Non-deterministic Characterisations*

Cynthia Kop

Department of Computer Science, University of Copenhagen (DIKU)
kop@di.ku.dk

Abstract

In this paper, we extend Jones' result—that cons-free programming with k^{th}-order data and a call-by-value strategy characterises $\texttt{EXP}^k\texttt{TIME}$—to a more general setting, including pattern-matching and non-deterministic choice. We show that the addition of non-determinism is unexpectedly powerful in the higher-order setting. Nevertheless, we can obtain a non-deterministic parallel to Jones' hierarchy result by appropriate restricting rule formation.

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

In [4], Jones introduces *cons-free programming*. Working with a small functional programming language, cons-free programs are defined to be *read-only*: recursive data cannot be created or altered (beyond taking sub-expressions), only read from the input. By imposing further restrictions on data order and recursion style, classes of cons-free programs turn out to characterise various deterministic classes in the time and space hierarchies of computational complexity. Most relevantly to this work, cons-free programs with data order k characterise the class $\texttt{EXP}^k\texttt{TIME}$ of decision problems decidable in $\mathcal{O}(\exp_2^k(a \cdot n^b))$ on a Turing Machine.

The classes thus characterised are all *deterministic*: they concern the time and space to solve decision problems on a deterministic Turing Machine. As the language considered by Jones is deterministic, a natural question is whether adding non-deterministic choice to the language would increase expressivity accordingly. The answer, at least in the base case, is *no*: following an early result by Cook [2], Bonfante shows [1] that adding a non-deterministic choice operator to cons-free programs with data order 0 makes no difference in expressivity: whether with or without non-deterministic choice, such programs characterise $\texttt{P}$.

In this paper, we consider the generalisation of this question: does adding non-deterministic choice give more expressivity when data of order greater than 0 is admitted? Surprisingly, the answer is yes! However, we do not obtain the non-deterministic classes; rather, non-deterministic cons-free programs of any data order ≥ 1 characterise $\texttt{ELEMENTARY}$, the class $\texttt{EXP}^0\texttt{TIME} \cup \texttt{EXP}^1\texttt{TIME} \cup \texttt{EXP}^2\texttt{TIME} \cup \dots$. As this is less useful for complexity arguments, we amend cons-freeness with a further restriction—unary variables—which allows us to obtain the expected generalisation: that (thus restricted) cons-free programs of data order k characterise $\texttt{EXP}^k\texttt{TIME}$, whether or not non-deterministic choice is allowed.

We also generalise Jones' language with pattern matching and user-defined constructors.

2 Cons-free programming

For greater generality—and greater ease of expressing examples—we extend Jones' language to a limited functional programming language with pattern matching. We will use terminology from the term rewriting world, but very little of the possibilities of this world.

* Supported by the Marie Skłodowska-Curie action "HORIP", program H2020-MSCA-IF-2014, 658162.

2.1 Higher-order Programs

We consider programs using simple types, including product types. The *type order* $o(\sigma)$ of a type σ is defined as follows: $o(\kappa) = 0$ for κ a *sort* (base type), $o(\sigma \times \tau) = \max(o(\sigma), o(\tau))$ and $o(\sigma \Rightarrow \tau) = \max(o(\sigma) + 1, o(\tau))$.

Assume given three disjoint set of identifiers: $\mathcal{C}$ of *constructors*, $\mathcal{D}$ of *defined symbols* and $\mathcal{V}$ of *variables*; each symbol is equipped with a type. Following Jones, we limit interest to constructors with a type $\iota_1 \Rightarrow \ldots \Rightarrow \iota_m \Rightarrow \kappa$ where all ι_i are types of order 0 and κ is a *sort*. *Terms* are expressions s such that $s : \sigma$ can be derived for some type σ using the clauses:

- $c\ s_1 \cdots s_m : \kappa$ if $c : \iota_1 \Rightarrow \ldots \Rightarrow \iota_m \Rightarrow \kappa \in \mathcal{C}$ and each $s_i : \iota_i$
- $a\ s_1 \cdots s_n : \tau$ if $a : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_n \Rightarrow \tau \in \mathcal{V} \cup \mathcal{D}$ and each $s_i : \sigma_i$
- $(s, t) : \sigma \times \tau$ if $s : \sigma$ and $t : \tau$

Thus, constructors cannot be partially applied, while variables and defined symbols can be. If $s : \sigma$, we say σ is the type of s, and let $\mathit{Var}(s)$ be the set of variables occurring in s. A term s is *ground* if $\mathit{Var}(s) = \emptyset$. We say t is a subterm of s, notation $s \trianglerighteq t$, if either $s = t$ or $s = a\ s_1 \cdots s_n$ with $a \in \mathcal{C} \cup \mathcal{F} \cup \mathcal{V}$ and $s_i \trianglerighteq t$ for some i, or $s = (s_1, s_n)$ and $s_i \trianglerighteq t$ for some i. Note that the head of an application is *not* a subterm of the application.

A *rule* is a pair of terms $f\ \ell_1 \cdots \ell_k \to r$ such that (a) $f \in \mathcal{D}$, (b) no defined symbols occur in any ℓ_i, (c) no variable occurs more than once in $f\ \ell_1 \cdots \ell_k$, (d) $\mathit{Var}(r) \subseteq \mathit{Var}(f\ \ell_1 \cdots \ell_k)$, and (e) r has the same type as $f\ \ell_1 \cdots \ell_k$. A *substitution* γ is a mapping from variables to ground terms of the same type, and $s\gamma$ is obtained by replacing variables x in s by $\gamma(x)$.

We fix a set $\mathcal{R}$ of rules, which are *consistent*: if $f\ \ell_1 \cdots \ell_k \to r$ and $f\ q_1 \cdots q_n \to s$ are both in $\mathcal{R}$, then $k = n$; we call k the *arity* of f. The set $\mathcal{DA}$ of *data terms* consists of all ground constructor terms. The set $\mathcal{VA}$ of *values* is given by: (a) all data terms are values, (b) if v, w are values, then (v, w) is a value, (c) if $f \in \mathcal{D}$ has arity k, $n < k$ and $s_1, \ldots, s_n$ are values, then $f\ s_1 \cdots s_n$ is a value if it is well-typed. Note that values whose type is a sort are data terms. The *call-by-value* reduction relation on ground terms is defined by:

- $(s, t) \to^* (v, w)$ if $s \to^* v$ and $t \to^* w$
- $a\ s_1 \cdots s_n \to^* a\ v_1 \cdots v_n$ if each $s_i \to^* v_i$ and either $a \in \mathcal{C}$, or $a \in \mathcal{D}$ and $n < \mathit{arity}(a)$
- $f\ s_1 \cdots s_m \to^* w$ if there are values $v_1, \ldots, v_m$ and a rule $f\ \ell_1 \cdots \ell_n \to r$ with $n \leq m$ and substitution γ such that each $s_i \to^* v_i = \ell_i\gamma$ and $(r\gamma)\ v_{n+1} \cdots v_m \to^* w$

Note that rule selection is non-deterministic; a choice operator might for instance be implemented by having rules `choose` $x\ y \to x$ and `choose` $x\ y \to y$.

2.2 Cons-free Programs

Since the purpose of this research is to find groups of programs which can handle *restricted* classes of Turing-computable problems, we must impose certain limitations. In particular, we will limit interest to *cons-free* programs:

▶ **Definition 1.** A rule $\ell \to r$ is cons-free if for all subterms $r \trianglerighteq s$ of the form $s = c\ s_1 \cdots s_n$ with $c \in \mathcal{C}$, we have: $s \in \mathcal{DA}$ or $\ell \triangleright s$. A program is cons-free if all its rules are.

This definition follows those for cons-free term rewriting in [3, 5] in generalising Jones' definition in [4]; the latter fixes the constructors in the program and therefore simply requires that the only non-constant constructor, `cons`, does not occur in any right-hand side.

In a cons-free program, if $v_1, \ldots, v_n, w$ are all data terms, then any data term occurring in the derivation of $f\ v_1 \cdots v_n \to^* w$ is a subterm of some v_i. This includes the result w.

3 Turing Machines and decision problems

In this paper, we particularly consider complexity classes of *decision problems*. A decision problem is a set $A \subseteq \{0,1\}^+$. A deterministic Turing Machine *decides* A in time $P(n)$ if every evaluation starting with a tape $\llcorner x_1 \dots x_n \llcorner\llcorner \dots$ completes in at most $P(n)$ steps, ending in the Accept state if $x_1 \dots x_n \in A$ and in the Reject state otherwise.

Let $\exp_2^0(m) = m$ and $\exp_2^{k+1}(m) = \exp_2^k(2^m) = 2^{\exp_2^k(m)}$. The class $\mathtt{EXP}^k\mathtt{TIME}$ consists of those decision problems which can be decided in $P(n) \leq \exp_2^k(a \cdot n^b)$ steps for some a, b.

▶ **Definition 2.** A program $(\mathcal{C}, \mathcal{D}, \mathcal{R})$ with constructors $\mathtt{true}, \mathtt{false} : \mathtt{bool}$, $[] : \mathtt{list}$ and $::$ (denoted infix) of type $\mathtt{bool} \Rightarrow \mathtt{list} \Rightarrow \mathtt{list}$, and a defined symbol $\mathtt{start} : \mathtt{list} \Rightarrow \mathtt{bool}$ *accepts* a decision problem A if for all $\vec{x} = x_1 \dots x_n \in \{0,1\}^+$: $\vec{x} \in A$ iff $\mathtt{start}\ (\overline{x_1}:: \dots ::\overline{x_n}::[]) \to^* \mathtt{true}$, where $\overline{x_i} = \mathtt{true}$ if $x_i = 1$ and $\mathtt{false}$ if $x_i = 0$. (Note that it is not required that *all* evaluations end in $\mathtt{true}$, just that there is at least one—and none if $x \notin A$).

4 A lower bound for expressivity

To give a lower bound on expressivity, we consider the following result paraphrased from [4]:

▶ **Lemma 3.** *Suppose that, given an input list* $cs ::= \overline{x_1}:: \dots ::\overline{x_n}::[]$ *of length* n*, we have a representation of* $0, \dots, P(n)$*, symbols* $\mathtt{seed}, \mathtt{pred}, \mathtt{zero} \in \mathcal{D}$*, and cons-free rules* $\mathcal{R}$ *with:*

- $\mathtt{seed}\ cs \to^* v$ *for* v *a value representing* $P(n)$
- *if* v *represents* $i > 0$*, then* $\mathtt{pred}\ cs\ v \to^* w$ *for* w *a value representing* $i - 1$
- *if* v *represents* i*, then* $\mathtt{zero}\ cs\ i \to^* \mathtt{true}$ *iff* $i = 0$ *and* $\mathtt{zero}\ cs\ i \to^* \mathtt{false}$ *iff* $i \neq 0$

Then any problem which can be decided in time $P(n)$ *is accepted by a cons-free program whose data order is the same as that of* $\mathcal{R}$*, and which is deterministic iff* $\mathcal{R}$ *is.*

Proof Idea. By simulating an evaluation of a Turing Machine. This simulation encodes all transitions of the machine as rules; a transition from state i to state j, reading symbol r, writing w and moving to the right is encoded by a rule $\mathtt{transition\ i\ r} \to (\mathtt{j}, (\mathtt{w}, \mathtt{R}))$. In addition, there are rules for $\mathtt{state}\ cs\ n$—which returns the state the machine is in at time n—, $\mathtt{position}\ cs\ n$—which returns the position of the tape reader—and $\mathtt{tape}\ cs\ n\ p$—for the symbol on the tape at position p and time n. Rules are for instance:

$$\mathtt{state}\ cs\ n \to \mathtt{ifthenelse}\ (\mathtt{zero}\ cs\ n)\ \mathtt{Start}\ (\mathtt{fst}\ (\mathtt{transitionat}\ cs\ (\mathtt{pred}\ cs\ n)))$$

This returns $\mathtt{Start}$ at time 0, and otherwise the state reduced to in the last transition. ◀

▶ **Example 4.** For $P(n) = (n+1)^2 - 1$, we can represent $i \in \{0, \dots, P(n)\}$ as any pair (l_1, l_2) of lists, where $i = |l_1| \cdot (n+1) + |l_2|$. For the counting functions, we define:

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{seed}\ cs & \to & ([], []) & \mathtt{zero}\ cs\ ([], []) & \to & \mathtt{true} \\
\mathtt{pred}\ cs\ (xs, y::ys) & \to & (xs, ys) & \mathtt{zero}\ cs\ (xs, y::ys) & \to & \mathtt{false} \\
\mathtt{pred}\ cs\ (x::xs, []) & \to & (xs, cs) & \mathtt{zero}\ cs\ (x::xs, []) & \to & \mathtt{false}
\end{array}$$

▶ **Lemma 5.** *For any* $a, b > 0, k \geq 0$*,there are cons-free, deterministic rules* $\mathcal{R}^k_{a,b}$ *defining counting functions as in Lemma 3 such that, for* $P(n) = \exp_2^k(a \cdot n^b) - 1$*, the numbers* $\{0, \dots, P(n)\}$ *can be represented. All function variables in* $\mathcal{R}^k_{a,b}$ *have a type* $\sigma \Rightarrow \mathtt{bool}$*.*

Proof Idea. For $k = 0$, we can count to $a \cdot n^b - 1$ using an approach much like Example 4. Given $\mathcal{R}^k_{a,b}$, which represents numbers as a type σ, we can define $\mathcal{R}^{k+1}_{a,b}$ by representing a number i with bit vector $b_0 \dots b_M$ (with $M = \exp_2^k(a \cdot n^b)$) as the function in $\sigma \Rightarrow \mathtt{bool}$ which maps a "number" j to $\mathtt{true}$ if $b_i = 1$ and to $\mathtt{false}$ otherwise. ◀

The observation that the functional variables take only one input argument will be used in Lemma 8 below. The counting techniques from Example 4 and Lemma 5 originate from Jones' work. However, in a non-deterministic system, we can do significantly more:

▶ **Lemma 6.** *Let $P_0(n) := n$, and for $k \geq 0$, $P_{k+1}(n) := 2^{P_k(n)} - 1$. Then for each k, we can represent all $i \in \{0, \dots, P_k(n)\}$ as a term of type $\mathtt{bool}^k \Rightarrow \mathtt{list}$, and accompanying counting functions $\mathtt{seed}_k, \mathtt{pred}_k$ and $\mathtt{zero}_k$ can be defined.*

Proof. The base case ($k = 0$) is Example 4. For larger k, let $i \in \{0, \dots, 2^{P_k(n)} - 1\}$ have bit vector $b_1 \dots b_{P_k(n)}$; we say $s : \mathtt{bool}^k \Rightarrow \mathtt{list}$ represents i at level k if for all $1 \leq j \leq P_k(n)$: $b_j = 1$ iff $s\ \mathtt{true} \to^* v$ for some v which represents j at level $k-1$, and $b_j = 0$ iff $s\ \mathtt{false} \to^* v$ for such v. This relies on non-determinism: $s\ \mathtt{true}$ reduces to a representation of *every* j with $b_j = 1$. A representation O of 0 at level $k-1$ is used as a default, e.g. $s\ \mathtt{false} \to^* O$ even if each $b_j = 0$. The $\mathtt{zero}$ and $\mathtt{pred}$ rules rely on testing bit values, using:

$\mathtt{bitset}_k\ cs\ F\ j \to \mathtt{bshelp}_k\ cs\ F\ j\ (\mathtt{equal}_{k-1}\ cs\ (F\ \mathtt{true})\ j)\ (\mathtt{equal}_{k-1}\ cs\ (F\ \mathtt{false})\ j)$

$\mathtt{bshelp}_k\ cs\ F\ j\ \mathtt{true}\ b \to \mathtt{true}$ $\qquad$ $\mathtt{bshelp}_k\ cs\ F\ j\ b\ \mathtt{true} \to \mathtt{false}$

$\mathtt{bshelp}_k\ cs\ F\ j\ \mathtt{false}\ \mathtt{false} \to \mathtt{bitset}_k\ cs\ F\ j.$

These rules are non-terminating, but if F represents a number at level k, and j at level $k-1$, then $\mathtt{bitset}_k\ cs\ F\ j$ reduces to exactly one value: $\mathtt{true}$ if $b_j = 1$, and $\mathtt{false}$ if $b_j = 0$. ◀

Thus, we can count up to arbitrarily high numbers; by Lemma 3, every decision problem in ELEMENTARY is accepted by a non-deterministic cons-free program of data order 1.

To obtain a more fine-grained characterisation which still admits non-deterministic choice, we will therefore consider a restriction of cons-free programming which avoids Lemma 6.

▶ **Definition 7.** A cons-free program *has unary variable* if all variables occurring in any rule in $\mathcal{R}$ have a type ι or $\sigma \Rightarrow \iota$, with $o(\iota) = 0$.

Intuitively, in a program with unary variables, functional variables cannot be *partially applied*; thus, such variables represent a function mapping to *data*, and not to some complex structure. Note that the input type σ of a unary variable $x : \sigma \Rightarrow \iota$ is allowed to be a product $\sigma_1 \times \dots \times \sigma_n$. Lemma 6 relies on non-unary variables, but Lemma 5 does not. We obtain:

▶ **Lemma 8.** *Any problem in $\mathtt{EXP}^k\mathtt{TIME}$ is accepted by a (non-deterministic) extended cons-free program of data order k.*

5 An upper bound for expressivity

To see that extended cons-free programs *characterise* the EXPTIME hierarchy, it merely remains to be seen that every decision problem that is accepted by a call-by-value cons-free program with unary variables and of data order k, can be solved by a deterministic Turing Machine—or, equivalently, an algorithm in pseudo code—running in polynomial time.

▶ **Algorithm 9** (Finding the values for given input in a fixed extended cons-free program $\mathcal{R}$).
Input: a term $\mathtt{start}\ v_1 \cdots v_n : \iota$ with each v_i a data term and $o(\iota) = 0$.
Output: all data terms w such that $\mathtt{start}\ v_1 \cdots v_n \to^* w$.

Let $\mathcal{B} := \bigcup_{1 \leq i \leq m} \{w \in \mathcal{DA} \mid v_i \trianglerighteq w\} \cup \bigcup_{\ell \to r \in \mathcal{R}} \{w \in \mathcal{DA} \mid r \trianglerighteq w\}$.

For all types σ occurring as data in $\mathcal{R}$, generate $[\![\sigma]\!]$ and a relation $\sqsupseteq$, as follows:

- $[\![\kappa]\!] = \{s \in \mathcal{B} \mid s : \kappa\}$ if κ is a sort; for $A, B \in [\![\kappa]\!]$, let $A \sqsupseteq B$ if $A = B$
- $[\![\sigma \times \tau]\!] = \{(A, B) \mid A \in [\![\sigma]\!] \wedge B \in [\![\tau]\!]\}$; $\quad (A_1, A_2) \sqsupseteq (B_1, B_2)$ if $A_1 \sqsupseteq B_1$ and $A_2 \sqsupseteq B_2$
- $[\![\sigma \Rightarrow \tau]\!] = \mathcal{P}(\{(A, B) \mid A \in [\![\sigma]\!] \wedge B \in [\![\tau]\!]\})$; for $A, B \in [\![\sigma \Rightarrow \tau]\!]$ let $A \sqsupseteq B$ if $A \supseteq B$

For all $f : \sigma_1 \Rightarrow \ldots \Rightarrow \sigma_m \Rightarrow \iota \in \mathcal{D}$, note that we can safely assume that $arity(f) \geq m-1$. For all such f, and all $A_1 \in [\![\sigma_1]\!], \ldots, A_m \in [\![\sigma_m]\!], v \in [\![\iota]\!]$, note down a statement: $f\ A_1 \cdots A_m \approx v$. If $arity(f) = m-1$, also note down $f\ A_1 \cdots A_{m-1} \approx O$ for all $O \in [\![\sigma_m \Rightarrow \iota]\!]$.

For all rules $\ell \to r$, all $s : \tau$ with $r \trianglerighteq s$ or $s = r\ x$, all $O \in [\![\tau]\!]$ and all substitutions γ mapping the variables $x : \sigma \in Var(s)$ to elements of $[\![\sigma]\!]$, note down a statement $s\gamma \approx O$. Mark all statements $x\gamma \approx O$ such that $x\gamma \sqsupseteq O$ as *confirmed*, and all other statements *unconfirmed*. Repeat the following steps until no new statements are confirmed anymore.

- For every unconfirmed statement $f\ A_1 \cdots A_n \approx O$, determine all rules $f\ \ell_1 \cdots \ell_k \to r$ (with $k = n$ or $k = n-1$) and substitutions γ mapping $x : \sigma \in Var(f\ \ell_1 \cdots \ell_k)$ to an element of $[\![\sigma]\!]$, such that each $A_i = \ell_i\gamma$, and mark the statement as confirmed if $(r\ x_{k+1} \cdots x_n)\gamma[x_{k+1} := A_{k+1}, \ldots, x_n := A_n] \approx O$ is confirmed.
- For every unconfirmed statement $(F\ s)\gamma \approx O$, mark the statement as confirmed if there exists A with $(A, O) \in \gamma(F)$ and $s\gamma \approx A$ is confirmed.
- For every unconfirmed statement $(f\ s_1 \cdots s_n)\gamma \approx O$, mark it as confirmed if there are $A_1, \ldots, A_n$ such that both $f\ A_1 \cdots A_n \approx O$ and each $s_i\gamma \approx A_i$ are confirmed.

Then return all w such that `start` $v_1 \cdots v_n \approx w$ is marked confirmed.

▶ **Lemma 10.** *Algorithm 9 is in* $\texttt{EXP}^k\texttt{TIME}$*—where k is the data order of $\mathcal{R}$—and returns the claimed output.*

Proof Idea. The complexity of Algorithm 9 is determined by the size of each $[\![\sigma]\!]$. The proof of soundness and completeness of the algorithm is more intricate; this fundamentally relies on replacing the values $f\ v_1 \cdots v_n$ with $n < arity(f)$ by subsets of the set of all tuples (A, w) with the property that, intuitively, $f\ v_1 \cdots v_n\ A \to^* w$. ◀

6 Conclusion

Thus, we obtain the following variation of Jones' result:

▶ **Theorem 11.** *A decision problem A is in* $\texttt{EXP}^k\texttt{TIME}$ *if and only if there is a cons-free program $\mathcal{R}$ of data order k and with unary variables, which accepts A. This statement holds whether or not the program is allowed to use non-deterministic choice.*

In addition, we have adapted Jones' language to be more permissive, admitting additional constructors and pattern matching. This makes it easier to specify suitable programs.

Using non-deterministic programs is a step towards further characterisations; in particular, we intend to characterise $\texttt{NEXP}^k\texttt{TIME} \subseteq \texttt{EXP}^{k+1}\texttt{TIME}$ using restricted non-deterministic cons-free programs of data order $k+1$.

References

1. G. Bonfante. Some programming languages for logspace and ptime. In *AMAST '06*, volume 4019 of *LNCS*, pages 66–80, 2006.
2. S.A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *ACM*, 18(1):4–18, 1971.
3. D. de Carvalho and J. Simonsen. An implicit characterization of the polynomial-time decidable sets by cons-free rewriting. In *RTA-TLCA '14*, volume 8560 of *LNCS*, pages 179–193, 2014.
4. N. Jones. Life without cons. *JFP*, 11(1):5–94, 2001.
5. C. Kop and J. Simonsen. Complexity hierarchies and higher-order cons-free rewriting. In *FSCD '16*, volume 52 of *LIPIcs*, pages 23:1–23:18, 2016.