Capturing Hiproofs in HOL Light

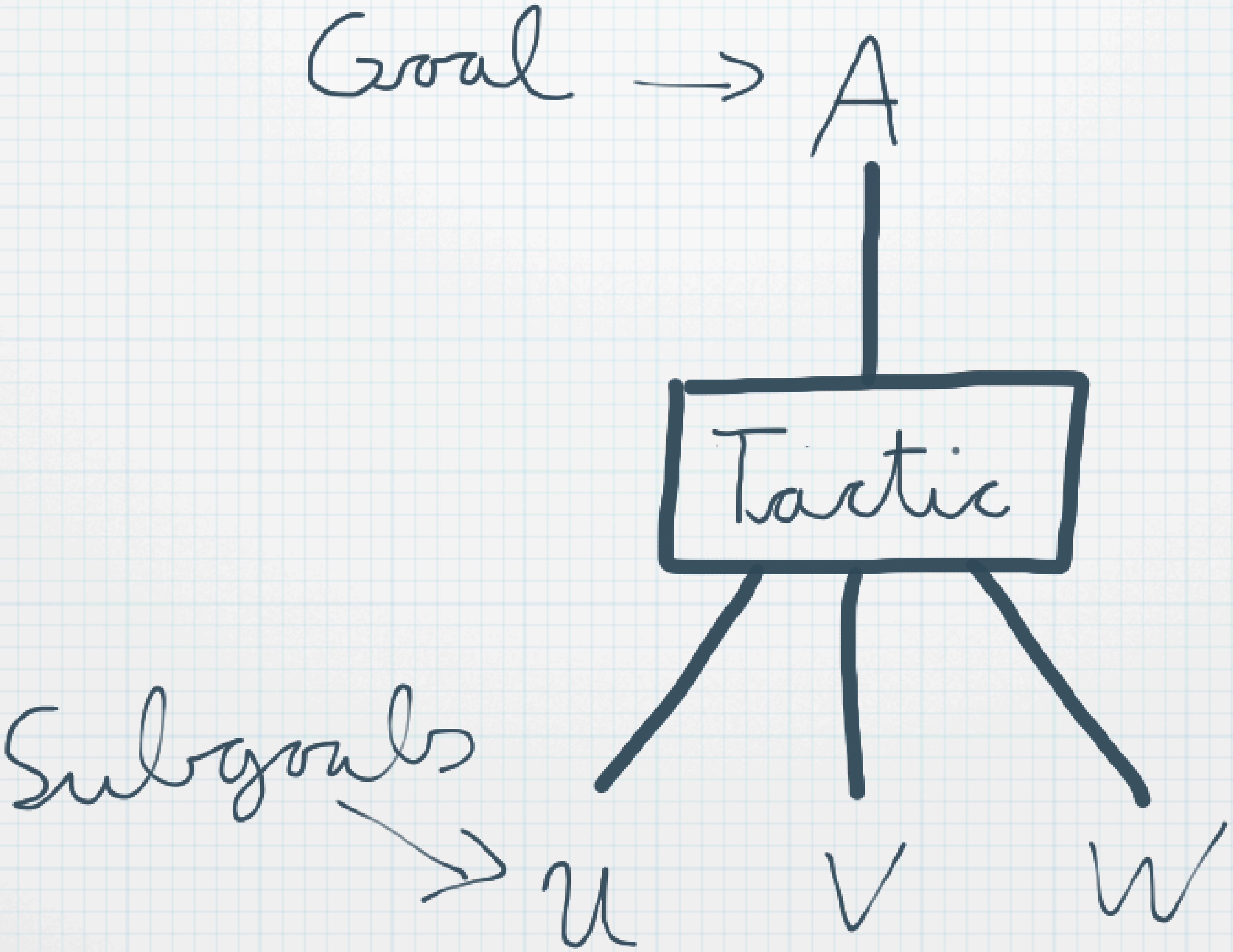
Steven Obua, David Aspinall, Mark Adams

Introduction to Hiproofs

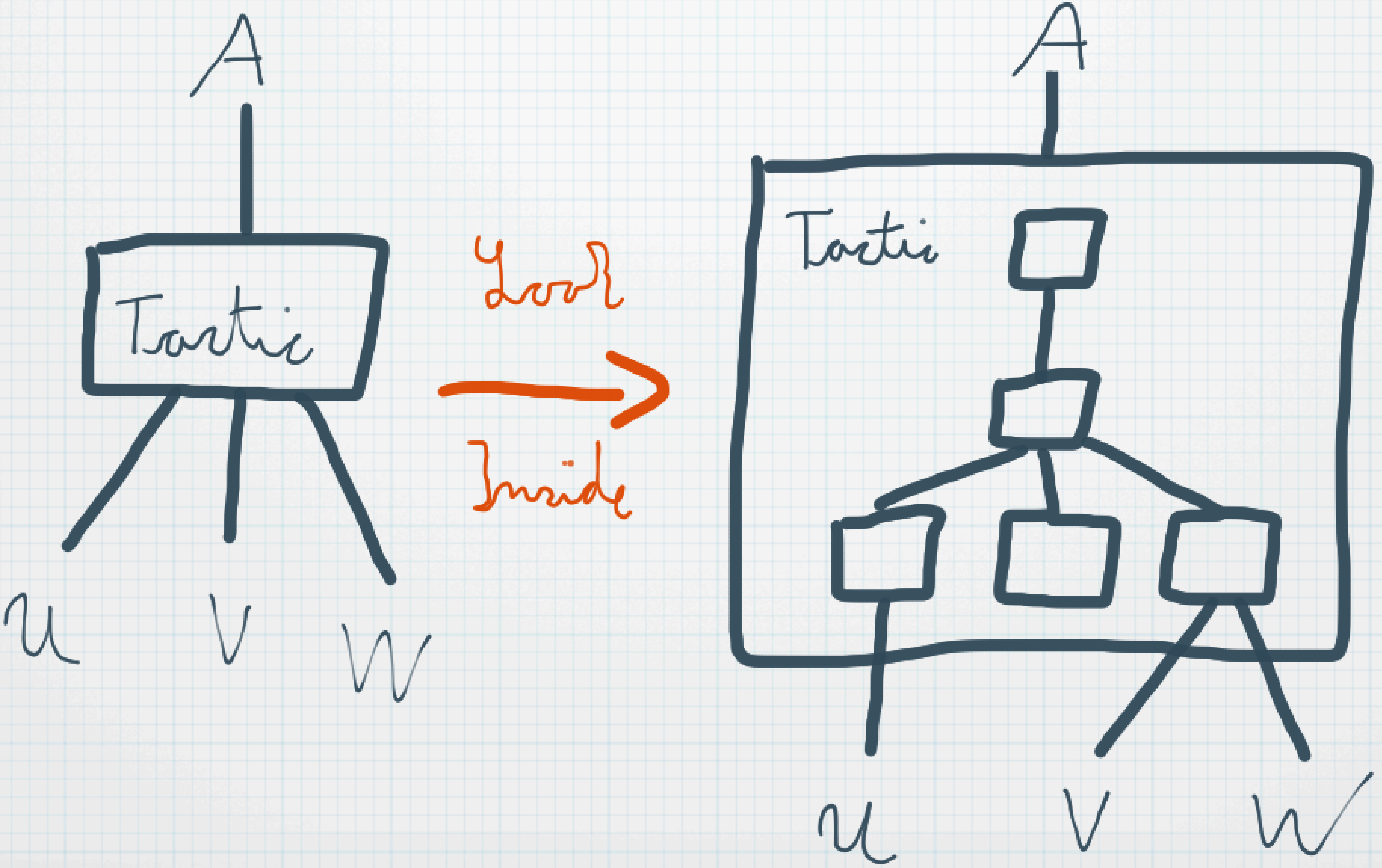
Hiproof History

- Hiproof = Hierarchical Proof
- Introduced by Ewen Denney et. al: **Hiproofs: A Hierarchical Notion of Proof Tree (2006)**
- Augmented with a syntax by David Aspinall, Ewen Denney et. al: **Tactics for Hierarchical Proof (2010)**

What's a Hiproof?

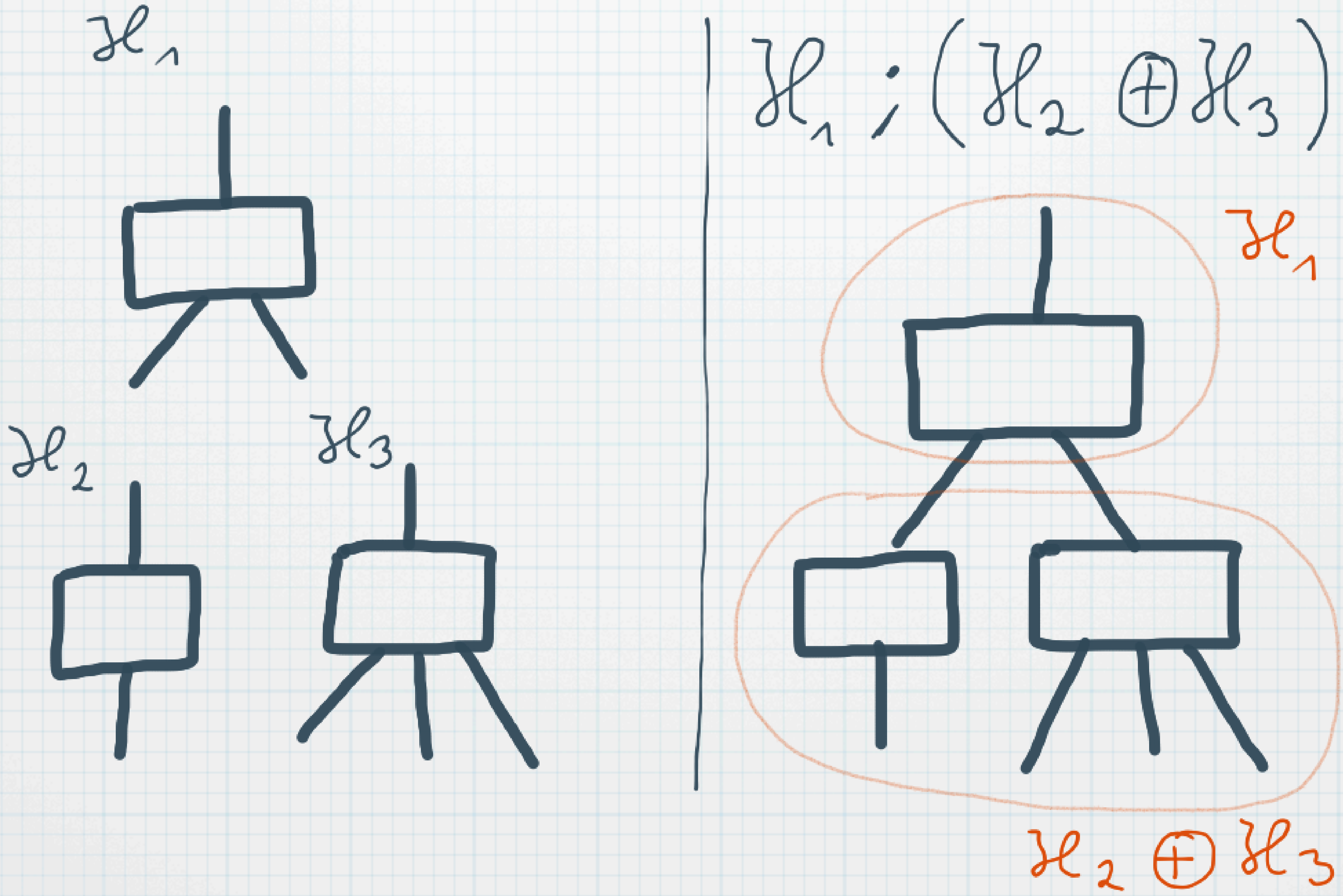


Looking Inside the Box



Monday, 11 February 13

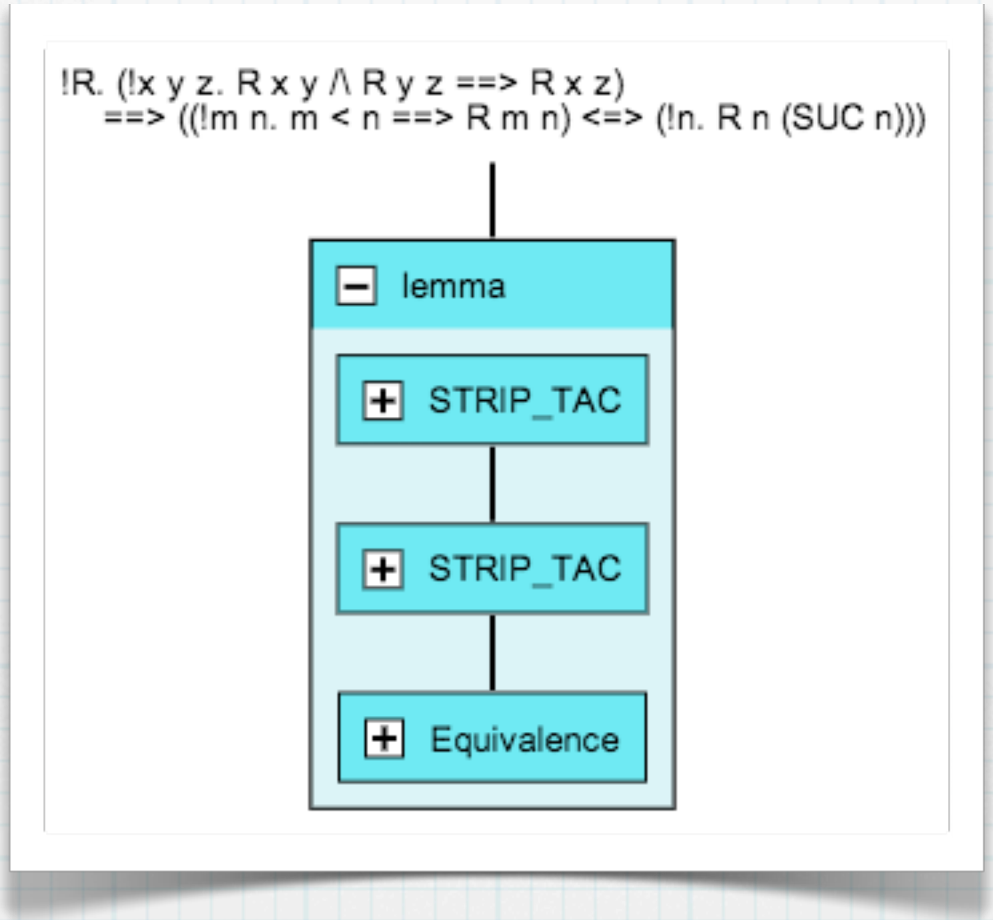
Tensors and Sequences


$\mathcal{H}_1$
$\mathcal{H}_2$
$\mathcal{H}_3$
$\mathcal{H}_1 ; (\mathcal{H}_2 \oplus \mathcal{H}_3)$
$\mathcal{H}_1$
$\mathcal{H}_2 \oplus \mathcal{H}_3$

Capturing HOL Light Hiproofs

Example:

TRANSITIVE_STEPWISE_LT_EQ



```
!R. (!x y z. R x y /\ R y z ==> R x z)
    ==> ((!m n. m < n ==> R m n) <=> (!n. R n (SUC n)))
```

Basic Idea

- augment `thm` datatype with hiproof:
  `hiproof : thm → hiproof`
- every kernel inference also produces corresponding hiproofs
- there is a labelling function for generating boxes around hiproofs:
  `hilabel : label → ? → ?`

Hiproof Datatype

(version 1)

```
type hiproof =
      Hi_atomic of int * label * goal
    | Hi_box of label * hiproof
    | Hi_tensor of hiproof list
    | Hi_sequence of hiproof list
```

Labels

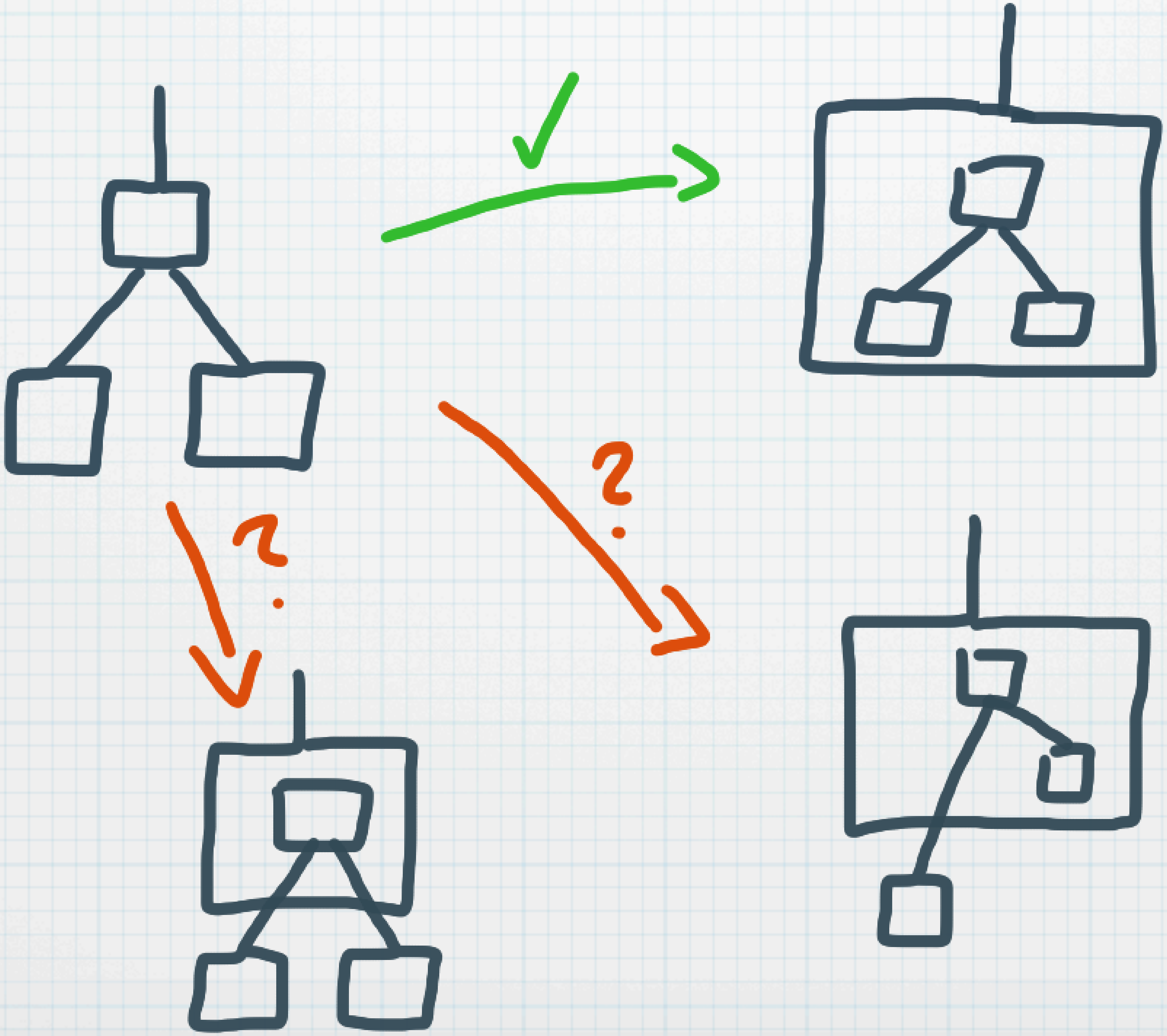
```
type label
type 'a labelconstr

val labelconstr : unit -> 'a labelconstr
val make_label : 'a labelconstr -> 'a -> label
val dest_label : 'a labelconstr -> label -> 'a option
```

How to use:

```
val make_string_label : string -> label
val dest_string_label : label -> string option

let lc_string = labelconstr ()
let make_string_label = make_label lc_string
let dest_string_label = dest_label lc_string
```

How to define hilabel ?

```
hilabel : label → thm → thm
```



Label rules, not (only) thms

```
type rule = thm list → thm

hilabel : label → rule → rule
```

Special Case: Theorems

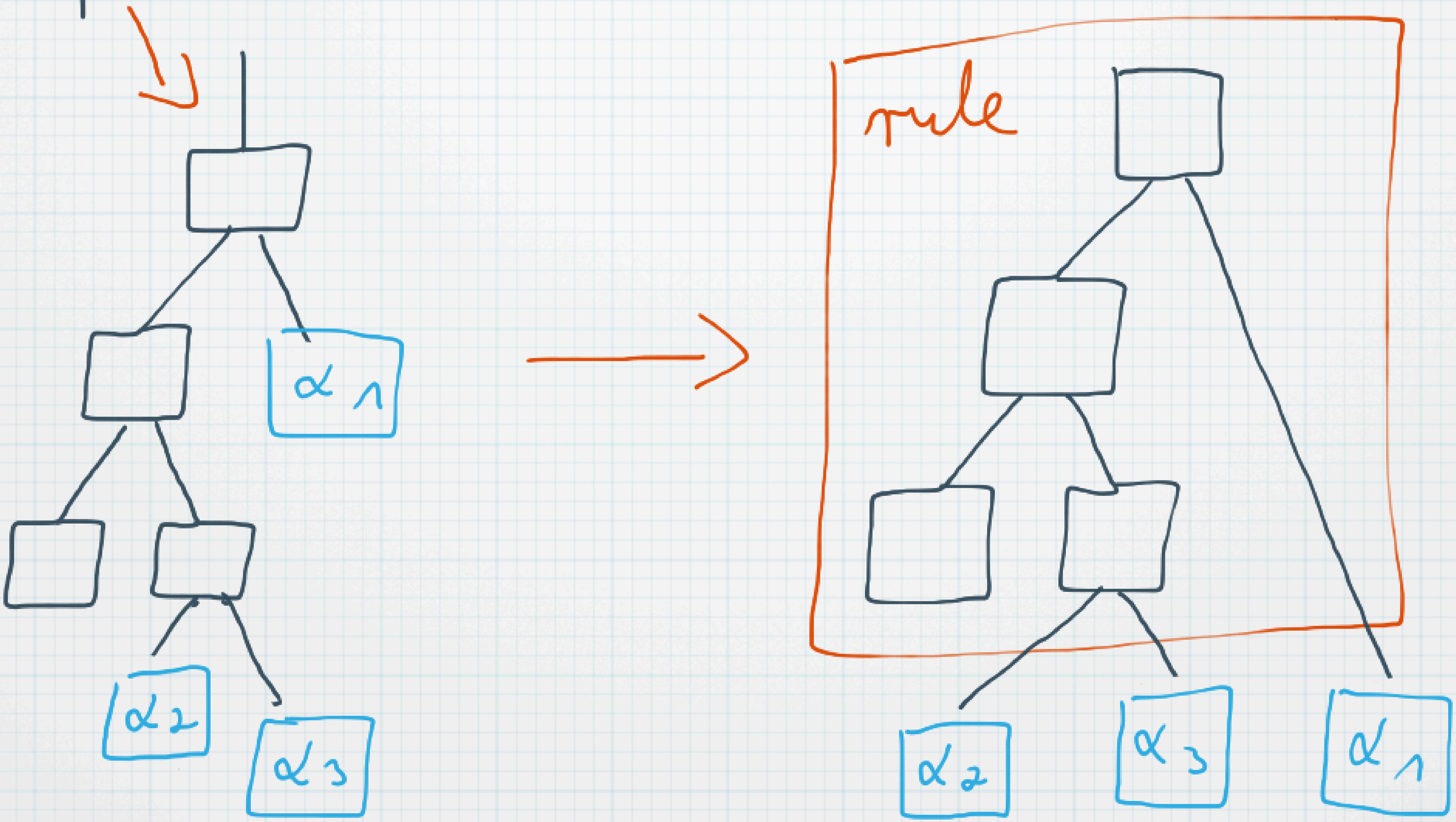
```
hilabel_thm : label → thm → thm

let hilabel_thm label thm =
  hilabel label (fun _ -> thm) []
```

Special Case: Tactics

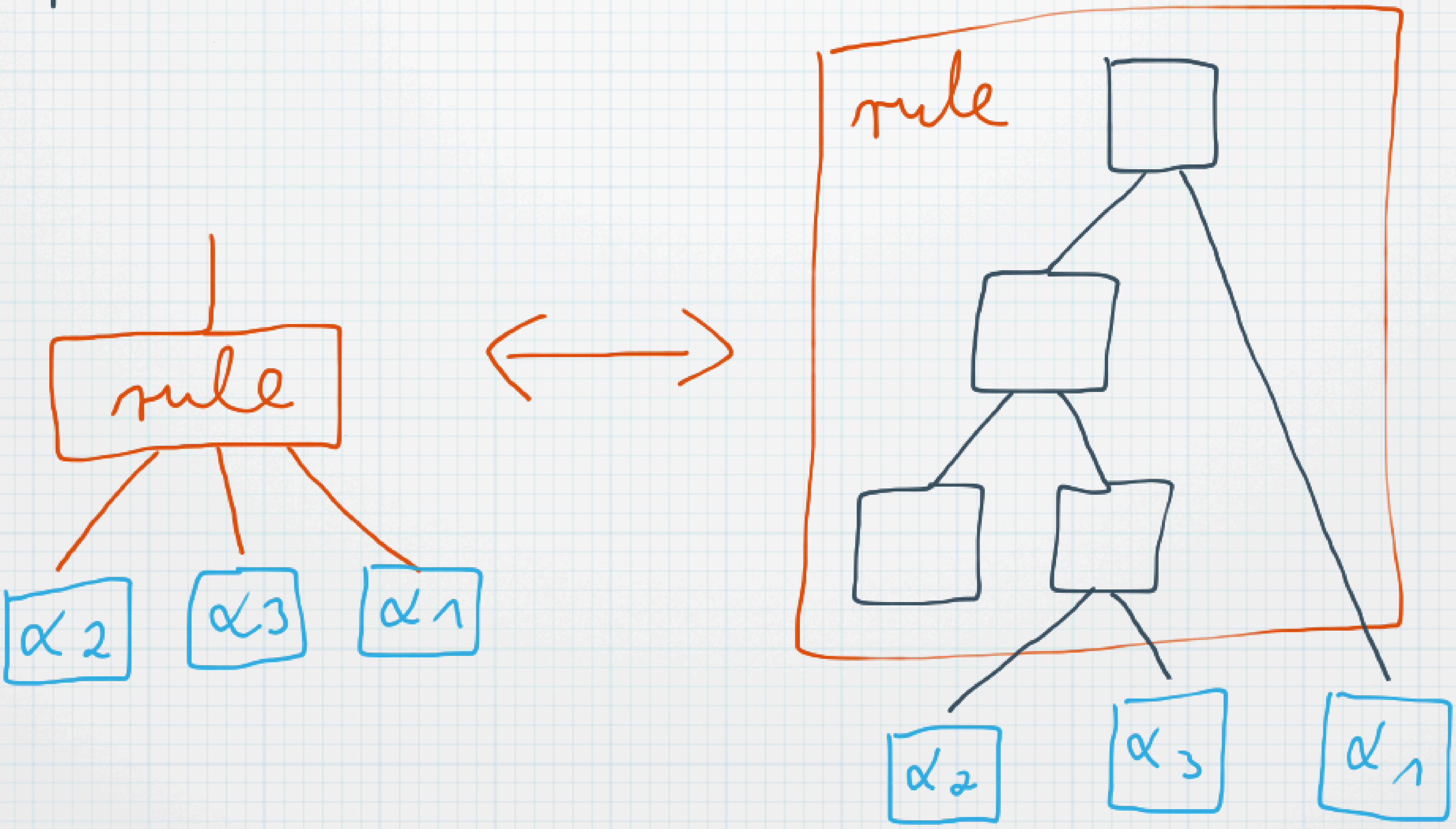
```
hilabel_tac : label → tactic → tactic

let hilabel_tac label tactic g =
  let (inst, gls, j) = tactic g in
  let k inst = hilabel label (j inst)
  in (inst, gls, k)
```

How to implement hilabel ?

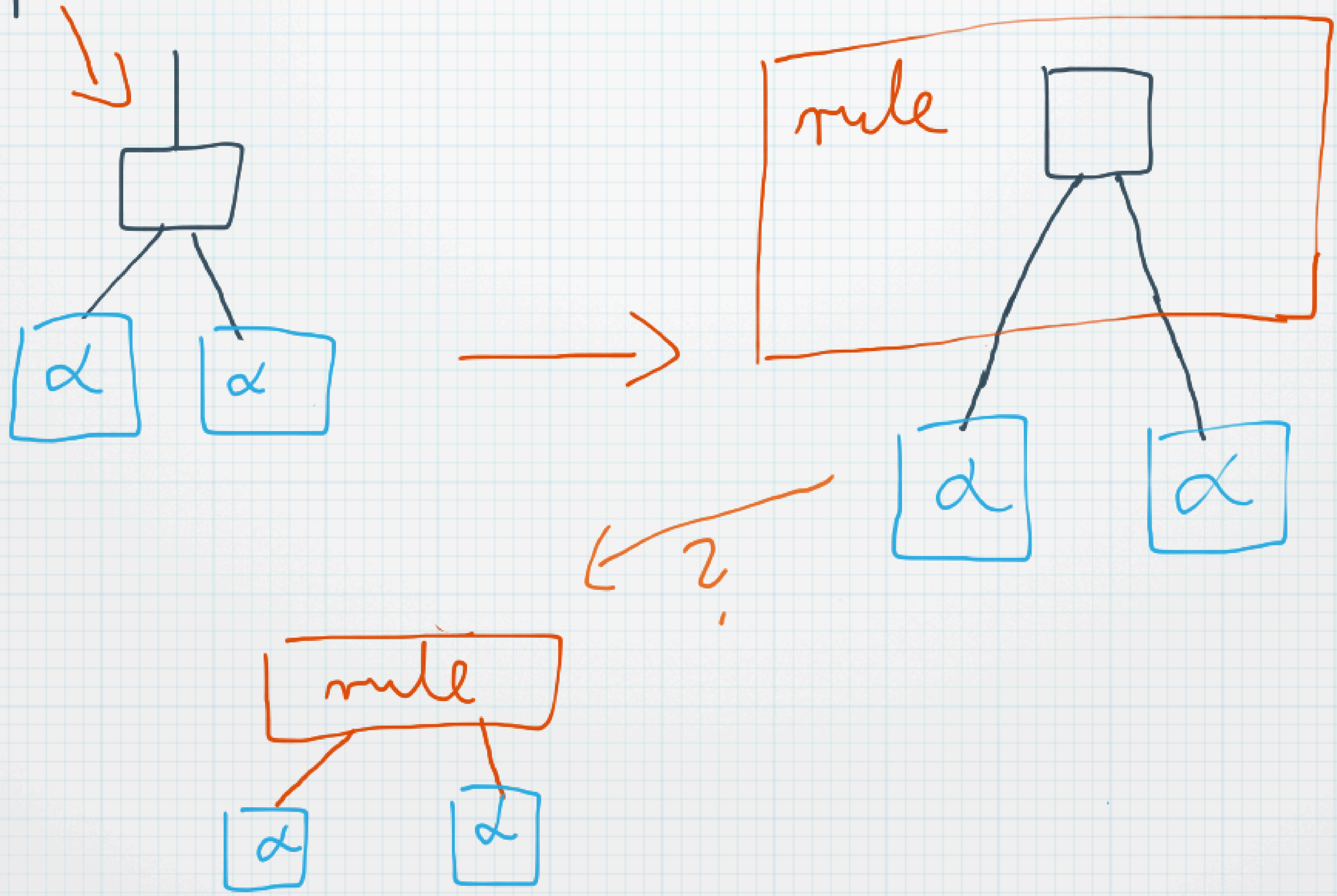
$$\beta = \text{rule}\,[\alpha_1, \alpha_2, \alpha_3]$$



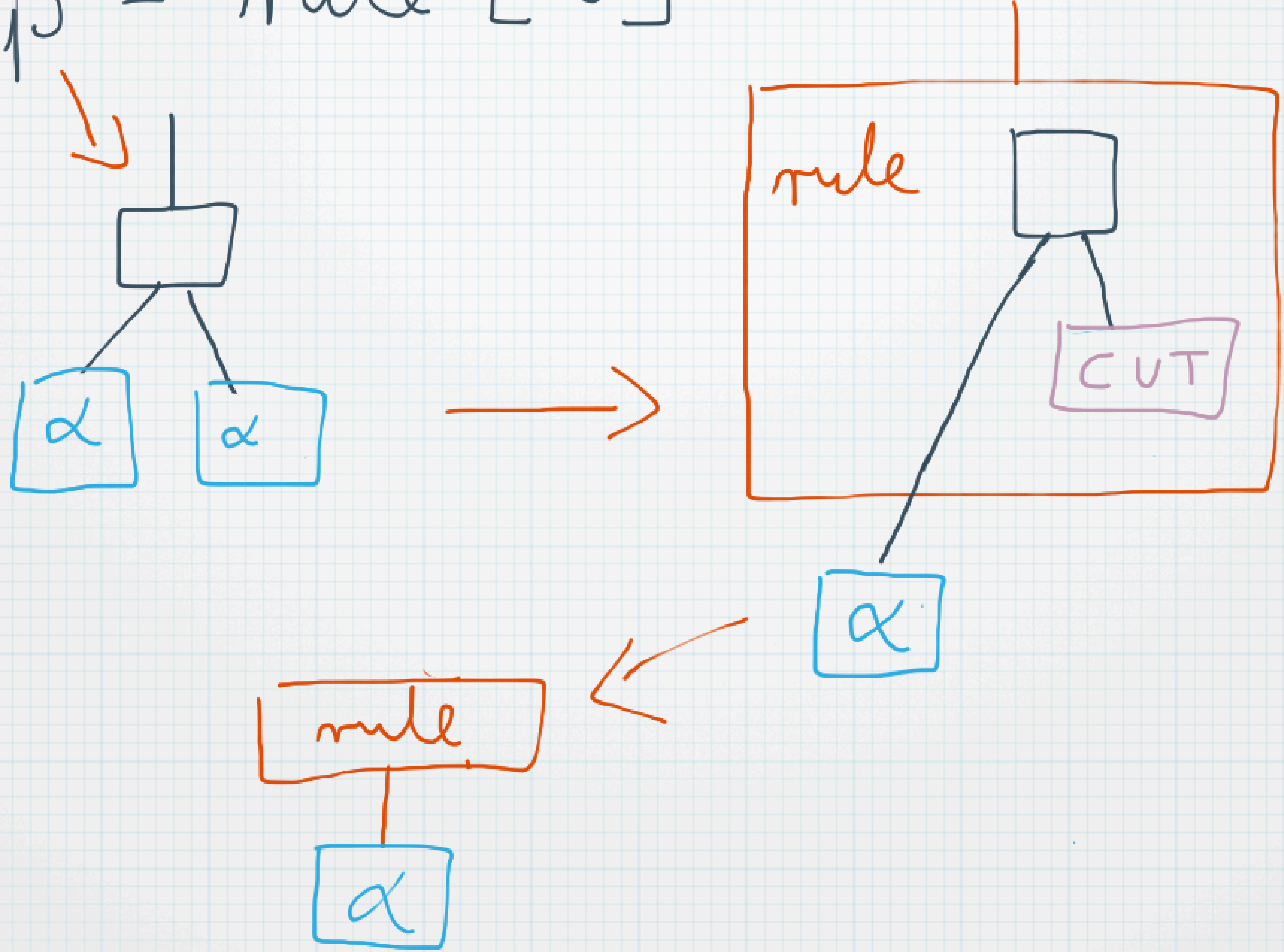
$$\beta = \mathrm{rule}\,[\alpha_1, \alpha_2, \alpha_3]$$



$\beta = \mathrm{rule}\,[\alpha]$



$\beta = \text{rule}[\alpha]$



Hiproof Datatype

(version 2)

```
type hiproof =
     Hi_atomic of int * label * goal
   | Hi_box of label * hiproof
   | Hi_tensor of hiproof list
   | Hi_sequence of hiproof list
   | Hi_id of goal
   | Hi_var of varname * goal
   | Hi_cut of goal
```

Dealing with Huge Hiproofs

- Very quickly hiproofs contain more than a billion elements. They fit into memory because of sharing.
- Naive manipulation of hiproofs breaks completely down. Even computing the size of a hiproof is not possible.

Solution: Make Hiproofs Even Bigger

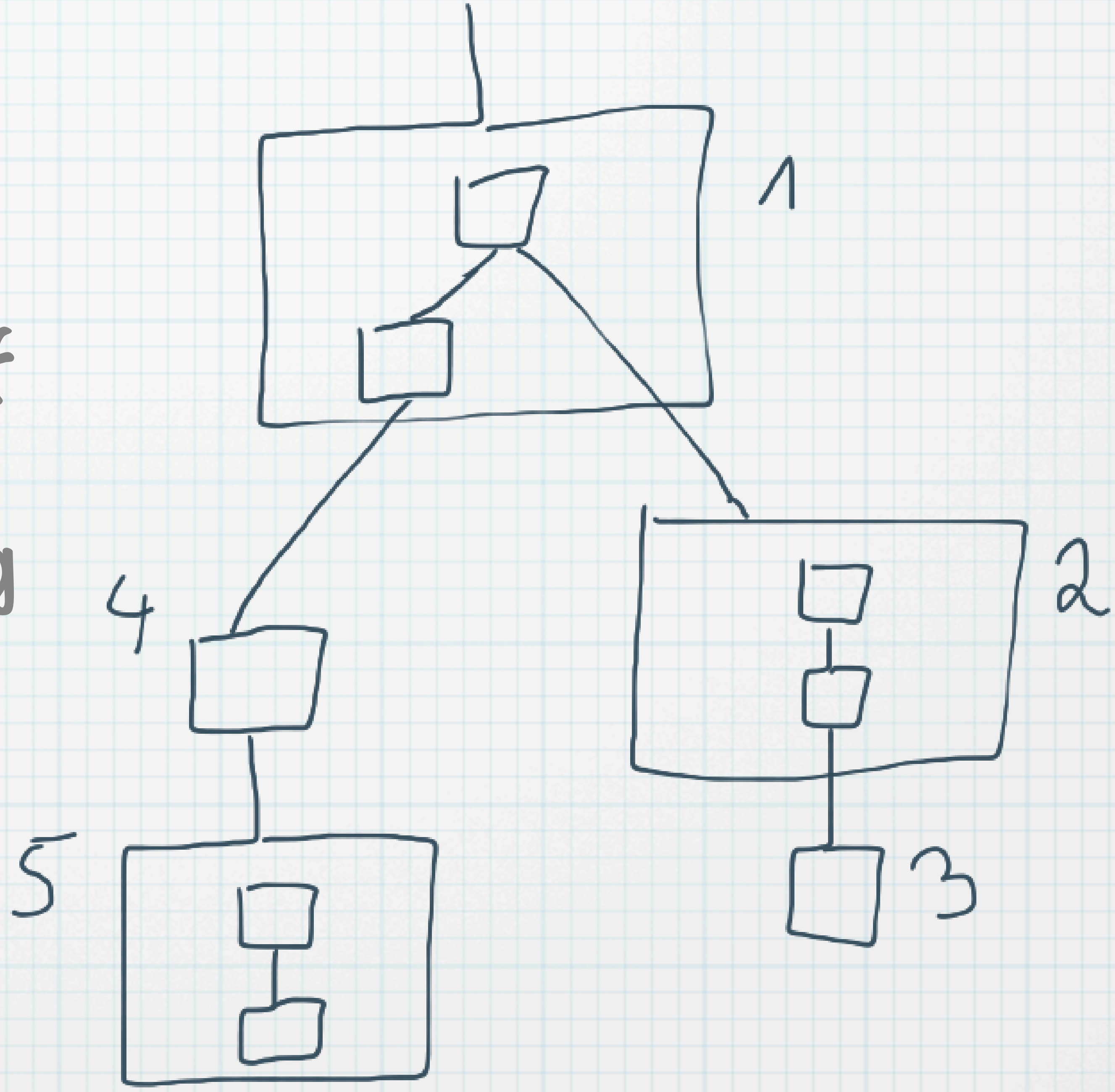
As a hiproof is created, store precomputed information:

- input arity
- output arity
- all variables it contains
- its "shallow size"

Shallow Size

There is a fixed (but configurable) treshold for how large the shallow size of boxes can be. This treshold is only relevant if the recording mechanism actually needs to look inside the box.

By default this treshold is **1000**.



Example: Hiproof with Shallow Size of 5

Hiproof Datatype

(final version)

```
type 'a varmap = (varname * 'a) list

type shallow_size = Above_treshold
                  | Below_treshold of int

type 'a info = int * int * ('a varmap) * shallow_size

type 'a hiproof =
    Hi_atomic of int * label * 'a * ('a varmap)
  | Hi_id of 'a
  | Hi_cut of 'a
  | Hi_var of varname * 'a
  | Hi_label of label * 'a * 'a hiproof * 'a info
  | Hi_tensor of 'a hiproof list * 'a info
  | Hi_sequence of 'a hiproof list * 'a info
```

Statistics

Proof	Original	Max-Detail Capturing	High-Level Capturing
#use "hol.ml"	4 min 30 sec 0.1 GB	7 min 2.2 GB	6 min 0.7 GB
Gödel 1	10 min 30 sec 0.1 GB	15 min 3.1 GB	13 min 0.9 GB
e is transcendental	13 min 0.2 GB	19 min 4.2 GB	16 min 1.5 GB
Jordan curve theorem	31 min 0.4 GB	out of memory	45 min 6.5 GB

Last Macbook Pro 17", Quad Core 2.4 GHz, 16GB RAM

To Do

Better Automatic Hierarchy

- Looking at the lowest levels of hiproofs is nice and fun for beginners, but rather worthless for more advanced users.
- A lot of noise makes it very hard to gain any insight into the proof.
- Automatic hierarchy creation should try to reduce this noise and convey the highlevel idea of the proof.

Display More Information

- Hiproofs record more information via rich labels than is actually displayed, like additional terms and theorems they depend on.
- Proofs of lemmas should be interlinked.

Better Layout

- Right now, hiproofs are pruned for size prior to visualisation. Usually only part of the hiproof is therefore displayed. Lazy layout could fix this.
- Hiproofs are already close to a visual representation. It should be possible to display them directly without prior conversion into graphs.

https://github.com/phlegmaticprogrammer/hiproofs