# Dependent Asymmetric Paths

by Sven Nilsen, 2019

*In this paper I introduce notation for dependent asymmetric paths.*

A dependent asymmetric path is an asymmetric path[1] which depends on the value on some previous argument. The idea is similar to dependent types[2], but for asymmetric paths.

Assume the following system:

∵ select : state × item → state
∵ selected : state × item → bool

∵ |state| > |item| The type of `state` is larger than the type of `item`

One would like to translate the following from first-order logic[3] to path semantical notation:

∀ s, it { selected(select(s, it), it) = true } First order logic

By using the notation for dependent asymmetric paths and notation for existential path equations[4]:

∵ select[unit × [id] it → (selected it)] <=> \true Dependent asymmetric path

∴ it : item
∴ (selected it) : state → bool
∴ select[unit × [id] it → (selected it)] : () × item → bool Can be treated as `item → bool`

The `[id] it` expression introduces the variable `it` at the same time as telling that the information about `item` is preserved in the asymmetric path. For more information about `id` and `unit`, see the standard dictionary for list of functions[5].

Notice that since the information about the state is erased, the following would be absurd:

select[unit × [id] it → id] : item → state ERROR: `|state| > |item|`

This is because one can not map deterministically from a smaller type with pure functions.
However, since `(selected it)` maps to a type `bool`, the type becomes `item → bool`.

Starting with decoupled syntax, one can see that the output path depends on the argument:

∵ select(_ : state, it : item) = [selected it] true
∴ select(_ : state, it : [id] it) = [selected it] true
∴ select[unit × id]((), it) = [selected it] true
∴ select[unit × [id] it → (selected it)] <=> \true

## References:


[1] “Algebraic Notation for Asymmetric Paths”
Sven Nilsen, 2017
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/algebraic-notation-for-asymmetric-paths.pdf

[2] “Dependent type”
Wikipedia
https://en.wikipedia.org/wiki/Dependent_type

[3] “First-order logic”
Wikipedia
https://en.wikipedia.org/wiki/First-order_logic

[4] “Higher Order Operator Overloading and Existential Path Equations”
Sven Nilsen, 2019
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/higher-order-operator-overloading-and-existential-path-equations.pdf

[5] “Standard Dictionary for Path Semantics – Alphabetic List of Functions”
Sven Nilsen, 2017
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/alphabetic-list-of-functions.pdf