Game semantics and Agda

Martin Churchill, University of Bath
Joint work with Jim Laird and Guy McCusker (theory)
and Makoto Takeyama (Agda formalisation)

1st/2nd February, 2010
Centre for Verification and Semantics, AIST, Osaka

Overview

- Games and strategies can be used to provide a denotational semantics for a higher-order imperative programs (many full abstraction results etc...)
- Some recent work in Bath includes a logical (and categorical) characterisation of the universe of games.
- Here we seek to formalise game semantics, and this logical characterisation, in Agda.

Games

Definition

A *game* is a triple (M_A, λ_A, P_A) where

- M_A is a set of moves
- $\lambda_A : M_A \to \{O, P\}$ divides these moves into *player*-moves and *opponent*-moves
- $M_A^{\circledast}$ is the set of λ_A-alternating sequences over M_A (we do not require that O starts)
- $P_A \subseteq M_A^{\circledast}$ is a set of valid plays (downwards-closed)

Example: $\mathbf{B} = (\{q, t, f\}, \{q \mapsto \mathrm{O}; t, f \mapsto \mathrm{P}\}, \{\epsilon, q, qt, qf\})$

In practice we deal with *positive* games (where P always starts) or *negative* games (where O always starts.)

Strategies

Definition

A *strategy* σ for a game (M_A, λ_A, P_A) is a nonempty subset of P_A such that

- All plays in σ end in a P-move
- If $sop \in \sigma$ then $s \in \sigma$
- If $s, t \in \sigma$ with $s \neq t$ then s and t first differ at an O-move

A strategy is *total* if P can always respond to valid O-moves.
Example : One total strategy on **B** for each Boolean.

Example

- For games M and N, we can define a derived game $M \multimap N$ whereby plays consist of plays in N interacting with an "inverted" version of M.
- Similarly we have a pairing operation $\otimes$.
- Strategy for and:

$\mathbf{B}$	$\otimes$	$\mathbf{B}$	$\Rightarrow$	$\mathbf{B}$	
				q	O
q					P
b_1					O
		q			P
		b_2			O
				$b_1 \wedge b_2$	O

Agda games

- A game is really a *tree* of plays, together with a polarity
  - The polarity of each node in the tree is determine by its position, and the polarity of the tree.
- But polarities are only relevant when we are considering strategies, so it is convinient to postpone its involvement...

```
data Game' : Set1 where
gam : {i : Set} -> (i -> Game) -> Game

data Game : Set where
gam : {i : MovEnc} -> (T i -> Game) -> Game
```

We will write `mv G` for its set of moves and `G > m` for the subgame of `G` occuring after move `m` is played.

Agda strategies

- A strategy on a game viewed as a tree, is just a subtree such that at all opponent nodes, only one child of that node is in the tree.
- Opponent nodes are either even-levelled nodes or odd-levelled nodes depending on polarity.

```
data Strat : Pol -> Game -> Set where
pos :  {G} -> (i : mv G) -> Strat - (G > i) -> Strat + G
neg :  {G} -> ((i : mv G) -> Strat + (G > i)) -> Strat - G
```

- So `Strat - G` represents a strategy on `G` viewed as a negative game, and `Strat + G` represents a strategy on `G` viewed as a positive game.

Examples

```
I : Game
I = gam {nil} bot-elim

eps : Strat - I
eps = neg (λ r -> bot-elim r)

B : Game
B = gam one (λ q -> gam {one + one} [ (λ t -> I) , (λ f -> I) ])

t : Strat - B
t = neg (λ q -> pos (inj1 q) eps)
```

Operators

- We also encode the various operators on games, e.g. $\multimap$ and $\otimes$:

```
mutual
—o : Game -> Game -> Game
(gam {j} g) —o (gam {i} f) = gam {i} (λ i' -> f i' ⊗ gam {j} g)

⊗ : Game -> Game -> Game
(gam {i} f) ⊗ (gam {j} g)
= gam {i + j} [ (λ i' -> gam {j} g —o f i') ,
(λ i' -> gam {i} f —o g i') ]
```

WS

- Recent work includes describing a logic where formulas represent games and proofs represent strategies.
- This logic is *fully complete* in the sense that all strategies are (uniquely) representable as a proof using some set of core rules.
- This is a very fine-grained logic (where individual moves of games are represented) inspired by the semantics

Formulas of WS

Positive and negative formulae:
$P := \mathbf{0} \mid \downarrow N \mid P \parr Q \mid P \oplus Q \mid P \lhd Q \mid P \oslash N$
$N := \mathbf{1} \mid \uparrow P \mid N \otimes M \mid M \& N \mid N \oslash M \mid N \lhd P$

- **Idea:** Positive (negative) formulae represent positive (negative) games. For each polarity, we have formulae representing the empty game, lifts, merge, sum, and leftmerge respectively.

Sequents are lists of formulas. Semantically, comma represents $\lhd$ or $\oslash$ depending on the polarity of the right hand side and is left-associative.

WS formulas in Agda

```
data Fml : Pol -> Set where
F0 : Fml +
F1 : Fml -
↓ : (P : Fml +) -> Fml -
↑ : (M : Fml -) -> Fml +
⊗ : (M N : Fml -) -> Fml -
⅋ : (P Q : Fml +) -> Fml +
⊕ : (P Q : Fml +) -> Fml +
& : (M N : Fml -) -> Fml -
⊘ :  p -> (A : Fml p) (M : Fml -) -> Fml p
◁ :  p -> (A : Fml p) (P : Fml +) -> Fml p
```

Proof rules of WS

$$\frac{}{\vdash \mathbf{1}, \Gamma}$$

$$\frac{\vdash M, N, \Gamma \qquad \vdash N, M, \Gamma}{\vdash M \otimes N, \Gamma}$$

$$\frac{\vdash P, Q, \Gamma}{\vdash P \parr Q, \Gamma}$$

$$\frac{\vdash Q, P, \Gamma}{\vdash P \parr Q, \Gamma}$$

$$\frac{\vdash P, \Gamma}{\vdash P \oplus Q, \Gamma}$$

$$\frac{\vdash Q, \Gamma}{\vdash P \oplus Q, \Gamma}$$

$$\frac{\vdash M, \Gamma \qquad \vdash N, \Gamma}{\vdash M \& N, \Gamma}$$

$$\frac{\vdash P}{\vdash \uparrow P}$$

$$\frac{\vdash \uparrow (P \parr Q), \Gamma}{\vdash \uparrow P, Q, \Gamma}$$

$$\frac{\vdash \uparrow (P \oslash N), \Gamma}{\vdash \uparrow P, N, \Gamma}$$

$$\frac{\vdash N}{\vdash \downarrow N}$$

$$\frac{\vdash \downarrow (M \otimes N), \Gamma}{\vdash \downarrow M, N, \Gamma}$$

$$\frac{\vdash \downarrow (N \triangleleft P), \Gamma}{\vdash \downarrow N, P, \Gamma}$$

$$\frac{\vdash A, N, \Gamma}{\vdash A \oslash N, \Gamma}$$

$$\frac{\vdash A, P, \Gamma}{\vdash A \triangleleft P, \Gamma}$$

Proof rules in Agda

```
data ⊢ : ∀ {p} -> Seq p -> Set where
P1 : ∀ Γ -> ⊢ F1 , Γ
P⊗ : ∀ M N Γ -> ⊢ M , N , Γ -> ⊢ N , M , Γ -> ⊢ M ⊗ N , Γ
P⅋ : ∀ P Q Γ -> ⊢ P , Q , Γ -> ⊢ P ⅋ Q , Γ
P⅋ : ∀ P Q Γ -> ⊢ Q , P , Γ -> ⊢ P ⅋ Q , Γ
P⊕₁ : ∀ P Q Γ -> ⊢ P , Γ -> ⊢ P ⊕ Q , Γ
P⊕₂ : ∀ P Q Γ -> ⊢ Q , Γ -> ⊢ P ⊕ Q , Γ
P& : ∀ M N Γ -> ⊢ M , Γ -> ⊢ N , Γ -> ⊢ M & N , Γ
P⊘ : ∀ pA : Fml pM Γ -> ⊢ A , M , Γ -> ⊢ A ⊘ M , Γ
P◁ : ∀ pA : Fml pP Γ -> ⊢ A , P , Γ -> ⊢ A ◁ P , Γ
P↑ : ∀ P -> ⊢ P ,  -> ⊢ ↑ P ,
P↑- : ∀ P N Γ -> ⊢ ↑(P ⊘ N) , Γ -> ⊢ ↑ P , N , Γ
P↑+ : ∀ P Q Γ -> ⊢ ↑(P ⅋ Q) , Γ -> ⊢ ↑ P , Q , Γ
P↓ : ∀ N -> ⊢ N ,  -> ⊢ ↓ N ,
P↓+ : ∀ N P Γ -> ⊢ ↓(N ◁ P) , Γ -> ⊢ ↓ N , P , Γ
P↓- : ∀ M N Γ -> ⊢ ↓(M ⊗ N) , Γ -> ⊢ ↓ M , N , Γ
```

Semantics of Proofs

The semantics of a proof is given as a strategy. If we represent the Booleans as $\uparrow (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1})$ then `not` is given as follows:

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \mathbf{1}, \mathbf{0}}{\vdash \mathbf{1} \lhd \mathbf{0}}}{\vdash \downarrow (\mathbf{1} \lhd \mathbf{0})}}{\vdash \downarrow \mathbf{1}, \mathbf{0}}\ \text{rule } \oplus_2}{\vdash \downarrow \mathbf{1} \oplus \downarrow \mathbf{1}, \mathbf{0}}}{\vdash \mathbf{0} \parr (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1})}}{\vdash \uparrow (\mathbf{0} \parr (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1}))}}{\vdash (\uparrow \mathbf{0}), (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1})}
\qquad
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \mathbf{1}, \mathbf{0}}{\vdash \mathbf{1} \lhd \mathbf{0}}}{\vdash \downarrow (\mathbf{1} \lhd \mathbf{0})}}{\vdash \downarrow \mathbf{1}, \mathbf{0}}\ \text{rule } \oplus_1}{\vdash \downarrow \mathbf{1} \oplus \downarrow \mathbf{1}, \mathbf{0}}}{\vdash \mathbf{0} \parr (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1})}}{\vdash \uparrow (\mathbf{0} \parr (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1}))}}{\vdash (\uparrow \mathbf{0}), (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1})}
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash (\uparrow \mathbf{0} \& \uparrow \mathbf{0}), (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1})}{\vdash (\uparrow \mathbf{0} \& \uparrow \mathbf{0}) \lhd (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1})}}{\vdash \downarrow ((\uparrow \mathbf{0} \& \uparrow \mathbf{0}) \lhd (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1}))}}{\vdash \downarrow (\uparrow \mathbf{0} \& \uparrow \mathbf{0}), (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1})}}{\vdash (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1}) \parr \downarrow (\uparrow \mathbf{0} \& \uparrow \mathbf{0})}}{\vdash \uparrow ((\downarrow \mathbf{1} \oplus \downarrow \mathbf{1}) \parr \downarrow (\uparrow \mathbf{0} \& \uparrow \mathbf{0}))}
$$

$$
\dfrac{\vdash \uparrow (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1}), \downarrow (\uparrow \mathbf{0} \& \uparrow \mathbf{0})}{\vdash \uparrow (\downarrow \mathbf{1} \oplus \downarrow \mathbf{1}) \lhd \downarrow (\uparrow \mathbf{0} \& \uparrow \mathbf{0})}
$$

Semantics in Agda

- We seek to formalise the semantics in Agda:

```
[[]] : ∀ {p}{Γ : Seq p} -> ⊢ Γ -> Strat p ⟦Γ⟧
```

- To do this, we need the semantic machinery (with strategies) in order to formalise the strategy-semantics in Agda.

Copycats

- To give the semantics of the rules, we require *copycat strategies* to mediate between various equivalent, but syntactically distinct, games.
- A copycat strategy comes from a more primitive structure on games, the ability to embed one into the other.

```
data sub : Game -> Game -> Set where
subp : ∀ {A B} -> (h : mv A -> mv B) ->
((i' : mv A) -> B > (h i') sub A > i' ) -> A sub B

copycat : ∀ {A B} -> B sub A -> Strat - (A —o B)
copycat {gam a} {gam b} (subp f g) = ...
```

This idea is actually more naturally expressed in the type-theoretic definition of games we are using, rather than the move-based one.

Where is this leading?

- Strategies represent a recipe for describing how a term should behave in response to the requests by its envrionment
- Thus, we can add a component to our system to "run" a strategy, whereby the user plays the role of the environment (Opponent) and the computer responds appropriately according to the strategy.
- The alternating nature of the games represent the computer and the user taking it in turns to play moves.
- "Copycat" on A then represents the strategy of having two versions of A running in parallel, where the computer throws the users moves back at the user in the other copy.

Pipeline

Idea:

- We have an imperative object, represented as a proof term
- We take its semantics as a strategy
- We can then run the strategy...

But it is useful to be able to use more high-level "features" in the proof-language than the movewise ones.

Admissibility

Full completeness implies rules for composing strategies are admissible:

$$\frac{}{\vdash N, N^{\perp}} \qquad \frac{\vdash A, \Gamma, N^{\perp} \qquad \vdash N, \Delta}{\vdash A, \Gamma, \Delta} \qquad \frac{\vdash M, \Gamma, \Delta \qquad \vdash N, \Delta'}{\vdash M, \Gamma, N, \Delta, \Delta'}$$

These actions represent general composition of strategies, which require merging the two strategies in question. Thus their semantics are clear, but can we emulate what's happening in the semantics in the proof system?

Cut elimination

Yes — we can define a cut-elimination procedure working syntactically on proofs.

```
cut : ∀ {p}{A : Fml p}(Γ : Ctx){N : Fml - }{P : Fml +} ->
⊢ A , Γ ,, (N ⊥ , ) -> ⊢ N , P ,  -> ⊢ A , Γ ,, (P , )
```

- Unfortunately, Agda doesn't currently see this as terminating.
  - Due to one use of `subst` applied to one of the arguments of an auxilary function — Agda does not infer that if f is smaller than g then so is `subst eq f`.
  - But `cut` is currently written in such a way to be maximally convincing to the reader that it is terminating.
  - This could potentially be solved using *sized types*

Next steps

- Giving other higher-level rules elimination procedures, thus increasing the tools available to be used in programming imperative objects in WS
- Formalising the full completeness theorem in Agda
- Continuing to work on the interactive elements of the system, e.g. annotating games to make the interaction more plesant for the user.

Working on the system in Agda has provided the many insights, including:

- Formalising game semantics inductively as a tree has been interesting, and nicely complements the traditional approach, making various things simpler (e.g. game embeddings).
- Significantly simplified the formulation of the cut-elimination procedure I had.

Questions?