Mechanisation of PDA and Grammar Equivalence for Context-Free Languages

Aditi Barthwal[1] and Michael Norrish[2,1]

[1] Australian National University
Aditi.Barthwal@anu.edu.au
[2] Canberra Research Lab., NICTA
Michael.Norrish@nicta.com.au

Abstract. We provide a formalisation of the theory of pushdown automata (PDAs) using the HOL4 theorem prover. It illustrates how provers such as HOL can be used for mechanising complicated proofs, but also how intensive such a process can turn out to be. The proofs blow up in size in way difficult to predict from examining original textbook presentations. Even a meticulous text proof has "intuitive" leaps that need to be identified and formalised.

1 Introduction

A context-free grammar provides a simple and precise mechanism for describing the methods by which phrases in languages are built from smaller blocks, capturing the "block structure" of sentences in a natural way. The simplicity of the formalism makes it amenable to rigorous mathematical study. Context-free grammars are also simple enough to allow the construction of efficient parsing algorithms using pushdown automata (PDAs). These "predicting machines" use knowledge about their stack contents to determine whether and how a given string can be generated by the grammar. For example, PDAs can be used to to build efficient parsers for LR grammars, some of which theory we have already mechanised [1].

This paper describes the formalisation of CFGs (Section 2) and PDAs (Section 3) using HOL4 [4], following Hopcroft & Ullman [2]. The formalisation of this theory is not only interesting in its own right, but also gives insight into the kind of manipulations required to port a pen-and-paper proof to a theorem prover. The mechanisation proves to be an ideal case study of how intuitive textbook proofs can blow up in size, and how details can change during formalisation. The crux of the paper is in Sections 4 and 5, describing the mechanisation of the result that the two formalisms are equivalent in power.

The theory outlined in this paper is part of the crucial groundwork for bigger results such as the SLR parser generation cited above. The theorems, even though well-established in the field, become novel for the way they have to be "reproven" in a theorem prover. Proofs must be recast to be concrete enough for the prover: patching deductive gaps which are easily grasped in a text proof, but beyond the automatic capabilities of the tool. The library of proofs, techniques and notations developed here provides the basis from which further work on verified language theory can proceed at a quickened pace.

2 Context-Free Grammars

A context-free grammar (CFG) is represented in HOL using the following type definitions:

```
symbol  = NTS of 'nts | TS of 'ts
rule = rule of 'nts => ('nts, 'ts) symbol list
grammar = G of ('nts, 'ts) rule list => 'nts
```

(The `=>` arrow indicates curried arguments to an algebraic type's constructor. Thus, the `rule` constructor is a curried function taking a value of type `'nts` (the symbol at the head of the rule), a list of symbols (giving the rule's right-hand side), and returning an `('nts,'ts) rule`.)

Thus, a rule pairs a value of type `'nts` with a symbol list. Similarly, a grammar consists of a list of rules and a value giving the start symbol. Traditional presentations of grammars often include separate sets corresponding to the grammar's terminals and non-terminals. It's easy to derive these sets from the grammar's rules and start symbol, so we shall occasionally write a grammar G as a tuple (V, T, P, S) in the proofs to come. Here, V is the list of non-terminals, T is the list of terminals, P is the list of productions and S is the start symbol.

Definition 1. *A list of symbols (or* sentential form*)* s derives t *in a single step if* s *is of the form* $\alpha A \gamma$*,* t *is of the form* $\alpha\beta\gamma$*, and if* $A \rightarrow \beta$ *is one of the rules in the grammar. In HOL:*

```
derives g lsl rsl ⟺
∃ s1 s2 rhs lhs.
(s1 ++ [NTS lhs] ++ s2 = lsl) ∧ (s1 ++ rhs ++ s2 = rsl) ∧
rule lhs rhs ∈ rules g
```

(The infix `++` *denotes list concatenation. The* ϵ *denotes membership.)*

We write `(derives g)`* sf_1 sf_2 to indicate that sf_2 is derived from sf_1 in zero or more steps, also written $sf_1 \Rightarrow^* sf_2$ (where the grammar g is assumed). This is concretely represented using what we call derivation lists. If an arbitrary binary relation R holds on adjacent elements of l which has x as its first element and y as its last element, then this is written $R \vdash l \lhd x \rightarrow y$. In the context of grammars, R relates sentential forms. Later we will use the same notation to relate derivations in a PDA. Using the concrete notation has simplified automating the proofs of many theorems. We will also use the rightmost derivation relation, `rderives`, and its closure.

Definition 2. *The* language *of a grammar consists of all the words (lists of only terminal symbols) that can be derived from the start symbol.*

```
language g =
{ tsl | (derives g)* [NTS (startSym g)] tsl ∧ isWord tsl }
```

3 Pushdown Automata

The PDA is modelled as a record containing the start state (`start` or q_0), the starting stack symbol (`ssSym` or Z_0), list of final states (`final` or F) and the next state transitions (`final` or δ).

```
pda = <| start : 'state; ssSym : 'ssym; final : 'state list;
         next : ('isym, 'ssym, 'state) trans list |>
```

The input alphabets (Σ), stack alphabets (Γ) and the states for the PDA (Q) can be easily extracted from the above information. In the proofs, we will refer to a PDA M as the tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ for easy access to the components. We have used lists instead of sets to avoid unncessary finiteness constraints in our proofs.

The `trans` type implements a single transition. A transition is a tuple of an 'optional' input symbol, a stack symbol and a state, and the next state along with the stack symbols (possibly none) to be added onto the current stack. The `trans` type describes a transition in the PDA's state machine. The `next` field of the record is a list of such transitions.

```
trans = ('isym option # 'ssym # 'state) # ('state # 'ssym list)
```

In HOL, a PDA transition in machine M is expressed using a binary relation on "instantaneous descriptions" of the tape, the machine's stack, and its internal state. We write $M \vdash (q, i\alpha, s) \rightarrow (q', i', s')$ to mean that in state q, looking at input i with stack s, m can transition to state q', with the input becoming i' and the stack becoming s'. The input i' is either the same as $i\alpha$ (referred to as an ϵ move) or is equal to α. Here, consuming the input symbol `i` corresponds to `SOME i` and ignoring the input symbol is `NONE` in the `trans` type.

Using the concrete derivation list notation, we write `ID` $M \vdash \ell \lhd x \rightarrow y$ to mean that the list ℓ is a sequence of valid instantaneous descriptions for machine M, starting with description x and ending with y. Transitions are not possible in the state where the stack is empty and only ϵ moves are possible in the state where the input is empty. In this paper, we will consider the language "accepted by empty stack" (`laes`):[3]

Definition 3 (Language accepted "by empty stack").

$$\texttt{laes}\ (M) \;=\; \{\ w \mid M \vdash (q_0, w, Z_0) \rightarrow^* (p, \epsilon, \epsilon)\ \ \textit{for some p in Q}\}$$

To be consistent with the notation in Hopcroft and Ullman, predicate `laes` is referred to as $N(M)$ in the proofs to follow. When the acceptance is by empty stack, the set of final states is irrelevant, so we usually let the list of final states be empty.

In the remainder of the paper we focus on the equivalence of PDAs and CFGs. Constructing a PDA for a CFG is a straightforward process so instead we devote much of the space to explaining the construction of a CFG from PDA and its equivalence proof. In order to illustrate the huge gap between a textbook *vs.* theorem prover formalisation, we try to follow Hopcroft and Ullman as closely as possible. As in the book, for the construction of a PDA from a CFG, we assume the grammar is in Greibach normal form.

[3] In the background mechanisation we have proved that this language is equivalent to the other standard notion: "accepted by final state".

4 Constructing a PDA for a CFG

Let $G = (V, T, P, S)$ be a context-free grammar in Greibach normal form generating L. We construct machine M such that $M = (q, T, V, \delta, q, S, \phi)$, where $\delta(q, a, A)$ contains (q, γ) whenever $A \rightarrow a\gamma$ is in P. Every production in a grammar that is in GNF has to be of the form $A \rightarrow a\alpha$, where a is a terminal symbol and α is a string (possibly empty) of non-terminal symbols (`isGnf`). The automaton for the grammar is constructed by creating transitions from the grammar productions, $A \rightarrow a\alpha$ that read the head symbol of the RHS (a) and push the remaining RHS (α) on to the stack. The terminals are interpreted as the input symbols and the non-terminals are the stack symbols for the PDA.

```
trans q (rule l r) = ((SOME (HD r),NTS l,q),q,TL r)

grammar2pda g q =
(let ts = MAP (trans q) (rules g) in
   <|start := q; ssSym := NTS (startSym g); next := ts;
     final := []|>)
```

(Here `HD` returns the first element in the list and `TL` returns the remaining list. Function `MAP` applies a given function to each element of a list.)

The PDA M simulates leftmost derivations of G. Since G is in Greibach normal form, each sentential form in a leftmost derivation consists of a string of terminals x followed by a string of variables α. M stores the suffix α of the left sentential form on its stack after processing the prefix x. Formally we show that

$$S \stackrel{l}{\Rightarrow}^* x\alpha \text{ by a leftmost derivation if and only if } (q, x, A) \rightarrow_M^* (q, \epsilon, \alpha) \qquad (1)$$

This turns out to be straightforward process in HOL and is done by representing the grammar and the machine derivations using derivation lists. Let dl represent the grammar derivation from S to $x\alpha$ and dl' represent the derivation from (q, x, A) to (q, ϵ, α) in the machine. Then an induction on dl gives us the "if" portion of (1) and induction on dl' gives us the "only if" portion of (1). Thus, we can conclude the following,

HOL Theorem 1

```
∀g. isGnf g ⇒ ∃m. x ∈ language g ⟺ x ∈ laes m
```

5 Constructing a CFG from a PDA

The CFG for a PDA is constructed by encoding every possible transition step in the PDA as a rule in the grammar. The LHS of each production encodes the starting and final state of the transition while the RHS encodes the contents of the stack in the final state.

Let M be the PDA $(Q, \delta, q_0, Z_0, \phi)$ and Σ and Γ the derived input and stack alphabets, respectively. We construct $G = (V, \Sigma, P, S)$ such that V is a set containing the new symbol S and objects of the form $[q, A, p]$; for q and p in Q, and A in Γ.

The productions P are of the following form: (**Rule 1**) $S \rightarrow [q_0, Z_0, q]$ for each q in Q; and (**Rule 2**) $[q, A, q_{m+1}] \rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3]...[q_m, B_m, q_{m+1}]$ for each $q, q_1, q_2, ..., q_{m+1}$ in Q, each a in $\Sigma \cup \{\epsilon\}$, and $A, B_1, B_2, ..., B_m$ in Γ, such that $\delta(q, a, A)$ contains $(q_1, B_1 B_2 ... B_m)$ (if $m = 0$, then the production is $[q, A, q_1] \rightarrow a$). The variables and productions of G have been defined so that a leftmost derivation in G of a sentence x is a simulation of the PDA M when fed the input x. In particular, the variables that appear in any step of a leftmost derivation in G correspond to the symbols on the stack of M at a time when M has seen as much of the input as the grammar has already generated.

From text to automated text: For **Rule 1** we only have to ensure that the state q is in Q. On the other hand, there are multiple constraints underlying the statement of **Rule 2** which will need to be isolated for mechanisation and are summarised below.

C2.1 The states q, q_1 and p belong in Q (a similar statement for terminals and non-terminals can be ignored since they are derived);
C2.3 the corresponding machine transition is based on the values of a and m and steps from state q to some state q_1 replacing A with $B_1 ... B_m$;
C2.3 the possibilties of generating the different grammar rules based on whether $a = \epsilon$, $m = 0$ or a is a terminal symbol;
C2.4 if $m > 1$ *i.e.* more than one nonterminal exists on the RHS of the rule then
- **C2.4.1** α is composed of only nonterminals;
- **C2.4.2** a nonterminal is an object of the form $[q, A, p]$ for PDA from-state q and to-state p, and stack symbol A;
- **C2.4.3** the from-state of the first object is q_1 and the to-state of the last object is q_{m+1};
- **C2.4.4** the to-state and from-state of adjacent nonterminals must be the same;
- **C2.4.5** the states encoded in the nonterminals must belong to Q.

Whether we use a functional approach or a relational one, the succinctness of the above definition is hard to capture in HOL. Using relations we can avoid concretely computing every possible rule in the grammar and thus work at a higher level of abstraction. The extent of details to follow are characteristic of mechanising such a proof. The relation `pda2grammar` captures the restrictions on the rules for the grammar corresponding to a PDA.

```
pda2grammar m g ⟺
(q ∈ states m ⟺
   rule (startSym g) [NTS (m.start,m.ssSym,q)] ∈ rules g) ∧
∀r. r ∈ rules g ⟺ p2gtrans m r
```

The first conjuct of the relation corresponds to **Rule 1** and the second conjunct (`p2gtrans`) ensures that each rule conforms with **Rule 2**. As already mentioned, **Rule 2** turns out to be more complicated to mechanise due to the amount of detail hidden behind the concise notation.

The `p2gtrans` predicate (see Figure 1) enforces the conditions **C2.1**, **C2.2**, **C2.3** (the three possibilities for the rule, $A \rightarrow \epsilon$; $A \rightarrow a$, where a is a terminal symbol and

```
p2gtrans m (rule l ntsl) ⟺
∃ isymo ssym q q' p mrhs.
  (l = (q,ssym,p)) ∧ q ∈ states m ∧ q' ∈ states m ∧
  p ∈ states m ∧ ((isymo,ssym,q),q',mrhs) ∈ m.next ∧
  ((ntsl = []) ∧ (isymo = NONE) ∧ (q' = p) ∧ (mrhs = []) ∨
   (∃ ts. (ntsl = [TS ts]) ∧ (isymo = SOME (TS ts)) ∧ (q' = p) ∧
      (mrhs = [])) ∨
   ∃ h t. ((ntsl = h::t) ∧ t ≠ []) ∧
     ((∃ ts. (h = TS ts) ∧ (isymo = SOME (TS ts)) ∧
         (MAP transSym t = mrhs) ∧ ntslCond m (q',p,mrhs) t) ∨
      (isymo = NONE) ∧ (MAP transSym ntsl = mrhs) ∧
      ntslCond m (q',p,mrhs) ntsl))
```

Fig. 1. Definition of `p2gtrans`.

$A \rightarrow a\alpha$) and the structure of the RHS of the rule which is based on the number of components in it (remaining three-way disjunction).

For the third type of production (more than one nonterminal *i.e.* $m > 1$), condition `ntslCond` captures **C2.4**. It enforces that $ntsl$ (α in **C2.4.1**) has only nonterminals, $[q, A, p]$ is interpreted as a non-terminal symbol and q (`frmState`) and p (`toState`) belong in the states of the PDA (**C2.4.2**), the conditions on q' and q_l that reflects **C2.4.3** condition on q_1 and q_{m+1} respectively, **C2.4.4** using relation adj and **C2.4.5** using the last conjunct.

```
ntslCond m (q',ql,mrhs) ntsl ⟺
EVERY isNonTmnlSym ntsl ∧
(∀ e1 e2 p s. (ntsl = p ++ [e1; e2] ++ s) ⇒ adj e1 e2) ∧
(frmState (HD ntsl) = q') ∧ (toState (LAST ntsl) = ql) ∧
(∀ e. e ∈ ntsl ⇒ toState e ∈ states m ∧ frmState e ∈ states m)
```

(The `;` is used to separate elements in a list and `LAST` returns the last element in a list.)

The constraints described above reflect exactly the information corresponding to the two criteria for the grammar rules. On the other hand, it is clear that the automated definition looks and is far more complex to digest. Concrete information that is easily gleaned by a human reader from abstract concepts has to be explicitly stated in a theorem prover.

Now that we have a CFG for our machine we can plunge ahead to prove the following.

Theorem 1. *If L is $N(M)$ for some PDA M, then L is a context-free language.*

To show that $L(G) = N(M)$, we prove by induction on the number of steps in a derivation of G or the number of moves of M that

$$(q, x, A) \rightarrow_M^* (p, \epsilon, \epsilon) \text{ iff } [q, A, p] \stackrel{l}{\Rightarrow}_G^* x\,. \tag{2}$$

5.1 Proof of the "if" portion of (2)

First we show by induction on i that if $(q, x, A) \rightarrow^i (p, \epsilon, \epsilon)$, then $[q, A, p] \Rightarrow^* x$.

HOL Theorem 2

```
ID m ⊢ dl ◁ (q,x,[A]) → (p,[],[]) ∧ isWord x ∧
pda2grammar m g ⇒ (derives g)* [NTS (q,A,p)] x
```

Proof. The proof is based on induction on the length of `dl`. The crux of the proof is breaking down the derivation such that a single stack symbol gets popped off after reading some (possibly empty) input.

Let $x = a\gamma$ and $(q, a\gamma, A) \rightarrow (q_1, \gamma, B_1B_2...B_n) \rightarrow^{i-1} (p, \epsilon, \epsilon)$. The single step is easily derived based on how the rules are constructed. For the $i-1$ steps, the induction hypothesis can be applied as long as the derivations involve a single symbol on the stack. The string γ can be written $\gamma = \gamma_1\gamma_2...\gamma_n$ where γ_i has the effect of popping B_j from the stack, possibly after a long sequence of moves. Note that B_1 need not be the n^{th} stack symbol from the bottom during the entire time γ_1 is being read by M. In general, B_j remains on the stack unchanged while $\gamma_1, \gamma_2...\gamma_{j-1}$ is read. There exist states $q_2, q_3, ..., q_{n+1}$, where $q_{n+1} = p$, such that $(q_j, \gamma_j, B_j) \rightarrow^* (q_j, \epsilon, \epsilon)$ by fewer than i moves (q_j is the state entered when the stack first becomes as short as $n - j + 1$). These observations are easily assumed by Hopcroft and Ullman or for that matter any human reader. The more concrete construction for mechanisation is as follows.

Filling in the gaps: For a derivation of the form, $(q_1, \gamma, B_1B_2...B_n) \rightarrow^i (p, \epsilon, \epsilon)$, this is asserted in HOL by constructing a list of objects $(q0, \gamma_j, B_j, q_n)$ (combination of the object's from-state, input, stack symbols and to-state), such that $(q0, \gamma_j, B_j) \rightarrow^i (q_n, \epsilon)$, where $i > 0$, γ_j is input symbols reading which stack symbol B_j gets popped off from the stack resulting in the transition from state q_0 to q_n. The from-state of the first object in the list is q_1 and the to-state of the last object is p. Also, for each adjacent object $e1, e2$, the to-state of $e1$ is the same as the from-state of $e2$. This process of popping off the B_j stack symbol turns out to be a lengthy one and is reflected in the proof statement of HOL Theorem 3.

To be able to prove this, it is neccessary to provide the assertion that each derivation in the PDA can be divided into two parts, such that the first part (list $dl0$) corresponds to reading n input symbols to pop off the top stack symbol. This is our HOL Theorem 4.

The proof of above is based on another HOL theorem that if $(q, \gamma\eta, \alpha\beta) \rightarrow^i (q', \eta, \beta)$ then we can conclude $(q, \gamma, \alpha) \rightarrow^i (q', \epsilon, \epsilon)$ (proved in HOL). This is a good example of a proof where most of the reasoning is "obvious" to the reader. This when translated into a theorem prover results in a cascading structure where one has to provide the proofs for steps that are considered "trivial". The gaps outlined here are just the start of the bridging process between the text proofs and the mechanised proofs.

Proof resumed: Once these gaps have been taken care of, we can apply the inductive hypothesis to get

$$[q_j, B_j, q_{j+1}] \stackrel{l}{\Rightarrow}^* \gamma_j \text{ for } 1 \leq j \leq n. \tag{3}$$

HOL Theorem 3

```
ID p ⊢ dl ◁ (q,inp,stk) → (qf,[],[]) ⇒
∃l. (inp = FLAT (MAP inp l)) ∧ (stk = MAP stk l) ∧
  (∀e. e ∈ MAP tost l ⇒ e ∈ states p) ∧
  (∀e. e ∈ MAP frmst l ⇒ e ∈ states p) ∧
  (∀h t. (l = h::t) ⇒ (frmst h = q) ∧ (stk h = HD stk) ∧
                      (tost (LAST l) = qf)) ∧
  ∀e1 e2 pfx sfx. (l = pfx ++ [e1; e2] ++ sfx) ⇒
                                  (frmst e2 = tost e1) ∧
    ∀e. e ∈ l ⇒ ∃m. m < |dl| ∧
        NRC (ID p) m (frmst e,inp e,[stk e]) (tost e,[],[])
```

(Relation `NRC` $R\ m\ x\ y$ *is the RTC closure of* R *from* x *to* y *in* m *steps.)*

HOL Theorem 4

```
ID p ⊢ dl ◁ (q,inp,stk) → (qf,[],[]) ⇒
∃dl0 q0 i0 s0 spfx. ID p ⊢ dl0 ◁ (q,inp,stk) → (q0,i0,s0) ∧
  (|s0| = |stk| - 1) ∧
  (∀q' i' s'. (q',i',s') ∈ FRONT dl0 ⇒ |stk| ≤ |s'|) ∧
  ((∃dl1. ID p ⊢ dl1 ◁ (q0,i0,s0) → (qf,[],[]) ∧
      |dl1| < |dl| ∧ |dl0| < |dl|) ∨
   ((q0,i0,s0) = (qf,[],[])))
```

(Predicate `FRONT` l *returns the list* l *minus the last element.)*

This leads to, $a[q_1, B, q_2][q_2, B_2, q_3]...[q_n, B_n, q_{n+1}] \stackrel{l}{\Rightarrow}{}^* x$.

Since $(q, a\gamma, A) \rightarrow (q_1, \gamma, B_1 B_2 ... B_n)$, we know that

$[q, A, p] \stackrel{l}{\Rightarrow} a[q_1, B, q_2][q_2, B_2, q_3]...[q_n, B_n, q_{n+1}]$, so finally we can conclude that $[q, A, p] \stackrel{l}{\Rightarrow}{}^* a\gamma_1\gamma_2...\gamma_n = x$.

The overall structure of the proof follows Hopcroft and Ullman but for each assertion made in the book, we have to provide concrete proofs before we can proceed any further. These proofs were quite involved, only a small subset of which has been shown above due to space restrictions.

5.2 Proof of the "only if" portion of (2)

Now suppose $[q, A, p] \Rightarrow^i x$. We show by induction on i that $(q, x, A) \rightarrow^* (p, \epsilon, \epsilon)$.

HOL Theorem 5

```
derives g ⊢ dl ◁ [NTS (q,A,p)] → x ⇒ isWord x ⇒
pda2grammar m g ⇒
(ID m)* (q,x,[A]) (p,[],[])
```

Proof. The basis, $i = 1$, is immediate, since $[q, A, p] \rightarrow x$ must be a production of G and therefore $\delta(q, x, A)$ must contain (p, ϵ). Note x is ϵ or in Σ here. In the inductive

step, there are three cases to be considered. The first is the trivial case, $[q, A, p] \Rightarrow a$, where a is a terminal. Thus, $x = a$ and $\delta(q, a, A)$ must contain (p, ϵ). The other two possibilities are, $[q, A, p] \Rightarrow a[q_1, B_1, q_2]...[q_n, B_n, q_{n+1}] \Rightarrow^{i-1} x$, where $q_{n+1} = p$ or $[q, A, p] \Rightarrow [q_1, B_1, q_2]...[q_n, B_n, q_{n+1}] \Rightarrow^{i-1} x$, where $q_{n+1} = p$. The latter case can be considered a specialisation of the first one such that $a = \epsilon$. Then x can be written as $x = ax_1x_2...x_n$, where $[q_j, B_j, q_{j+1}] \Rightarrow^* x_j$ for $1 \leq j \leq n$ and possibly $a = \epsilon$. This has to be formally asserted in HOL. Let α be of length n. If $\alpha \Rightarrow^m \beta$, then α can be divided into n parts, $\alpha = \alpha_1\alpha_2...\alpha_n$ and $\beta = \beta_1\beta_2...\beta_n$, such that $\alpha_i \Rightarrow^i \beta_i$ in $i \leq m$ steps.

HOL Theorem 6

```
derives g ⊢ dl ◁ x  y ⇒
∃l. (x = MAP FST l) ∧ (y = FLAT (MAP SND l)) ∧
  ∀a b. (a,b) ∈ l ⇒ ∃dl'. |dl'| ≤ |dl| ∧ derives g ⊢ dl' ◁ [a]  b
```

(The FLAT function returns the elements of (nested) lists, SND returns the second element of a pair.)

Inserting $B_{j+1}...B_n$ at the bottom of each stack in the above sequence of ID's gives us,

$$(q_j, x_j, B_jB_{j+1}...B_n) \rightarrow^* (q_{j+1}, \epsilon, B_{j+1}...B_n). \tag{4}$$

The first step in the derivation of x from $[q, A, p]$ gives us,

$$(q, x, A) \rightarrow (q_1, x_1x_2...x_n, B_1B_2...B_n) \tag{5}$$

is a legal move of M. From this move and (4) for $j = 1, 2, ..., n$, $(q, x, A) \rightarrow^* (p, \epsilon, \epsilon)$ follows. In Hopcroft and Ullman, the above two equations suffice to deduce the result we are interested in.

Unfortunately, the sequence of reasoning here is too coarse-grained for HOL4 to handle. The intermediate steps need to be explicitly stated for the proof to work out using a theorem prover. These steps can be further elaborated as follows[4]. By our induction hypothesis,

$$(q_j, x_j, B_j) \rightarrow^* (q_{j+1}, \epsilon, \epsilon). \tag{6}$$

Now consider the first step, if we insert $x_2...x_n$ after input x_1 and $B_2...B_n$ at the bottom of each stack, we see that

$$(q_1, x_1...x_n, B_1...B_n) \rightarrow^* (p, \epsilon, \epsilon). \tag{7}$$

Another fact that needs to be asserted explicitly is reasoning for (7).

This is done by proving the affect of inserting input/stack symbols on the PDA transitions. Now from the first step, (5) and (7), $(q, x, A) \rightarrow^* (p, \epsilon, \epsilon)$ follows.

Equation (2) with $q = q_0$ and $A = Z_0$ says $[q_0, Z_0, p] \Rightarrow^* x$ iff $(q_0, x, Z_0) \rightarrow^* (p, \epsilon, \epsilon)$. This observation, together with **Rule 1** of the construction of G, says that $S \Rightarrow^* x$ if and only if $(q_0, x, Z_0) \rightarrow^* (p, \epsilon, \epsilon)$ for some state p. That is, x is in $L(G)$ if and only if x is in $N(M)$ and we have

[4] Their HOL versions can be found as part of the source code

HOL Theorem 7

$$\texttt{pda2grammar}\ m\ g \wedge \texttt{isWord}\ x \Rightarrow \texttt{(derives}\ g\texttt{)}^{*}\ \texttt{[NTS (}q\texttt{,}A\texttt{,}p\texttt{)]}\ x \iff \texttt{(ID}\ m\texttt{)}^{*}\ \texttt{(}q\texttt{,}x\texttt{,[}A\texttt{]) (}p\texttt{,[],[])}$$

To avoid the above being vacuous, we additionally prove the following:

HOL Theorem 8

$$\forall m.\ \exists g.\ \texttt{pda2grammar}\ m\ g$$

6 Related work and conclusions

In the field of language theory, Nipkow [3] provided a verified and executable lexical analyzer generator. This work is the closest in nature to the mechanisation we have done.

A human reader is not concerned with issues such as finiteness of sets which have to be dealt with explicitly in a theorem prover. The form of definitions (relations *vs.* functions) has a huge impact on the size of the proof as well as the ease of automation. These do not necessarily overlap. A number of what we call "gap" proofs have been omitted due to space restrictions. These "gaps" cover the deductive steps that get omitted in a textbook proof and the intermediate results needed because of the particular mechanisation technique. Formalisation of a theory results in tools, techniques and an infrastructure that forms the basis of verifying tools based on the theory for example parsers, compilers, etc. Working in a well understood domain is useful in understanding the immense deviations that automation usually results in. More often than not the techniques for dealing with a particular problem in a domain are hard to generalise. The only solution in such cases is to have an extensive library at one's call.

The mechanised theory of PDAs is ~9000 lines and includes various closure properties of CFGs such as union, substitution and inverse homomorphism. It took 6 months to complete the work which includes over 600 lemmas/theorems. HOL sources for the work are available at `http://users.rsise.anu.edu.au/~aditi/`.

References

1. Aditi Barthwal and Michael Norrish. Verified, executable parsing. In Giuseppe Castagna, editor, *Programming Languages and Systems: 18th European Symposium on Programming*, volume 5502 of *Lecture Notes in Computer Science*, pages 160–174. Springer, March 2009.
2. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Ma., USA, 1979.
3. Tobias Nipkow. Verified lexical analysis. In J. Grundy and M. Newey, editors, *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'98)*, pages 1–15, Canberra, Australia, 1998. Springer-Verlag LNCS 1479.
4. Konrad Slind and Michael Norrish. A brief overview of HOL4. In O. A. Mohamed, C. Muñoz, and S. Tahar, editors, *Theorem Proving in Higher Order Logics*, volume 5170 of *LNCS*, pages 28–32. Springer, 2008. See also the HOL website at `http://hol.sourceforge.net`.