

Activity: Revising Robs

A. Why?

When defining formal systems, it's often possible to play with the definitions so that the system is easier to use. For the Rob system in particular, the $()$ values were there to enforce a particular property Robs should have. Here, we'll look at a different (easier?) way to set up that property.

B. Outcomes

At the end of this activity, you should be able to more easily calculate the f and g functions on Robs. It may also be clearer just what a Rob is.

C. Redefining Robs

A Rob r follows the syntax $r ::= * \mid @ \mid (*, r) \mid (@, r)$. (We're dropping the weird $()$ values from last time.)

The straightforward cases for the Rob f and g definitions are when the arguments are both base case or both inductive case: Below, $s1$ and $s2$ are $*$ or $@$ (base case Robs).

Rule fBase: $f(*, *) = *$; $f(*, @) = f(@, *) = @$; $f(@, @) = (*, @)$

Rule gBase: $g(*, *) = @$; $g(*, @) = g(@, *) = (*, @)$; $g(@, @) = (@, @)$

For the next two rules, let $t = f(r1, r2)$ and $u = g(r1, r2)$.

Rule fRec: $f((s1, r1), (s2, r2)) = (*, t)$ if $s1=s2=*$, $(*, u)$ if $s1=s2=@$, and $(@, t)$ otherwise

Rule gRec: $g((s1, r1), (s2, r2)) = (@, t)$ if $s1=s2=*$, $(@, u)$ if $s1=s2=@$, and $(*, u)$ otherwise

Last time, the weird definitions with $()$ got added because f and g might have arguments of different lengths (i.e., depth of nested pairs). A different way around the problem is to behave as though the shorter Rob is right-extended with extra $*$ symbols. We can get this effect by adding two more rules: If only one of the two arguments is a base-case Rob, we add a $*$ to it.

Rule Extend: $h(s1, (s2, r2)) = h((s1, *), (s2, r2))$ where h is f or g

and $h((s1, r1), s2) = h((s1, r1), (s2, *))$ where h is f or g

D. Sample Calculations

- | | | |
|--|-------------------|--|
| <p>1. $f(@, *)$
 $= @$</p> | <p>Rule fBase</p> | <p>5. $f(@, (*,(*,@)))$
 $= f((@, *), (*,(*,@)))$ Rule Extend
 $= (@, f(*, (*,@)))$ Rule fBase
 $= (@, f((*, *), (*,@))$ Rule Extend
 $= (@, (*, f(*,@)))$ Rule fRec
 $= (@, (*, @))$ Rule fBase</p> |
| <p>2. $f(@, @)$
 $= (*, @)$</p> | <p>Rule fBase</p> | |
| <p>3. $f(@, (*,@))$
 $= f((@, *), (*,@))$ Rule Extend
 $= (@, f(*,@))$ Rule fRec
 $= (@, @)$ Rule fBase</p> | | <p>6. $f(@, (@,(*,@)))$
 $= f((@, *), (@,(*,@)))$ Rule Extend
 $= (*, g(*, (*,@)))$ Rule fRec
 $= (*, g((*, *), (*,@))$ Rule Extend
 $= (*, (@, f(*,@)))$ Rule gRec
 $= (*, (@, @))$ Rule fBase</p> |
| <p>4. $f(@, (@,@))$
 $= f((@, *), (@,@))$ Rule Extend
 $= (*, g(*,@))$ Rule fRec
 $= (*, (*, @))$ Rule gRec</p> | | |

E. Questions

1. Complete the following calculation:

- $f(@, (*,(@,@)))$
 $=$ _____ Rule Extend $f((@, *), (*,(@,@)))$
 $=$ _____ Rule fBase $(@, f(*, (@,@)))$
 $=$ _____ Rule Extend $(@, f((*, *), (@,@)))$
 $=$ _____ Rule fBase $(@, (@, f(*,@)))$
 $=$ _____ Rule fBase $(@, (@, @))$

2. Complete the following calculation.

- $f(@, (@,(@,@)))$ Extend
 $= f((@, *), (@,(@,@)))$ fRec
 $= (*, g(*, (@,@)))$ Extend
 $= (*, g((*, *), (@,@)))$ gRec
 $= (*, (*, g(*,@)))$ gBase
 $= (*, (*, (*, @)))$

Abbreviate things by omitting parentheses and comma

Going through the f(@, previous line)

@

*@

@@

**@

@*@

*@@

@@@

***@

@**@

@@

@@*@

**@@

@*@@

*@@@

@@@@

****@

What if we use 0 for * and 1 for @?

1

01

11

001

101

011

111

0001

1001

0101

1101

0011

1011

0111

1111

Rob = Reverse order binary number

f(x,y) = addition

g(x,y) = addition with carry bit