Non size-increasing computation

Martin Hofmann

LMU München

Geocal '06

Luminy, 9.-10. Feb. 2006

Main result of these talks

Description: The system $LFPL_\omega$ is a linearly typed functional language with recursive datatypes and higher-order functions. The growth of definable functions is controlled by an abstract type $\diamond$ elements of which must be present in order to construct a member of a recursive datatype and which become available upon destruction of such a member in a pattern match.

Theorem: A function $f : \mathbb{N} \to \{0, 1\}$ is in the complexity class $EXPTIME = DTIME(2^{n^{O(1)}})$ if and only if it is representable in $LFPL_\omega$. The restriction of $LFPL_\omega$to structural recursion captures polynomial time.

Outline

Today:

- Motivation and background
- Definition of *LFPL* and $LFPL_\omega$
- Proof that structural recursion captures polytime (part I)
- New connective: The "LFPL-bang".

Tomorrow morning:

- Proof that structural recursion captures polytime (part II)
- Representation of polyspace in $LFPL_\omega$

Tomorrow afternoon:

- Proof that $LFPL_\omega$ programs can be evaluated in *EXPTIME*.

Motivation and Background

1. **Programming languages capturing complexity classes.**
   - Bounded recursion (Cobham, Cook-Urquhart)
   - Safe and ramified recursion (Cook-Bellantoni, Leivant, Caseiro)
   - Finite model theory (Gurevich, Goerdt)
   - Higher-order and type-theoretic extensions (BLL, LLL, SLL, Bellantoni et al, Aehlig-Schwichtenberg, MH)
2. **Memory management**
   - Alias analysis (Morrisett-Walker, Reps-Wilhelm)
   - Logic for pointers (O'Hearn-Calcagno, Reynolds)
   - Pointer analysis (Klarlund et al.)
   - In-place update (Shankar, Mackie, Cooper)

Non size-increasing functions

Want to discover statically whether

- the complexity of a function does not "explode" under iteration
- a function can be evaluated in a given amount of space.

Useful (to an extent) answer: isolate statically functions that do not increase the size of their input.

A (affine) linear functional language

- *Types:* booleans (B), lists (L(A)), binary trees (T(A)), products ($A \otimes B$), disjoint union ($A + B$), resource type ($\diamond$).

  In examples: N = B $\otimes \cdots \otimes$ B (32 times), type variables.

- *Signatures:* mapping of function symbols f to "arities" $\Sigma(f) = A_1, A_2, \ldots, A_n \to B$, e.g., append : L(N), L(N) $\to$ L(N).

- *Programs:* Signature + for each function symbol f with $\Sigma(f) = A_1, \ldots, A_n \to B$ a term e_f of type B containing free variables $x_1{:}A_1, \ldots, x_n{:}A_n$. The term e_f may contain calls to f and other functions declared in Σ.

Terms

built up from function calls, constructors, and pattern matching like in functional programming with the following exceptions:

- Constructors of recursive types take an extra argument of type $\Diamond$ (unless they are nil):

$$\mathsf{cons}(e_1^{\Diamond}, e_2^{A}, e_3^{\mathsf{L}(A)}) : \mathsf{L}(A)$$

$$\mathsf{match}\ e_1^{\mathsf{L}(A)}\ \mathsf{with}\ \mathsf{nil} \Rightarrow e_2^{C} \mid \mathsf{cons}(x^{\Diamond}, y^{A}, z^{\mathsf{L}(A)}) \Rightarrow e_3^{C}$$

  (as always pattern matching binds variables)

- Free and bound variables occur at most once (in the usual sense of affine linear types, e.g. occurrences in different branches of case distinction count only once.)

- Variables of type $\mathsf{B}, \mathsf{B} \otimes \mathsf{B}, \mathsf{N}, \ldots$ may be used more than once.

Examples

$\texttt{append} : \mathsf{L}(C), \mathsf{L}(C) \to \mathsf{L}(C)$

$\texttt{append}(l, \mathsf{nil}) = l$

$\texttt{append}(\mathsf{cons}(d, h, t), l) = \mathsf{cons}(d, h, \texttt{append}(t, l))$

Formally:

$$\begin{aligned} \texttt{append}(l_1, l_2) = \mathsf{match}\ l_1\ \mathsf{with}\ & \mathsf{nil} \Rightarrow l_2 \\ & \mid \mathsf{cons}(d, h, t) \Rightarrow \mathsf{cons}(d, h, \texttt{append}(t, l_2)) \end{aligned}$$

$\texttt{reverse} : \mathsf{L}(C) \to \mathsf{L}(C)$

$\texttt{reverse}(\mathsf{nil}) = \mathsf{nil}$

$\texttt{reverse}(\mathsf{cons}(d, h, t)) = \texttt{append}(\texttt{reverse}(t), \mathsf{cons}(d, h, \mathsf{nil}))$

$$\texttt{insert} : \diamond, C, \mathsf{L}(C) \to \mathsf{L}(C)$$

$$\texttt{insert}(d, x, \mathsf{nil}) = \mathsf{cons}(d, x, \mathsf{nil})$$

$$\texttt{insert}(d_1, x, \mathsf{cons}(d_2, y, l)) = \mathsf{let}\ (x, y, b) = \texttt{compare}(x, y)\ \mathsf{in}$$

$$\quad \mathsf{if}\ b\ \mathsf{then}\ \mathsf{cons}(d_1, x, \mathsf{cons}(d_2, y, l))$$

$$\quad\ \mathsf{else}\ \mathsf{cons}(d_1, y, \texttt{insert}(d_2, x, l))$$

$$\texttt{sort} : \mathsf{L}(C) \to \mathsf{L}(C)$$

$$\texttt{sort}(\mathsf{nil}) = \mathsf{nil}$$

$$\texttt{sort}(\mathsf{cons}(d, x, l)) = \texttt{insert}(d, x, \texttt{sort}(l))$$

$$\texttt{bst} : \mathsf{L}(C) \to \mathsf{T}(C)$$

$$\texttt{bst}(\mathsf{nil}) = \mathsf{leaf}$$

$$\texttt{bst}(\mathsf{cons}(d, h, t)) = \texttt{ins}(d, h, \texttt{bst}(t))$$

$$\texttt{ins} : (\diamond, C, \mathsf{T}(C)) \to \mathsf{T}(C)$$

$$\texttt{ins}(d, c, \mathsf{leaf}) = \mathsf{node}(d, c, \mathsf{leaf}, \mathsf{leaf})$$

$$\texttt{ins}(d_1, c_1, \mathsf{node}(d_2, c_2, l, r)) = \mathsf{if}\ c_1 \leq c_2\ \mathsf{then}$$

$$\mathsf{node}(d_1, c_2, \texttt{ins}(d_2, c_1, l), r)$$

$$\mathsf{else}\ \mathsf{node}(d_1, c_2, l, \texttt{ins}(d_2, c_1, r))$$

$$\texttt{head\_tail} : \mathsf{L}(C) \to C \otimes \diamond \otimes \mathsf{L}(C)$$

$$\texttt{head\_tail}(\mathsf{nil}) = \texttt{head\_tail}(\mathsf{nil})$$

$$\texttt{head\_tail}(\mathsf{cons}(d, x, l)) = d \otimes x \otimes l$$

$\texttt{duplist} : \mathsf{L}(\Diamond \otimes \mathsf{B}) \to \mathsf{L}(\mathsf{B}) \otimes \mathsf{L}(\mathsf{B})$

$\texttt{duplist}(\mathsf{nil}) = \mathsf{nil} \otimes \mathsf{nil}$

$\texttt{duplist}(\mathsf{cons}(d_1, d_2 \otimes h, t)) = \mathsf{match}\ \texttt{duplist}(t)\ \mathsf{with}$

$\quad u \otimes v \Rightarrow \mathsf{cons}(d_1, h, u) \otimes \mathsf{cons}(d_2, h, v)$

$\texttt{twice} : \mathsf{L}(\Diamond \otimes \mathsf{B}) \to \mathsf{L}(\mathsf{B})$

$\texttt{twice}(l) = \mathsf{match}\ \texttt{duplist}(l)\ \mathsf{with}$

$\quad u \otimes v \Rightarrow \mathsf{append}(u, v)$

Note: Function `twice` duplicates length. There is *no* definable function that squares or exponentiates length. So, really, $\Diamond$ enforces linear growth, not zero growth.

Would like to see more examples?

See

`http://www.dcs.ed.ac.uk/home/resbnd/prototypes/by_Robert_Atkey/`

for more examples and online experiments.

Functions are non-size increasing in standard model

$$[\![\mathsf{B}]\!] = \{\mathsf{tt}, \mathsf{ff}\} \qquad [\![\mathsf{L}(A)]\!] = [\![A]\!]^*$$
$$[\![A \otimes B]\!] = [\![A]\!] \times [\![B]\!] \qquad [\![\diamond]\!] = \{\star\}$$
$$\ldots$$

If $f : A_1, \ldots, A_n \to B$ then $[\![f]\!] : [\![A_1]\!] \times \cdots \times [\![A_n]\!] \rightharpoonup [\![B]\!]$ defined as least fixpoint (recursively).

If $v_1 \in [\![A_1]\!], \ldots, v_n \in [\![A_n]\!]$ then

$$|[\![f]\!](v_1, \ldots, v_n)|_B \leq |v_1|_{A_1} + \cdots + |v_n|_{A_n}$$

where $|-|_C : [\![C]\!] \to \mathbb{N} \cup \{\infty\}$, e.g., $|[u_1, \ldots, u_\ell]|_{\mathsf{L}(C)} = \ell + \sum_i |u_i|_C$ and $|\star|_\diamond = 1$.

NB: Evaluation in a finite model is *sound*. Cf. Gurevich, Goerdt.

Characterisation of definable functions

A function $f : \{0,1\}^* \to \{0,1\}$ is *representable* if there exists a program containing a function symbol $\mathtt{f} : \mathsf{L}(\mathsf{B}) \to \mathsf{B}$ with $f = [\![\mathtt{f}]\!]$.

Theorem: f is representable iff f is computable in time $O(2^{cn})$ for some c (here $n = |w|$). Equivalently f is computable on an $O(n)$ space-bounded Turing machine with unbounded stack (Cook 1972).

If

$W = \mathsf{L}(\mathsf{B}^{\otimes c}) \otimes \mathsf{B}^{\otimes q}$ represents c work tapes and finite control with q states. $S = \mathsf{B}^{\otimes s}$ represents stack symbols. Define helper functions

$$\texttt{init} : \mathsf{L}(\mathsf{B}) \to W$$

$$\texttt{output} : W \to \mathsf{L}(\mathsf{B})$$

$$\texttt{run} : W, S \to W$$

where $[\![\texttt{run}]\!](w, s)$ is the tape inscription+state of the machine when started on one-element stack s.

$$[\![\texttt{run}]\!](w, s) = \begin{cases} w', \text{ if } s \text{ is popped} \\ [\![\texttt{run}]\!]([\![\texttt{run}]\!](w', s_1), s), \text{ if } s_1 \text{ is pushed} \end{cases}$$

(w' = successor configuration)

Only if

Translate programs into malloc()-free C, in particular $\diamond$ as `void *`.

```
typedef ... C;

list_C cons(void *d, C hd, list_C tl)
{
  (list_C)d->hd = hd;
  (list_C)d->tl = tl;
  return (list_C)d;
}
```

Show using logical relation that for well-typed programs this translation yields the correct result.

Linearity is really needed: append(l, l) creates a circular list.

Further elaboration

- Read-only types: quest for laxer conditions than linearity which still guarantee that translation to C is correct. ESOP 02 with David Aspinall.
- Inference of $\diamond$-resource: given linear functional program (without $\diamond$) return integer linear program whose solutions yield annotations of the program with $\diamond$. Diploma thesis by Steffen Jost based on earlier work by Dilsun Kırlı.
- Automatic amortised space analysis for functional (POPL03) and object-oriented (ESOP06) programs (Jost & MH).

Higher-order functions

Add type former $A \multimap B$ and terms

$e_1^{A \multimap B} e_2^A : B$ Application; $\mathrm{FV}(e_1) \cup \mathrm{FV}(e_2) = \emptyset$

$\lambda x^A.e^B : A \multimap B$ Abstraction.

$[\![A \multimap B]\!] = [\![A]\!] \multimap [\![B]\!]$

Definition: *LFPL*$_\omega$is *LFPL* together with linear function spaces.

Example: Continuation passing style:

$$\mathtt{rev} : \mathsf{L}(A), \mathsf{L}(A) \multimap \mathsf{L}(A) \to \mathsf{L}(A)$$

$$\mathtt{rev}(\mathsf{nil}, k) = k\mathsf{nil}$$

$$\mathtt{rev}(\mathsf{cons}(d, h, t), k) = \mathtt{rev}(t, \lambda l.k(\mathtt{append}(t, \mathsf{cons}(d, h, \mathsf{nil}))))$$

Idea: $\mathtt{rev}(l, k) = k(\mathtt{reverse}(l))$.

Higher-order functions and size

With

$$|f|_{A \multimap B} = \sup\{|f(a)|_B - |a|_A \;\Big|\; a \in [\![A]\!]\}$$

functions continue to be non size increasing.

However, storing f takes more space than $|f|_{A \multimap B}$.

Structural recursion with HO functions

Theorem: (MH1999, Aehlig-Schwichtenberg 2000, Dal Lago& MH 2005):

The functions definable with structural recursion and HO functions are polynomial time computable.

Proof idea: semantic interpretation of programs (also higher-order ones) as time and space bounded algorithms. AS use normalisation.

Let us call that system (structural recursion formalised e.g. with iterators) $LFPL_{\mathrm{poly}}$

Resource monoid

Define M as the set of pairs (l, p) where $l \in \mathbb{N}$ and p is a unary polynomial.

Define a binary addition on M by $(l_1, p_1) + (l_2, p_2) = (l_1 + l_2, p_1 + p_2)$.

Define a partial ordering on M by $(l_1, p_1) \leq (l_2, p_2)$ if $l_1 \leq l_2$ and furthermore $p_1(x) \leq p_2(x)$ for all $x \geq l_2$.

$(M, +)$ is a monoid and $+$ is monotone w.r.t. $\leq$.

If $\alpha_1 = (l_1, p_1)$ and $\alpha_2 = (l_2, p_2)$ and $\alpha_1 \leq \alpha_2$ define $dist(\alpha_1, \alpha_2) = p_2(l_2) - p_1(l_2)$. Note that $dist(\alpha_1, \alpha_2) \in \mathbb{N}$.

The elements of the form $(0, p)$ form a submonoid M_0 of M.

LFPL space

An LFPL-space A comprises

- A set $|A|$ or A for short
- A relation $\mathbf{maj}_A \subseteq \mathit{Machines} \times M \times A$ such that:
  - for each $a \in |A|$ there exists t, α such that $t, \alpha\ \mathbf{maj}_A\ a$.
  - if $t, \alpha\ \mathbf{maj}_A\ a$ and $\beta \geq \alpha$ then $t, \beta\ \mathbf{maj}_A\ a$.
  - $e, \alpha \Vdash_A a$ implies $dist(0, \alpha) \geq |e|$ where $|e|$ is the size of machine e and 0 is $(0, 0) \in M$.
  - $t, \alpha_1\ \mathbf{maj}_A\ a_1$ and $t, \alpha_2\ \mathbf{maj}_A\ a_2$ implies $a_1 = a_2$ so that elements are uniquely characterised by their realisers.

Examples of LFPL-spaces

- $|\mathsf{B}| = \{\mathsf{tt}, \mathsf{ff}\}$, $t, \alpha \; \mathbf{maj}_{\mathsf{B}} \; x$ when t encodes x and $\alpha = (l, p)$ where $dist(0, \alpha) \geq |t|$.
- $|I| = \{0\}$, $t, \alpha \; \mathbf{maj}_{I} \; 0$ iff $dist(0, \alpha) \geq |t|$.
- $|\mathsf{L}(\mathsf{B})| = \{\text{lists of Booleans}\}$ and $t, \alpha \; \mathbf{maj}_{A} \; a$ iff t encodes a and $\alpha = (l, p)$ where $l \geq \mathrm{lh}(a)$ and $dist(0, \alpha) \geq |t|$.
- $|\Diamond| = \{0\}$, $t, \alpha \; \mathbf{maj}_{\Diamond} \; 0$ iff $\alpha \geq (1, 0)$ and $dist(0, \alpha) \geq |t|$.

LFPL-spaces as a category

A map from LFPL-space A to B is a function $f : |A| \longrightarrow |B|$ such that

- there exists $e \in$ *Machines* and $\phi \in M_0$ (not M!!). If we only find a $\phi \in M$ then f is an M-map, not a map.
- whenever $t, \alpha \ \mathbf{maj}_A \ a$ then $\{e\}(t), \beta \ \mathbf{maj}_B \ f(b)$ for some $\beta \leq \phi + \alpha$ and, moreover, *Time*$(\{e\}(t)) \leq dist(\beta, \phi + \alpha)$.

Examples of maps

The maps from L(B) to L(B) are precisely the non size-increasing polytime functions.

Suppose that $f : |\mathsf{L(B)}| \to |\mathsf{L(B)}|$ satisfies $\mathrm{lh}(f(a)) \leq \mathrm{lh}(a)$ and $f(a)$ is computable in time $p(\mathrm{lh}(a))$. Then f is a map from L(B) to L(B) by $(0, p)$ as follows: Suppose that e is an algorithm for f running in time p and put $\phi = (0, p)$. Suppose that $t, \alpha\ \mathbf{maj}_{\mathsf{L(B)}}\ a$. We may assume $t = (\mathrm{lh}(a), q)$ where $|t| \leq q(\mathrm{lh}(a))$.

Now $\{e\}(t), \alpha\ \mathbf{maj}_{\mathsf{L}(B)}$ and $dist(\alpha, \alpha + \phi) = p(\mathrm{lh}(a))$ which equals the runtime of $\{e\}(t)$.

The converse is clear.

LFPL-spaces as a model of $\boldsymbol{LFPL}_{\mathrm{poly}}$

LFPL-spaces form a symmetric monoidal closed category (thus model $\otimes, \multimap$) irrespective of the choice of the resource monoid.

To wit, $|A \otimes B| = |A| \times |B|$ and $\langle s, t\rangle, \gamma\ \mathbf{maj}_{A\otimes B}\ (a, b)$ when $\gamma \geq \alpha + \beta$ for some α, β such that $s, \alpha\ \mathbf{maj}_A\ a$ and $t, \beta\ \mathbf{maj}_B\ b$ and also $dist(0, \gamma) \geq |\langle s, t\rangle$.

We require here that $|\langle s, t\rangle = |s| + |t| + O(1)$.

Linear function space

We put $|A \multimap B|$ the set of all M-maps from A to B and

$e, \phi \; \mathbf{maj}_{A \multimap B} \; f$ if e, ϕ witness that f is an M-map and furthermore $dist(0, \phi) \geq |e|$.

In this way, LFPL-spaces become a symmetric monoidal closed category (model of multiplicative linear logic, linear lambda calculus) and this works for any resource monoid (obvious abstraction from M, M_0).

An LFPL-map from I to A is the same as an element a of $|A|$ such that there exists t and $\alpha \in M_0$ with $t, \alpha \; \mathbf{maj}_A \; a$.

An LFPL-map from I to $A \multimap B$ is the same as an LFPL-map from A to B.

Recursion Theorem

Proposition: cons : $\Diamond \multimap$ B $\multimap$ L(B) $\multimap$ L(B) and nil : $I \multimap$ L(B) are maps.

Theorem: When X is any LFPL-space, and $f : \Diamond \multimap$ B $\multimap X \multimap X$ then *rec*(f) defined by *rec*(f)(nil)(x) $= x$ and *rec*(f)(cons(d, b, a))(x) $= f(d)(b)(a)$(*rec*(f)(x)) is an LFPL map of type L(B) $\multimap X \multimap X$.

We will deduce this from a more general result.

Powering

Let A be an LFPL-space and $n \in \mathbb{N}$. We form a new LFPL-space $A^{(n)}$ by

$|A^{(n)}| = |A|$ and $t, \alpha\ \mathbf{maj}_{A^{(n)}}$ if there exists α' such that $t, \alpha'\ \mathbf{maj}_A\ a$ and $\alpha \geq \alpha' + \cdots + \alpha'$ (n summands).

If A_i is a family of LFPL-spaces indexed by $i \in I$, e.g., $I = \mathbb{N}$ or $I = \{\text{LFPL-spaces}\}$.

We form the LFPL-space $\forall i.A_i$ with $|\forall i.A_n| = \prod_{i \in I} A_i$ and $t, \alpha\ \mathbf{maj}_{\forall n.A_n}\ a$ if $t, \alpha\ \mathbf{maj}_{A_i}\ a_i$ for all $i \in I$.

Likewise, $\exists i.A_i$ with $|\exists i.A_n| = \sum_{i \in I} A_i$ and $t, \alpha\ \mathbf{maj}_{\exists n.A_n}\ (i, a)$ if $t, \alpha\ \mathbf{maj}_{A_i}\ a$. For cardinality reasons we have $\exists i.A_i$ only for $I = \mathbb{N}$.

Examples of LFPL-maps:

$$\forall m.\forall n.A^{(m+n)} \longrightarrow A^{(m)} \otimes A^{(n)}$$

$$\forall X.\forall Y.\forall Z.(X \otimes Y \multimap Z) \multimap (X \multimap Y \multimap Z)$$

Finally: The LFPL-bang

If A is an LFPL-space we form a new LFPL-space $!A$ with $|!A| = |A|$ and $t, \alpha \; \mathbf{maj}_{!A} \; a$ if there exists $\alpha_0 = (0, p) \in M_0$ such that $t, \alpha_0 \; \mathbf{maj}_A \; a$ and $\alpha \geq (0, (1 + x)p)$.

"!" is a functor and the following obvious functions are LFPL-maps:

$$!(A \otimes B) \leftrightarrow !A \otimes !B$$

$$\forall n.!A \otimes \Diamond^{(n)} \multimap A^{(n)} \otimes \Diamond^{(n)}$$

This allows us to *define*

$$\mathsf{L}(\mathsf{B}) = \exists n.\Diamond^{(n)} \otimes \forall X.X \multimap (\mathsf{B} \multimap X \multimap X)^{(n)} \multimap X$$

and similarly for other datatypes and recover constructors and iterators, e.g.,

$$\mathit{rec} : !(\Diamond \multimap \mathsf{B} \multimap X \multimap X) \multimap \mathsf{L}(\mathsf{B}) \multimap X \multimap X$$

HO functions & general recursion

$\texttt{Contrived} : A, A \multimap A \to A$

$e_{\texttt{Contrived}}(x, f) =$ match $\mathsf{a}(x)$ with $x' \otimes b \otimes \mathit{done} \Rightarrow$ if *done* then $f(x')$

else if b then $\texttt{Contrived}(x', \lambda y.\mathsf{g}(f(y)))$

else $\texttt{Contrived}(x', \lambda y.f(\mathtt{h}(y))$

where $\mathsf{a}, \mathsf{g}, \mathtt{h}$ are arbitrary function symbols of appropriate types.

Evaluating $\texttt{Contrived}(\lambda y.y, x)$ by term rewriting may require an arbitrary amount of space.

Is general recursion with higher-order linear functions Turing complete?

Do higher-order linear functions buy any extra expressive power beyond first-order functions?

Higher-order functions represent polynomial space

Theorem: Every function $f : \{0,1\}^* \to \{0,1\}$ computable in polynomial space with stack (equivalently *EXPTIME* can be represented in $LFPL_\omega$.) □

An element $w : \mathsf{L}(\mathsf{B}^{\otimes c})$ can store $c|w|$ bits. For p a polynomial we need a type S_p such that an element $w : S_p$ can store $p(|w|)$ bits.

If we had such a type S_p we could use it to store configurations of a p-space bounded TM and use general recursion as before to represent stack.

What does it mean to "store x bits"?

What should the type S_p be?

Notation

$W = [\![\mathsf{L}(\mathsf{B})]\!]$

$W_k = \{w \in W \mid |w| = k\}$

We identify W_k with integers $< 2^k$. E.g. $W_5 \ni [\mathsf{ff}, \mathsf{tt}, \mathsf{tt}, \mathsf{ff}, \mathsf{ff}] = 12$.

Using higher-order functions as store

Proposition: For any type S there is a program containing a symbol

$$\Phi : \mathsf{L}(S) \longrightarrow (\mathsf{L}(\mathsf{B}) \multimap \mathsf{L}(S)) \otimes \mathsf{L}(\mathsf{B})$$

such that $[\![\Phi]\!](l) = (f, w)$ implies

- $\mathsf{lh}(l) = \mathsf{lh}(w)$,
- whenever $\mathsf{lh}(w') = \mathsf{lh}(w)$ then $f(w') = l$

Thus, if S has size 0, we can store the information contained in a size n object (l) in a size 0 object (f). To retrieve the information we need to provide an arbitrary size n object (w') as a "catalyst".

Storage device

Let $s : \mathbb{N} \to \mathbb{N}$ be a function with $s(n) < 2^n$ and $k \in \mathbb{N}$ be a number. A (k, s)-*storage device* is given by the following data:

- a type S
- a family of subsets $S_n \subseteq [\![S]\!]$ for $n \in \mathbb{N}$.
- a represented function $\texttt{init} :\to S$,
- a represented function
  $\texttt{read} : \mathsf{L}(\mathsf{B}) \otimes \mathsf{L}(\mathsf{B}) \otimes S \to \mathsf{L}(\mathsf{B}) \otimes \mathsf{L}(\mathsf{B}) \otimes \mathsf{B} \otimes S$
- a represented function
  $\texttt{write} : \mathsf{L}(B) \otimes \mathsf{L}(\mathsf{B}) \otimes \mathsf{B} \otimes S \to \mathsf{L}(\mathsf{B}) \otimes \mathsf{L}(\mathsf{B}) \otimes S$.

such that, *please turn*

For all $n \in \mathbb{N}$ and $w, w_1, w_2, w_3 \in W_{kn}$, $a, a' \in W_n$ and $s \in S_n$ the following are satisfied:

- $[\![\texttt{init}]\!]() \in S_n$
- $[\![\texttt{read}]\!](w, a, s) = (w', a', b, s')$ implies $w' \in W_{kn}, a' \in W_n, s' = s$
- $[\![\texttt{write}]\!](w, a, b, s) = (w', a', s')$ implies $w' \in W_{kn}, a' \in W_n, s' \in S_n$
- $[\![\texttt{read}]\!](w_1, a, [\![\texttt{write}]\!](w_2, a, b, s).2.2).2.2.1 = b$ provided that $a < s(n)$
- $[\![\texttt{read}]\!](w_1, a, [\![\texttt{write}]\!](w_2, a', b, s).2.2) = [\![\texttt{read}]\!](w_3, a, s)$ provided that $a, a' < s(n)$ and $a \neq a'$.

Building storage devices

Fact: $S := \mathsf{B}^{\otimes c}$ can be extended to a $(0, \lambda n.c)$-storage device.

I.e., a constant number of bits is stored, no scratchpad is required.

Lemma: If S is a (k, s) storage device then $S' := \mathsf{L}(\mathsf{B}) \multimap \mathsf{L}(S)$ can be extended to a $(k+1, \lambda n.n \cdot s(n))$ storage device.

Corollary: If p is a polynomial of degree k then there exists a (k, p) storage device.

Representing polyspace + stack

Let $f : W \to \{0, 1\}$ be computable in space $p(n)$ plus stack.

We let S be a $(k, \lambda n.p(2kn))$ storage device.

Given input l : L(B) we first split l into w : L(B) and l' : L(B $\otimes$ B) so that $|w| = |l'| = |l|/2$ but l' contains the same information as l.

Using w as scratchpad our storage device can thus store $p(n)$ bits as required.

Upper bound on expressivity by dynamic programming

Theorem: If f is representable with general recursion and HO-functions then $f \in \mathrm{DTIME}(2^{n^{O(1)}})$.

Proof: Intuitions:

- When evaluating $\mathtt{f}(w)$ all intermediate first-order results should be bounded (length-wise) by $|w|$, so a finite model should suffice.
- Assuming nondeterminism a linear (apply once) function can be represented by an argument-result pair.

Let $N \in \mathbb{N}$ be a fixed parameter. We define finite sets $(\!|A|\!)$ for types A inductively as follows.

$$(\!|\Diamond|\!) = \{0\} \qquad (\!|\mathsf{B}|\!) = \{\mathsf{tt}, \mathsf{ff}\}$$

$$(\!|\mathsf{L}(A)|\!) = \{w \in (\!|A|\!)^* \mid |w|_{\mathsf{L}(A)} \leq N\} \qquad (\!|A \multimap B|\!) = (\!|A|\!) \times (\!|B|\!)$$

$$(\!|A \otimes B|\!) = \{x \in (\!|A|\!) \times (\!|B|\!) \mid |x|_{A \otimes B} \leq N\}$$

$$(\!|\Gamma|\!) = \{\eta \mid \forall x \in \mathrm{dom}(\Gamma).\eta(x) \in (\!|\Gamma(x)|\!) \wedge |\eta|_\Gamma \leq N\}$$

If $\Gamma \vdash e : A$ then $(\!|e|\!) \subseteq (\!|\Gamma|\!) \times (\!|A|\!)$. E.g.

$$(\!|e_1 e_2|\!)(\eta_1, \eta_2) = \{y \mid \exists x.x \in (\!|e_2|\!)(\eta_2) \wedge (x, y) \in (\!|e_1|\!)(\eta_1)\}$$

General recursion rendered by (finite) iteration of monotone operator.

If $R \subseteq A \times B$ and $U \subseteq A$ write $R(A)$ for $\{b \mid \exists a \in A.aRb\}$

Main result about $(\!|-|\!)$

Recall that $[\![-]\!]$: *Types* $\rightarrow$ *cpos*; *Terms* $\rightarrow$ *cts fctns*

In particular $[\![A \multimap B]\!] = [\![A]\!] \rightarrow [\![B]\!]_\perp$.

Proposition: Suppose that $f : \mathsf{L(B)} \rightarrow \mathsf{L(B)}$ in some program and let $l \in W$ where $|l| \leq N$ (recall that N is a fixed parameter). Notice that in this case $l \in [\![\mathsf{L(B)}]\!]$ as well as $l \in (\!|\mathsf{L(B)}|\!)$. Then

$$l(\!|f|\!)l' \iff [\![f]\!](l) = l'$$

for all $l' \in W$.

This will follow from a more general property proved by induction on typing derivations.

Correspondence between $[\![-]\!]$ and $(\!|-|\!)$

For each type A and $n \in \mathbb{N}$ define relation
$\sim_A^n \subseteq [\![A]\!] \times \{U \subseteq (\!|A|\!) \mid U \neq \emptyset \wedge |U|_A \leq n\}$ where $|U|_A = \max_{x \in U} |x|_A$.

Intention: $x \sim_A^n U$ means U consists of size $\leq n$ elements all encoding x.

Clauses for $\sim^n_A$

$$\mathsf{tt} \sim^n_{\mathsf{B}} \{\mathsf{tt}\} \quad \mathsf{ff} \sim^n_{\mathsf{B}} \{\mathsf{ff}\} \quad 0 \sim^{n+1}_{\Diamond} \{0\} \quad [] \sim^n_{\mathsf{L}(A)} \{[]\}$$

$$(a, b) \sim^n_{A \otimes B} W \iff$$
$$\exists n_1, n_2, U, V. n_1 + n_2 = n$$
$$\wedge\ a \sim^{n_1}_A U \wedge b \sim^{n_2}_B V \wedge W = U \times V$$

Clauses for $\sim^n_A$

$$f \sim^n_{A \multimap B} U \iff$$
$$\forall n_1, x. n + n_1 \leq N$$
$$\wedge\ x \sim^{n_1} V \Rightarrow f(x) \sim^{n+n_1}_B U(V)$$

$$x :: l \sim^n_{\mathsf{L}(A)} W \iff$$
$$\exists n_1, n_2, U, V. n_1 + n_2 + 1 \leq n$$
$$\wedge\ x \sim^{n_1}_A U \wedge l \sim^{n_2}_{\mathsf{L}(A)} V \wedge W = U :: V$$

Formally extend by $\bot \sim^n_A \emptyset$.

Main Lemma

Suppose given denotations for function symbols $\psi(f) \subseteq [\![\vec{A}]\!] \to [\![B]\!]\perp$ and $\rho f \subseteq (\!|\vec{A}|\!) \times (\!|B|\!)$ such that whenever $n = n_1 + \cdots + n_r \leq N$ one has $\bigwedge_i u_i \sim^{n_i}_{A_i} U_i \Longrightarrow \psi(f)(u_1, \ldots, u_r) \sim^n_B \rho(f)^n(U_1, \ldots, U_r)$

Lemma: If $\Gamma \vdash e : B$ and $\eta \sim^n_\Gamma \vec{U}$ then $[\![e]\!]_{\psi,\eta} \sim^n_B (\!|e|\!)_{\rho,\vec{U}}$.

Proof by induction on typing derivations.

For a fixed set of types $\mathcal{T}$, e.g., those occurring in a given program there is a polynomial p such that $\log \#(\!|A|\!) \leq p(N)$ for all $A \in \mathcal{T}$.

A value table for $(\!|\mathtt{f}|\!)$ takes space $2^{p(N)}$.

By maintaining space for two value tables for $(\!|\mathtt{f}|\!)$ we can compute $(\!|\mathtt{f}|\!)$ in time $2^{p(N)}$.

Hence all those functions are computable in time $O(2^{p(N)})$.

To know more See papers on www.tcs.ifi.lmu.de/ mhofmann or citeseer.

Results on $LFPL_\omega$ (General recursion and higher-order linear functions): POPL02 — The strength of non-size-increasing computation. This paper also shows that all polytimes functions are representable in $LFPL_{\mathrm{poly}}$.

Results on $LFPL_{\mathrm{poly}}$ (Definable functions contained in PTIME, extensions for PSPACE, divide and conquer algorithms etc.): Non size-increasing polynomial time computation (Inf. & Comp. 2001).

New proof using resource monoids: with Ugo Dal Lago. Quantitative models and Implicit Complexity. FSTTCS 05. Full version on Ugo's home page. Also contains soundness proofs for LLL, ELL, SLL.

Inference of Diamond types: Static Prediction of Heap Space Usage for First-Order Functional Programs. POPL03. With Steffen Jost.