Cons-free programs and complexity classes between Logspace and Ptime

Verification and Program Termination 2020

Siddharth Bhaskar, Neil D. Jones, Cynthia Kop, and Jakob Simonsen

Univeristy of Copenhagen and Radboud University Nijmegen

Computational complexity of languages

- Fix an alphabet Σ. A language is a subset $X \subseteq \Sigma^\star$.
- A Turing machine $\mathcal{M}$ decides X in case, for all $x \in \Sigma^\star$,
  - if $x \in X$, then there is an accepting computation of $\mathcal{M}$ on input x, and
  - if $x \notin X$, then there is a rejecting computation of $\mathcal{M}$ on input x.
- By imposing various resource bounds on $\mathcal{M}$, we get complexity classes.
- PTIME: class of all languages decidable in polynomial time
- LOGSPACE: class of all languages decidable in logarithmic space

Computational complexity of languages

Other relevant models of computation:

	captures $\textsc{Ptime}$ when	natural resources
circuit families	logspace uniform	size, depth
alternating TMs	logspace work tape	time, treesize
AuxPDAs	logspace work tape	time, stack size

- Variants: circuits can have *bounded*, *unbounded*, or *semi-unbounded* fan-in.
- AuxPDAs can be *deterministic* or *nondeterministic*
- close connection between (log) circuit depth, AuxPDA time, (log) ATM time, and ATM treesize.

Computational complexity of languages

These models naturally interpolate between $\textsc{Logspace}$ and $\textsc{Ptime}$

$$\textsc{Logspace} \subseteq \textsc{LogDCFL}, \textsc{NLogspace} \subseteq \textsc{LogCFL} = \mathrm{SAC}^1 \subseteq \mathrm{AC}^1 \subseteq \mathrm{NC}^2 \subseteq \mathrm{SAC}^2 \cdots \subseteq \mathrm{NC} \subseteq \textsc{Ptime}$$

- $\textsc{LogCFL}/\textsc{LogDCFL}$: polynomial-time nondeterminstic/deterministic AuxPDAs
- $\mathrm{AC}^k/\mathrm{SAC}^k/\mathrm{NC}^k$: depth-$O(\log^k n)$ (unbounded/semi-unbounded/bounded fan-in) circuit families
- NC $(= \mathrm{AC} = \mathrm{SAC})$ polylogarithmic depth circuit families, quasi-polynomial time AuxPDAs
- "Orthogonal" to ramification of $\textsc{Ptime}$ by time-$O(n^c)$ TMs.

Program-based complexity

Consider the following first-order functional programming language:

$$
\begin{array}{lcl}
\texttt{p} \in \text{Program} & \leftarrow & \texttt{def}_1\ \texttt{def}_2\ \ldots\ \texttt{def}_m \\
\texttt{def} \in \text{Definition} & \leftarrow & \texttt{f}\ \texttt{x}_1\texttt{x}_2 \ldots \texttt{x}_n = \texttt{e}^f \\
 & & \\
\texttt{e} \in \text{Expression} & \leftarrow & \texttt{x} \mid \texttt{f}\ \texttt{e}_1 \ldots \texttt{e}_n \mid \texttt{0} \mid \texttt{1} \\
 & \mid & \texttt{if}\ \texttt{e}_0\ \texttt{then}\ \texttt{e}_1\ \texttt{else}\ \texttt{e}_2 \\
 & \mid & \texttt{[]} \mid \texttt{cons}\ \texttt{e}_1\ \texttt{e}_2 \\
 & \mid & \texttt{head e} \mid \texttt{tail e} \mid \texttt{null e} \\
 & & \\
\texttt{x} \in \text{Parameter} & \leftarrow & \text{identifier} \\
\texttt{f} \in \text{FcnName} & \leftarrow & \text{identifier, disjoint from Parameter}
\end{array}
$$

Program-based complexity

- Programs can be augmented with any manner of semantics, which give meaning to the phrase "program $\mathtt{p}$ decides language X."
- All natural semantics come with a roughly equivalent measure of running time.
- Moreover program running time is roughly equivalent to time on a Turing machine.
- For any $X \subseteq \Sigma^\star$, $X \in \textsc{Ptime} \iff X$ is decidable by a program in polynomial time.

Cons-free programs

Definition

A program is cons-free in case it contains no occurrence of `cons`.

- For any $X \subseteq \Sigma^\star$, $X \in \textsc{Ptime} \iff X$ is decidable by a cons-free program.
- Let CF be the class of all cons-free programs. For a class $\mathcal{P}$ of programs, we write $\{\{\mathcal{P}\}\}$ to indicate the class of languages decided by some program in $\mathcal{P}$.

Theorem (J '99)

$\{\{\mathrm{CF}\}\} = \textsc{Ptime}$.

Cons-free programs

Definition

A program is tail recursive if all of its recursive calls are operationally final.

- Non-example:

```
even  x = if (null x) then 1 else not(even(tail x))
```

- Example:

```
even x = f x 1
f x b = if (null x) then b else f (tail x) (not b)
```

Theorem (J '99)

$\{\{\text{CF-TR}\}\} = \text{LOGSPACE}$.

Elementary observations

- Note that the maximum depth of the call stack of a cons-free program is bounded by a polynomial in the length of the input.

$\mathtt{f}(\mathtt{x}_0, y_0)$	$\mathtt{f}(\mathtt{x}_1, y_1)$	...	$\mathtt{f}(\mathtt{x}_\ell, y_\ell)$

- Hence (convergent) cons-free programs run in exponential time $(2^{n^{O(1)}})$
- Convergent cons-free tail recursive programs run in polynomial time $(n^{O(1)})$

Elementary observations

- Hence, separating exponential time from polynomial time for cons-free programs would suffice to separate Ptime from Logspace.
- **Motivating question:** What is the expressive power of time-bounded cons-free computation? In particular,
  - Does cons-free running time correspond to any known resource?
  - Are there robust cons-free time classes? (For example, cons-free polynomial time?)

Our results

- At a first approximation, **cons-free running time corresponds to AuxPDA running time**
- (and therefore, to related resources)
- We can characterize $\mathrm{LogDCFL}$, LogCFL, SAC^k, and NC as cons-free time classes.
- Thus: a correlation between cons-free running time and complexity-theoretic hardness, **unlike** ordinary running time!

Our results

- The real story: we can isolate 6 **dialects** of cons-free programs, depending on whether we allow nondeterminism, and what types of output we allow in the recursive function symbols.
- Original language types: e.g., $\mathtt{f} : (\Sigma^\star)^k \to 2$ or $\mathtt{f} : (\Sigma^\star)^k \to \Sigma^\star$.
- Boolean variant: only allow, e.g., $\mathtt{f} : (\Sigma^\star)^k \to 2$.
- Tuple variant: allow output tuples, e.g., $\mathtt{f} : (\Sigma^\star)^k \to (\Sigma^\star)^\ell$.

Our results

Diagram of inclusions:

$$\begin{array}{ccccc} \mathrm{nCF}_0 & \longrightarrow & \mathrm{nCF}_1 & \longrightarrow & \mathrm{nCF}_2 \\ \uparrow & & \uparrow & & \uparrow \\ \mathrm{dCF}_0 & \longrightarrow & \mathrm{dCF}_1 & \longrightarrow & \mathrm{dCF}_2 \end{array}$$

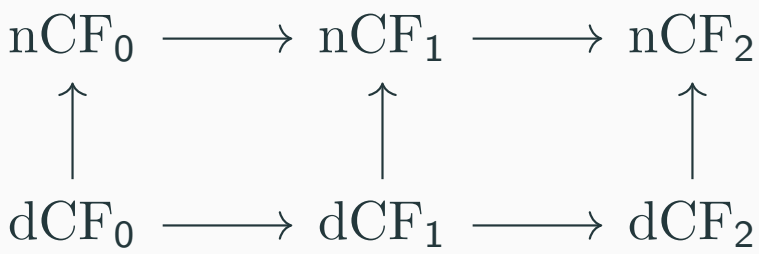
- dCF_1: original cons-free language
- all are equally expressive:

$$\{\{\mathrm{dCF}_0\}\} = \{\{\mathrm{dCF}_1\}\} = \{\{\mathrm{dCF}_2\}\} =$$

$$\{\{\mathrm{nCF}_0\}\} = \{\{\mathrm{nCF}_1\}\} = \{\{\mathrm{nCF}_2\}\} = \textsc{Ptime}$$

Our results

Diagram of inclusions:



- dCF_1: original cons-free language
- However, considering *time-bounded computation* might distinguish them!

Our results

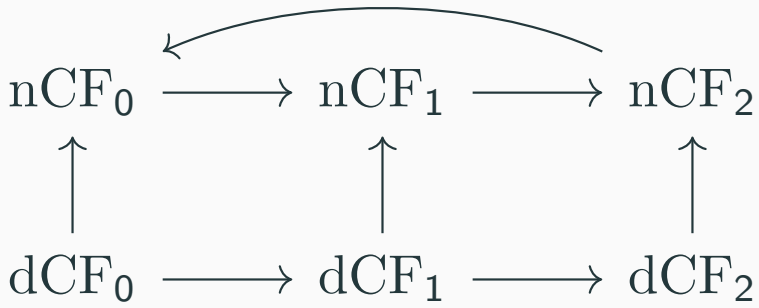
Lemma

nCF_2 programs can be transformed into nCF_0 programs with at most a polynomial loss in running time.

Hence for any class $\mathcal{C}$ of time bounds closed under composition by polynomials,

$$\{\{\mathrm{nCF}_0\text{–}\mathcal{C}\}\} = \{\{\mathrm{nCF}_1\text{–}\mathcal{C}\}\} = \{\{\mathrm{nCF}_2\text{–}\mathcal{C}\}\}.$$

Picture:



Our results

Theorem (with Ben-Amram)

For $k \geq 1$, $\{\{\text{nCF–time}(2^{O(\log^k n)})\}\} = \text{SAC}^k$. Hence,

- $\{\{\text{nCF–poly}\}\} = \text{LogCFL}$. (poly: $n^{O(1)}$)
- $\{\{\text{nCF–quasipoly}\}\} = \text{NC}$. (quasipoly: $2^{(\log n)^{O(1)}}$)

- In fact, SAC^k is known to be closed under complementation.
- **Open problem.** Is this closure realized by a natural program transformation?
- This might also generalize to the closure of NL $(= \{\{\text{nCF–TR}\}\})$ under complementation.

Our results

Theorem (with Ben-Amram)

In the deterministic case,

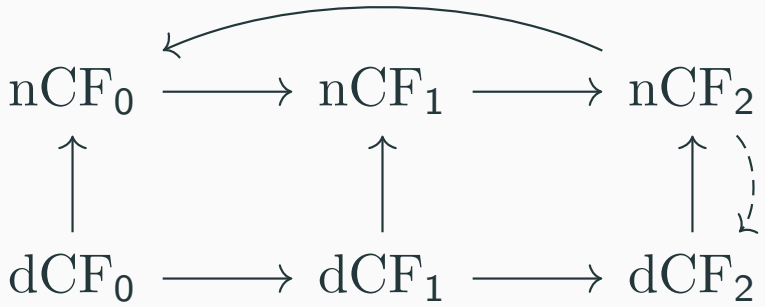
- $\{\{\mathrm{dCF}_2\text{–poly}\}\} = \mathrm{LogDCFL}$.
- $\{\{\mathrm{dCF}_i\text{–quasipoly}\}\} = \mathrm{NC}$ for $i \in \{0, 1, 2\}$.

- Complexity theory tells us that

$$\{\{\mathrm{nCF}\text{–time}(t(n))\}\} \subseteq \{\{\mathrm{dCF}_2\text{–time}(t(n)^{O(\log n)})\}\}.$$

- **Open problem.** Is this containment realized by a natural program transformation?
- $\mathrm{LogDCFL} = \mathrm{LogCFL}$? is the "P vs NP" problem for cons-free programs.

Discussion



Recap of technical open problems. Realize each of the following via natural program translations:

- closure of nCF-classes under complementation
- determinization of cons-free programs, with minimal loss of efficiency
- elimination of tuples/strings in the output, with minimal loss of efficiency

Discussion

Long-term goals:

- *Thesis* (Neil Jones, 1999!): "We maintain that Computability and Complexity theory, and Programming Language and Semantics have much to offer each other, in both directions."
- Interpolate $\textsc{Logspace}$ and $\textsc{Ptime}$ using cons-free programs, and recover all known relationships between complexity classes via natural program transformations.
- Develop a theory of cons-free algorithms, with an eye towards implementing complete problems.