Object-Oriented Semantics

Objects are not special.

Properties:

* Constructing objects — initialization which is user-defined
* Object : combination of data and behavior
* Inheritance
* Classes — general defn of sorts of objects
* Encapsulation
* Self-aware

Encoding objects in FbSR

Record Rule

$$\frac{\langle S_0, e_1\rangle \Rightarrow \langle S_1, v_1\rangle \quad \cdots \quad \langle S_{n-1}, e_n\rangle \Rightarrow \langle S_n, v_n\rangle}{\langle S_0, \{l_1 = e_1, \ldots, l_n = e_n\}\rangle \Rightarrow \langle S_n, \{l_1 = v_1, \ldots, l_n = v_n\}\rangle}$$

Encode:
object w/ 2 fields: "wins" & "losses"
1 method: "totalGames".

```
Let obj =
  { wins = Ref 0;
    losses = Ref 0;
    totalGames = Function this -> !this.wins + !this.losses
  }
In
obj.wins := !obj.wins + 1;
obj.totalGames obj
```

Object Extension

```
Let obj = ...
In
Let obj2 =
  { ties = Ref 0;
    totalGames = Function this -> !this.ties + obj.totalGames this
    wins = obj.wins;
    losses = obj.losses;
  }
In
```

pass itself

```
class Foo {
    void foo();      <-- "statically-dispatched"
    virtual void bar(); <-- "dynamically-dispatched"
```

pass subobjects

```
Let obj =
  { rate = Function this ->
      !this.wins + !this.wins;
    compare = Function this -> Function other ->
      lessThan (!this.rate this) (!other.rate other);
    ⋮
  }
In
Let obj2 =
  { rate = Function this ->
      !this.ties + obj.rate this;
    compare = ...
    ⋮
  }
In
obj2.compare obj2
```

```
{
    new = Function this -> Function w -> Function l ->
        this.createdCount := !this.createdCount + 1;
        Let private =
            { wins = Ref w;
              losses = Ref l;
            }
        In
        {
            totalGames = Function this -> !private.wins + !private.losses
        };
    createdCount = Ref 0;
}
```