Automata and Logic for Floyd Languages

Violetta Lonati[1], Dino Mandrioli[2], Matteo Pradella[2]

[1] DSI - Università degli Studi di Milano, via Comelico 39/41, Milano, Italy
lonati@dsi.unimi.it
[2] DEI - Politecnico di Milano, via Ponzio 34/5, Milano, Italy
{dino.mandrioli, matteo.pradella}@polimi.it

Floyd languages (FL), as we renamed Operator Precedence Languages after their inventor, were originally introduced to support deterministic parsing of programming and other artificial languages [1]; then, interest in them decayed for several decades, probably due to the advent of more expressive grammars, such as LR ones [2] which also allow for efficient deterministic parsing.

In another context Visual Pushdown Languages (VPL) have been introduced and investigated [3] with the main motivation to extend to them the same or similar automatic analysis techniques -noticeably, model checking- that have been so successful for regular languages. Recently we discovered that VPL are a proper subclass of FL, which in turn enjoy the same properties that make regular and VP languages amenable to extend to them typical model checking techniques; in fact, to the best of our knowledge, FL are the largest family closed w.r.t. Boolean operation, concatenation, Kleene * and other classical operations [4]. Another relevant feature of FL is their "locality property", i.e., the fact that partial strings can be parsed independently of the context in which they occur within a whole string. This enables more effective parallel and incremental parsing techniques than for other deterministic languages.

Originally, Floyd languages were defined in terms of grammars. In this work we present an appropriate automata family that recognizes exactly FL [5], together with a complete characterization of FL in terms of a suitable Monadic Second-Order (MSO) logic [6]. In this way, as well as with regular and VP languages, one can, for instance, state a language property by means of a MSO formula, then automatically verify whether a given FA accepts a language that enjoys that property.

Operator precedence alphabet and chains

Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet. The empty string is denoted ϵ. We use a special symbol # not in Σ to mark the beginning and the end of any string. This is consistent with the typical operator parsing technique that requires the look-back and look-ahead of one character to determine the next parsing action [2].

An *operator precedence matrix* (OPM) M over an alphabet Σ is a partial function $(\Sigma \cup \{\#\})^2 \to \{\lessdot, \doteq, \gtrdot\}$, that with each ordered pair (a, b) associates the OP relation $M_{a,b}$ holding between a and b. We call the pair (Σ, M) an *operator precedence alphabet* (OP). Relations $\lessdot, \doteq, \gtrdot$, are named yields precedence, equal in precedence, takes precedence, respectively. By convention, the initial # can only yield precedence, and other symbols can only take precedence on the ending #.

If $M_{a,b} = \circ$, where $\circ \in \{\lessdot, \doteq, \gtrdot\}$, we write $a \circ b$. For $u, v \in \Sigma^*$ we write $u \circ v$ if $u = xa$ and $v = by$ with $a \circ b$. M is *complete* if $M_{a,b}$ is defined for every a and b in Σ. Moreover in the following we assume that M is $\doteq$*-acyclic*, which means that $c_1 \doteq c_2 \doteq \ldots \doteq c_k \doteq c_1$ does not hold for any $c_1, c_2, \ldots c_k \in \Sigma, k \geq 1$. See [7,4,5] for a discussion on this hypothesis.

Given an OP alphabet, the OPM M assigns a structure to strings in Σ^*, i.e., a string can be uniquely associated with a tree.

A *simple chain* is a string $c_0 c_1 c_2 \ldots c_\ell c_{\ell+1}$, written as ${}^{c_0}[c_1 c_2 \ldots c_\ell]^{c_{\ell+1}}$, such that: $c_0, c_{\ell+1} \in \Sigma \cup \{\#\}$, $c_i \in \Sigma$ for every $i = 1, 2, \ldots \ell$, and $c_0 \lessdot c_1 \doteq c_2 \ldots c_{\ell-1} \doteq c_\ell \gtrdot c_{\ell+1}$. A *composed chain* is a string $c_0 s_0 c_1 s_1 c_2 \ldots c_\ell s_\ell c_{\ell+1}$, where ${}^{c_0}[c_1 c_2 \ldots c_\ell]^{c_{\ell+1}}$ is a simple chain, and $s_i \in \Sigma^*$ is the empty string or is such that ${}^{c_i}[s_i]^{c_{i+1}}$ is a chain (simple or composed), for every $i = 0, 1, \ldots, \ell$. Such a composed chain will be written as ${}^{c_0}[s_0 c_1 s_1 c_2 \ldots c_\ell s_\ell]^{c_{\ell+1}}$. A string $s \in \Sigma^*$ is *compatible* with the OPM M if ${}^{\#}[s]^{\#}$ is a chain.

Floyd automata

Floyd automata are stack-based automata perfectly carved on the generation mechanism of the traditional Floyd grammars [1]. Not surprisingly they inherit some features of VPA (mainly a clear separation between push and pop operations) and maintain some typical behavior of shift-reduce parsing algorithms [2]; however, they also exhibit some distinguishing features.

A nondeterministic *Floyd automaton* (FA) is a tuple $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ where: (Σ, M) is a precedence alphabet, Q is a set of states (disjoint from Σ), $I, F \subseteq Q$ are sets of initial and final states, respectively, $\delta : Q \times (\Sigma \cup Q) \to 2^Q$ is the transition function. The transition function is the union of two disjoint functions: $\delta_{\text{push}} : Q \times \Sigma \to 2^Q$ and $\delta_{\text{flush}} : Q \times Q \to 2^Q$.

To define the semantics of the automaton, we introduce some notations. We use letters $p, q, p_i, q_i, \ldots$ for states in Q and we set $\Sigma' = \{a' \mid a \in \Sigma\}$; symbols in Σ' are called *marked* symbols. Let $\Gamma = (\Sigma \cup \Sigma' \cup \{\#\}) \times Q$; we denote symbols in Γ as $[a\ q]$, $[a' q]$, or $[\#\ q]$, respectively. We set $smb([a\ q]) = smb([a' q]) = a$, $smb([\#\ q]) = \#$, and $st([a\ q]) = st([a' q]) = st([\#\ q]) = q$.

A *configuration* of a FA is any pair $C = \langle B_1 B_2 \ldots B_n,\ a_1 a_2 \ldots a_m \rangle$, where $B_i \in \Gamma$ and $a_i \in \Sigma \cup \{\#\}$. The first component represents the contents of the stack, while the second component is the part of input still to be read.

A computation is a finite sequence of moves $C \vdash C_1$; there are three kinds of moves, depending on the precedence relation between $smb(B_n)$ and a_1:

(push) if $smb(B_n) \doteq a_1$ then $C_1 = \langle B_1 \ldots B_n [a_1\ q],\ a_2 \ldots a_m \rangle$, with $q \in \delta_{push}(st(B_n), a_1)$;
(mark) if $smb(B_n) \lessdot a_1$ then $C_1 = \langle B_1 \ldots B_n [a_1' q],\ a_2 \ldots a_m \rangle$, with $q \in \delta_{push}(st(B_n), a_1)$;
(flush) if $smb(B_n) \gtrdot a_1$ then let i be the greatest index such that $smb(B_i) \in \Sigma'$ and $C_1 = \langle B_1 \ldots B_{i-2} [smb(B_{i-1})\ q],\ a_1 a_2 \ldots a_m \rangle$, with $q \in \delta_{flush}(st(B_n), st(B_{i-1}))$.

Push and mark moves both push the input symbol on the top of the stack, together with the new state computed by δ_{push}; such moves differ only in the marking of the symbol on top of the stack. The flush move is more complex: the symbols on the top of the stack are removed until the first marked symbol (*included*), and the state of the next symbol below them in the stack is updated by δ_{flush} according to the pair of states that delimit the portion of the stack to be removed; notice that in this move the input symbol is not consumed and it remains available for the following move.

Finally, we say that a configuration $[\# \, q_I]$ is *starting* if $q_I \in I$ and a configuration $[\# \, q_F]$ is *accepting* if $q_F \in F$. The language accepted by the automaton is defined as:

$$L(\mathcal{A}) = \left\{ x \mid \langle [\# \, q_I], \; x\# \rangle \overset{*}{\vdash} \langle [\# \, q_F], \; \# \rangle, q_I \in I, q_F \in F \right\}.$$

The chains fully determine the structure of the parsing of any automaton over (Σ, M). Indeed, if the automaton performs the computation $\langle [a \, q_0], \; sb \rangle \overset{*}{\vdash} \langle [a \, q], \; b \rangle$, then ${}^{a}[s]^{b}$ is necessarily a chain over (Σ, M) and the first move in the above computation is a mark from state q_0, whereas the last one is a flush towards state q labelled by q_0. Such a computation corresponds to the parsing by the automaton of the string $s_0 c_1 \ldots c_\ell s_\ell$ within the context a,b; this context contains all information needed to build the subtree whose frontier is that string. This is a distinguishing feature of FL, not shared by other deterministic languages: we call it the *locality principle* of Floyd languages.

In other terms, given an OP alphabet, the OPM M assigns a structure to any string in Σ^* compatible with M; a FA defined on the OP alphabet selects an appropriate subset within such a “universe”. In some sense this property is yet another variation of the fundamental Chomsky-Shützenberger theorem.

Logic characterization of Floyd languages

Our characterization of FL in terms of a suitable Monadic Second Order (MSO) logic follows the approach originally proposed bu Büchi for regular languages and subsequently extended by Alur and Madhusudan for VPL. The essence of the approach consists in defining language properties in terms of relations between the positions of characters in the strings: first order variables are used to denote positions whereas second order ones denote subsets of positions; then, suitable constructions build an automaton from a given formula and conversely, in such a way that formula and corresponding automaton define the same language. The extension designed by [3] introduced a new basic binary predicate $\rightsquigarrow$ in the syntax of the MSO logic, $x \rightsquigarrow y$ representing the fact that in positions x and y two matching parentheses –named call and return, respectively in their terminology– are located. In the case of FL, however, we have to face new problems.

Both finite state automata and VPA are real-time machines, i.e., they read one input character at every move; this is not the case with more general machines such as FA, which do not advance the input head when performing flush transitions, and may also apply many flush transitions before the next push or mark which are the transitions that consume input. As a consequence, whereas in the logic characterization of regular and VP languages any first order variable can belong to only one second order variable representing an automaton state, in this case –when the automaton performs a flush– the same position may correspond to different states and therefore belong to different second-order variables.

In VPL the $\rightsquigarrow$ relation is one-to-one, since any call matches with only one return, if any, and conversely (with the exception of unmatched calls and returns, where many call positions can be in relation with +infinite and symmetrically). In FL, instead the same position y can be "paired" with different positions x in correspondence of many flush transitions with no push/mark in between, as it happens for instance when parsing a derivation such as $A \overset{*}{\Rightarrow} \alpha^k A$, consisting of k immediate derivations $A \Rightarrow \alpha A$; symmetrically the same position x can be paired with many positions y.

Consider an OP alphabet (Σ, M). We introduce a relation over positions of characters in any word $s \in \Sigma^*$. For $0 \le x < y \le |s| + 1$, we say that (x, y) is a *chain boundary* iff there exists a sub-string of $\#s\#$ which is a chain ${}^a[r]^b$, such that a is in position x and b is in position y. In general if (x, y) is a chain boundary, then $y > x + 1$, and a position x may be in such a relation with more than one position and vice versa. Moreover, if s is compatible with M, then $(0, |s| + 1)$ is a chain boundary.

Let us define a countable infinite set of first-order variables $x, y, \ldots$ and a countable infinite set of monadic second-order (set) variables $X, Y, \ldots$. The MSO$_{\Sigma,M}$ (*monadic second-order logic* over (Σ, M)) is defined by the following syntax:

$$\varphi := a(x) \mid x \in X \mid x \le y \mid x \curvearrowright y \mid x = y + 1 \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

where $a \in \Sigma$, x, y are first-order variables and X is a set variable.

MSO$_{\Sigma,M}$ formulae are interpreted over (Σ, M) strings and the positions of their characters in the following natural way: first-order variables are interpreted over positions of the string; second-order variables are interpreted over sets of positions; $a(x)$ is true iff the character in position x is a; $x \curvearrowright y$ is true iff (x, y) is a chain boundary; the other logical symbols have the usual meaning.

A sentence is a formula without free variables. The language of all strings $s \in \Sigma^*$ such that $\#s\# \models \varphi$ is denoted by $L(\varphi) = \{s \in \Sigma^* \mid \#s\# \models \varphi\}$, where $\models$ is the standard satisfaction relation.

This characterization completes a research path that began more than four decades ago and was resumed only recently with new -and old- goals. FL enjoy most of the nice properties that made regular languages highly appreciated and applied to achieve decidability and, therefore, automatic analysis techniques.

References

1. Floyd, R.W.: Syntactic analysis and operator precedence. Journ. ACM **10** (1963) 316–333
2. Grune, D., Jacobs, C.J.: Parsing techniques: a practical guide. Springer, New York (2008)
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journ. ACM **56** (2009)
4. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. Journal of Computer and System Science (2012) to appear.
5. Lonati, V., Mandrioli, D., Pradella, M.: Precedence automata and languages. In Kulikov, A.S., Vereshchagin, N.K., eds.: CSR. Volume 6651 of Lecture Notes in Computer Science., Springer (2011) 291–304
6. Lonati, V., Mandrioli, D., Pradella, M.: Logic characterization of Floyd languages. CoRR-arXiv **1204.4639** (2012) http://arxiv.org/abs/1204.4639.
7. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic properties of operator precedence languages. Information and Control **37** (1978) 115–133