OPLSS22: Game Semantics - Lecture 4

Speaker: Pierre-Louis Curien

June 25, 2022

1 Full abstraction

The work on game semantics presented in these lectures was motivated by the full abstraction problem for PCF. Since full abstraction is a desirable property for denotational semantics of any language, game semantics is not applicable only to PCF, but rather is a more general framework that could be applied to develop a denotational semantics of other languages.

We denote *operational equality* between two terms M and N by $M =_{op} N$, meaning that for all contexts C such that $C[M]$ and $C[N]$ are closed and of base type, we have $C[M] \to^* c$ if and only if $C[N] \to^* c$

We say that a model is *fully abstract* when denotational equivalence between terms M and N implies $M =_{op} N$, and vice versa:

$$[\![M]\!] = [\![N]\!] \text{ if and only if } M =_{op} N$$

To prove this, we introduce the notion of *computational adequacy.* Computational adequacy, as discovered by Plotkin, is stated in lemma 1.1.

Lemma 1.1. $\vdash P : C$*, then* $[|P|] = v$ *if and only if* $P \to^* v$.

The proof of lemma 1.1 follows by constructs of logical relations. Observe that any model that interprets natural numbers as the corresponding actual natural numbers (e.g. game model, intermediate model) satisfies computational adequacy.

Now we have all the ingredients to complete the proof of full abstraction. The forward direction is immediate. Here, we want to show that if two terms have the same denotational semantics, then they are operationally equal. Indeed, let $[\![M]\!] = [\![N]\!]$ and let C be a context as above. Then $[\![C[M]]\!] = [\![C[N]]\!]$, and we can conclude by lemma 1.1

2 Towards a fully abstract PCF

2.1 Inadequacy of Scott semantics

The Scott semantics model for PCF cannot be fully abstract because it accepts the *parallel-or* function *por* (discussed below) that cannot be defined in PCF.

The function $por : Bool \to Bool \to Bool$ precludes full-abstraction because it is possible to construct two PCF terms N and M each of type $(Bool \to Bool \to Bool) \to Bool$ that differ only in their denotational interpretation when applied to *por*, hence $M =_{op} N$ (because *por* cannot be defined in "real" PCF) yet $[[M]] \neq [[N]]$ (when applied to *por* in the model).

2.2 Research on PCF full abstraction

In the late 1970's arose the question of finding a fully abstract model of PCF. There were multiple candidate models:

- Milner's term model solves the full abstraction problem when quotiented by operational equivalence. The necessity of the quotienting step reduces the appeal of the model.
- Scott semantics is not a fully abstract model of PCF, but it is a fully abstract model for PCF+*por*. Scott semantics models a computer program with a *Scott-continuous function* f (any single piece of the output $f(x)$ may be computed using a finite part of the input x (where x itself could be infinite), which one can take to be minimal). But Scott pointed out the barrier to full abstraction posed by $por(\bot, T) = T$ and $por(T, \bot) = T$ which is not definable in PCF (discussed above). Plotkin observed that Scott continuous functions are a fully abstract model for the extension PCF + *por* of PCF.
- The stable model of PCF, was developed by Gérard Berry and uses a "stability" condition to exclude the problematic *por* from the denotational semantics. A function f is stable if for a fixed x and fixed piece of $f(x)$, such a minimal input is unique, and is thus minimum. But this condition was not sufficient to obtain full abstraction, as witnessed by the following function:

  $$Gustave(T, F, \bot) = T \quad Gustave(F, \bot, T) = T \quad Gustave(\bot, T, F) = T$$

  Gustave[1] is problematic because on one hand any pair of elements do not have a common upper bound (viz. first components in the first two equations) hence it is stable, but on the other hand, no sequential program can compute it.

2.3 Definable separability

We say that a model has the property of *definable separability* if for any two terms with different denotations there exists a definable function that produces a different value when applied to each term (hence the terms are not operationally equivalent). Definable separability is a sufficient condition to get a fully abstract model.

[1]Gustave was the nickname given to Berry when he joined INRIA as a PhD student.

Theorem 2.1. *If the model is such that for every distinct f,g of the same type A (interpreting some syntactic type) there exists a definable h of type $A \to bool$ such that $hf \neq hg$, then it is fully abstract.*

Proof. Suppose that $[\![M]\!] \neq [\![N]\!]$, i.e. M and N are not denotationally equivalent. We want to show that $M \not\equiv_{op} N$. Let h be given by our assumption, and let P be such that $h = [\![P]\!]$, $v_1 = h([\![M]\!])$, $v_2 = h([\![N]\!])$, and context $C = P\,[\,]$. Then $C\,[M] \to^* v_1$ and $C\,[N]] \to^* v_2$, and hence $M \not\equiv_{op} N$.

3 Imperative semantics

We now step away from PCF to apply game semantics to imperative programming languages. For this, we introduce a new class of moves that correspond to statements in an imperative program. We'll start with a very small arena `comm` containing just two moves:

- `run`, an opponent move, and the initial move
- `done`, a player move, enabled by `run`

We'll introduce another imperative arena `var` that expresses what it means to read and write from a single memory cell that can hold a natural number. The moves are:

- `write`(n) for any $n \in \omega$. This is an opponent move and is initial.
- `OK`. This is a player move and is the response sent after a write is successfully committed.
- `read`. This is an opponent move that asks for the value of the cell.
- n for any $n \in \omega$. This is a player move that represents responding to a read request with the current value of the cell.

As an example of a strategy in this arena, consider the strategy "cell" whose plays are as follows: `write(0) OK read 0`. Here, the opponent begins the game by requesting to write 0. The player responds `OK` signifying that the write has completed. Then the opponent sends a `read` request. The player responds with the value of the cell of memory.

As a second example, consider `write(0) OK write(2) OK read 0`. Similar to the previous example, the opponent writes 0 and the game continues with the player's turn. Then opponent then writes 2. Now, when the player sends a read request, 2 is obtained. For this reason, the strategy is not innocent as "reads" refer to the most recent "write."

3.1 Cell discipline enforced by interaction

This scheme naturally allows for expansion to interaction plays, e.g. write after reads. For example, consider the program `x := 0; x := x+1`. The denotational semantics of this program is defined as follows:

$$\mathtt{run}_\epsilon\ \mathtt{write}(0)_1\ \mathtt{OK}_1\ \mathtt{read}_1 \begin{cases} \vdots \\ n_1\ \mathtt{write}(n+1)_1\ \mathtt{OK}_1\ \mathtt{done}_\epsilon \\ \vdots \end{cases}$$

Observe that after the $\mathtt{read}_1$ there are branches for any natural number. This may seem counterintuitive, because $\mathtt{write}(0)_1$ appears earlier in the denotation of the program, and we expect a cell to retain the most recent value stored in it. This is not a bug but a feature, because it allows us to model concurrency: if another program is also interacting with the same cell of memory, then there is no guarantee as to what may be found in that cell immediately prior to the read. In this way, our game semantics (which we developed with reference to the completely sequential execution of PCF) is "ready" to express concurrent execution, provided that when the above strategy is applied to the strategy "cell" (see above), then the allowed interaction must impose non interference.

Here is an example of a play in the strategy denoted above:

$$\mathtt{run}_\epsilon\ \mathtt{write}(0)_1\ \mathtt{OK}_1\ \mathtt{read}_1\ 0_1\ \mathtt{write}(1)_1\ \mathtt{OK}_1\ \mathtt{done}_\epsilon$$

3.2 Sequencing commands

The sequencing of commands (noted $\mathtt{comm}_1; \mathtt{comm}_2$) can be modeled with the strategy for $\mathtt{comm}_1 \times \mathtt{comm}_2 \to \mathtt{comm}_\epsilon$:

$$\mathtt{run}_\epsilon\ \mathtt{run}_1\ \mathtt{done}_1\ \mathtt{run}_2\ \mathtt{done}_2\ \mathtt{done}_\epsilon$$

3.3 Assignment

The assignment operation can be interpreted as a strategy for $\mathtt{var}_1 \to \mathtt{comm}_\epsilon$, where `var` is the type of variables. For example, the strategy for $x := 2$ is

$$\mathtt{run}_\epsilon\ \mathtt{write}(2)_1\ \mathtt{OK}_1\ \mathtt{done}_\epsilon$$

Mixing assignment and sequencing we can interpret $(x := 0)\ ;\ (x := x + 1)$ as

$$\mathtt{run}_\epsilon\ \mathtt{write}(0)_1\ \mathtt{OK}_1\ \mathtt{read}_1 \begin{cases} 0_1\ \mathtt{write}(0+1)_1\ \mathtt{OK}_1\ \mathtt{done}_\epsilon \\ 1_1\ \mathtt{write}(1+1)_1\ \mathtt{OK}_1\ \mathtt{done}_\epsilon \\ 2_1\ \mathtt{write}(2+1)_1\ \mathtt{OK}_1\ \mathtt{done}_\epsilon \\ \vdots \end{cases}$$