Notes on HORS

William Blum

May 23, 2008

1 Notes on HORS

What are the infinite words languages generated by HO recursion scheme?

At order 0, do we get the omega-regular languages? (*i.e.* languages of infinite words recognized by Buchi automata.)

At order 1, what do we get? Context-free languages of infinite words? Is there such a thing?

1.1 Urzycyn language U

U is not context-free. The standard pumping lemma for context-free language (Bar-Hillel lemma) cannot be used to prove it. Instead one must Ogden's lemma, a a stronger version of the pumping lemma:

Let n be a number ¿ 0.

Take $w = [[^{n+1}]^n][*^{n+3}$.

where the underlined] denotes the n distinguished position of the Ogden Lemma.

Then w admits no decomposition that would allow to pump two (or even more) subwords at the same time. Hence by Ogden Lemma, U is not context free.

It is not even possible to decompose w so that three or more subwords can be pumped at the same time!

This suggests the following conjecture:

Conjecture: Ogden pumping lemma for n-PDA Let L be a language recognized by an order-n PDA A. Then there is a number n¿0 such that for all word w with $|w| > 0$ and all marking of at least n distinguished positions in w, there is a decomposition $w = up0u2p1...unpnun + 1$ such that - the word p0 p1 ... pn contains at least one marked position - the word p0 u2 p1 ... un pn contains at most n marked position - for all i¿=0, $up0^iu2p1^i...unpn^iun + 1$ is in L.

1.2 Conjecture

The language L defined as follows is intrinsically unsafe (for any order).

$\Sigma = \{a, b, c\}$

$L = \{w *_1^{(|w|_b - |w|_a)} *_2^{(|w|_c - |w|_a)} \mid w \in \Sigma^* \wedge |w|_b \geq |w|_a \wedge |w|_c \geq |w|_a\}$

Lemma Let A be a n-PDA then there is a n+1-PDA recognizing the reverse of L(A).

Proof: just replace the production of terminals by a $push_1$ of a corresponding element on the stack followed by a $push_{n+1}$. When the end of string symbol is emitted, the reversed string can be retrieved using the following algorithm:

while stack is not empty do emit top1(s) pop_{n+1} done EOP

1.3 safe HORS to HO-PDA

Hague, Murawsky, Ong and Serre proposed an algorithm that transforms a given order-n recursive scheme G to an order-n collapsible pushdown automaton (CPDA) A_G that computes the value tree

of G (i.e. the tree generated by G); precisely, A_G computes exactly the traversals over the computation tree of G. (The paper is at `http://users.comlab.ox.ac.uk/luke.ong/publications/stoc07-long.pdf`, the relevant part is section 5).

We prove that if G is safe, then A_G is a pushdown automaton (of order n). This gives a new proof of the fact that order-n pushdown automata compute trees generated by order-n safe recursion schemes, independent of the original KNU02 approach. (There is a similar conjecture/result in the other direction.)

Proof idea: In fact for incrementally-bound computation trees, the n-CPDA that computes the traversals is such that only links created with the $push_1$ operation will ever be used by a collapse operation later on. Links that are created during the duplication of a stack by the operations $push_j$ (for j¿=2) can all be ignored. Hence the collapse operation can safely be simulated with the operation $pop_{n-ord(u)+1}$ where u is the top_1 lambda-node.

1.4 2

For a grammar G of order 2, the eta-long normal form of G introduces variables of order 0 only.

Proof: First necessarily, the variables introduced are of order 1 at most (otherwise the eta-expanded grammar would not be of order 2).

Suppose a subterm M is eta-expanded to $\lambda u.Mu$ where u is a variable of order 1. Then this implies that M is of order 2. Consequently M is a non-terminal! Indeed any variable or an applicative term is of order 1 at most in an order 2 grammar.

However non-terminals of order 2 cannot occur at operand position, otherwise this would imply that the grammar is of order 3. Hence M is not eta-expanded when computing the eta-long normal form, which contradicts the hypothesis.

1.5 2-DPDA equivalence problem:

Question: Is 2-DPDA equivalence decidable? Equivalence of 1-DPDA was shown to be decidable but it is still not known whether it is the case at higher orders. Can we encode the set of traversals of a Safe PCF_2 term into a 2-DPDA?

1.5.1 Does Undecidability of Finitary Safe PCF_2 implies that of 2-DPDA equivalence?

The simply-typed λ-calculus augmented with the Y-combinator written $\lambda^{\rightarrow} + Y$ is as expressive as recursion schemes in the sense that any *ground type* term of $\lambda^{\rightarrow} + Y$ with redexes of order n at most can be encoded into an order-n recursion scheme (furthermore, the transformation is such that safe terms correspond to safe recursion schemes). For example, take the ground type term $s^n z$ where $s : o \rightarrow o$, $z : o$ for some natural number n. This term can be transformed into the recursion scheme containing a single equation $S = s^n z$. The value tree of this recursion scheme is a flat tree over the alphabet $\Sigma = \{s : o \rightarrow o, z : o\}$ where nodes are labelled s and leaves are labelled z.

Suppose that we are able to do the same for any PCF term, then we could convert SIM into an equivalent safe recursion scheme. And since order-n safe recursion schemes are as expressive as n-DPDA[1], we could build a 2-DPDA computing SIM. So the Queue program terminates iff this DPDA is equivalent to the constant representing termination of the Queue program. But since Queue-Halting is undecidable, this would imply the undecidability of 2-DPDA equivalence!

Instead of trying to encode PCF terms into recursion schemes, we can look at the equivalent problem of simulating PCF constructs in $\lambda^{\rightarrow} + Y$. It is well-known that the Church Numeral representation does not permit to represent the predecessor function in $\lambda^{\rightarrow}$ [1]. It is however possible to represent the predecessor function provided that we allow a different representation of input and output numbers [1]. Unfortunately, the substraction function is not definable in $\lambda^{\rightarrow}$

[1]Ssee [2] where an effective transformation in both directions is given.

(equivalently the equality test function $\lambda xy.\mathsf{if}\ x = y\ \mathsf{then}\ u\ \mathsf{else}\ v$ is not definable) [1]. Hence we know that PCF cannot be simulated into the simply-typed λ-calculus without the Y combinator.

Would the Y combinator help to simulate the PCF constructs? It does not seem so. Indeed, as we have just seen, substraction – which is definable in PCF using recursion and the conditional operator – is not definable in $\lambda^{\rightarrow}$. Throwing in the recursion can be useful only if we have some destruction function, such as the predecessor function. Indeed, only this can permit us to make recursive call with decreasing parameters (which is necessary to guarantee termination). Therefore we are bound to use a numeral representation that can differ from input to output. In other words, the type of the input may differ from the type of the output. But with such difference in the representation, recursion cannot be used anymore! Note that this problem is due to the fact that PCF uses simple types. With other kinds of types, for instance second-order types, it would be possible to encode the substraction operation.

References

[1] Steven Fortune, Daniel Leivant, and Michael O'Donnell. The expressiveness of simple and second-order type structures. *J. ACM*, 30(1):151–185, 1983.

[2] T. Knapik, D. Niwiński, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FOSSACS'02*, pages 205–222. Springer, 2002. LNCS Vol. 2303.

[3] C.-H. L. Ong. Observational equivalence of third-order Idealized Algol is decidable. In *Proceedings of IEEE Symposium on Logic in Computer Science, 22-25 July 2002, Copenhagen Denmark*, pages 245–256. Computer Society Press, 2002.