Order Preserving Embeddings

James Chapman
Institute of Cybernetics, Tallinn

Joint work with Peter Morris, original idea due to Altenkirch, Hofmann and Streicher. It was shown to me by Conor McBride

This work in context

- I am interested in:
  - Semantics of programming languages based on lambda-calculus.
  - Formalised in type theoretic theorem provers such as Agda, Epigram and Coq
    - In these systems:
      - Proofs and programs are the same
      - The types of programs can express their specifications

What’s the problem?

- Doing these things in detail, as you are forced to do in a theorem prover, makes it very important to use appropriate representations of things
- The greater the level of detail, the greater the choice of representation
- A general problem for formal mathematics and programming language semantics

Finite sets

```
data Fin : Nat -> Set where
  fzero : Fin (suc n)
  fsuc  : Fin n -> Fin (suc n)
```

Pictorial enumeration of the first three instances:

`Fin zero`

`Fin (suc zero)`

`Fin (suc (suc zero))`

Example usage:

```
safe_lookup : Fin n -> Array X n -> X
```

Untyped Lambda calculus

The well-scoped lambda terms

```
data Lam : Nat -> Set where
  var : Fin n -> Lam n
  λ   : Lam (suc n) -> Lam n
  app : Lam n -> Lam n -> Lam n
```

Example expressions:

```
λ (var fzero)                -- identity function
λ (λ (var (fsuc fzero))) -- 'K' combinator
```

What can you go from here?

- Define syntactic operations:
  - Weakening, substitution, etc.
- Implement an evaluator/normaliser
- Extend it:
  - Data types, effects, annotate with simple or dependent types, etc.

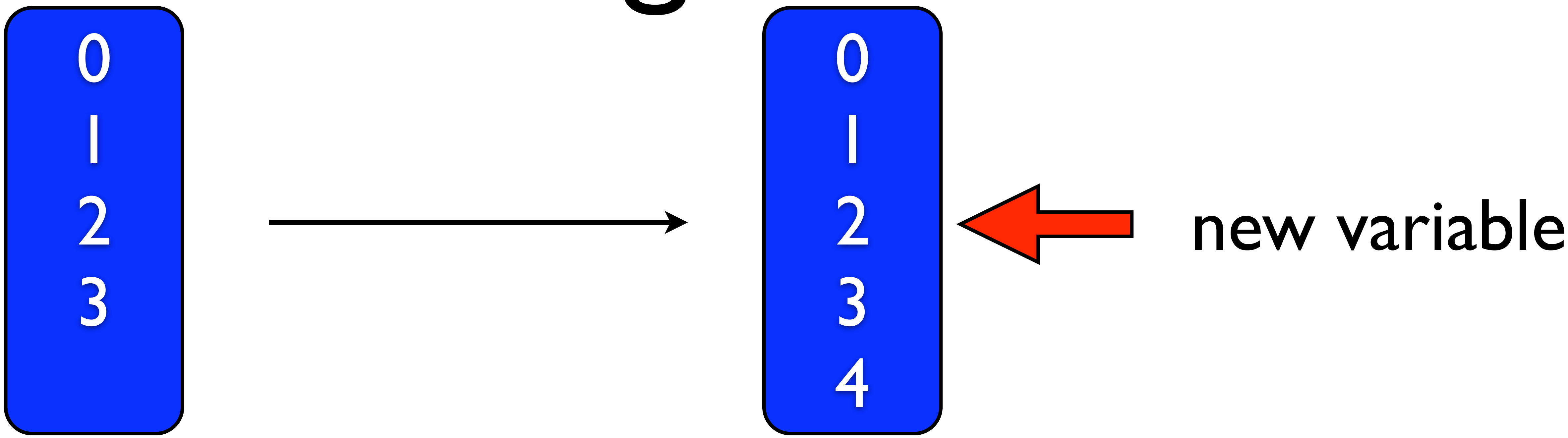
Implementing Weakening

Weakening adds a fresh variable at the 'zero' position and increments the rest

```
weak : Lam n -> Lam (suc n)
weak (var x)   = var (fsuc x)
weak (app t u) = app (weak t) (weak u)
weak (λ t)     = λ ? -- we're stuck
```

We need to generalise from adding a new variable at the end of the context to an arbitrary position

Thinning: A solution?



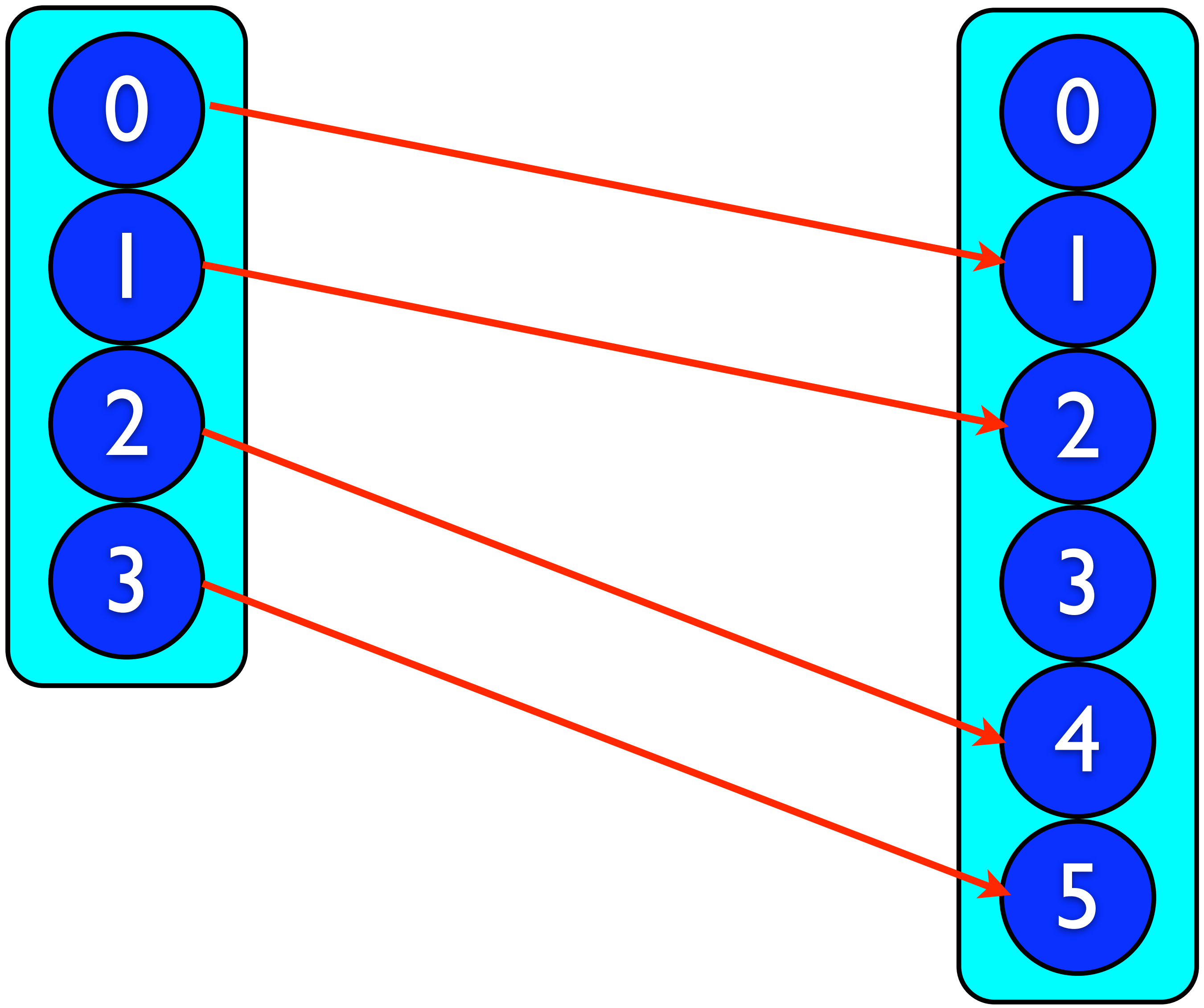
We implement it for terms as follows:

```
tlam : Fin (suc n) -> Lam n -> Lam (suc n)
tlam x (var y)   = var (tvar x y)
tlam x (app t u) = app (tlam x t) (tlam x u)
tlam x (λ t)     = λ (tlam (fsuc x) t)
```

Ordinary weakening for terms is now easy:

```
weak : Lam n -> Lam (suc n)
weak t = tlam fzero t
```

A better solution: Order Preserving Embeddings



OPEs

```
data OPE : Nat -> Nat -> Set where
  done : OPE zero zero
  skip : OPE m n -> OPE m (suc n)
  keep : OPE m n -> OPE (suc m) (suc n)
```

The identity OPE (`id : OPE n n`) is recursively defined

The weakening OPE is a now easy to define:

```
oweak : OPE n (suc n)
oweak = skip id
```

Action of OPEs

We can easily define the following operation lifting an OPE to a function on lambda expressions

```
olam : OPE m n -> (Lam m -> Lam n)
olam f (var x)   = var (ovar x)
olam f (app t u) = app (olam f t) (olam f u)
olam f (λ t)     = λ (olam (keep f) t)
```

Ordinary weakening for terms is now easy:

```
weak : Lam n -> Lam (suc n)
weak t = olam oweak t
```

OPEs form a category

- The objects are natural numbers: `0`, `1`, ...
- The morphisms are OPEs: `f`, `g`, ...
- For every object `n` an OPE `id : OPE n n`
- For every `f : OPE l m` and `g : OPE m n` there is an operation `●` such that `f ● g : OPE l n`
- and the following properties hold:
  - `f ● id = f` and `id ● f = f`
  - `f ● (g ● h) = (f ● g) ● h`

Why is this a good representation?

- Avoids reasoning about functions, first order structures are easier to deal with
- Avoids junk, captures only what we want
- Simple (elegant?) algebraic structure

Big-step Normalisation

- Central part of my thesis:
  - based on “Big-step Normalisation” by Altenkirch and Chapman
  - Published (soon!) in Special issue of Journal of Functional Programming. 2009. Eds. C. McBride and T. Uustalu
- Big win: simplified reasoning about weakenings; avoids problematic reasoning about context extensions.

Dependent types

- OPEs are helpful here for well-typed terms, as even defining variables requires reference to weakening.
- If we implement weakening by referring to variables we quickly get into a knot.
- OPEs avoid this and the fact the form a category become an integral part of the definition.