Transition Systems over Games

Paul Blain Levy, University of Birmingham
Sam Staton, University of Oxford

April 30, 2015

Outline

1. Examples of game semantics

2. Transition systems

3. Framework
   - Single game
   - Tensor
   - Transfers between games

What is game semantics?

- A form of semantics for many different language features.
- Game between P (Proponent, Patricia, the program)
- and O (Opponent, Oliver, the environment)

What is game semantics?

- A form of semantics for many different language features.
- Game between P (Proponent, Patricia, the program)
- and O (Opponent, Oliver, the environment)
- Particularly good for modelling private store
- because you don't see it in the semantics.

What is game semantics?

- A form of semantics for many different language features.
- Game between P (Proponent, Patricia, the program)
- and O (Opponent, Oliver, the environment)
- Particularly good for modelling private store
- because you don't see it in the semantics.

  (U. Reddy, Global state considered unnecessary, 1996)

Example with storage

In OCaml, a computation of type `int` $\rightarrow$ `int`

does some stuff, then returns a function.

Example with storage

In OCaml, a computation of type `int` $\rightarrow$ `int`

does some stuff, then returns a function.

- P returns a function `f`.
- O calls `f` with argument 7.
- P returns 5.
- O calls `f` with argument 7.
- P returns 29.

Example with storage

In OCaml, a computation of type `int` $\to$ `int`

does some stuff, then returns a function.

- P returns a function `f`.
- O calls `f` with argument 7.
- P returns 5.
- O calls `f` with argument 7.
- P returns 29.

This is a play.
The meaning of the computation is a strategy:
a set of plays describing P's response to every possible O-move.

Example with storage

In OCaml, a computation of type `int` $\rightarrow$ `int`

does some stuff, then returns a function.

- P returns a function `f`.
- O calls `f` with argument 7.
- P returns 5.
- O calls `f` with argument 7.
- P returns 29.

This is a play.
The meaning of the computation is a strategy:
a set of plays describing P's response to every possible O-move.
The above play cannot be achieved without private store, e.g. of booleans.

Example with storage of functions

A computation of type $(\mathtt{int} \to \mathtt{int}) \to \mathtt{int}$

does some stuff, then returns a second-order function whose argument is a function.

- P returns a second-order function $\mathtt{f}$.
- O calls $\mathtt{f}$ with function argument g.
- P returns 7.
- O calls $\mathtt{f}$ with function argument g'.
- P calls g with argument 2.
- O returns 3.
- P returns 5.

Example with storage of functions

A computation of type $(\mathtt{int} \to \mathtt{int}) \to \mathtt{int}$

does some stuff, then returns a second-order function whose argument is a function.

- P returns a second-order function $\mathtt{f}$.
- O calls $\mathtt{f}$ with function argument g.
- P returns 7.
- O calls $\mathtt{f}$ with function argument g'.
- P calls g with argument 2.
- O returns 3.
- P returns 5.

This play cannot be achieved without storage of functions.

Example with storage of functions

A computation of type $(\mathtt{int} \to \mathtt{int}) \to \mathtt{int}$

does some stuff, then returns a second-order function whose argument is a function.

- P returns a second-order function $\mathtt{f}$.
- O calls $\mathtt{f}$ with function argument g.
- P returns 7.
- O calls $\mathtt{f}$ with function argument g'.
- P calls g with argument 2.
- O returns 3.
- P returns 5.

This play cannot be achieved without storage of functions.
Arguments and return values that are functions
are represented as fresh names.

Two kinds of game semantics

Denotational game semantics (1994 onwards)

- Hyland, Ong; Nickau; Abramsky, Honda, McCusker; Laird; . . . , .
- The meaning of a complex piece of code is defined from the meaning of its parts.
- This is called compositionality

Two kinds of game semantics

Denotational game semantics (1994 onwards)

- Hyland, Ong; Nickau; Abramsky, Honda, McCusker; Laird; . . . , .
- The meaning of a complex piece of code is defined from the meaning of its parts.
- This is called compositionality

Operational game semantics (1994 onwards)

- Sangiorgi; Jeffrey, Rathke; Lassen; . . . .
- The meaning of a piece of code directly describes its interaction with the environment.
- Given by a transition system.
- Not obviously compositional.

Two kinds of game semantics

Denotational game semantics (1994 onwards)

- Hyland, Ong; Nickau; Abramsky, Honda, McCusker; Laird; . . . , .
- The meaning of a complex piece of code is defined from the meaning of its parts.
- This is called compositionality

Operational game semantics (1994 onwards)

- Sangiorgi; Jeffrey, Rathke; Lassen; . . . .
- The meaning of a piece of code directly describes its interaction with the environment.
- Given by a transition system.
- Not obviously compositional.

Very different but semantically the same.

Our goal: a combined account

- Start with the direct operational description.
- Obtain compositionality as a theorem, not a definition.

Our goal: a combined account

- Start with the direct operational description.
- Obtain compositionality as a theorem, not a definition.
- And treat fresh names in a rigorous yet unobtrusive way.

Our goal: a combined account

- Start with the direct operational description.
- Obtain compositionality as a theorem, not a definition.
- And treat fresh names in a rigorous yet unobtrusive way.

We needed to think more carefully about transition systems.

Example of transitions

A program in BASIC

```
10 IF X>3 THEN PRINT 'd'
20 X = 5
30 GOTO 10
```

Example of transitions

A program in BASIC

```
10 IF X>3 THEN PRINT 'd'
20 X = 5
30 GOTO 10
```

A behaviour of the program

$$\begin{pmatrix} \text{line } 10 \\ \mathtt{X} = 4 \end{pmatrix} \overset{\mathtt{d}}{\rightsquigarrow} \begin{pmatrix} \text{line } 20 \\ \mathtt{X} = 4 \end{pmatrix} \rightsquigarrow \begin{pmatrix} \text{line } 30 \\ \mathtt{X} = 5 \end{pmatrix}$$

$$\rightsquigarrow \begin{pmatrix} \text{line } 10 \\ \mathtt{X} = 5 \end{pmatrix} \overset{\mathtt{d}}{\rightsquigarrow} \begin{pmatrix} \text{line } 20 \\ \mathtt{X} = 5 \end{pmatrix}$$

Some transitions perform an observable action, while others are silent.

Labelled transition system

Let L be a set of actions.

Definition

A labelled transition system over the set L consists of

- a set $\mathbb{S}$ of states
- a relation $\overset{a}{\leadsto}$ on $\mathbb{S}$ for each $a \in L$
- a relation $\leadsto$ on $\mathbb{S}$, representing silent transitions.

Traces

A trace is a sequence of observable actions.

Traces

A trace is a sequence of observable actions.

A behaviour of the program

$$\begin{pmatrix} \text{line } 10 \\ \mathtt{X} = 4 \end{pmatrix} \overset{\mathtt{d}}{\rightsquigarrow} \begin{pmatrix} \text{line } 20 \\ \mathtt{X} = 4 \end{pmatrix} \rightsquigarrow \begin{pmatrix} \text{line } 30 \\ \mathtt{X} = 5 \end{pmatrix}$$

$$\rightsquigarrow \begin{pmatrix} \text{line } 10 \\ \mathtt{X} = 5 \end{pmatrix} \overset{\mathtt{d}}{\rightsquigarrow} \begin{pmatrix} \text{line } 20 \\ \mathtt{X} = 5 \end{pmatrix}$$

The state $\begin{pmatrix} \text{line } 10 \\ \mathtt{X} = 4 \end{pmatrix}$ has a trace $\mathtt{dd}$.

What's good about transition systems

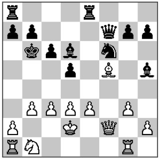
- Easy to set up.
- We can show that two states have the same behaviour, using a kind of relation called a bisimulation.

What's odd about transition systems

- Actions may represent outputs, inputs or sychronizations.
- The set of actions does not change over time.

Example: White chess-playing system

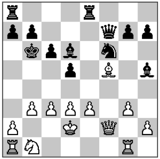
- In one state of the system, the chessboard looks like this:

  with White to play. The line number is 370, and $\mathtt{X} = 7$. From this state, White moves the knight to A3.
- In another state, the chessboard looks the same, with White to play. The line number is 520 and $\mathtt{X} = 2$. From this state, White performs a silent transition, changing the line number to 530, and then moves the pawn to F4.

Example: White chess-playing system

- In one state of the system, the chessboard looks like this:

  with White to play. The line number is 370, and $\mathtt{X} = 7$. From this state, White moves the knight to A3.
- In another state, the chessboard looks the same, with White to play. The line number is 520 and $\mathtt{X} = 2$. From this state, White performs a silent transition, changing the line number to 530, and then moves the pawn to F4.

Every state is in a position.

The position determines what actions are legitimate.

The position is the "type" of the state.

Example: higher-order functions

Each player has an inventory of function-names they are allowed to call.

Position = two finite sets of function-names

Example: higher-order functions

Each player has an inventory of function-names they are allowed to call.

Position = two finite sets of function-names

Suppose a computation performs a higher-order function call:

$$\mathtt{f}(V, W)$$

The arguments V and W are functions.

Example: higher-order functions

Each player has an inventory of function-names they are allowed to call.

Position = two finite sets of function-names

Suppose a computation performs a higher-order function call:

$$\mathtt{f}(V, W)$$

The arguments V and W are functions.

P performs the move call `f`.

Example: higher-order functions

Each player has an inventory of function-names they are allowed to call.

Position = two finite sets of function-names

Suppose a computation performs a higher-order function call:

$$\mathtt{f}(V, W)$$

The arguments V and W are functions.

P performs the move call `f`.

The new position is the same as before,
except that O has received two fresh function-names a and b,
which he can call in subsequent moves.

Example: higher-order functions

Each player has an inventory of function-names they are allowed to call.

Position = two finite sets of function-names

Suppose a computation performs a higher-order function call:

$$\mathtt{f}(V, W)$$

The arguments V and W are functions.

P performs the move call `f`.

The new position is the same as before,
except that O has received two fresh function-names a and b,
which he can call in subsequent moves.

The new state contains the bindings

$$\begin{array}{ccc} a & \mapsto & V \\ b & \mapsto & W \end{array}$$

The Framework

	Single game $\mathcal{G}$	Tensor $\mathcal{G} \otimes \mathcal{G}'$	Transfer $\mathcal{G} \to \mathcal{H}$
Games			
Strategies			
Transition systems			
Relating strategies to transition systems		Compositionality theorems	

Game = bipartite graph

Definition

A game consists of

- a set of passive positions (O to move)
- a set of active positions (P to move)
- from each passive position P, a set of O-moves m, each with an active target position $P.m$
- from each active position Q, a set of P-moves n, each with a passive target position $Q.n$.

Notation	$P \circ\!\xrightarrow{m} Q$	O-move
	$Q \bullet\!\xrightarrow{n} P$	P-move

Plays

Let P be a passive position. (The starting position.)

A play from position P is a sequence of moves

$$P \circ\!\xrightarrow{m_0} \cdot \bullet\!\xrightarrow{n_0} \cdot \circ\!\xrightarrow{m_1} \cdot \bullet\!\xrightarrow{n_1} \cdot$$

Strategies

A strategy tells P how to respond to any O-move

either playing a move or diverging.

Strategies

A strategy tells P how to respond to any O-move

either playing a move or diverging.

Definition

A strategy σ from passive position P is a set of passive-ending plays such that

- $\varepsilon \in \sigma$
- (prefix-closure) $smn \in \sigma \Rightarrow s \in \sigma$
- (determinacy) $tn, tn' \in \sigma \Rightarrow n = n'$

Definition

- In each passive position, a set of passive states.
- In each active position, a set of active states.
- For each passive state x and O-move m, an active state $x@m$.
- Each active state y either
  - performs a P-move $y \overset{n}{\rightsquigarrow} x$
  - or performs a silent transition $y \rightsquigarrow y'$ (same position).

Derived notation

$$y \overset{n}{\Longrightarrow} z \quad \text{when } y \rightsquigarrow^{*} \overset{n}{\rightsquigarrow} z$$

$$y \Uparrow \quad \text{when } y \rightsquigarrow^{\omega}$$

From states to strategies

For a state x in passive position P, suppose

$$x = x_0 \quad x_0 @ m_0 \stackrel{n_0}{\Longrightarrow} x_1 \quad x_1 @ m_1 \stackrel{n_1}{\Longrightarrow} x_2 \quad \cdots$$

then the play $m_0, n_0, m_1, n_1 \cdots$ is a trace of x.

From states to strategies

For a state x in passive position P, suppose

$$x = x_0 \quad x_0 @ m_0 \stackrel{n_0}{\Longrightarrow} x_1 \quad x_1 @ m_1 \stackrel{n_1}{\Longrightarrow} x_2 \quad \cdots$$

then the play $m_0, n_0, m_1, n_1 \cdots$ is a trace of x.

The set of all traces of x is a strategy $[\![x]\!]$.

Tensor game $\mathcal{G} \otimes \mathcal{G}'$ (Lamarche)

We wish to play the two games concurrently.

A passive position of $\mathcal{G} \otimes \mathcal{G}'$ is a pair of passive positions $\begin{pmatrix} P \\ P' \end{pmatrix}$

O can choose which game to play in
and P has to respond in the same game.

So an active position has one active and one passive component.

Play in tensor game

$$\begin{pmatrix} P \\ P' \end{pmatrix} \begin{matrix} \circ\!\xrightarrow{m} \\ = \end{matrix} \begin{pmatrix} P.m \\ P' \end{pmatrix} \begin{matrix} \bullet\!\xrightarrow{n} \\ = \end{matrix} \begin{pmatrix} P.m.n \\ P' \end{pmatrix} \begin{matrix} = \\ \circ\!\xrightarrow{m'} \end{matrix} \begin{pmatrix} P.m.n \\ P'.m' \end{pmatrix} \begin{matrix} = \\ \bullet\!\xrightarrow{n'} \end{matrix} \begin{pmatrix} P.m.n \\ P'.m'.n' \end{pmatrix}$$

Tensor strategy $\sigma \otimes \sigma'$

A play is in the strategy $\sigma \otimes \sigma'$ when
its first component is in σ
and its second component is in σ'.

Tensor of transition systems

Given transition systems over $\mathcal{G}$ and $\mathcal{G}'$,

the tensor system has states $\begin{pmatrix} x \\ x' \end{pmatrix}$

Compositionality theorem for tensors

$$\left[\!\left[\begin{pmatrix} x \\ x' \end{pmatrix} \right]\!\right] = [\![x]\!] \otimes [\![x']\!]$$

Transfer from $\mathcal{G}$ to $\mathcal{H}$

I am going to play the external game $\mathcal{H}$ (chess) against external-O.

In my attic lives a player of the internal game $\mathcal{G}$ (draughts) called internal-P.

I shall transfer moves between the two games using a transfer.

Positions and linkers

At any time, either

- both games are in passive position and I'm waiting for external-O
- or both games are in active position and I'm waiting for internal-P.

Positions and linkers

At any time, either

- both games are in passive position and I'm waiting for external-O
- or both games are in active position and I'm waiting for internal-P.

My own state is called a linker

$$\begin{array}{cc} P & \text{internal position} \\ \big\downarrow & \\ R & \text{external position} \end{array}$$

Example: a binary transfer

λ**Game** is the game for λ-calculus.

- A position is two finite sets of function-names.

Example: a binary transfer

λ**Game** is the game for λ-calculus.

- A position is two finite sets of function-names.

We define a transfer from $\lambda\textbf{Game} \otimes \lambda\textbf{Game} \to \lambda\textbf{Game}$.

Intended purpose of the transfer

To provide a semantic counterpart to syntactic substitution.

Example: a binary transfer

λ**Game** is the game for λ-calculus.

- A position is two finite sets of function-names.

We define a transfer from $\lambda\mathbf{Game} \otimes \lambda\mathbf{Game} \to \lambda\mathbf{Game}$.

Intended purpose of the transfer

To provide a semantic counterpart to syntactic substitution.

What's a linker in this transfer?

A function saying that certain names in one game correspond to certain names in the other.

Responding to an external O-move

Given a passive linker and external O-move

$$\begin{tikzcd} P \arrow[d, "h"'] & \\ R \arrow[r, "m'"] & {} \end{tikzcd}$$

I play either

$$\begin{tikzcd} P \arrow[r, dotted, "m"'] \arrow[d, "h"'] & \cdot \arrow[d, dotted, "k"] \\ R \arrow[r, "m'"] & \cdot \end{tikzcd}$$

O-move square

or

$$\begin{tikzcd} P \arrow[rr, equal] \arrow[d, "h"'] & & P \arrow[d, dotted, "h'"] \\ R \arrow[r, "m'"] & \cdot \arrow[r, dotted, "n'"] & \cdot \end{tikzcd}$$

external square

Responding to an internal P-move

Given an active linker
and internal P-move

$$\begin{tikzcd}
Q \arrow[r, "n"'] \arrow[d, "k"'] & {} \\
S &
\end{tikzcd}$$

I play either

$$\begin{tikzcd}
Q \arrow[r, "n"'] \arrow[d, "k"'] & \cdot \odot \arrow[r, dotted, "m"'] & \cdot \arrow[d, dotted, "k'"] \\
S \arrow[rr, equal] & & S
\end{tikzcd}$$

internal
square

or

$$\begin{tikzcd}
Q \arrow[r, "n"'] \arrow[d, "k"'] & Q.n \arrow[d, dotted, "h"] \\
S \arrow[r, dotted, "n'"] & S.n'
\end{tikzcd}$$

P-move
square

What is a transfer?

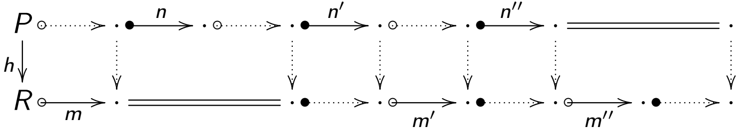
A transfer from $\mathcal{G}$ to $\mathcal{H}$ consists of

- a collection of passive linkers
- a collection of active linkers
- a collection of interaction squares (of four kinds) saying how to respond to every external O-move and every internal P-move.

Interaction sequence

An interaction sequence from a linker is a sequence of interaction squares.

internal play



external play

Transferring strategies

Given a transfer $\mathcal{O}$ from $\mathcal{G}$ to $\mathcal{H}$

and a linker $h : P \to R$,

each strategy σ from P gives a strategy $\mathcal{O}(\sigma)$ from R

viz. the set of all external plays of interaction sequences from h whose internal play is in σ.

Compositionality theorem for transfers

Recall

Given a transfer $\mathcal{O}$ from $\mathcal{G}$ to $\mathcal{H}$

and a linker $h : P \to R$,

each strategy σ from P gives a strategy $\mathcal{O}(\sigma)$ from R

Compositionality theorem for transfers

Recall

Given a transfer $\mathcal{O}$ from $\mathcal{G}$ to $\mathcal{H}$

and a linker $h : P \to R$,

each strategy σ from P gives a strategy $\mathcal{O}(\sigma)$ from R

Suppose x is a state of a system over $\mathcal{G}$

and x' is a state of a system over $\mathcal{H}$,

we want $[\![x']\!] = \mathcal{O}[\![x]\!]$

i.e. the transfer correctly predicts the semantics of x'.

Compositionality theorem for transfers

Recall

Given a transfer $\mathcal{O}$ from $\mathcal{G}$ to $\mathcal{H}$

and a linker $h : P \to R$,

each strategy σ from P gives a strategy $\mathcal{O}(\sigma)$ from R

Suppose x is a state of a system over $\mathcal{G}$

and x' is a state of a system over $\mathcal{H}$,

we want $[\![x']\!] = \mathcal{O}[\![x]\!]$

i.e. the transfer correctly predicts the semantics of x'.

This is achieved with a special kind of relation between states, called a stepped bisimulation.

What have we gained?

To describe a game semantics we first give a transition system over a game.

Then for each term constructor we give a transfer, and a stepped bisimulation to demonstrate its correctness.

We only need to talk about individual moves. Plays and strategies are handled by our compositionality theorems.

Further directions

- Game semantics for many different languages
- Creating new instances of the same game.
- Equations between operations on strategies arising from transfers
- Nondeterminism, probability, ...
- Concurrency?