On space efficiency of Krivine’s abstract machine and Hyland-Ong games

Kazushige Terui

terui@nii.ac.jp

National Institute of Informatics, Tokyo
Laboratoire d'Informatics de Paris Nord, CNRS
ICC

- Intensional Computational Complexity (ICC)
- algorithms (eg. $\lambda$ terms)
- evaluation mechanisms (eg. $\beta$-reduction)
Space sensitive Turing machines

- **Input tape** (read only)
- **Work tapes** (read/write)
- **Output tape** (write only)

- Space = the num of cells used on worktapes.
- The input and output can be significantly larger than the space: If $M$ works in space $f(n)$ then the output is of size $O(2^{f(n)})$.
- Given two Turing machines $M$, $N$, we want to compose them. How?
Space sensitive Turing machines

- **Functional** composition is not good:
  
  If $M, N$ work in space $f(n), g(n)$, then $M; N$ works in space $O(f(n) + g(2f(n)) + 2f(n))$.

- In particular, logspace functions **do not** compose.

- Composition must be **interactive**: $M \Leftrightarrow N$ works in $O(f(n) + g(2f(n)))$.

- From functional to interactive computation!
Towards a logical theory of computational complexity

- Shortcomings of Turing Machines:
  - Poor data structures (only tapes)
  - Poor control mechanisms (only transitions)
  - Ad hoc constructions: no canonical way of composition
  - Results in lack of beautiful mathematical theory

- Logic and lambda calculus
  - Rich data structures and controls
  - Canonical composition $T \circ U = \lambda x. T(U(x))$
  - Hope to make complexity theory more mathematical
Space in lambda calculus

- Composition does not preserve good space bounds when using $\beta$-reduction, because $\beta$-reduction is not interactive!

- Interactive evaluation mechanisms
  - Geometry of Interaction
  - Krivine’s abstract machine (KAM)
  - Game semantics
KAM implements weak head linear reduction:

\[(\lambda \cdots \lambda x \cdots .x\tilde{U}) \cdots T \cdots \Rightarrow (\lambda \cdots \lambda x \cdots .T\tilde{U}) \cdots T \cdots\]

GoI with “jumps” under \(A \Rightarrow B = !(A \multimap B)\) (Danos-Regnier 94)

Syntax:

Terms \( t ::= x \mid \lambda x. t_1 \mid t_1 t_2 \)

Environments \( \rho ::= [x_1 \mapsto t_1 \rho_1, \ldots, x_n \mapsto t_n \rho_n] \)

Closures \( t\rho \)

Stacks \( \pi ::= t_1 \rho_1 : \cdots : t_n \rho_n \)

States \( S ::= (t\rho, \pi) \)
**KAM**

- **Initial state:** \((t[], \text{nil})\)

- **Transitions**

  \[
  (x\rho, \pi) \rightarrow (\rho(x), \pi) \quad \text{if} \ x \in \text{Dom}(\rho)
  \]

  \[
  ((tu)\rho, \pi) \rightarrow (t\rho, u\rho : \pi)
  \]

  \[
  ((\lambda x.t)\rho, u\rho' : \pi) \rightarrow (t\rho[x \mapsto u\rho'], \pi)
  \]

- **Terminations**

  \[
  (x\rho, \pi) \text{ with } x \notin \text{Dom}(\rho) \quad \rightarrow \quad \text{Success with output } x
  \]

  \[
  ((\lambda x.t)\rho, \text{nil}) \quad \rightarrow \quad \text{Failure}
  \]

- **Not space efficient:** \(\rho\) contains a lot of redundant bindings.
Optimized KAM

- Optimized transitions

\[(x \rho, \pi) \rightarrow (\rho(x), \pi) \quad \text{if } x \in \text{Dom}(\rho)\]

\[((tx) \rho, \pi) \rightarrow (t(\rho|_t), \rho(x) : \pi) \quad \text{if } x \in \text{Dom}(\rho)\]

\[((tu) \rho, \pi) \rightarrow (t(\rho|_t), u(\rho|_u) : \pi) \quad \text{otherwise}\]

\[((\lambda x.t) \rho, u \rho' : \pi) \rightarrow (t\rho[x \mapsto u \rho'], \pi)\]

- \(\rho|_t\) is a restriction of \(\rho\) to the free variables of \(t\).
Example

Suppose \((K[], \text{nil}) \overset{*}{\longrightarrow} (\lambda z. z, \text{nil})\), \(\overline{2} = \lambda x.f(fx)\).

\((\overline{2}Kc, \text{nil})\)
Example

Suppose \((K[], \text{nil}) \longrightarrow^* (\lambda.z.z, \text{nil}), \ \bar{2} = \lambda f.x.f(fx)\).

\((\bar{2}Kc, \text{nil})\)
\((\bar{2}K, c)\)
Example

Suppose \((K[], \text{nil}) \xrightarrow{\ast} (\lambda z. z, \text{nil}), \quad \overline{2} = \lambda x. f(fx)\).

\((\overline{2}Kc, \text{nil})\)
\((\overline{2}K, c)\)
\((\overline{2}, K : c)\)
Example

Suppose $(K[], \text{nil}) \xrightarrow{*} (\lambda z. z, \text{nil}), \quad \overline{2} = \lambda f. f(fx).$

$(\overline{2}Kc, \text{nil})$
$(\overline{2}K, c)$
$(\overline{2}, K : c)$
$(\lambda x. f(fx)[f \mapsto K], c)$
Example

Suppose $(K[], \text{nil}) \xrightarrow{*} (\lambda z. z, \text{nil}), \quad \overline{2} = \lambda f. f(fx)$.

$(\overline{2}Kc, \text{nil})$
$(\overline{2}K, c)$
$(\overline{2}, K : c)$
$(\lambda x.f(fx)[f \mapsto K], c)$
$(f(fx)[f \mapsto K, x \mapsto c], \text{nil})$
Example

Suppose \((K[], \text{nil}) \rightarrow^* (\lambda z.z, \text{nil})\), \(\overline{2} = \lambda f. f(fx)\).

\((\overline{2}Kc, \text{nil})\)
\((\overline{2}K, c)\)
\((\overline{2}, K : c)\)
\((\lambda x.f(fx)[f \mapsto K], c)\)
\((f(fx)[f \mapsto K, x \mapsto c], \text{nil})\)
\((f[f \mapsto K], fx[f \mapsto K, x \mapsto c])\)
Example

Suppose $(K[], \text{nil}) \longrightarrow^* (\lambda z.z, \text{nil}), \quad \overline{2} = \lambda f.x.f(fx)$.

$(\overline{2}Kc, \text{nil})$
$(\overline{2}K, c)$
$(\overline{2}, K : c)$
$(\lambda x.f(fx)[f \mapsto K], c)$
$(f(fx)[f \mapsto K, x \mapsto c], \text{nil})$
$(f[f \mapsto K], fx[f \mapsto K, x \mapsto c])$
$(K, fx[f \mapsto K, x \mapsto c])$
Example

Suppose $(K[], \text{nil}) \rightarrow^* (\lambda z.z, \text{nil}), \quad \bar{2} = \lambda f.x.f(fx)$.

$(\bar{2}Kc, \text{nil})$
$(\bar{2}K, c)$
$(\bar{2}, K : c)$
$(\lambda x.f(fx)[f \mapsto K], c)$
$(f(fx)[f \mapsto K, x \mapsto c], \text{nil})$
$(f[f \mapsto K], fx[f \mapsto K, x \mapsto c])$
$(K, fx[f \mapsto K, x \mapsto c])$
$(\lambda z.z, fx[f \mapsto K, x \mapsto c])$
Example

Suppose $(K[], \text{nil}) \rightarrow^* (\lambda z.z, \text{nil}), \quad 2 = \lambda f x.f(f x)$.

$(2 K c, \text{nil})$
$(2 K, c)$
$(2, K : c)$
$(\lambda x.f(f x)[f \mapsto K], c)$
$(f(f x)[f \mapsto K, x \mapsto c], \text{nil})$
$(f[f \mapsto K], f x[f \mapsto K, x \mapsto c])$
$(K, f x[f \mapsto K, x \mapsto c])$
$(\lambda z.z, f x[f \mapsto K, x \mapsto c])$
$(f x[f \mapsto K, x \mapsto c], \text{nil})$
Example

Suppose $(K[], \text{nil}) \xrightarrow{\ast} (\lambda z.z, \text{nil}), \quad \overline{2} = \lambda f.x.f(fx)$.

$(\overline{2}Kc, \text{nil})$
$(\overline{2}K, c)$
$(\overline{2}, K : c)$
$(\lambda x.f(fx)[f \mapsto K], c)$
$(f(fx)[f \mapsto K, x \mapsto c], \text{nil})$
$(f[f \mapsto K], fx[f \mapsto K, x \mapsto c])$
$(K, fx[f \mapsto K, x \mapsto c])$
$(\lambda z.z, fx[f \mapsto K, x \mapsto c])$
$(fx[f \mapsto K, x \mapsto c], \text{nil})$
$(f[f \mapsto K], c)$
Example

Suppose \((K[], \text{nil}) \rightarrow^* (\lambda z. z, \text{nil})\), \(\bar{2} = \lambda f x. f(f x)\).

\((\bar{2}Kc, \text{nil})\)
\((\bar{2}K, c)\)
\((\bar{2}, K : c)\)
\((\lambda x. f(f x)[f \mapsto K], c)\)
\((f(f x)[f \mapsto K, x \mapsto c], \text{nil})\)
\((f[f \mapsto K], f x[f \mapsto K, x \mapsto c])\)
\((K, f x[f \mapsto K, x \mapsto c])\)
\((\lambda z. z, f x[f \mapsto K, x \mapsto c])\)
\((f x[f \mapsto K, x \mapsto c], \text{nil})\)
\((f[f \mapsto K], c)\)
\((K, c)\)
Size of a state

Subterm property: if \((t[], \text{nil}) \xrightarrow{*} S\), all terms occurring in \(S\) are subterms of \(t\).

Allows us to think of each state as made of pointers rather than actual terms.

\(\#S = \) the number of pointers in \(S\):

\[
\begin{align*}
\#\rho &= \#\rho_1 + \cdots + \#\rho_n + n \quad \text{if} \ \rho = [x_1 \mapsto t_1 \rho_1, \ldots, x_n \mapsto t_n \rho_n] \\
\#\pi &= \#\rho_1 + \cdots + \#\rho_n + n \quad \text{if} \ \tau_1 \rho_1 : \cdots : \tau_n \rho_n \\
\#(t\rho, \pi) &= \#\rho + \#\pi + 1
\end{align*}
\]

In particular, \(\#(t[], \text{nil}) = 1\).

KAM evaluates \(t\) with \(n\) pointers if for any intermediate state \((t[], \text{nil}) \xrightarrow{*} S\), \(\#S \leq n\).
TM space-efficiently simulates KAM

- Naively, TM requires of $\log n$ overhead to simulate KAM:
  if KAM evaluates $t$ with $n$ pointers, then $M(t)$ works in space $O(n \cdot \log |t|)$.

- We are interested in terms $td$, where $t$ (program) is fixed while $d$ (input) is varying.

- Consider a huge alphabet

\[ \Sigma = \{ u \mid u \text{ is a subterm occ. of } t \} \cup \{0, 1\} \]
TM space-efficiently simulates KAM

t satisfies the input condition if in any intermediate state \((td[], \text{nil}) \rightarrow^* S\), the number of pointers pointing subterms of \(d\) is constant (independent of \(d\)).

Analogy: the num of heads on input tape is fixed.

Theorem: There is a TM \(M\) which simulates KAM linearly: if KAM evaluates \(td\) with \(f(|d|)\) pointers, \(t\) satisfies the input condition and \(f(x) \geq \log x\), then \(M(t)\) works in space \(O(f(|d|))\).
Data structures

- **Booleans:** \( t = \lambda xy.x, \ f = \lambda xy.y \)

- **Scott numerals:** \( w \) for \( w \in \{0, 1\}^* \)

\[
\varepsilon = \lambda x_\varepsilon x_0 x_1 . x_\varepsilon
\]
\[
0 \cdot w = \lambda x_\varepsilon x_0 x_1 . x_0 w
\]
\[
1 \cdot w = \lambda x_\varepsilon x_0 x_1 . x_1 w
\]

- Purely linear. Successor, predecessor and conditional are also linear. Useful for **worktapes**.

- What are **read-only/write-only tapes**?

- They should be **interactive entities** (analogy: Book vs Dialogue)
Data structures

**Databases:** For \( d \in \{0, 1\}^{2^n} \), define a term \( d : \text{Word} \rightarrow \text{Bool} \) s.t.

Given \( w \in \{0, 1\}^n \), \( dw \) returns the \( w \)th bit of \( d \).

\[
\begin{align*}
  d & = \lambda w. w \Omega \Omega \quad \text{if } d = i \in \{0, 1\} \\
  & = \lambda w. w \Omega d_0 d_1 \quad \text{if } d_1 = i_{2^n} \cdots i_{2^{n-1}+1}, \\
  & \quad d_0 = i_{2^n-1} \cdots i_1, \ d = d_1 d_0
\end{align*}
\]
KAM space-efficiently simulates TM

- Theorem: For any TM $M$, there is a term $t_M$ s.t.
  - Given $d \in \{0, 1\}^*$, $t_M \cdot d \cdot \cdot \cdot \cdot$ returns $\cdot \cdot \cdot \cdot \cdot \cdot$ or $\cdot \cdot \cdot \cdot \cdot \cdot$ according to the output of $M(d)$.
  - When $M$ works in space $f(n) \geq \log n$, KAM evaluates $t_M \cdot d \cdot \cdot \cdot \cdot$ with $O(f(n))$ pointers.

- Now the space hierarchy theorem implies
  - There is no TM $M$ that evaluates $t_M \cdot d$ in space $O(f(n)^\epsilon)$ with $\epsilon < 1$.

- Corollary: There is no evaluator that is uniformly (i.e. for all terms) and significantly (i.e. super-linearly) more space-efficient than KAM.
Composition

Let $M, N$ be transducer TMs which work in space $f(n), g(n)$ and produce outputs of size $2^{f(n)}$ and $2^{g(n)}$.

$$t_M : \text{Word} \to \text{Bool} \rightarrow \text{Word} \to \text{Bool}$$

$$t_N : \text{Word} \to \text{Bool} \rightarrow \text{Word} \to \text{Bool}$$

$$t_M \circ t_N : \text{Word} \to \text{Bool} \rightarrow \text{Word} \to \text{Bool}$$

$(t_N \circ t_M)\text{dw} \ast_0 \ast_1$ returns the $w$th bit of $N \circ M(d)$.

KAM evaluates it with $O(f(|d|) + g(|d'|))$ pointers, where $d'$ is the output of $M(d)$.

KAM + canonical composition simulates the interactive composition of TMs!
Moral

<table>
<thead>
<tr>
<th></th>
<th>Turing Machines</th>
<th>$\lambda$-calculus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition</td>
<td>non canonical</td>
<td>canonical</td>
</tr>
<tr>
<td>Evaluation</td>
<td>canonical</td>
<td>non canonical</td>
</tr>
</tbody>
</table>

In $\lambda$-calculus, TIME-SPACE tradeoff shows up at the stage of evaluation:

- KAM is space-efficient
- CBV seems to be time-efficient (cf. Dal Lago-Martini).

Work to be done: identify a subclass of $\lambda$-terms (via a type system) on which space bounds are well preserved by KAM composition.
Warning

There is no uniformly more efficient evaluator than KAM.

KAM is like a student Bob with score

<table>
<thead>
<tr>
<th>Math</th>
<th>Science</th>
<th>Grammer</th>
<th>Latin</th>
<th>Music</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

Although there is no uniformly better student than Bob, this does not mean Bob is the best student.
From KAM to HO games

Empirically, KAM is good at "tall" terms:

\[ t_n = (\lambda f \cdot f^n(x))(\lambda y.y)^* \]

KAM evaluates \( t_n \) with \( O(1) \) pointers.

KAM is not good at "wide" terms:

\[ u_n = (\lambda x_1 \cdots \lambda x_n.x_n)^*_{1 \cdots n} \]

KAM evaluates \( u_n \) with \( O(n) \) pointers.

For simply typed terms with unbounded width and fixed rank, HO games seem to be more efficient.
HO games

- A fully abstract model of PCF (Hyland-Ong 00). Successfully applied to various programming languages.
- Useful for compositional and higher-order model checking (Ghica-Mckusker 00, Ong 04, etc.)

\[
\begin{align*}
\text{Types} & \quad = \quad \text{Arenas} \\
\text{Terms} & \quad = \quad \text{Strategies} \\
\text{Computation} & \quad = \quad \text{play}
\end{align*}
\]

- Play = interactive composition of two strategies
- Composition is effective (cf. Ong 04). Corresponds to Pointer Abstract Machine (Danos-Herbelin-Regnier 96).
Ranks and complexity

- The rank of a type: \( \text{rank}(\tau) = 0, \)
  \[ \text{rank}(A \rightarrow B) = \max(\text{rank}(A) + 1, \text{rank}(B)) \]

- A term is at rank \( n \) if all of its subterms have rank \( n \) types.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Quantative</th>
<th>Qualitative</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>boolean formulas</td>
<td>(open) addition</td>
</tr>
<tr>
<td></td>
<td>( \neg b = \lambda xy. b^{t\rightarrow t} y x )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>boolean circuits</td>
<td>multiplication</td>
</tr>
<tr>
<td></td>
<td>( (\lambda x.F(x, \ldots, x)^{\text{bool}\rightarrow\text{bool}} G(y) )</td>
<td>( \overline{n} \circ \overline{m} )</td>
</tr>
<tr>
<td>3</td>
<td>QBFs ( \forall x.F(x) = )</td>
<td>exponentiation</td>
</tr>
<tr>
<td></td>
<td>( (\lambda X.X(0) \wedge X(1))^{\text{bool} \rightarrow \text{bool}} \rightarrow \text{bool} \lambda x.F(x) )</td>
<td>( (\overline{n})^{\text{int} \rightarrow \text{int}} \overline{m} )</td>
</tr>
</tbody>
</table>
Ranks and complexity

Normalization problem:

Given two terms $t, u$ where $u$ is normal, does $t$ reduce to $u$?

(Schubert 2001) shows that this problem is

1. in $DTIME(n \log n)$ for rank 1 terms
2. P-complete for rank 2 terms
3. PSPACE-complete for rank 3 terms

Hardness is easy. Membership is difficult, since $\beta$-reduction does not work.

Schubert uses graph rewriting. We use HO games.
HO games for simple types

- **Arena** for type $A$:
  
  - **Moves** = occurrences of atomic types in $A$
  - **P-moves** = negative occurrences
  - **O-moves** = positive occurrences
  - **Initial move** = $\text{tail}(A)$

  $$\text{tail}(B) \vdash \text{tail}(C)$$ where $C$ is an immediate subformula of $B$

- **Legal play**: a PO-alternating pointing sequence like

  $$s = m_0 m_1 \cdots m_j \cdots m_i \cdots$$

  - $m_0$ is the initial O-move
  - each $m_i$ ($1 \leq i$) is justified by a preceding move $m_j$
    
    $$(m_j \vdash m_i, 0 \leq j < i).$$
Let’s play the game!

- **P-view**: a legal play in which any O-move is justified by the move immediately before.

- Given an arbitrary legal play \( s \), a P-view \( [s] \) can be extracted.

- **Innocent strategy (as view function)**: a P-deterministic tree made of P-ending P-views.

- Every closed normal term \( t : A \) can be interpreted by an innocent strategy.

- **HO-dialogue**: Given innocent strategies \( \sigma : A \rightarrow B \) and \( \tau : A \),
  1. Start with the initial move \( m_0 = \text{tail}(B) \)
  2. Expand an odd length play \( s \) to \( s \cdot m \) such that \( [s] \cdot m \in \sigma \)
  3. Expand an even length play \( m_0 \cdot s \) to \( m_0 \cdot s \cdot m \) such that \( [s] \cdot m \in \tau \)
A lemma in game semantics

- **Difficulty:** HO game is **history sensitive**. But recording the whole history leads to \( \text{TIME} = \text{SPACE} \).

- **Lemma** (taught by P. Boude): For any legal play satisfying P- and O-visibility

\[
s = \cdots m_1 \cdots \cdots m_2 \cdots \cdots
\]

if \( m_2 \) justifies no moves, it never happens that

\[
s = \cdots m_1 \cdots \cdots m_2 \cdots \cdots
\]

- So one can safely contract \( s \) to

\[
s' = \cdots m_1 m_2 \cdots
\]

and continue the play.
Complexity of Rank 3 games

Theorem: Given a term $t$ at rank 3 (which is an applicative combination of closed normal terms), the length of play can be kept within $O(|t|)$.

Corollary: One can determine the head variable of $nf(t)$ in polynomial space via HO games.

Work to be done: What about rank $\geq 4$? Do we have a hierarchy result as in Kristiansen’s talk?
Conclusion

- Lambda calculus admits a canonical composition $M \circ N$
- **KAM** is optimal for simulating TM.
- Composition works nicely with KAM.
- (KAM allows natural programming eg. boolean matrix closure)
- **HO games** seem to be good at least for rank bounded terms.
- Good understanding of game semantics leads to clever space management. **Game semantics may implicitly explain complexity.**