CDM
Context-Sensitive Grammars

Klaus Sutner
Carnegie Mellon Universality

70-cont-sens 2017/12/15 23:17

Where Are We? 3

Context-free languages based on grammars with productions $A \to \alpha$ are very important since they describe many aspects of programming languages and admit very efficient parsers.

CFLs have a natural machine model (PDA) that is useful e.g. to evaluate arithmetic expressions.

Properties of CFLs are mostly undecidable, Emptiness and Finiteness being notable exceptions.

Postfix Calculators 4

Hewlett-Packard figured out 40 years ago the reverse Polish notation is by far the best way to perform lengthy arithmetic calculations. Very easy to implement with a stack.

The old warhorse dc also uses RPN.

```
10 20 30 + *
n
500
10 20 30
f
30
20
10
^
n
10737418240000000000000000000000000000
```

Languages versus Machines 5

Why the hedging about "aspects of programming languages"?

Because some properties of programming language lie beyond the power of CFLs. Here is a classical example: variables must be declared before they can be used.

begin
    int x;
    ...
    x = 17;
    ...
end

Obviously, we would want the compiler to address this kind of situation.

Languages versus Machines 6

To deal with problems like this one we need to strengthen our grammars. The key is to remove the constraint of being "context-free."

This leads to another important grammar based class of languages: context-sensitive languages. As it turns out, CSL are plenty strong enough to describe programming languages—but in the real world it does not matter, it is better to think of programming language as being context-free, plus a few extra constraints.

As we will see shortly, though, the associated machines (linear bounded automata) are quite important in complexity theory.

Context-Sensitive Grammars

Definition (CSG)

A context-sensitive grammar (CSG) is a grammar where all productions are of the form

$$\alpha A \beta \to \alpha \gamma \beta \qquad \text{where } \gamma \neq \varepsilon$$

Some authors also allow $S \to \varepsilon$ in which case S may not appear on the righthand side of any production. A language is context-sensitive if it can be generated by a context-sensitive grammar.

Note the constraint that the replacement string $\gamma \neq \varepsilon$; as a consequence we have

$$\alpha \Rightarrow \beta \qquad \text{implies} \qquad |\alpha| \leq |\beta|$$

This should look familiar from our discussion of ε-free CFG.

Brute-Force Recognition

Lemma

Every context-sensitive language is decidable.

Proof.

Suppose $w \in \Sigma^\star$ and $n = |w|$. In any potential derivation $(\alpha_i)_{i<N}$ we have $|\alpha_i| \leq n$.

So consider the digraph D with vertices $\Gamma^{\leq n}$ and edges $\alpha \Rightarrow^1 \beta$.

Then w is in L if w is reachable from S in D.

□

Of course, the size of D is exponential, so this method won't work in the real world.

Needless to say . . .

Not all decidable languages are context-sensitive.

Here is a cute diagonalization argument for this claim.

Let $(x_i)_i$ be an effective enumeration of $\Sigma^\star$ and $(G_i)_i$ an effective enumeration of all CSG over Σ (say, both in length-lex order). Set

$$L = \{\, x_i \mid x_i \notin \mathcal{L}(G_i) \,\}$$

Clearly, L is decidable.

But L cannot be context-sensitive by the usual diagonal mumbo-jumbo.

Example: Counting

We know that the language

$$L = \{\, a^n b^n c^n \mid n \geq 1 \,\}$$

is not context free. Here is a context-sensitive grammar G for L:

let $V = \{S, B\}$ and set

$$\begin{aligned} S &\to aSBc \mid abc \\ cB &\to Bc \\ bB &\to bb \end{aligned}$$

A typical derivation look like

$$S \Rightarrow a^{n+1} bc(Bc)^n \Rightarrow a^{n+1} bB^n c^{n+1} \Rightarrow a^{n+1} b^{n+1} c^{n+1}$$

Right?

Not so fast . . .

The "typical" derivation easily produces a proof that $L \subseteq \mathcal{L}(G)$.

But we also need to show that $\mathcal{L}(G) \subseteq L$.

This is a bit harder: we need to show that the productions cannot be abused in some unintended way to generate other strings.

E.g., the canonical order is not necessary:

$$S \Rightarrow aaSBcBc \Rightarrow aaSBBcc$$

Exercise

Figure out the details.

Example: More Counting

We also know that the language

$$L = \{\, x \in \{a, b, c\}^\star \mid \#_a x = \#_b x = \#_c x \,\}$$

is not context free. But, again, it is easily context-sensitive:

let $V = \{S, A, B, C\}$ and set

$$\begin{aligned} S &\to S' \mid \varepsilon \\ S' &\to S'ABC \mid ABC \\ XY &\to YX \qquad \text{for all } X, Y \in \{A, B, C\} \\ A &\to a \\ B &\to b \\ C &\to c \end{aligned}$$

Note that most productions are actually context free. The critical part is the commutation productions for $\{A, B, C\}$.

Closure Properties

Theorem

context-sensitive languages are closed under union, concatenation, Kleene star and reversal.

They are also closed under ε-free homomorphisms.

Proof is straightforward by manipulating the grammar.

Note that arbitrary homomorphisms do not work in this case: they erase too much information.

Innocent Questions

- Are CSL closed under intersection?
- Are CSL closed under complement?

The answer is Yes in both cases (so this is quite different from context-free languages).

The proof for intersection has to wait until we have a machine model, and the proof for complement requires a special and very surprising counting technique (later).

Normal Form

Theorem (Kuroda)

Every context-sensitive grammar can be written with productions of the form

$$A \to BC \qquad AB \to CD \qquad A \to a$$

The proof is very similar to the argument for Chomsky normal form for CFG.

Note that the recognition algorithm becomes particularly simple when the CSG is given in Kuroda normal form: we first get rid of all terminals and then operate only on pairs of consecutive variables.

Undecidability

Recall the trick to produce undecidability with CFGs: encode accepting computations of a Turing machine as strings

$$\# C_0 \# C_1 \# \ldots \# C_n \#$$

For CFG this requires some treachery (complement plus alternating configurations), but for CSG there is no problem: we can directly generate all strings corresponding to valid computations by a context-sensitive grammar. As a consequence even the most basic question about CSG is undecidable.

Theorem

It is undecidable whether a CSG generates the empty language.

Undecidability Overview

	$x \in L$	$L = \emptyset$	$L = \Sigma^\star$	$L = K$	$L \cap K = \emptyset$
regular	Y	Y	Y	Y	Y
DCFL	Y	Y	Y	Y	N
CFL	Y	Y	N	N	N
CSL	Y	N	N	N	N
decidable	Y	N	N	N	N
semi-dec.	N	N	N	N	N

Standard decision problems for the language classes we have seen so far.

A Machine Model? 19

So far we have have managed to associate language classes with machines:

- semidecidable — Turing machines
- context free — pushdown automata
- regular — finite state machines

The question arises whether a context-sensitive language can be similarly accepted by some suitable type of machine.

It is clear that we need more memory than for pushdown automata (stacks), but we cannot allow arbitrary memory (as in Turing machines).

Linear Bounded Automata 20

Definition

A linear bounded automaton (LBA) is a type of one-tape, nondeterministic Turing machine acceptor where the input is written between special end-markers and the computation can never leave the space between these markers (nor overwrite them).

Thus the initial configuration looks like

$$\#q_0x_1x_2\ldots x_n\#$$

and the tape head can never leave the block of n consecutive cells.

It may seem that there is not enough space to perform any interesting computations on an LBA, but note that we can use a sufficiently large tape alphabet to "compress" the input to a fraction of its original size and make room.

Digression: Physical Realizability 21

As an aside: this type of compression argument flies in the face of physical reality.

For example, it was recently demonstrated how to store bits by placing chlorine atoms on a copper substrate (using a scanning electron microscope). This produces fantastic densities, the whole Library of Congress would fit into a few cubic millimeters.

But one cannot store an ASCII symbols instead of a single bit.

Of course, there are lots of other problems: time, space, energy requirements.

The Myhill-Landweber-Kuroda Theorem 22

The development happened in stages:

- Myhill 1960 considered deterministic LBAs.
- Landweber 1963 showed that they produce only context-sensitive languages.
- Kuroda 1964 generalized to nondeterministic LBAs and showed that this produces precisely the context-sensitive languages.

Theorem

A language is accepted by a (nondeterministic) LBA iff it is context-sensitive.

Proof 23

First suppose L is context-sensitive via G. The idea is to run a derivation of G backwards, starting at a string of terminals.

To this end we can use a nondeterministic Turing machine that "randomly" tries to apply the productions $\alpha A\beta \to \alpha\gamma\beta$ in G backwards.

Since $\gamma \neq \varepsilon$, we do not need a full Turing machine: a LBA can handle this type of computation. Essentially, we replace γ by A (given the right context).

We will ultimately reach S iff the original word was in L.

Proof, Cont'd 24

For the opposite direction it is a labor of love to check in great gory detail that the workings of an LBA can be described by a context-sensitive grammar.

The critical point is that an LBA can overwrite a tape symbol but can not append new cells at the end (the head never moves beyond the end markers).

□

Note that nondeterminism is critical here: grammars are naturally nondeterministic and a deterministic machine would have to search through all possible choices. That seems to require more than just linear space (open problem, see below).

Digression: Physical Realizability II

It was recognized already in the 1950s that Turing machines are, in many ways, too general to describe anything resembling the type of computation that was possible on emerging digital computers.

The Rabin-Scott paper was one radical attempt to impose a radical constraint on Turing machines that brings them into the realm of "feasible" computation.

Myhill's introduction of LBA is another attempt at constructive restrictions. As we now know, LBAs are still a rather generous interpretation of the notion of feasible computation; real, practical algorithms need further constraints.

Still, it is a perfectly good model and there are many interesting problems that fit perfectly into this framework.

Intersection Closure

Theorem

CSL are closed under intersection.

Proof.

Given two LBAs M_i for L_i. we can construct a new LBA for $L_1 \cap L_2$ by using a 2-track tape alphabet (recall our synchronous transducers).

The upper track is used to simulate M_1, the lower track is used to simulate M_2.

It is easy to check that the simulating machine is again a LBA (it will sweep back and forth over the whole tape, updating both tracks by one step on the way).

□

Why Not CFG?

In essence, the argument says that we can combine two LBAs into a single one that checks for intersection. This is entirely similar to the argument for FSMs.

Burning Question: Why can't we do the same for PDAs?

Because we cannot in general combine two stacks into a single one (though this works in some cases; the stack height differences need to be bounded).

Quantified Boolean Formulae

Definition

A quantified Boolean formula (QBF) is a formula consisting of propositional connectives "and," "or" and "not," as well as existential and universal quantifiers.

If all variables in a QBF are bounded by a quantifier then the formula has a truth value: we can simply expand it out as in

$$\exists x\, \varphi(x) \mapsto \varphi(0) \vee \varphi(1)$$
$$\forall x\, \varphi(x) \mapsto \varphi(0) \wedge \varphi(1)$$

In the end we are left with an exponential size propositional formula without variables which can simply be evaluated.

QBF

Note that many properties of propositional formulae can easily be expressed this way. For example, $\varphi(x_1, x_2, \ldots, x_n)$ is satisfiable iff

$$\exists\, x_1, \ldots, x_n\, \varphi(x_1, x_2, \ldots, x_n) \text{ is valid}$$

Likewise, the formula is a tautology iff

$$\forall\, x_1, \ldots, x_n\, \varphi(x_1, x_2, \ldots, x_n) \text{ is valid}$$

Claim

Validity of QBF can be checked by a deterministic LBA.

Proof

Consider, say,

$$\forall\, x_1, \ldots, x_n\, \exists\, y_1, \ldots, y_m\, \varphi(\boldsymbol{x}, \boldsymbol{y})$$

To check validity we use two loops, one counting from 0 to $2^n - 1$ for $\boldsymbol{x}$ and another counting from 0 to $2^m - 1$ for $\boldsymbol{y}$.

The machine checks that for each value of $\boldsymbol{x}$ there exists a $\boldsymbol{y}$ that satisfies the quantifier-free part.

Of course, the running time is exponential — but not much space is required.

QBF Solvers

`http://qbf.satisfiability.org/gallery/`

There is a lot of current research on building powerful QBF solvers.

Kuroda's Problems

Kuroda stated two open problems in his 1964 paper:

- Is nondeterminism in LBA really needed?
- Are context-sensitive languages closed under complements?

The first problem is still open and seems very difficult.

But the second one has been solved.

Immerman-Szelepsényi

Kuroda's second problem was solved independently by two researchers in the late 1980s.

Theorem (Immerman-Szelepsényi, 1988)

Context-sensitive languages are closed under complement.

The easiest way to prove this is to consider a seemingly unrelated and harmless graph problem first.

Problem:	**Graph Reachability**
Instance:	A digraph G, two nodes s and t.
Question:	Is there a path from s to t?

Of course, this can easily be solved in linear time using standard graph algorithms.

Prototype Reachability Algorithm

The strategy is clear, but there is a problem. We build the set $R \subseteq V$ of vertices reachable from s in G in stages.

Let's say $(u, v) \in E$ requires attention if $u \in R$ but $v \notin R$.

```
R = { s };

while( some edge (u,v) requires attention )
        add v to R;

return t in R;
```

DFS and BFS are both instances of this strategy. Alas, these algorithms require linear space: we have to keep track of R.

How About Logarithmic Space?

Any algorithm using logarithmic space cannot in general keep track of the set of all reachable vertices, so this seems tricky.

It works, though, if we allow nondeterminism. Let $n = |V|$.

// path guessing
$\ell = 0$
$x = s$
while $\ell < n - 1$ **do**
    if $x = t$ **then** return Yes
    guess an edge (x, y)
    $\ell{+}{+}$
    $x = y$
return No

Works! 37

This works in the following sense:

- If there is a path s to t, then, making the right guesses, the algorithm can return Yes.
- If there is no path, then the algorithm always returns No.

So there may be false negatives, but there can never by false positives.

So, the symmetry between true and false is broken. But we already know that this turns out to be a good thing: nondeterministic finite state machines work that way. They can be made deterministic, but at a potentially exponential cost.

And semi-decision algorithms cannot be fixed at any cost.

Non-Reachability 38

How about the opposite problem: t is not reachable from s?

Problem:	**Graph Non-Reachability**
Instance:	A digraph G, two nodes s and t.
Question:	Is there no path from s to t?

Note that for this version nondeterminism seems totally useless: what exactly would we guess? A non-path? A unicorn???

It is a major surprise that Non-Reachability can also be handled in nondeterministic logarithmic space, though the logical complexity of the algorithm is substantially higher.

Counting Reachable Points 39

Let

$$R = \{\, x \in V \mid \text{ exists path } s \to x \,\}$$

be the set of vertices reachable from s (the weakly connected component of s).

From the perspective of nondeterministic logarithmic space, R can be exponentially large (it may have cardinality up to n). But, it suffices to know just its cardinality to determine non-reachability.

Claim

Given the cardinality $r = |R|$, we can solve Non-Reachability in nondeterministic logarithmic space.

Deciding Non-Reachability 40

```
// non-reachability algorithm
i = 0
forall x ≠ t in V do            // length-lex order
    guess path s → x
    if path found
    then i++
return i == r
```

This works since $i = r$ means that all reachable vertices are different from t.

Nice, but useless unless we can actually compute r.

Onion Trick 41

Instead of trying to compute r directly, let

$$R_\ell = \{\, x \in V \mid \text{ exists path } s \to x \text{ of length } \leq \ell \,\}$$

and set $r_\ell = |R_\ell|$.

Obviously $r_0 = 1$ and $r_{n-1} = r$.

So we only have to figure out how to compute r_ℓ, given $r_{\ell-1}$.

Counting 42

```
// inductive counting algorithm
// inputs ℓ, ρ; output ρ'

ρ' = 0
forall x in V do
    b = 0                       // increment 0 or 1
    c = 0                       // path counter
    forall y in V do
        guess path s → y of length < ℓ
        if path found
        then c++
        if y = x or (y, x) ∈ E
        then b = 1
    assert c == ρ               // all paths found
    ρ' = ρ' + b

return ρ'
```

Quoi? 43

Suppose ρ is the correct value of $r_{\ell-1}$.

Since we check that $c = \rho$, we know that all paths $s \to y$ have been guessed correctly.

But then we have properly found and counted all paths of length at most ℓ: we just have to add one more edge. Hence, the output ρ' is indeed r_ℓ.

Note that the algorithm requires only storage for a constant number of vertices, so logarithmic space suffices, plus nondeterminism to guess the right path from s to y.

Nondeterministic Functions? 44

The counting algorithm is heavily nondeterministic: we have to guess multiple paths $s \to y$ (which would never work out if we were to flip a coin to make nondeterministic decisions).

But note the **assert** statement: if we make a mistake, the whole computation crashes. This means in particular that no function value is produced on any of these branches.

The branches where all the guesses are correct all produce the exactly same value ρ.

Back to Immerman-Szelepsényi 45

The theorem now follows easily.

For suppose M is some LBA (nondeterministic!) which accepts some CSL L. We would like to build another LBA N that accepts $\Sigma^\star - L$.

Consider some input $x \in \Sigma^\ell$. Clearly the number of configurations of M on x is bounded by $n = c^\ell$ for some constant $c > 1$. Note that n also bounds the running time of M (remove loops).

So we can consider the graph of all such configurations with s the initial configuration for input x and t the accepting configuration, and edges given by the one-step relation for M.

The construction for Non-Reachability from above can then be handled by another LBA N, done.

Full Immerman-Szelepsényi 46

In a while, we will introduce *space complexity classes*: things that can be done using some amount $s(n)$ of space. Then we have:

Theorem

For any reasonable function $s(n) \geq \log n$ *we have* $\mathrm{NSPACE}(s(n)) = \mathrm{co{-}NSPACE}(s(n))$.

In particular $\mathrm{NL} = \mathrm{co{-}NL}$ *and* $\mathrm{NSPACE}(n) = \mathrm{co{-}NSPACE}(n)$.

For this version one considers graphs determined by the computations of a nondeterministic Turing machine (vertices are instantaneous descriptions, edges correspond to the one-step relation).

YABA 47

Yet another Boolean algebra: the context-sensitive languages form a Boolean algebra, just like the regular and decidable ones.

But the following is somewhat surprising:

Theorem

The Boolean algebras of regular, context-sensitive and decidable languages are all isomorphic.

The proof requires a bit of lattice theory; we'll skip.

Application: Determinization 48

For an NFA $\mathcal{A} = \langle\, Q, \Sigma, \delta; I, F \,\rangle$, recall that $\mathrm{pow}(\mathcal{A})$ denotes the accessible part of the power automaton produced by the Rabin-Scott determinization procedure.

We have $1 \leq |\mathrm{pow}(\mathcal{A})| \leq 2^n$ where $n = |Q|$, and we have examples that show that the upper bound is hard, even when $Q = I = F$.

Now consider the following problem:

Problem:	**Powerautomaton Reachability**
Instance:	An NFA $\mathcal{A}$ and a state set $P \subseteq Q$.
Question:	Is P in $\mathrm{pow}(\mathcal{A})$?

Powerautomaton Reachability can be "solved" by an LBA: The LBA can guess a string $x \in \Sigma^\star$ and verify that $\delta(I, x) = P$.

Note that we have to guess the witness $x = x_1 x_2 \dots x_m$ one symbol at a time: m may be exponentially larger than n.

Surprisingly, by Immerman-Szelepsényi, the corresponding non-reachability problem can also be solved by an LBA: we can "guess" in linear space that $P \subseteq Q$ is not part of pow($\mathcal{A}$).

This is another case where your intuition might suggest otherwise: what's there to guess?

Size Matters

Similarly the following problem can be solved by a LBA:

Problem:	**Powerautomaton Size**
Instance:	An NFA $\mathcal{A}$ and a bound B.
Question:	Does pow($\mathcal{A}$) have size at least B?

The LBA can

- guess B-many subsets $P \subseteq Q$, say, in lexicographic order, and
- guess a string $x \in \Sigma^\star$ and verify that $\delta(I, x) = P$.

Again by Immerman-Szelepsényi the corresponding problems $|\text{pow}(\mathcal{A})| \leq B$, $|\text{pow}(\mathcal{A})| = B$, $|\text{pow}(\mathcal{A})| \neq B$ can all be solved by an LBA.