CS162 Final Project Writeup (Rough Draft)
NotWorlds

David Schnurr
James Ebentier
Vincent Alindogan

Our original vision for notWorlds was a language that gives programmers the ability to create and execute code in a completely self-contained environment called a "World", which has no effect on the original environment in which it was created. We have followed through with the goals that we originally set out, and our current iteration of notWorlds supports the following features:

- Referencing the current world
- Sprouting new worlds from an existing world
- Executing arbitrary code within a specific world
- Committing changes from a child world back to its parent
- Updating a child world by pulling changes from its parent
- Comparing worlds by testing equality of the values contained within them

(At the time of writing this rough draft, commit and update will affect all values in the world. For our final milestone we plan on supporting an argument of a list of variables that will commit or update only the specified changes)

notWorlds is an extension of notJS, and as such, the syntax and semantics are largely the same. The things that we did change are outlined below:

Abstract Syntax changes:

We add the following expression for executing code inside a world:

within ω t

Semantic Domains changes:

The abstract machine configuration now takes a World instead of an Env:

$C \in Config = Term \times World \times Store$

The language values now includes a new value for Worlds:

$\omega \in World = Env \times World$ (The second world is the parent)

notWorlds Semantics changes:

We added a WITHIN case, for executing code within a world:

$$\frac{[[e \cdot \omega\sigma]] \Rightarrow (\omega_1, \sigma_1)}{[[\ \textbf{within}\ e\ t \cdot \omega\sigma\ ]] \Rightarrow [[\ t \cdot \omega_1\sigma_1]]}\ \text{(WITHIN)}$$

Our EQ case has changed to handle worlds but we're still in the process of formally defining the semantics.

Our BLOCK case has changed to handle worlds but we're still in the process of formally defining the semantics.

Our ACC case has changed to handle accessing a variable from a world and also calling methods on a world, but we're still in the process of formally defining the semantics.