Undecidable Problems for Context-free Grammars

We discuss some basic undecidable problems for context-free languages, mainly based on *Valid and invalid computations of TM's: a tool for proving CFL problems undecidable*, Section 8.6 of the "Cinderella Book" by Hopcroft and Ullman [HU79]. The main results relating TM computations to CFG's are from Hartmanis [Ha67]. The main undecidability results were known, but by reduction from PCP, Section 2, rather than directly from TM computations.

It is assumed that basic notions of context-free grammars and Turing machines are known. In particular it is should be understood how to construct CFG's for the languages $\{x\#x^R\# \mid x \in \Sigma^*\}^*$ and $\{x\#y^R \mid x, y \in \Sigma^*, x \neq y\}$. Here $\#$ is a symbol not in Σ and $\cdot^R$ denotes the mirror operator.

$L(G_1) \cap L(G_2) = \varnothing$	disjointness	Theorem 2
$L(G) = \Sigma^*$	universality	Theorem 7
$L(G) = R$		
$L(G_1) = L(G_2)$	equality	
$L(G)$ is ambiguous	ambiguity	Theorem 5
$L(G)$ is regular	regularity	Theorem 8

1 Coding Turing Machine Computations

The basic tool for proving the various undecidability results for CF languages is a proper coding of TM computations.

A *configuration* of TM $\mathcal{M}$ is a string $w \in \Gamma^* Q \Gamma^*$, where Γ is the tape alphabet of $\mathcal{M}$ and Q is the state set of $\mathcal{M}$. Thus $w = xqy$ with $x, y \in \Gamma*$ and $q \in Q$, meaning that the tape has left and right parts x and y, and the TM is in state q with its head at the first letter of x. The tape alphabet Γ contains a special symbol $\mathtt{B}$ to represent blank cells. Clearly we only represent a finite number of $\mathtt{B}$'s in a configuration. A single computational step from configuration x to configuration y is denoted as $x \vdash y$.

A *valid computation* (of length n) is represented by string of the form $w_0\#w_1^R\#w_2\#w_3^R\#\ldots\#w_{2k-1}^R\#$ ($n = 2k-1$ is odd) or $w_0\#w_1^R\#w_2\#w_3^R\#\ldots\#w_{2k}\#$ ($n = 2k$ is even) in which the consecutive w_i represent the consecutive configurations from initial to halting. Adding the mirror at alternating positions makes the coding feasible for context-free grammars, as will be clear next. Thus, the requirements for a valid computation are

- $w_0 = \mathtt{B}q_0x$ is the initial configuration for some input $x \in \Sigma^*$,
- consecutive configurations are obtained by a TM step $w_i \vdash w_{i+1}$ for $0 \leq i < n$.
- w_n is accepting, i.e., of the form yhz where h is a halting state, and

The language of valid computations of TM $\mathcal{M}$ is denoted by valid($\mathcal{M}$).

Theorem 1. *Given a* TM $\mathcal{M}$ *one effectively constructs two* CFG G_1, G_2 *such that* valid($\mathcal{M}$) $= L(G_1) \cap L(G_2)$.

Proof. We use the fact that TM steps ony locally rewrites the tape contents. Let $x, \beta \in \Gamma^*$. Then

$$\begin{array}{rll} x\,ZpX\,\beta \vdash x\,qZY\,\beta & \text{if} \quad (p,X,q,Y,L) \in \delta & \text{'move left'} \\ x\,pX\,\beta \vdash x\,Yq\,\beta & \text{if} \quad (p,X,q,Y,R) \in \delta & \text{'move right'} \\ x\,pX\,\beta \vdash x\,qY\,\beta & \text{if} \quad (p,X,q,Y,S) \in \delta & \text{'stay'} \end{array}$$

If we take note of the mirror operation, consecutive configurations are coded as $\#\,xZpXy\,\#\,y^RYZqx^R\,\#$, move left for even steps, etc.

Now we define two grammars: G_1 generates repeated occurrences of even steps $w_i\#w_{i+1}^R$, while G_2 takes care of the odd steps $w_i^R\#w_{i+1}$.

Let $L_e = \{u\#v^R\# \mid u,v \in \Gamma^*Q\Gamma^*, u \vdash v\}$ and $L_o = \{u^R\#v\# \mid u,v \in \Gamma^*Q\Gamma^*, u \vdash v\}$. Then G_1 generates $L_e^*(\{\lambda\} \cup \Gamma^*H\Gamma^*)$, while similarly G_2 generates $\mathtt{B}q_0(\{\mathtt{B}\} \cup \Sigma^+)\,L_o^*(\{\lambda\} \cup \Gamma^*H\Gamma^*)$. Note how the first and last position contain initial and final configurations. Then the intersection of both languages yields computations of $\mathcal{M}$.

Checking whether w_n is final is simple, the form of the possible strings is regular. This restriction can be added to the grammars. □

Now the first undecidability result for CFG's is a simple consequence from the fact that it is undecidable whether the language accepted by a Turing machine is empty.

Theorem 2. *It is undecidable whether $L(G_1) \cap L(G_2) = \varnothing$ for two CFG's G_1, G_2.*

2 Post Correspondence Problem

We can improve Theorem 2 using PCP. It enables us to simplify the proof, and to restrict the type of CFG used.

An instance of *Post Correspondence Problem* is given by two lists of equal length, $A = (x_1, x_2, \ldots, x_n)$ and $B = (y_1, y_2, \ldots, x_n)$, of words over an alphabet Σ. It has a solution if there is a sequence of integers $i_1, i_2, \ldots, i_m$, $m > 0$, such that $x_{i_1}x_{i_2}\ldots x_{i_m} = y_{i_1}y_{i_2}\ldots y_{i_m}$.

Coding TM computations the following result can be obtained.

Proposition 3. PCP *is undecidable.*

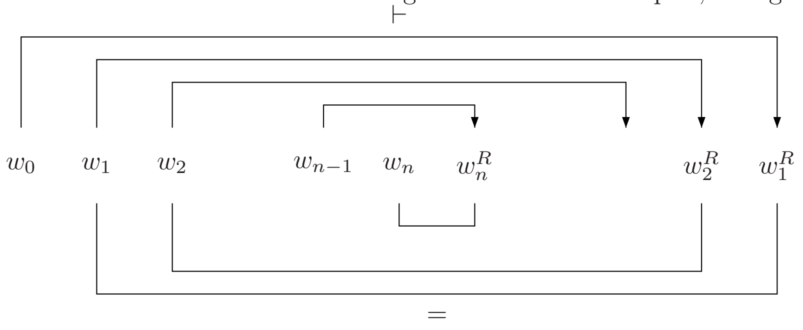
A CFG is called linear if it only has productions of the form $A \to xBy$ and $A \to x$ (with A, B nonterminal, and x, y terminal).

Consider a list $A = (x_1, x_2, \ldots, x_n)$ over an alphabet Σ. It is quite easy to encode the strings of words genereted by A together with their indexes using a linear grammar. Its productions are $A \to x_iAi$, $A \to x_i\#i$, $i = 1, \ldots n$. The language generated is $x_{i_1}x_{i_2}\ldots x_{i_m}\#i_m\ldots i_2i_1$, $1 \le i_1, i_2, \ldots, i_m \le n$, over the alphabet $\Sigma \cup \{1,2,\ldots,n\} \cup \{\#\}$.

Given two lists A, B obviously $L(G_A) \cup L(G_B)$ codes solutions of the corresponding PCP. Hence we have.

Theorem 4. *It is undecidable whether $L(G_1) \cap L(G_2) = \varnothing$ for two linear grammars G_1, G_2.*

Alternatively, we can restrict ourselves to linear grammars if we change the representation for a computation $w_0, w_1, \dots, w_n$ to $w_0\#w_1\#w_2\#\dots\#w_n\#w_n^R\#\dots\#w_2^R\#w_1^R\#$. This representation can be obtained as the intersection of two linear languages, one which checks the steps, and one which checks the mirror image between the two copies, see figure below.



A context-free grammar G is *ambiguous* if there exists a string x such that there are two different derivation trees for x in G. For a list of words A the grammar G_A as given above is unambiguous: each string has a unique derivation (tree). Given two lists we may assume that the grammars G_A and G_B have different nonterminals. In particular that means they have no derivations in common. Thus the grammar with productions $S \to A, S \to B$ together with the productions for G_A and G_B is unambiguous, unless there is a string that is in both $L(G_A)$ and $L(G_B)$. The latter is undecidable. Hence

Theorem 5. *It is undecidable whether G is ambiguous for linear grammar G.*

3 Coding Invalid Computations

We can obtain an even more striking result by focussing on the strings that do code invalid computations (which include both non-accepting computations and strings that do not code a computation at all).

Thus we define $\text{invalid}(\mathcal{M}) = (\Gamma \cup Q \cup \{\#\})^* - \text{valid}(\mathcal{M})$. It is easy to see that a string of the form $x_1\#x_2\#\dots x_m\#$ is invalid if one or more of the following cases hold.

- x_i not a configuration
- x_1 not initial
- x_n not final
- not $x_i \vdash x_{i+1}^R$ for even i
- not $x_i^R \vdash x_{i+1}$ for odd i

Hence we have the following result.

Theorem 6. *Given a TM $\mathcal{M}$ one effectively constructs a* CFG *G such that* $\text{invalid}(\mathcal{M}) = L(G)$.

A TM accepts the empty language iff all its computations are invalid. Consequently

Theorem 7.

1. *It is undecidable whether $L(G) = \Sigma^*$ for a* CFG *G over Σ.*
2. *It is undecidable whether $L(G) = R$ for* CFG *G and regular language R.*
3. *It is undecidable whether $L(G_1) = L(G_2)$ for* CFG*'s G_1, G_2.*

In the second item we assume both G and R are part of the input of the problem. For fixed R the problem may be undecidable (as for $R = \Sigma^*$) or decidable (as for $R = \varnothing$).

Note that $R \subseteq L(G)$ is undecidable, by having $R = \Sigma^*$. The question $L(G) \subseteq R$ on the other hand *is* decidable, it holds iff $L(G) \cap (\Sigma^* \setminus R) = \varnothing$. This is decidable as emptiness of context-free languages is decidable (and CFL are effectively closed under intersection with regular languages).

Not only we cannot decide whether $L(G) = R$ for a given regular R, but we cannot decide whether $L(G)$ is regular at all.

Theorem 8. *It is undecidable whether $L(G)$ is regular for* CFG *G.*

Proof. Start with a fixed nonregular context-free language $L_0 \subseteq \Sigma^*$. Let # be a symbol not in Σ.

Now for given G consider $L_1 = L_0\#\Sigma^* \cup \Sigma^*\#L(G)$. L_1 is context-free. We argue that L_1 is regular iff $L(G) = \Sigma^*$.

Assume we find a string $w \notin L(G)$ then $L_1 \cap (\Sigma^*\#w) = L_0\#w$. As L_0 is nonregular, also $L_0\#w$ is nonregular. Context-free languages are closed under intersection with regular languages so L_1 cannot be regular.

On the other hand, when $L(G) = \Sigma^*$ then $L_1 = \Sigma^*\#\Sigma^*$, which is regular.

So deciding regularity of L_1 would be equivalent to deciding whether $L(G) = \Sigma^*$, which is impossible. □

The above proof technique works for larger classes of grammars, and is called *Greibach's Theorem* [Gr68].

Theorem 9. *Let $\mathcal{C}$ be a family of languages effectively closed under union and concatenation with regular sets, and for which equality to Σ^* is undecidable (for sufficiently large Σ). Let $\mathcal{P}$ be a proper subset of $\mathcal{C}$, which contains all regular languages, and is closed under $/a$ (single letter quotient). Then membership of $\mathcal{P}$ is undecidable for $\mathcal{C}$.*

Even when it is known that $L(G)$ regular, finding a FSA A with $L(A) = L(G)$ is not effective. This is seen as follows. Let $\Delta = \Gamma \cup Q \cup \{\#\}$, and consider $L_0 = \text{invalid}(\mathcal{M}) \cup q_0\,\Sigma^+\,\#\Delta^*$. Apart from all invalid computations, L_0 also contains all computations on non-empty words. L_0 is effectively context-free.

We consider two cases. If $\lambda \in L(M)$, then $L_0 = \Delta^* - \{w\}$, where w codes the accepting computation for λ. Otherwise, if $\lambda \notin L(M)$, then $L_0 = \Delta^*$.

The language L_0 is regular in both cases, but because of the empty tape halting problem it is impossible to decide which of the two forms it has.

References

[Gr68] S.A. Greibach: A Note on Undecidable Properties of Formal Languages. Mathematical Systems Theory 2 (1968) 1–6.

[Ha67] J. Hartmanis: Context-free Languages and Turing Machine Computations. Mathematical Aspects of Computer Science, Proceedings of Symposia in Applied Mathematics, vol. 19, American Mathematical Society, 1967, pages 42–51.

[HU79] J.E. Hopcroft, J.D. Ullman: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.

Hendrik Jan Hoogeboom
Leiden University
May 19, 2015