

[9] David J. Weir. A geometric hierarchy beyond context-free languages. *Theor. Comput. Sci.*, 104(2):235–261, 1992.

[10] David J. Weir. Linear iterated pushdowns. *Computational Intelligence*, 10:431–439, 1994.