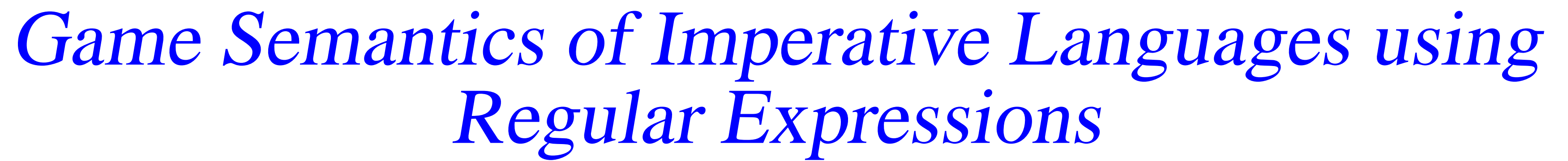
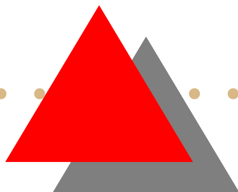


Game Semantics of Imperative Languages using Regular Expressions

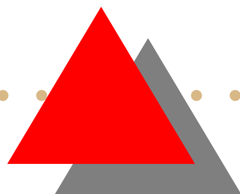
Guy McCusker



The game semantics universe has proved itself to be very flexible: it provides accurate models for a wide variety of programming languages.

But how useful are these models? Can they be used to reason about programs? Do they tell us anything we didn't already know?

This talk describes some work which attempts to provide a usable reasoning technique for the games model of Idealized Algol, developed in conjunction with Dan Ghica [1].

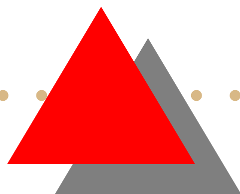


Idealized Algol

Idealized Algol (IA) is Reynolds's theoretical distillation of Algol 60 [9, 3]. IA can be seen as:

- a basic functional language extended with state
  - $x := 3$
  - $!x$
  - new x in $\ldots$
- alternatively, a basic imperative language extended with
  - block structure (new)
  - higher-order procedures (λ-calculus).

Either way, it is very expressive, elegant and powerful.



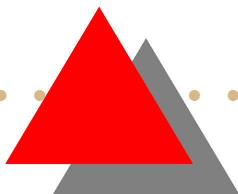
Programmers in Algol-like languages have many intuitions about the behaviour of programs.

- The use of local variables is invisible from outside a block: privacy, modularity, representation independence.
- State changes are irreversible: there is no “snapback” construct

$$\mathsf{snapback}(P)$$

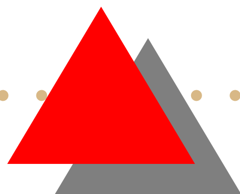
which runs P and then undoes all its state-changes.

A good semantics should capture and formalize these intuitions, and the program optimizations which they justify.



Garbage collection If x does not occur free in P then

$$\text{new } x \text{ in } P \cong P.$$



Some program equivalences

Garbage collection If x does not occur free in P then

$$\text{new } x \text{ in } P \cong P.$$

No snapback

$$\text{new } x \text{ in } P(x := 1);\text{if } !x = 1 \text{ then } \Omega \text{ else skip} \cong P(\Omega)$$

Some program equivalences

Garbage collection If x does not occur free in P then

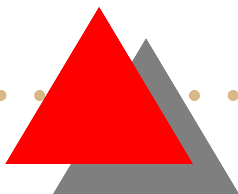
$$\text{new } x \text{ in } P \cong P.$$

No snapback

$$\text{new } x \text{ in } P(x := 1); \text{if } !x = 1 \text{ then } \Omega \text{ else skip} \cong P(\Omega)$$

Representation Independence

$$\begin{aligned} & \text{new } x : \text{bool in } x := \text{true}; P(!x, x := \neg !x) \\ \cong \quad & \text{new } x : \text{int in } x := 1; P(!x > 0, x := -!x) \end{aligned}$$

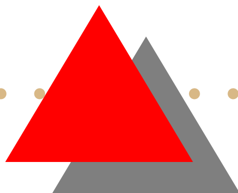


- Classical Milne-Strachey “marked store” models. These can fail to validate garbage collection [2].

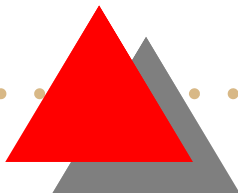
- Classical Milne-Strachey “marked store” models. These can fail to validate garbage collection [2].
- Functor-category models handle locality well. Parametricity constraints can be added to handle representation independence. Snapback remains problematic [6, 5].

Semantic approaches

- Classical Milne-Strachey "marked store" models. These can fail to validate garbage collection [2].
- Functor-category models handle locality well. Parametricity constraints can be added to handle representation independence. Snapback remains problematic [6, 5].
- Reddy's "object spaces" handle snapback too. Full abstraction up to order 2 via the Yoneda embedding [8, 4].

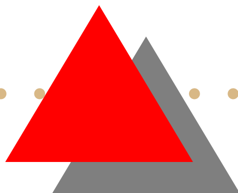


- Classical Milne-Strachey “marked store” models. These can fail to validate garbage collection [2].
- Functor-category models handle locality well. Parametricity constraints can be added to handle representation independence. Snapback remains problematic [6, 5].
- Reddy’s “object spaces” handle snapback too. Full abstraction up to order 2 via the Yoneda embedding [8, 4].
- Operationally-based reasoning [7] .



Semantic approaches

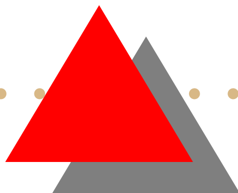
- Classical Milne-Strachey “marked store” models. These can fail to validate garbage collection [2].
- Functor-category models handle locality well. Parametricity constraints can be added to handle representation independence. Snapback remains problematic [6, 5].
- Reddy’s “object spaces” handle snapback too. Full abstraction up to order 2 via the Yoneda embedding [8, 4].
- Operationally-based reasoning [7] .
- Game semantics: full abstraction.



Syntax of IA

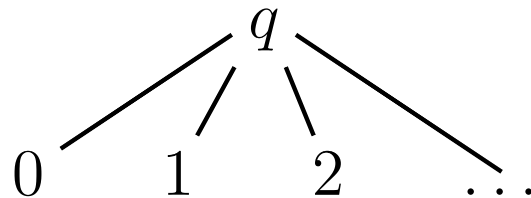
The language we will consider is a simply-typed λ-calculus with the following base types.

- Expression types $\mathbb{N}$, $\mathbb{B}$, with the usual constants and operations, including dereferencing of storage variables $!x$.
- The type comm of *commands*. Commands include assignment $x := M$, sequential composition $C_1; C_2$, skip, and local blocks new x in M.
- The type var of storage variables. Such variables are allocated using new and manipulated via assignment and dereferencing.

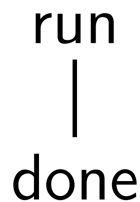


Semantics of base types

The expression types are interpreted as usual. For the natural numbers:

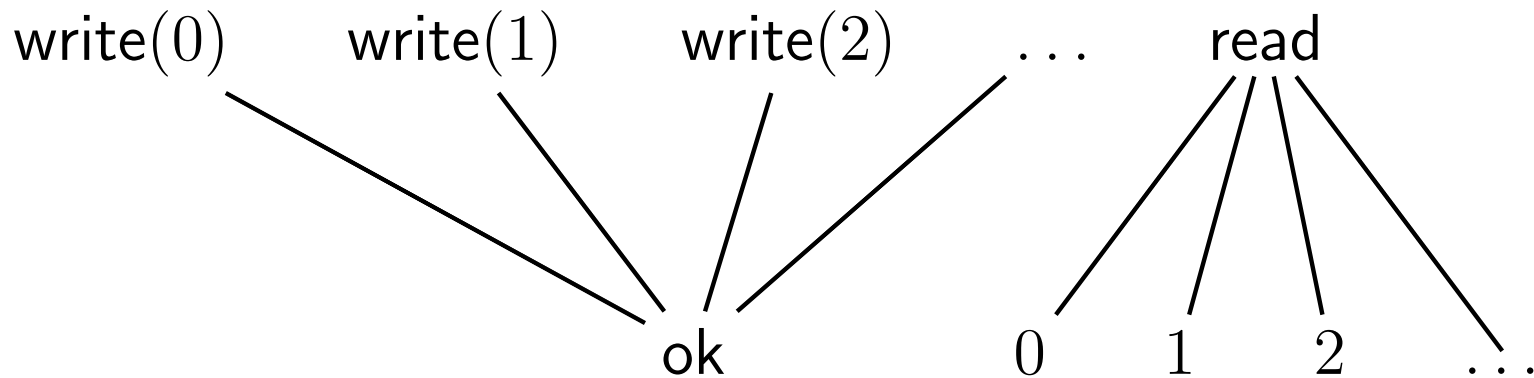


Commands are interpreted with a similar game:



Semantics of var

Variables have two kinds of initial move, for reading and for writing:



Some examples

$$x : \mathsf{var},\quad y : \mathsf{var} \vdash x := !y + 1 : \mathsf{comm}$$

run

read

n

write$(n+1)$

ok

done

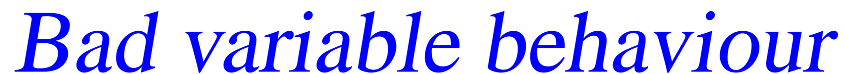
Some examples

c : comm,	x : var	$\vdash$ $x := 3; c; x :=!x + 1$:	comm
			run
	write(3)		
	ok		
run			
done			
	read		
	n		
	write$(n+1)$		
	ok		
			done

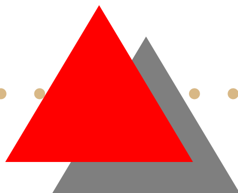
Some examples

$c : \mathsf{comm}, \quad x : \mathsf{var} \quad \vdash \quad x := 3; c; x := !x + 1 : \quad \mathsf{comm}$

c : comm,	x : var	⊢ $x := 3; c; x := !x + 1$: comm
		run
	write(3)	
	ok	
run		
done		
	read	
	n	(n need not be 3)
	write$(n+1)$	
	ok	
		done



In the previous example, O *must* be allowed to respond to read with any value, since the command c may later be bound to something like $x := 19$.
If we wrap the command in a new x in $\ldots$, this changes: the command c cannot now access x, and nor can anything else.
The variable x is now a *good variable*.



Variable allocation

The command new x in P is just like P, except that:

- x is bound to a storage cell, so P's interactions with x have the expected causal relationship between values written and values read.
- The outside world can no longer see x.

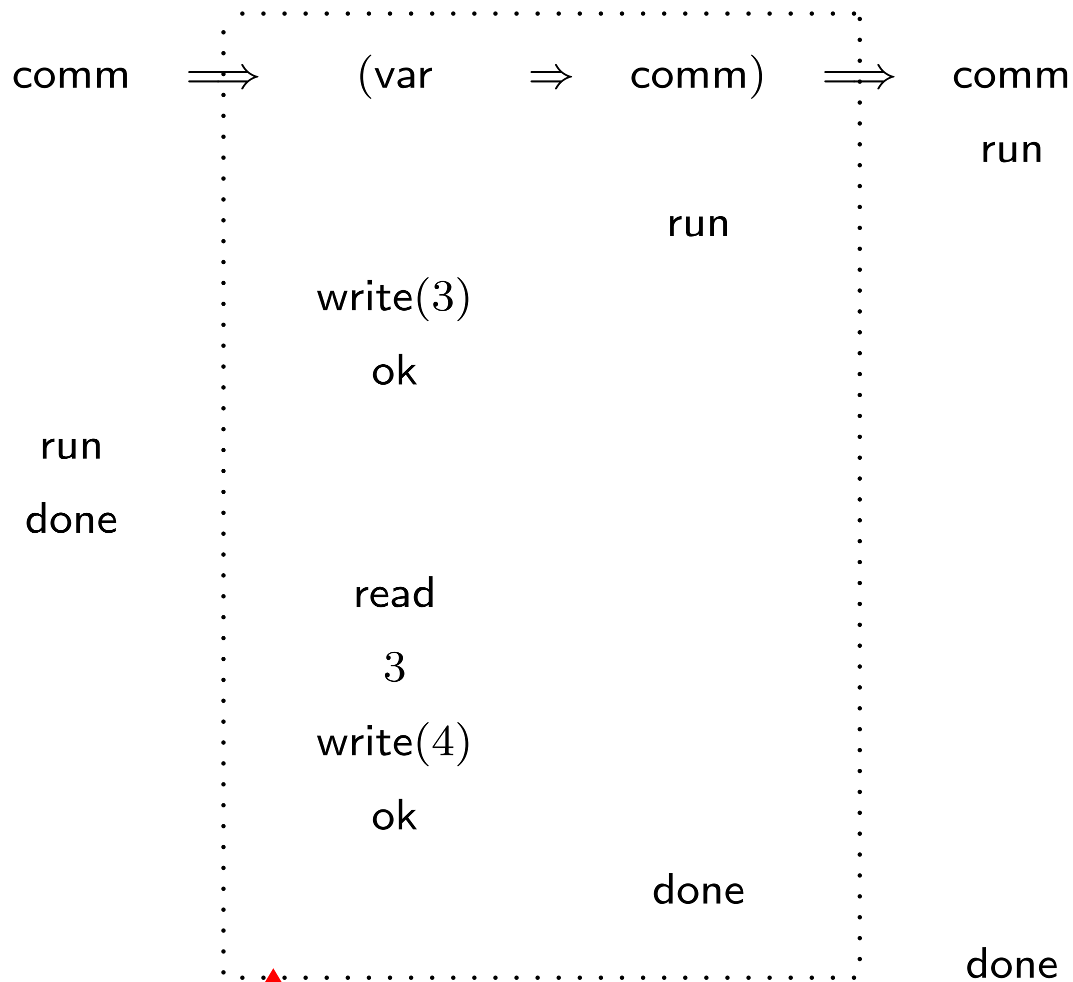
Semantics of allocation

In game semantics, variable allocation is handled by composition with this strategy:

$$
\begin{array}{ccccc}
(\mathsf{var} & \Rightarrow & \mathsf{comm}) & \Rightarrow & \mathsf{comm} \\
 & & & & \mathsf{run} \\
 & & \mathsf{run} & & \\
s & & & & \\
 & & \mathsf{done} & & \\
 & & & & \mathsf{done}
\end{array}
$$

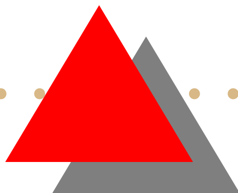
where s is any sequence of reads and writes for which the values read match the last values written at all times.

Allocation in action

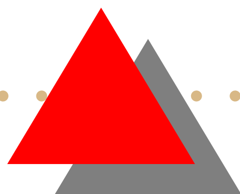


This composition achieves two things:

- good variable behaviour is enforced.
- the interactions in the var type are hidden.



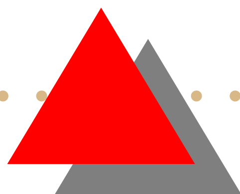
We will now see how to reason about this games model using regular expressions, for a restricted language.



We will now see how to reason about this games model using regular expressions, for a restricted language.
We consider only terms of the form

$$x_1 : \Theta_1, \ldots, x_n : \Theta_n \vdash M : B$$

where B is a base type and the Θ_i are first-order types.

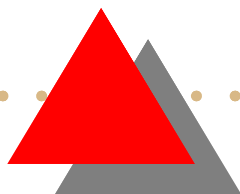


A restricted IA

We will now see how to reason about this games model using regular expressions, for a restricted language.
We consider only terms of the form

$$x_1 : \Theta_1, \ldots, x_n : \Theta_n \vdash M : B$$

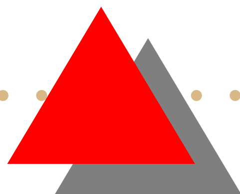
where B is a base type and the Θ_i are first-order types.
We do not handle general recursion, but we do include while-loops.



We will now see how to reason about this games model using regular expressions, for a restricted language.
We consider only terms of the form

$$x_1 : \Theta_1, \ldots, x_n : \Theta_n \vdash M : B$$

where B is a base type and the Θ_i are first-order types.
We do not handle general recursion, but we do include while-loops.
We assume a finite set of data-values.

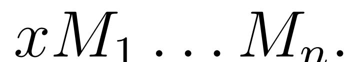


A restricted IA

We will now see how to reason about this games model using regular expressions, for a restricted language.
We consider only terms of the form

$$x_1 : \Theta_1, \ldots, x_n : \Theta_n \vdash M : B$$

where B is a base type and the Θ_i are first-order types.
We do not handle general recursion, but we do include while-loops.
We assume a finite set of data-values.
For convenience, we assume all terms are β-normal, so there are no λ-abstractions, and the only application terms we consider are those of the form



$$xM_1 \ldots M_n.$$

RegExps++

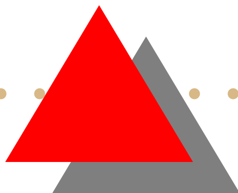
We will give the game semantics of this subset using a mildly extended syntax of regular expressions.

Constants a (singleton), ϵ (empty string), $\perp$ (empty language).

Standard operations $R + S$ (union), $R \cdot S$ (concatenation), R^* (repetition).

Intersection $R \cap S$.

Hiding $R \setminus \alpha$: delete all symbols in the set α.



Encoding of plays

We must make the disjoint union operation from game semantics explicit. We annotate moves as follows:

$$
\begin{array}{ccccccccc}
x : \mathbb{N}, & f : & \mathbb{N} & \Rightarrow & \mathbb{N} & \vdash & f(x) : & \mathbb{N} \\
 & & & & & & & q \\
 & & & & q_f & & & \\
 & & q_f^1 & & & & & \\
q_x & & & & & & & \\
n_x & & & & & & & \\
 & & n_f^1 & & & & & \\
 & & & & m_f & & & \\
 & & & & & & & m
\end{array}
$$

Denotation of terms

The strategy for a term

$$x_1 : \Theta_1, \ldots, x_n : \Theta_n \vdash M : \mathbb{N}$$

consists of a set of sequences of the form

$$q \ldots n$$

We will give a set of regular languages $(\![M]\!)_n$ such that

$$[\![M]\!] = \sum_n q \cdot (\![M]\!)_n \cdot n$$

Denotation of terms

For M : comm,

$$[\![M]\!] = \mathsf{run} \cdot (\![M]\!) \cdot \mathsf{done}.$$

For M : var,

$$\begin{aligned} [\![M]\!] &= \sum_n \mathsf{read} \cdot (\![M]\!)_{\mathsf{read}(n)} \cdot n \\ &+ \sum_n \mathsf{write}(n) \cdot (\![M]\!)_{\mathsf{write}(n)} \cdot \mathsf{ok} \end{aligned}$$

(Note: this notation differs from that used in the paper).

Constants

In the usual game semantics, $[\![n]\!] = \{q \cdot n\}$.
We define

$$(\![n]\!)_n = \epsilon, \qquad (\![n]\!)_m = \bot \text{ for } m \neq n.$$

Similarly:

$$(\![\mathsf{skip}]\!) = \epsilon, \qquad (\![\Omega]\!) = \bot.$$

Variables

For variables $x : \mathbb{N}$, the semantics is the copycat strategy

$$\begin{array}{ccc} x : \mathbb{N} & \vdash & x : \mathbb{N} \\ & & q \\ q_x & & \\ n_x & & \\ & & n \end{array}$$

so we define

$$(\![x : \mathbb{N}]\!)_n = q_x \cdot n_x.$$

More semantic definitions

$$
\begin{aligned}
(\!|C; C'|\!) &= (\!|C|\!) \cdot (\!|C'|\!) \\
(\!|M + M'|\!)_n &= \sum_{m+m'=n} (\!|M|\!)_m \cdot (\!|M'|\!)_{m'}. \\
(\!|\mathsf{if}\ B\ \mathsf{then}\ C\ \mathsf{else}\ C'|\!) &= (\!|B|\!)_{\mathsf{true}} \cdot (\!|C|\!) + (\!|B|\!)_{\mathsf{false}} \cdot (\!|C'|\!) \\
(\!|\mathsf{while}\ B\ \mathsf{do}\ C|\!) &= \left((\!|B|\!)_{\mathsf{true}} \cdot (\!|C|\!)\right)^* \cdot (\!|B|\!)_{\mathsf{false}} \\
(\!|V := M|\!) &= \sum_n (\!|M|\!)_n \cdot (\!|V|\!)_{\mathsf{write}(n)} \\
(\!|!V|\!)_n &= (\!|V|\!)_{\mathsf{read}(n)}
\end{aligned}
$$

Semantics of Application

Given $x : \mathsf{comm} \Rightarrow \mathsf{comm} \Rightarrow \mathsf{comm}$, $M_1, M_2 : \mathsf{comm}$, we define

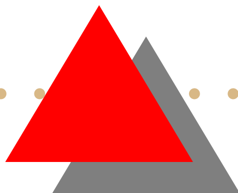
$$(\![x M_1 M_2]\!) = \mathsf{run}_x \cdot (\mathsf{run}^1_x \cdot (\![M_1]\!) \cdot \mathsf{done}^1_x + \mathsf{run}^2_x \cdot (\![M_2]\!) \cdot \mathsf{done}^2_x)^* \cdot \mathsf{done}_x.$$

Similarly, if $x : \mathsf{var} \Rightarrow \mathbb{N}$ and $M : \mathsf{var}$, $(\![x M]\!)_k$ is

$$q_x \cdot \left(\sum_n \mathsf{read}^1_x \cdot (\![M]\!)_{\mathsf{read}(n)} \cdot n^1_x + \sum_n \mathsf{write}(n)^1_x \cdot (\![M]\!)_{\mathsf{write}(n)} \cdot \mathsf{ok}^1_x\right)^* \cdot k_x.$$

Semantics of allocation

Let X denote the set of symbols tagged with an x. Let Y be the rest of the alphabet.

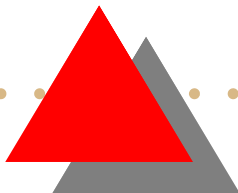


Semantics of allocation

Let X denote the set of symbols tagged with an x. Let Y be the rest of the alphabet.
The regular language

$$G = Y^{*}\cdot(\mathsf{read}_x\cdot 0_x\cdot Y^{*})^{*}\cdot(\sum_n \mathsf{write}(n)_x\cdot\mathsf{ok}_x\cdot Y^{*}\cdot(\mathsf{read}_x\cdot n_x\cdot Y^{*})^{*})^{*}$$

describes all those strings in which x has good-variable behaviour.

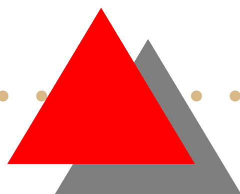


Let X denote the set of symbols tagged with an x. Let Y be the rest of the alphabet.
The regular language

$$G = Y^* \cdot (\mathsf{read}_x \cdot 0_x \cdot Y^*)^* \cdot (\sum_n \mathsf{write}(n)_x \cdot \mathsf{ok}_x \cdot Y^* \cdot (\mathsf{read}_x \cdot n_x \cdot Y^*)^*)^*$$

describes all those strings in which x has good-variable behaviour.
Now we can define

$$(\![\mathsf{new}\ x\ \mathsf{in}\ P]\!) = ((\![P]\!) \cap G) \setminus X.$$



A theorem

$$P \cong Q \Leftrightarrow [\![P]\!] = [\![Q]\!] \Leftrightarrow (\![P]\!) = (\![Q]\!).$$

A simple example

$$\begin{aligned}
(\![\text{while true do } C]\!) &= ((\![\text{true}]\!)_{\text{true}} \cdot (\![C]\!))^* \cdot (\![\text{true}]\!)_{\text{false}} \\
&= (\epsilon \cdot (\![C]\!))^* \cdot \bot \\
&= \bot \\
&= (\![\Omega]\!).
\end{aligned}$$

So while true do $C \cong \Omega$.

Garbage collection

$$\begin{aligned} (\![\mathsf{new}\ x\ \mathsf{in}\ M]\!) &= ((\![M]\!) \cap G) \setminus X \\ &= (\![M]\!) \end{aligned}$$

since x is not free in M so no move of $(\![M]\!)$ is tagged with an x.

No snapback

Let M be $P(x := 1)$; if $!x = 1$ then Ω else skip.
We will show that new x in $M \cong P(\Omega)$.

$$
\begin{aligned}
(\![x := 1]\!) &= \mathsf{write}(1)_x \cdot \mathsf{ok}_x \\
(\![P(x := 1)]\!) &= \mathsf{run}_P \cdot \\
&\quad (\mathsf{run}^1_P \cdot \mathsf{write}(1)_x \cdot \mathsf{ok}_x \cdot \mathsf{done}^1_P)^* \cdot \\
&\quad \mathsf{done}_P \\
(\![!x = 1]\!)_{\mathsf{true}} &= \mathsf{read}_x \cdot 1_x \\
(\![!x = 1]\!)_{\mathsf{false}} &= \sum_{n \neq 1} \mathsf{read}_x \cdot n_x
\end{aligned}
$$

No snapback, continued

$$
\begin{aligned}
(\![\mathsf{if}\ !x = 1\ \mathsf{then}\ \Omega\ \mathsf{else\ skip}]\!) &= (\![!x = 1]\!)_{\mathsf{true}} \cdot (\![\Omega]\!) \\
&+ (\![!x = 1]\!)_{\mathsf{false}} \cdot (\![\mathsf{skip}]\!) \\
&= (\![!x = 1]\!)_{\mathsf{true}} \cdot \bot + (\![!x = 1]\!)_{\mathsf{false}} \cdot \epsilon \\
&= (\![!x = 1]\!)_{\mathsf{false}} \\
&= \sum_{n \neq 1} \mathsf{read}_x \cdot n_x
\end{aligned}
$$

Therefore

$$
(\![M]\!) = \sum_{n \neq 1} \mathsf{run}_P \cdot (\mathsf{run}_P^1 \cdot \mathsf{write}(1)_x \cdot \mathsf{ok}_x \cdot \mathsf{done}_P^1)^* \cdot \mathsf{done}_P \cdot \mathsf{read}_x \cdot n_x
$$

No snapback, concluded

$$(\llbracket M \rrbracket) = \sum_{n \neq 1} \mathsf{run}_P \cdot (\mathsf{run}^1_P \cdot \mathsf{write}(1)_x \cdot \mathsf{ok}_x \cdot \mathsf{done}^1_P)^* \cdot \mathsf{done}_P \cdot \mathsf{read}_x \cdot n_x$$

We therefore have

$$\begin{aligned} (\llbracket M \rrbracket) \cap G &= \mathsf{run}_P \cdot \mathsf{done}_P \cdot \mathsf{read}_x \cdot 0_x \\ (\llbracket \mathsf{new}\ x\ \mathsf{in}\ M \rrbracket) &= ((\llbracket M \rrbracket) \cap G) \setminus X = \mathsf{run}_P \cdot \mathsf{done}_P. \end{aligned}$$

On the other hand, $(\llbracket \Omega \rrbracket) = \bot$ so

$$\begin{aligned} (\llbracket P(\Omega) \rrbracket) &= \mathsf{run}_P \cdot (\mathsf{run}^1_P \cdot \bot \cdot \mathsf{done}^1_P)^* \cdot \mathsf{done}_P \\ &= \mathsf{run}_P \cdot \mathsf{done}_P. \end{aligned}$$

Decidability

Our theorem shows that observational equivalence for this fragment of Idealized Algol is decidable.

In principle, we could mechanize this reasoning to build a model-checker.

However, language equivalence for regular expressions with intersection is EXPSPACE complete. . .

More research needed!

References

[1] Dan R. Ghica and Guy McCusker. Reasoning about Idealized Algol using regular languages. In *Proceedings, Twenty-Seventh International Colloquium on Automata, Languages and Programming*, pages 103–115, 2000.

[2] R.E. Milne and C. Strachey. *A Theory of Programming Language Semantics*. Chapman and Hall, London, 1976.

[3] Peter Naur, J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauer, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Revised report on the algorithmic language ALGOL 60. *Communications of the ACM*, 6(1):1–17, January 1963.

[4] P. W. O'Hearn and U. Reddy. Objects, interference and the Yoneda embedding. In M. Main and S. Brookes, editors, *Mathematical Foundations of Programming Semantics: Proceedings of 11th International Conference*, Electronic Notes in Theoretical Computer Science. Elsevier Science Publishers B.V., 1995.

[5] P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *Journal of the ACM*, 42(3):658–709, May 1995.

[6] Peter W. O'Hearn and R. D. Tennent. Semantics of local variables. In M. P. Fourman, P. T. Johnstone, and A. M.

Pitts, editors, *Applications of Categories in Computer Science: Proceedings of the LMS Symposium, Durham, 1991*. Cambridge University Press, 1992. LMS Lecture Notes Series, 177.

[7] A. M. Pitts. Reasoning about local variables with operationally-based logical relations. In *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 152–163. IEEE Computer Society Press, 1996.

[8] Uday S. Reddy. Global state considered unnecessary: Object-based semantics for interference-free imperative programs. *Lisp and Symbolic Computation*, 9(1), 1996.

[9] John C. Reynolds. The essence of Algol. In *Proceedings of the 1981 International Symposium on Algorithmic Languages*, pages 345–372. North-Holland, 1981.