Control Flow as Contours of Data Flow

Malin Altenmüller

(jww Dan Ghica)

MSP 101 09/06/23

Control Flow

≙ order of execution of program elements

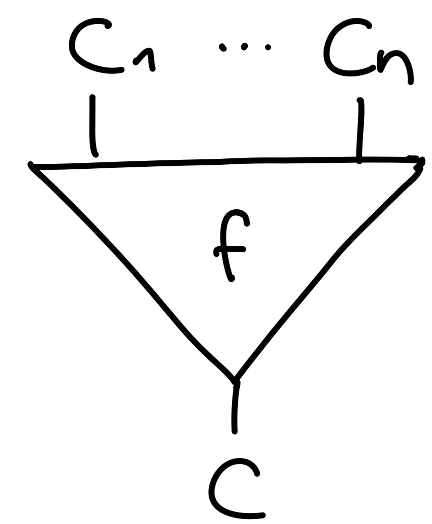
```
a = 5
b = 10
c = a+b
return c
```

```
a = 5
b = 10
if (a<b)
then    c = b-a
else    c = a-b
return c
```

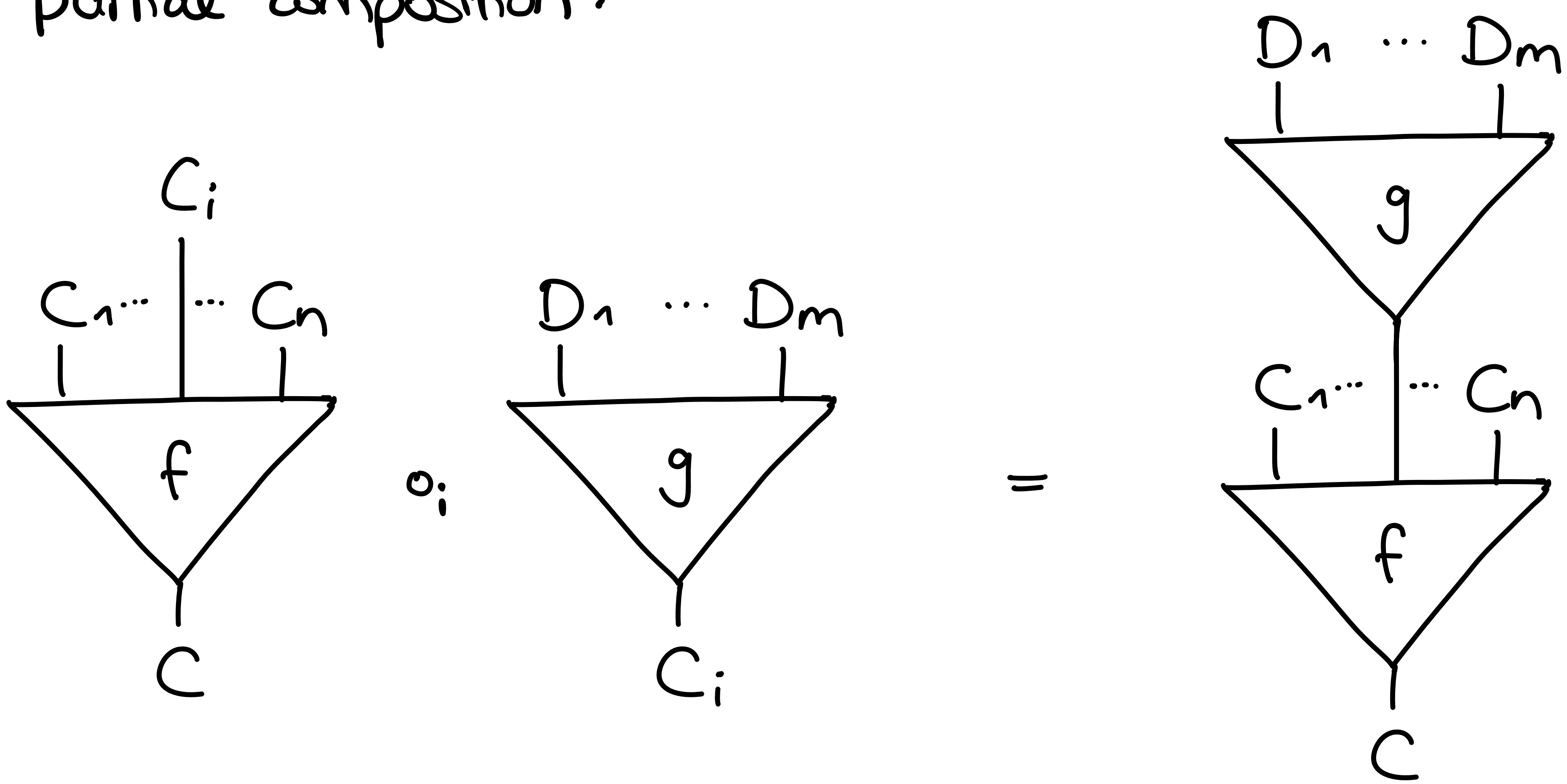
```
a = 5
c = 0
while (c < 100)
    c += a
    a ++
return c
```

Operads (aka Multicategories)

generalisation of categories: maps take multiple inputs



- colours $\vec{C}$
- n-ary maps $f : C_1, \dots, C_n \to C$
- identity $C \to C$
- partial composition:



+ laws

Operads

- functors of operads : $F : \mathcal{C} \to \mathcal{D}$

  function on colours : $C \mapsto D$

  arity-preserving function on arrows :

  $$f : C_1, \ldots, C_n \to C \longmapsto Ff : D_1, \ldots, D_n \to D$$

- intuition : take multiple things & explain the space "in between" them, how the connect to make one whole thing

- idea : define 2 operads
  - a simple one, mainly contains the wiring
  - a more complex one, adding more information

Related Work

work by Paul-André Melliès & Noam Zeilberger [1]

```
1 : S → NP VP
2 : NP → mom
3 : NP → tom
4 : VP → loves NP
```

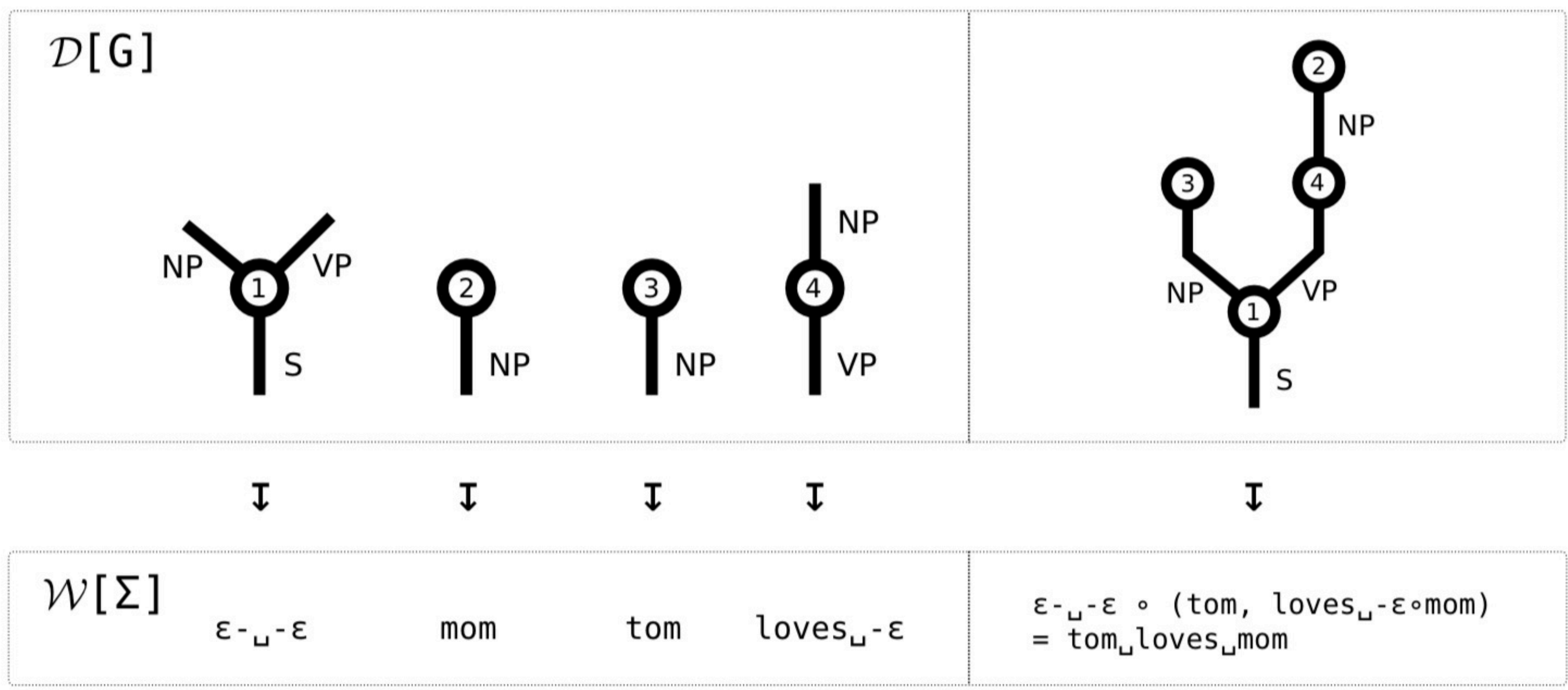


Fig. 1. Example of a context-free grammar and the corresponding functor $\mathcal{D}[G] \to \mathcal{W}[\Sigma]$, indicating the action of the functor on the generating operations of $\mathcal{D}[G]$ as well the induced action on a closed derivation.

[1] "Parsing as a Lifting Problem and the Chomsky-Schützenberger Representation Theorem" MFPS '22

Related Work

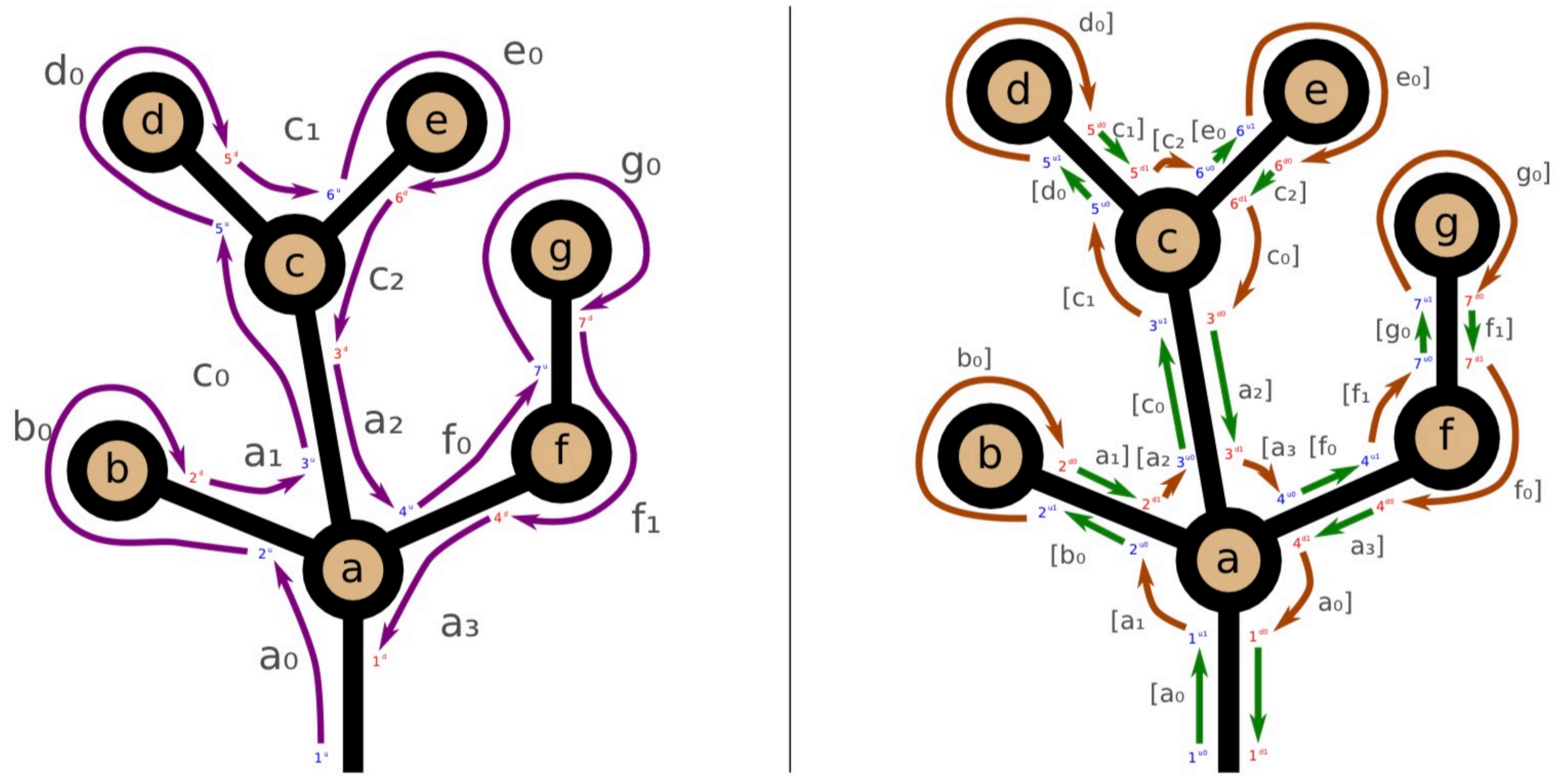
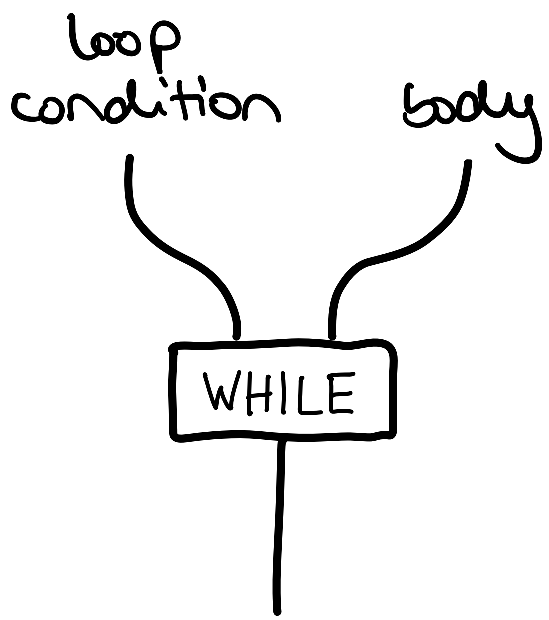


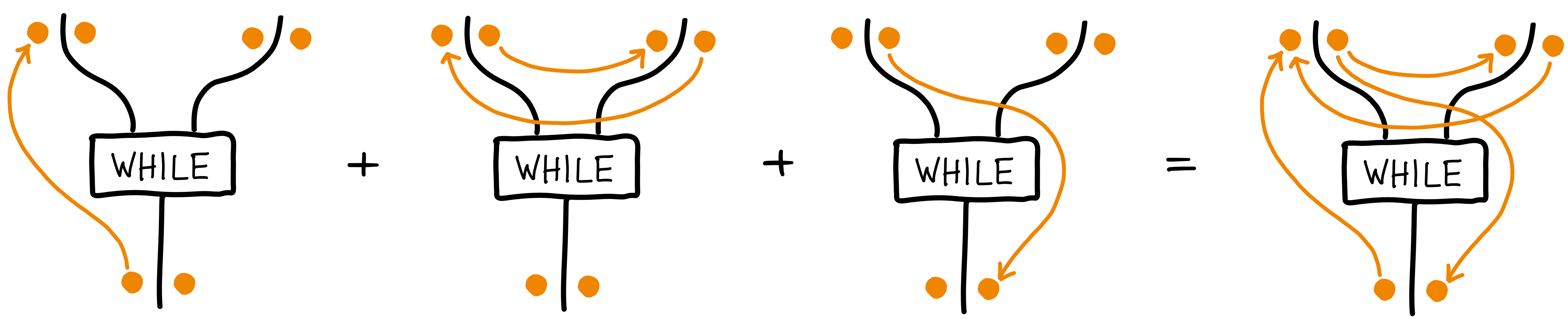
Fig. 4. Left: an $\mathcal{S}$-rooted tree of root color 1 and its corresponding contour word $\mathsf{a}_0\mathsf{b}_0\mathsf{a}_1\mathsf{c}_0\mathsf{d}_0\mathsf{c}_1\mathsf{e}_0\mathsf{c}_2\mathsf{a}_2\mathsf{f}_0\mathsf{g}_0\mathsf{f}_1\mathsf{a}_3 : 1^u \to 1^d$. Right: the corresponding Dyck word obtained by first decomposing each corner of the contour into alternating actions of walking along an edge and turning around a node, and then annotating each arrow both by the orientation (with $u = [, d =]$) and the node-edge pair of its target.

linear & deterministic notion of contour

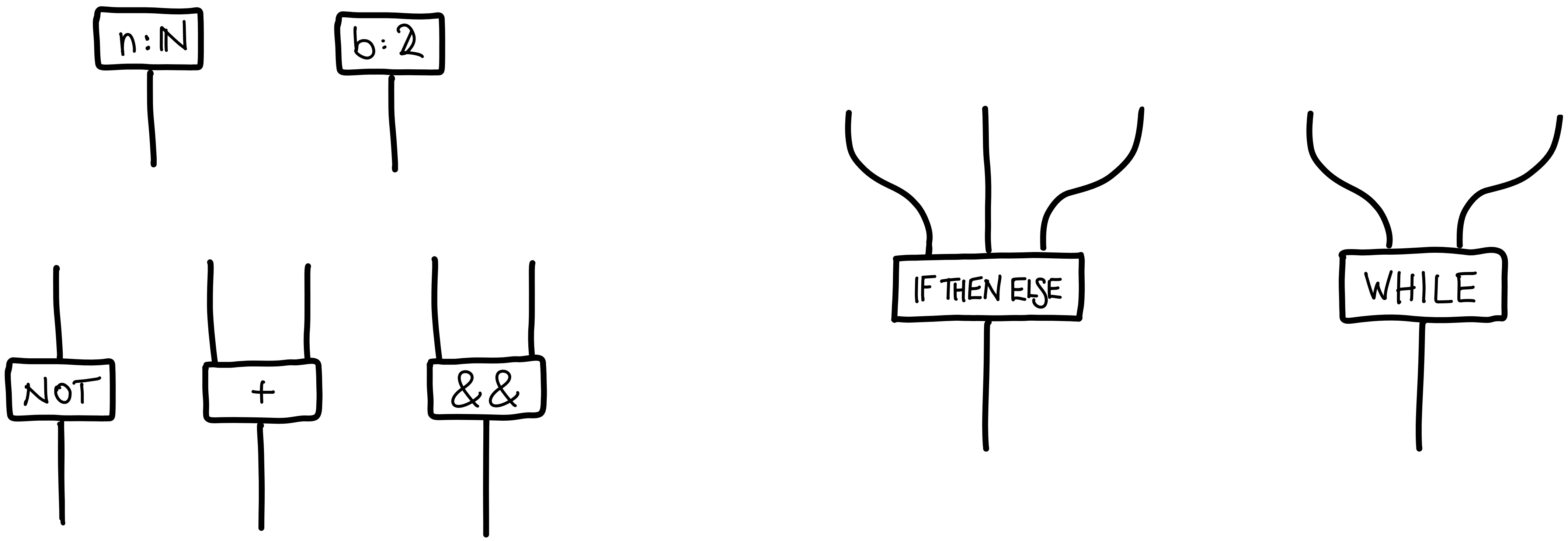
Motivating Example



- start bottom left, return bottom right
- for while:
    - check condition
    - if true: enter body (& loop back)
    - if false: return
- encode all options in one



Language Generators



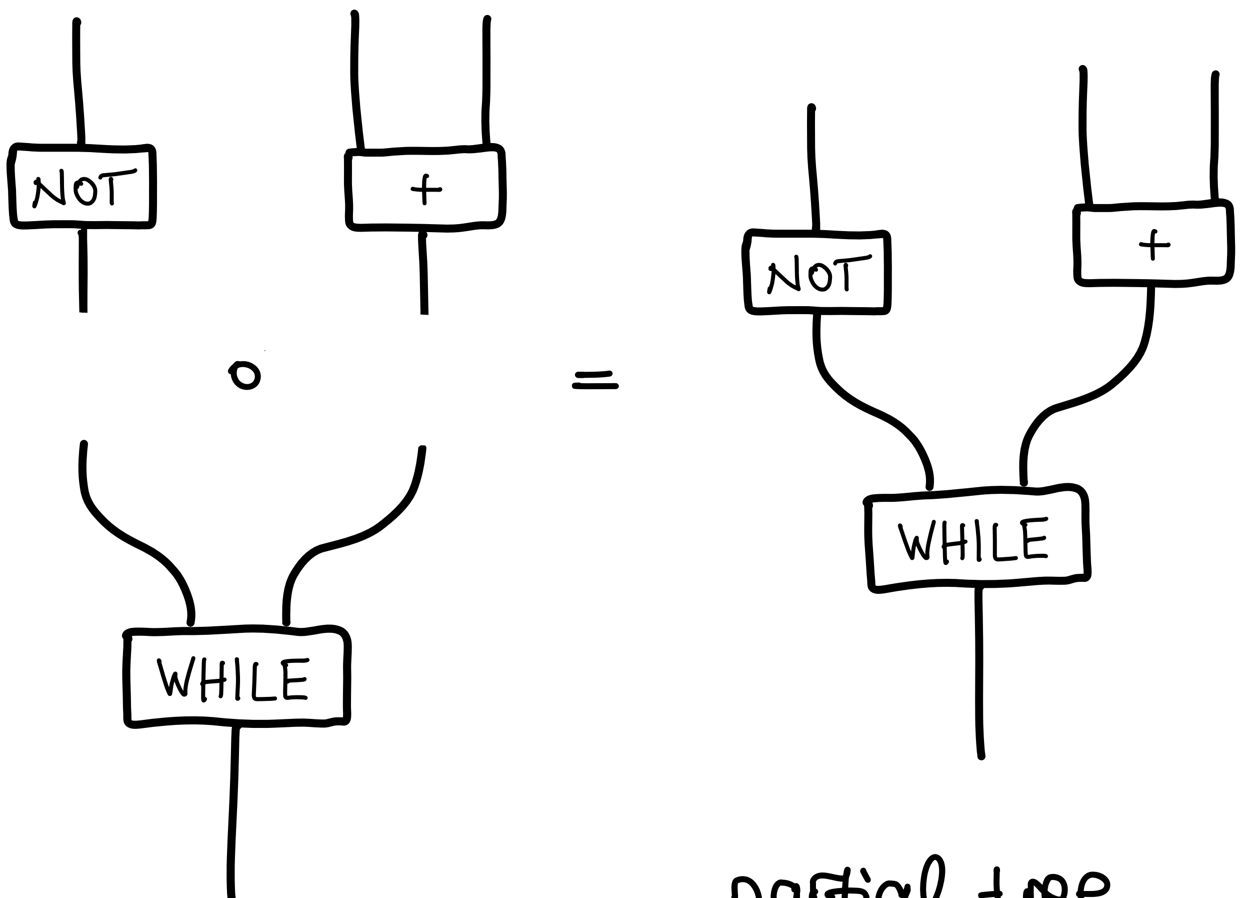
- operations with multiple (including none) inputs and one output
- don't care about composition just yet:

  elements of spans $C^* \leftarrow O \rightarrow C$ "species"
- typically have types on the wires, not needed for control flow though

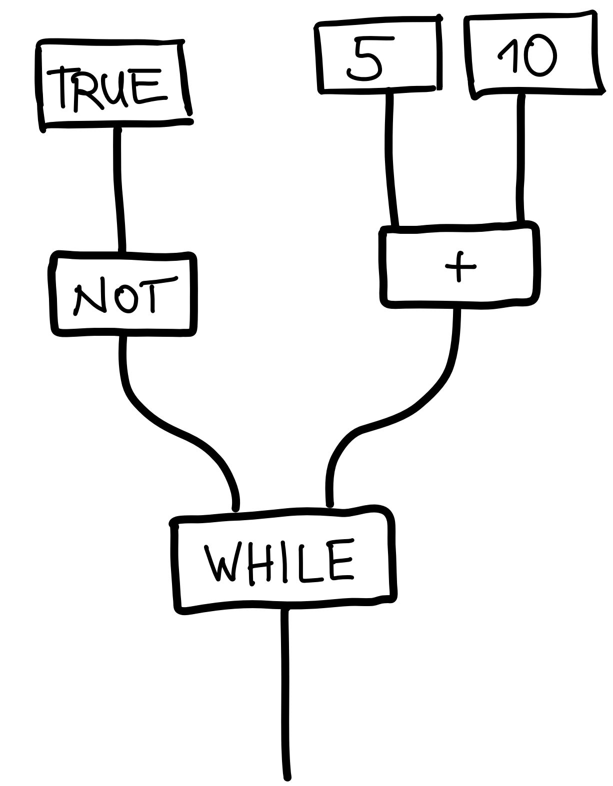
Abstract Syntax Trees

take the free operad on a species of generators :

now we get composition!



partial tree
$3 \to 1$



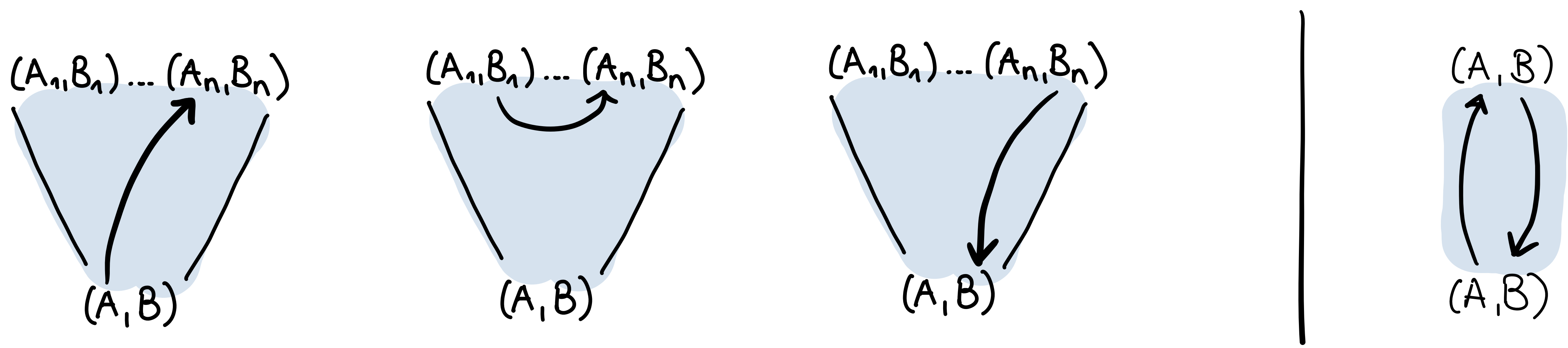
total tree $0 \to 1$

Contour Operads – Definition (1)

given category $\underline{A}$. A contour operad $\mathcal{C}(\underline{A})$ consists of

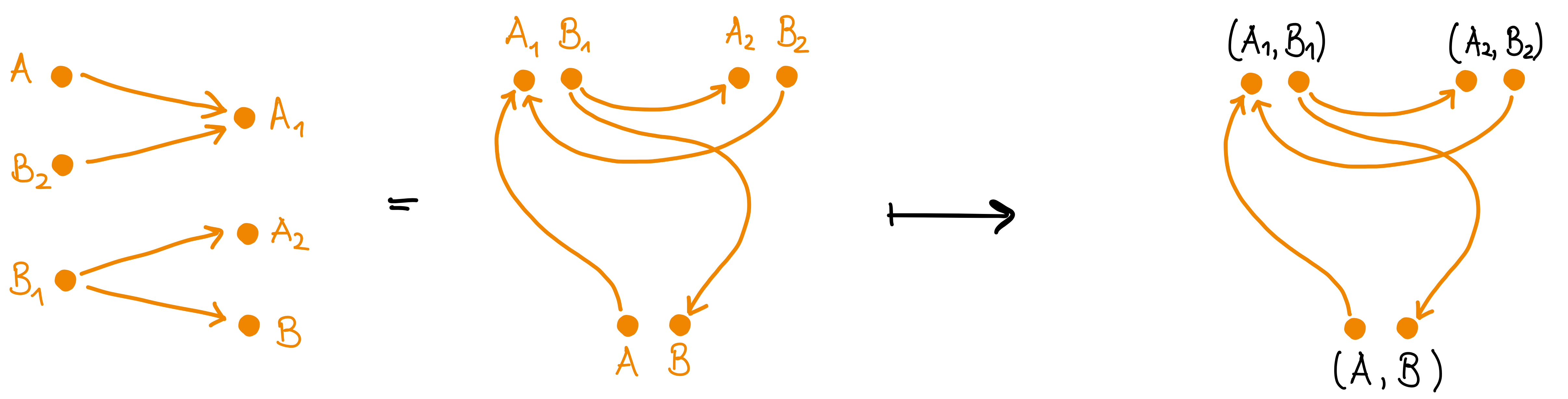
- colours are pairs of $\underline{A}$-objects $(A,B), (A_1,B_1), (A_2,B_2), \ldots$
- n-ary map $f : (A_1,B_1),\ldots,(A_n,B_n) \to (A,B)$
  is a finite set of $\underline{A}$-morphisms, each of the format:

$A \to A_j$ $\qquad$ $B_i \to A_j$ $\qquad$ $B_i \to B$



- the identity operad $(A,B) \to (A,B)$ is the pair id_A, id_B from $\underline{A}$

Contour Operads – Example

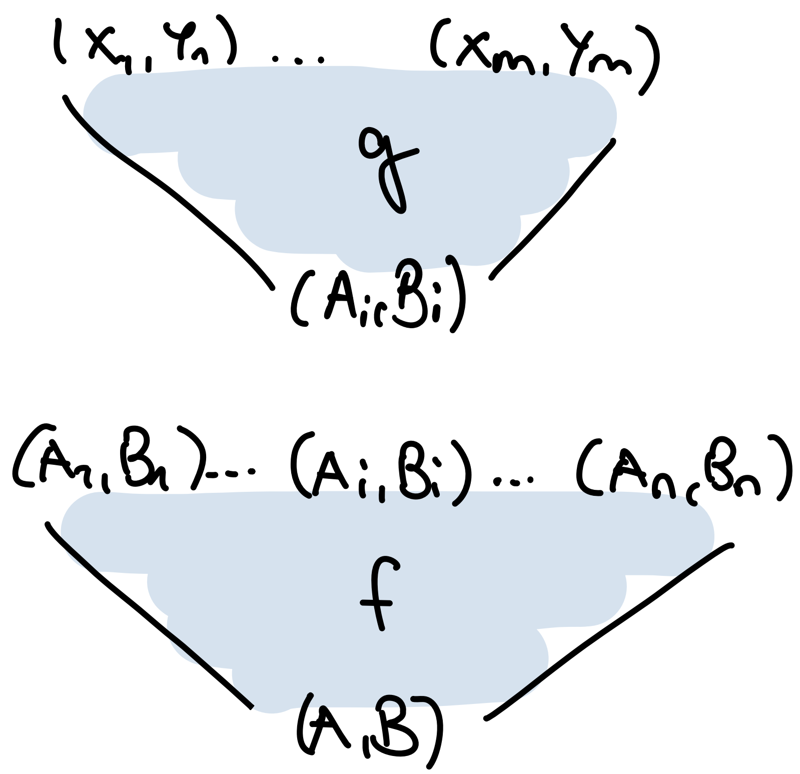


category

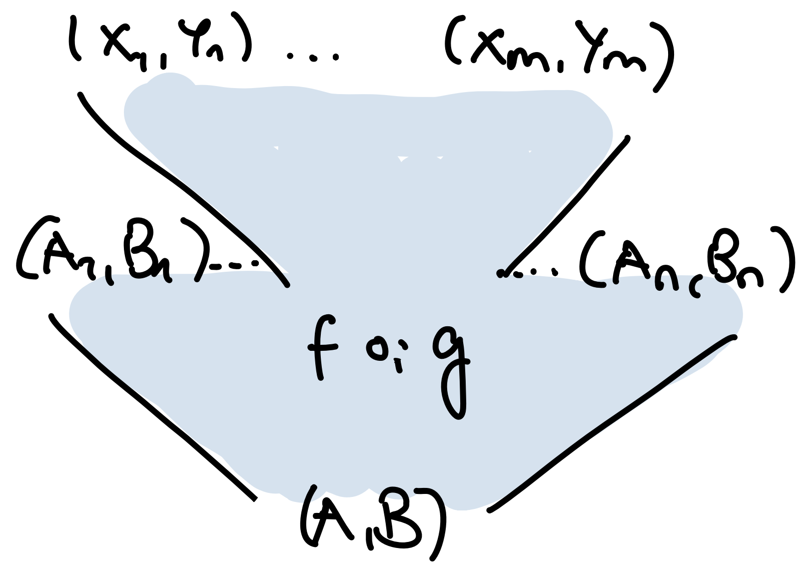
contour operad

Contour Operads – Definition (2)

- given f and g:



their composition:



is computed by composing all $\underline{A}$-morphisms

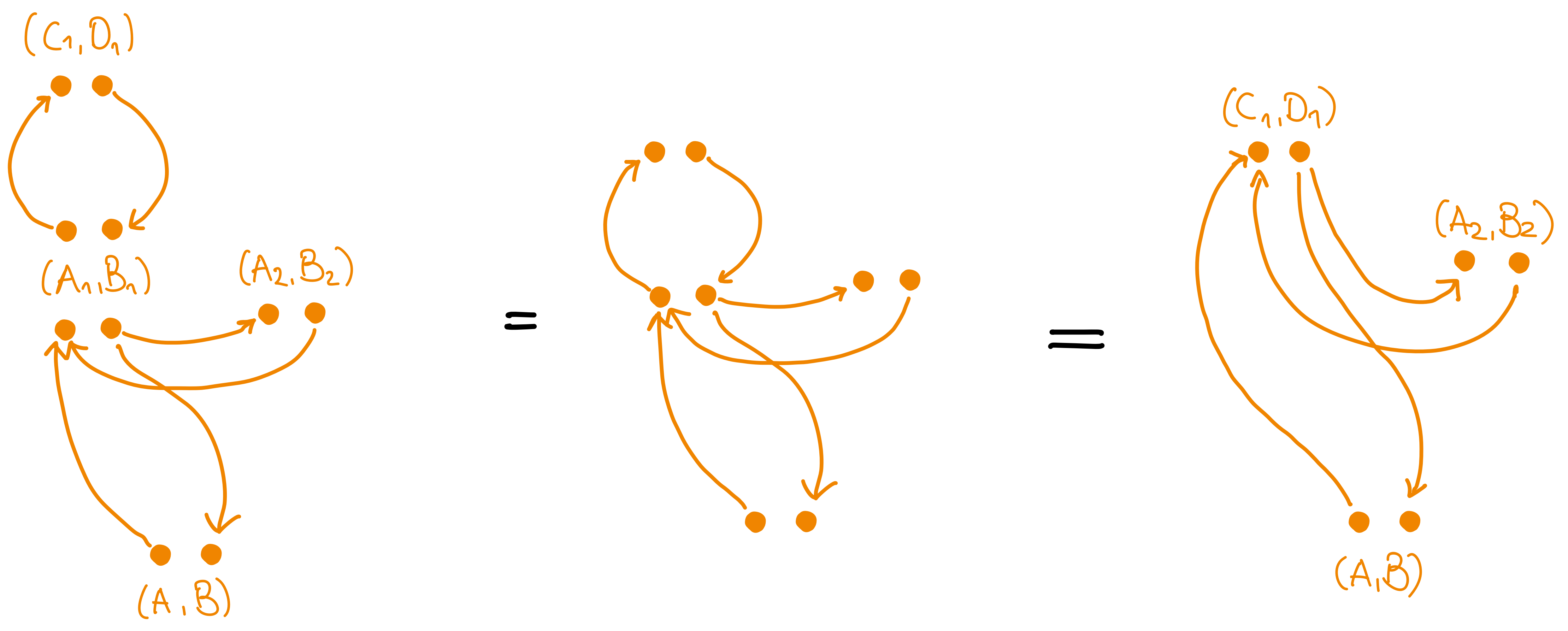
$$X \to A_i \,;\, A_i \to Y$$

and

$$Y \to B_i \,;\, B_i \to X$$

- identity & composition laws hold because they hold in $\underline{A}$

Contour Operads - Example



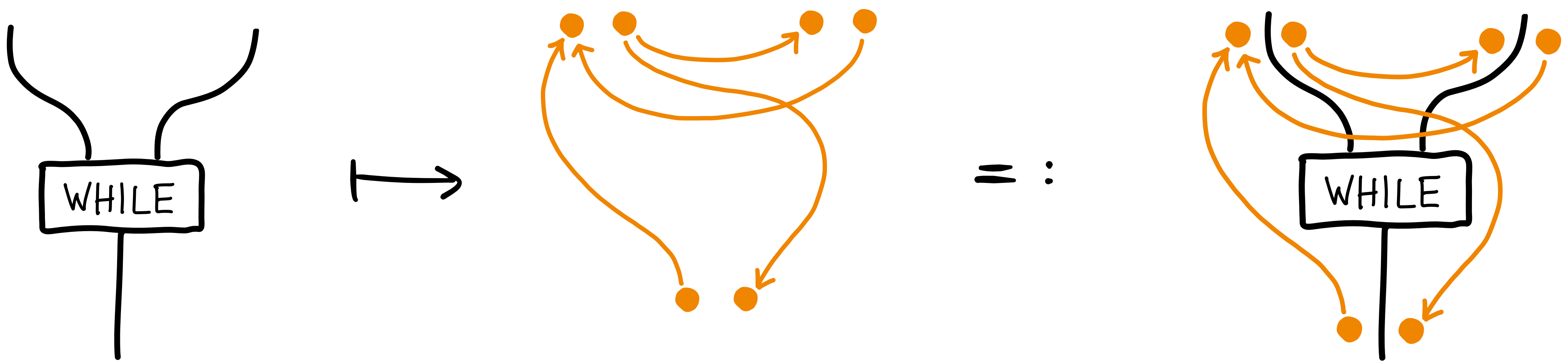
Assigning Control Flow Information

given the free operad on the language generators $\mathrm{Free}(S)$

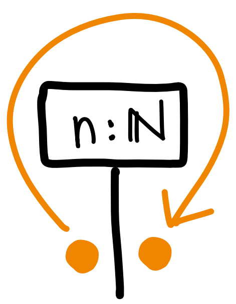
and the contour operad $\mathcal{C}(\underline{A})$

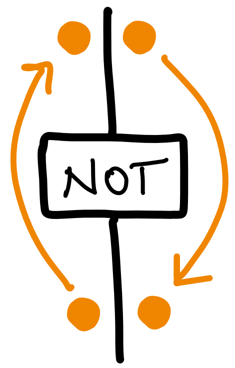
assigning control flow to the abstract syntax tree amounts to a functor

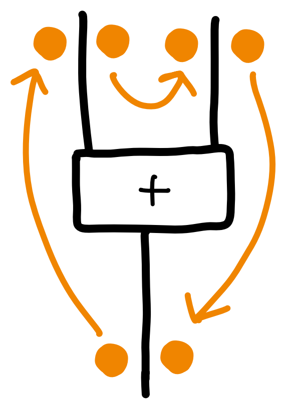
$$\mathrm{Free}(S) \to \mathcal{C}(\underline{A})$$

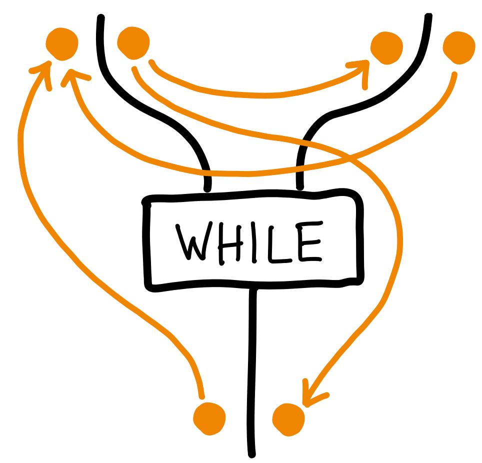


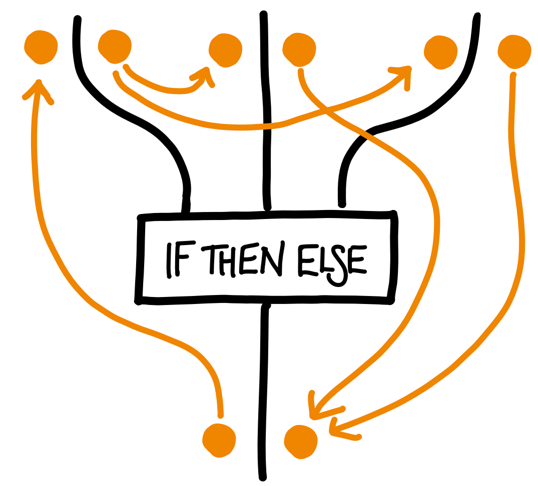
Control Flow for Language Generators

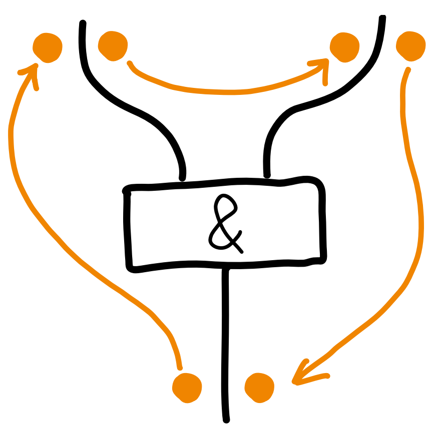




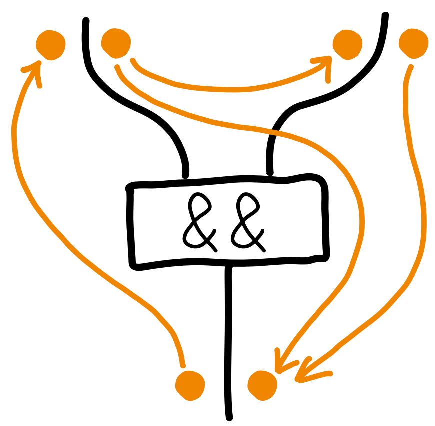




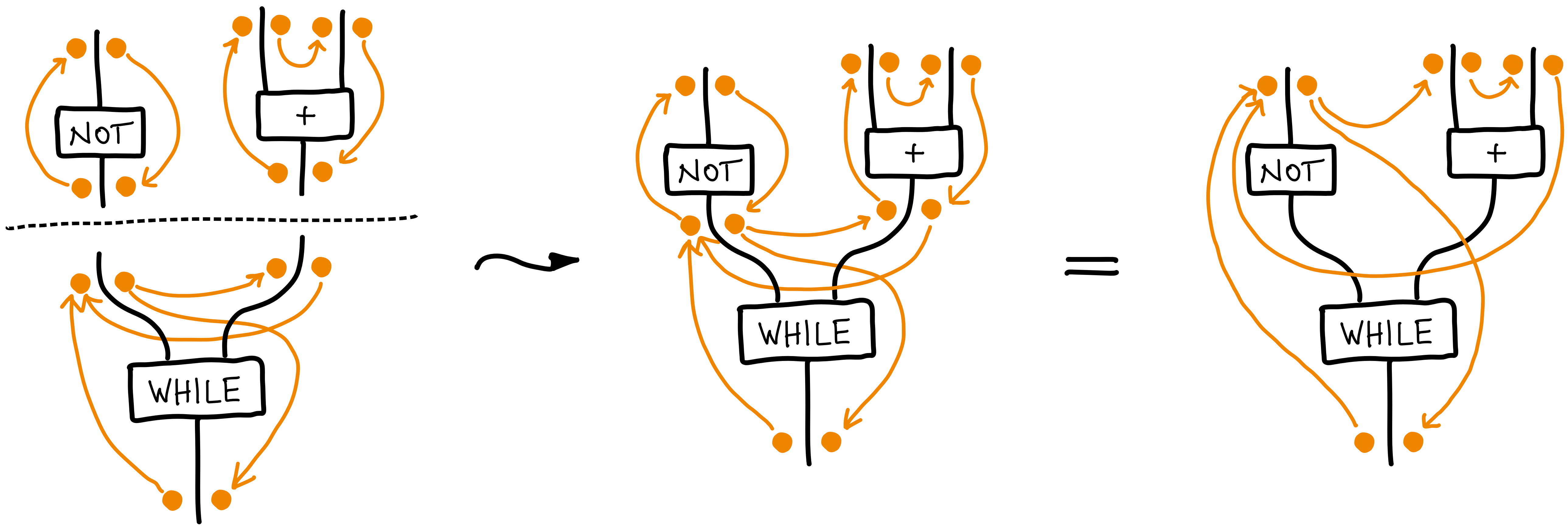




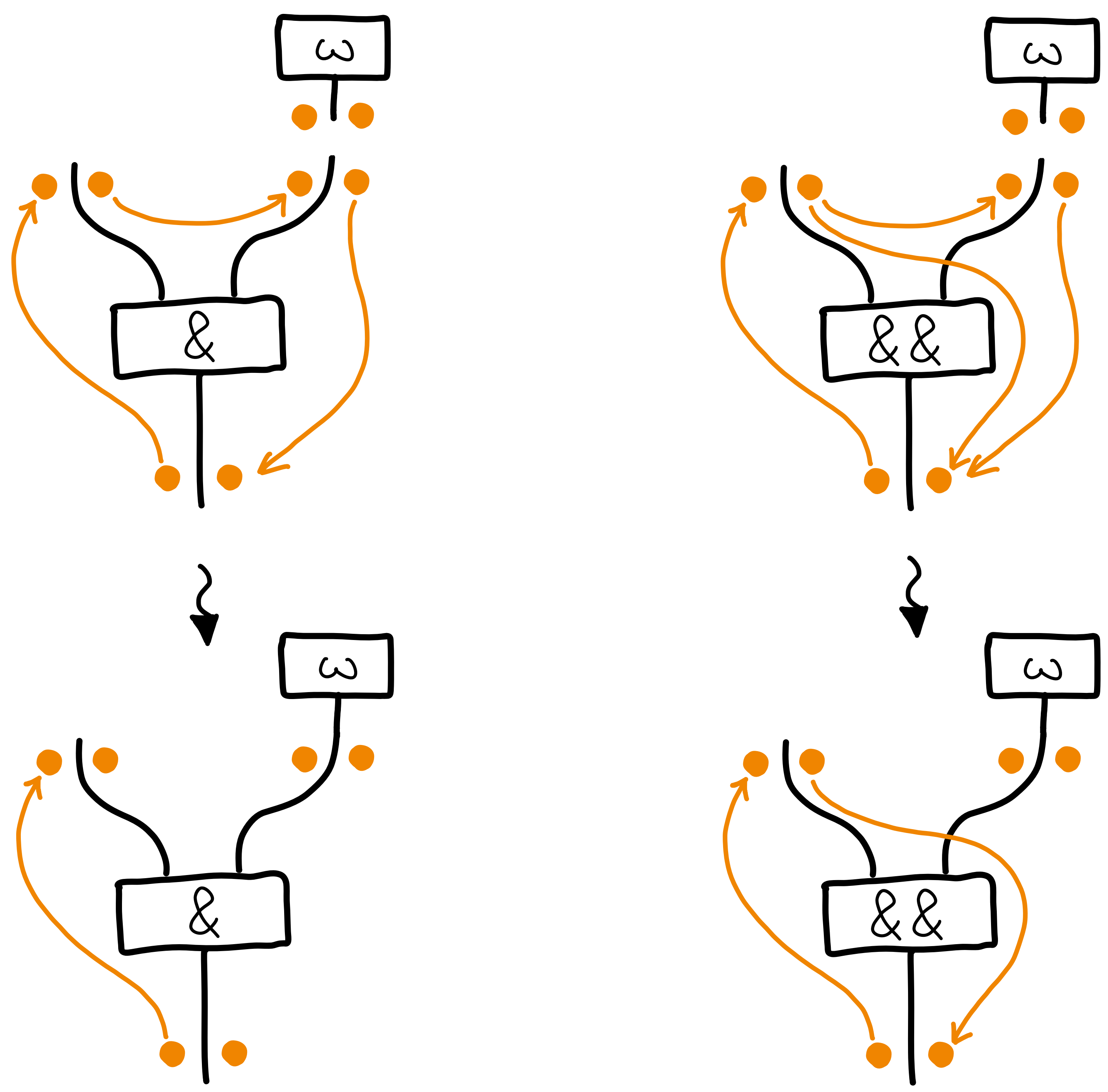
vs.



Composing Control Flow (1)



Composing Control Flow (2)



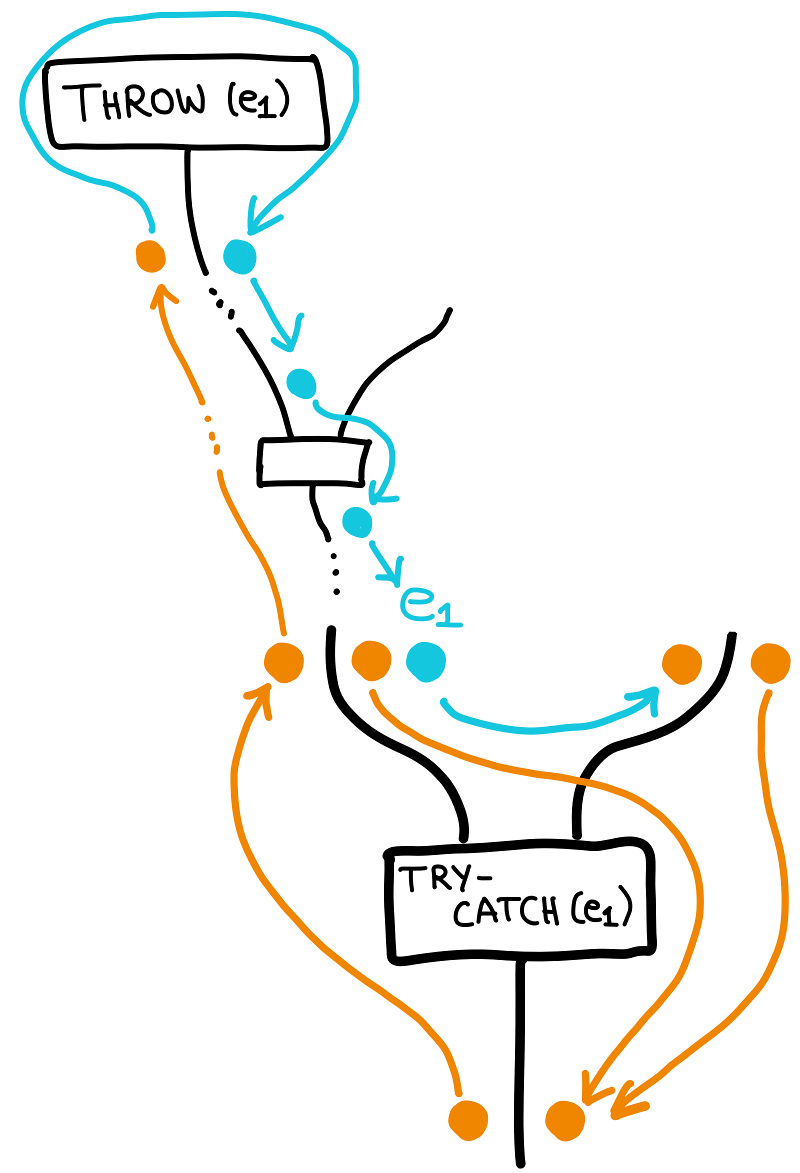
Composing Control Flow

- which operations get composed is defined by the underlying operad
  → control flow composes in the corresponding way
- composing contour operads may delete control flow paths
  (decreasing degree of approximation)
- in the normal case, a complete tree has one (or none)
  control flow path (i.e. deterministic control flow)

Exceptional Control Flow

- non-standard/alternative program behaviour
- obvious example: throwing & catching exceptions
- provide an alternative control flow:
  - interrupt normal flow
  - bypass all operations until caught
- plan: add another option to control flow
  - composition needs to do the right thing
  - normal & exceptional flow need to exclude each other

Exceptional Control Flow



Exceptional Control Flow

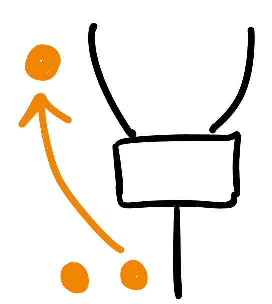
version of the contour operad:

- colours are pairs of finite sets of $\underline{A}$-objects
- maps: $[(A_1^1, A_1^2, \dots), (B_1^1, B_1^2, \dots)], \dots, [(A_n^1, A_n^2, \dots), (B_n^1, B_n^2, \dots)]$

$$\longrightarrow [(A^1, A^2, \dots), (B^1, B^2, \dots)]$$

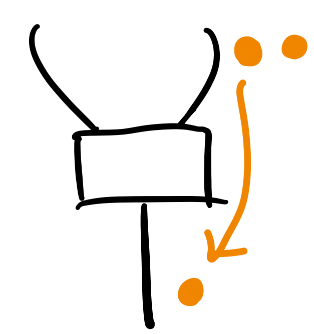
are finite sets of $\underline{A}$-morphisms, each of the format

$A^i \to A_j^\ell$



$B_i^k \to A_j^\ell$

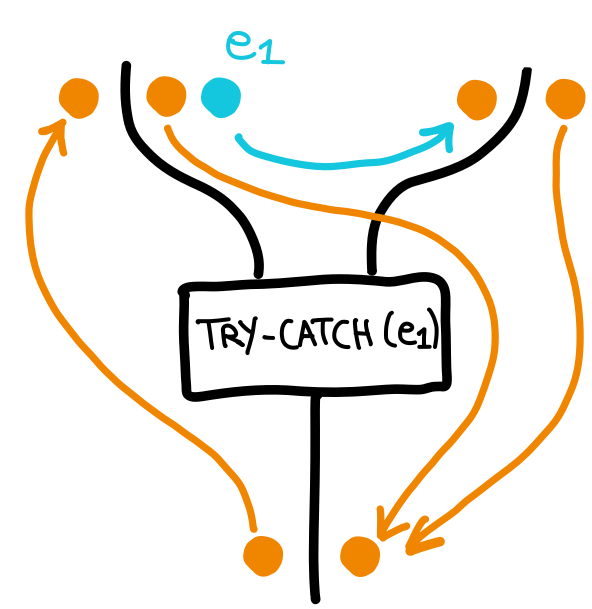
$B_i^k \to B^\ell$



Exceptional Control Flow

two kind of operations :

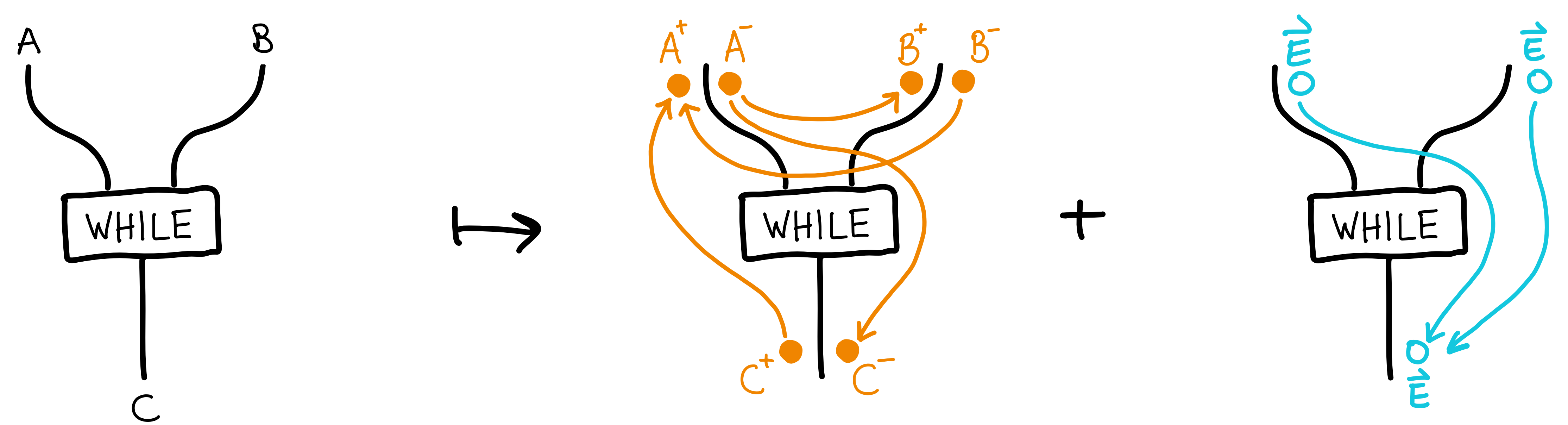
1) actively involved in creating / handling exceptional behaviour:



Exceptional Control Flow

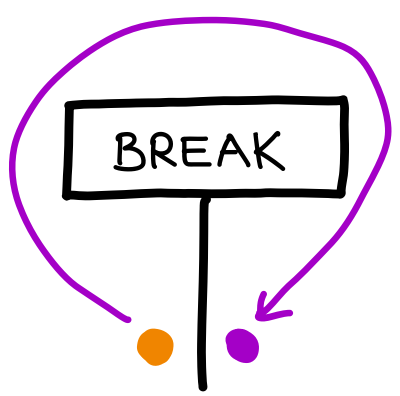
2) not involved in any way with exceptions

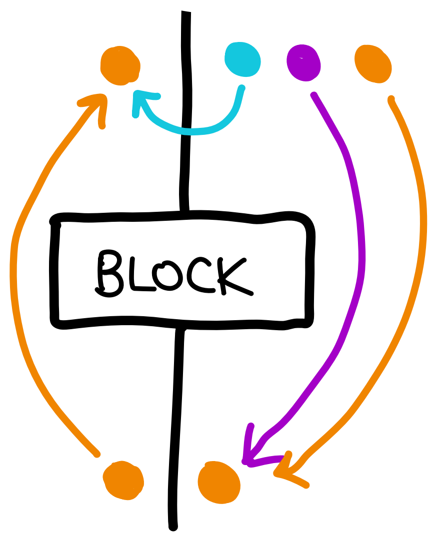
→ immediately passing on any exception to environment



Another kind of exceptional control flow

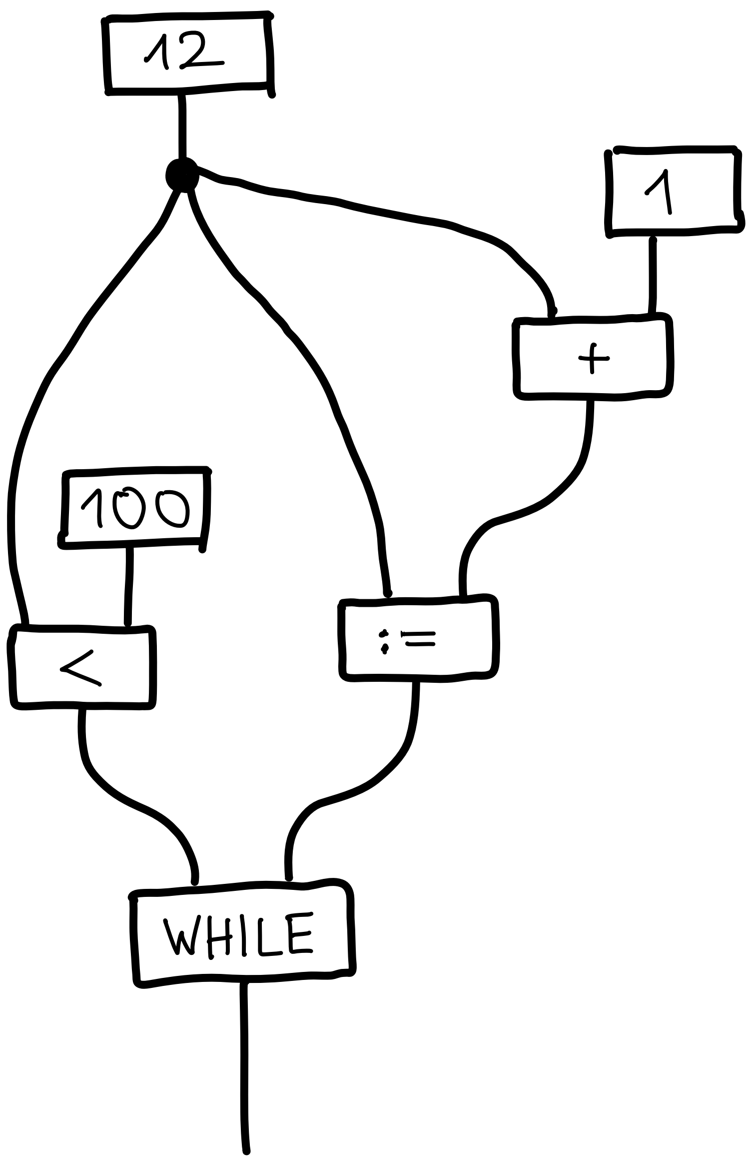
- early termination of a loop iteration
- two kinds of exceptions: BREAK and CONTINUE
- BLOCK acts like a handler for these two exceptions





Abstract Syntax Graphs

from tree to graphs: encoding shared memory access



for control flow →

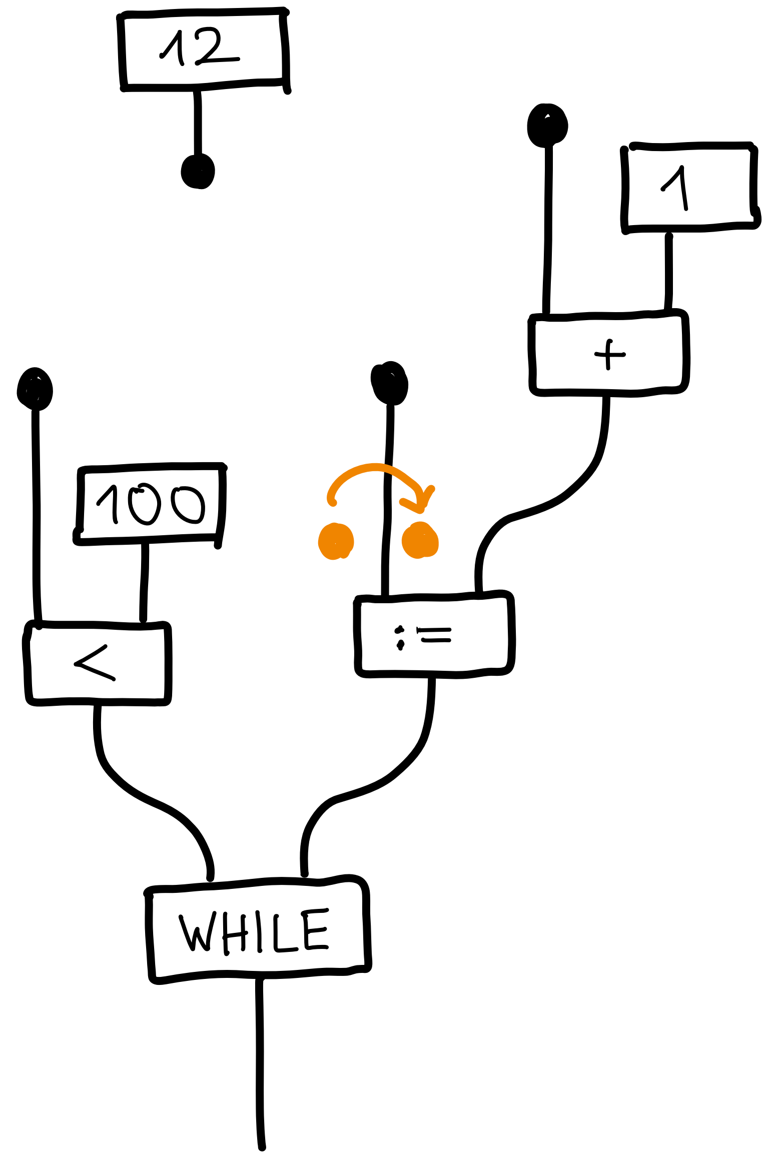
it doesn't really matter

e.g. could have a

distinguished colour

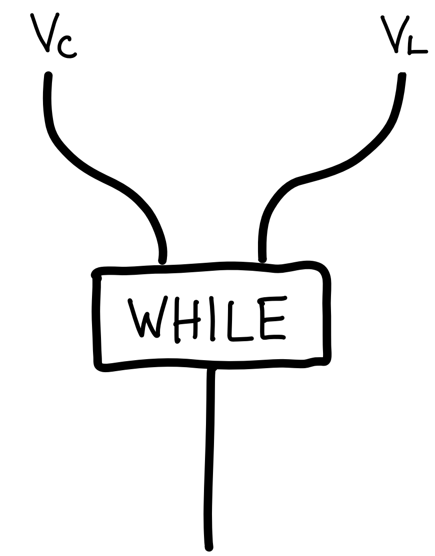
to highlight

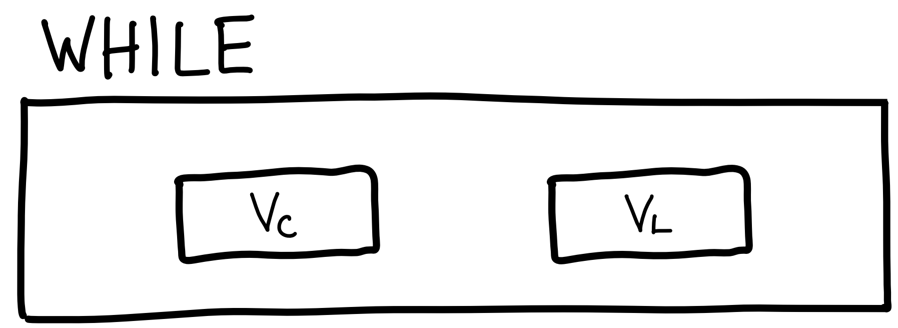
signals memory access



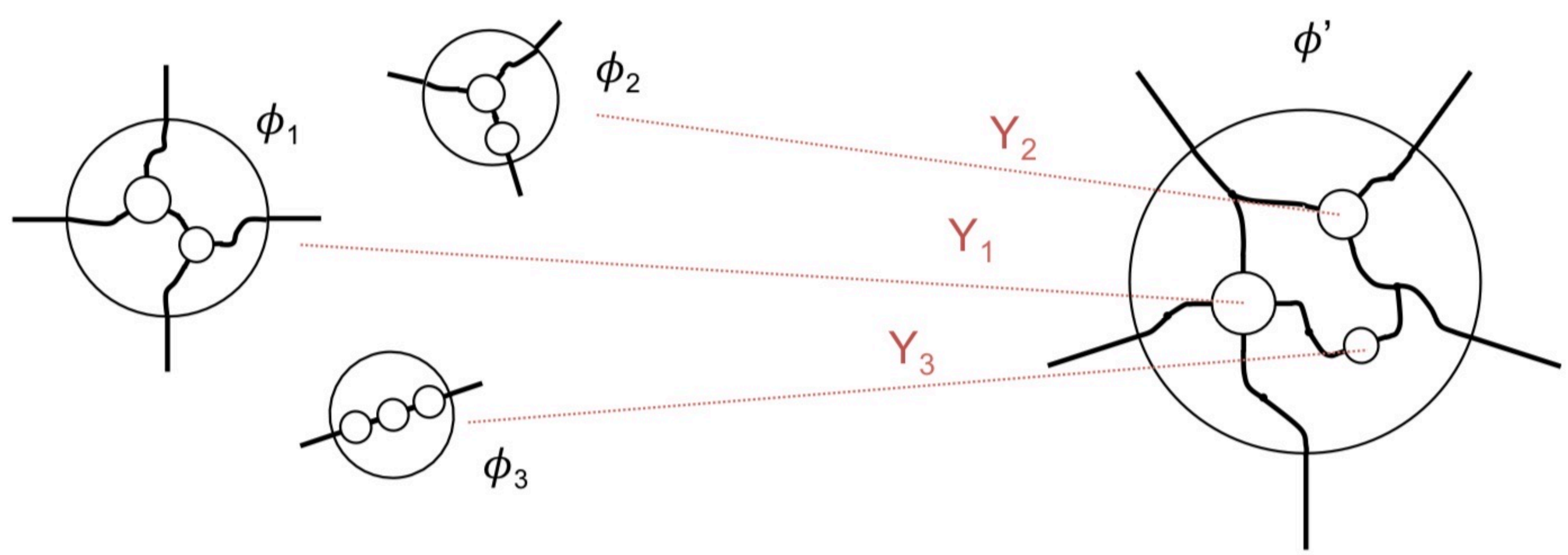
→ only interested in the order of accesses

Control Flow for Terms

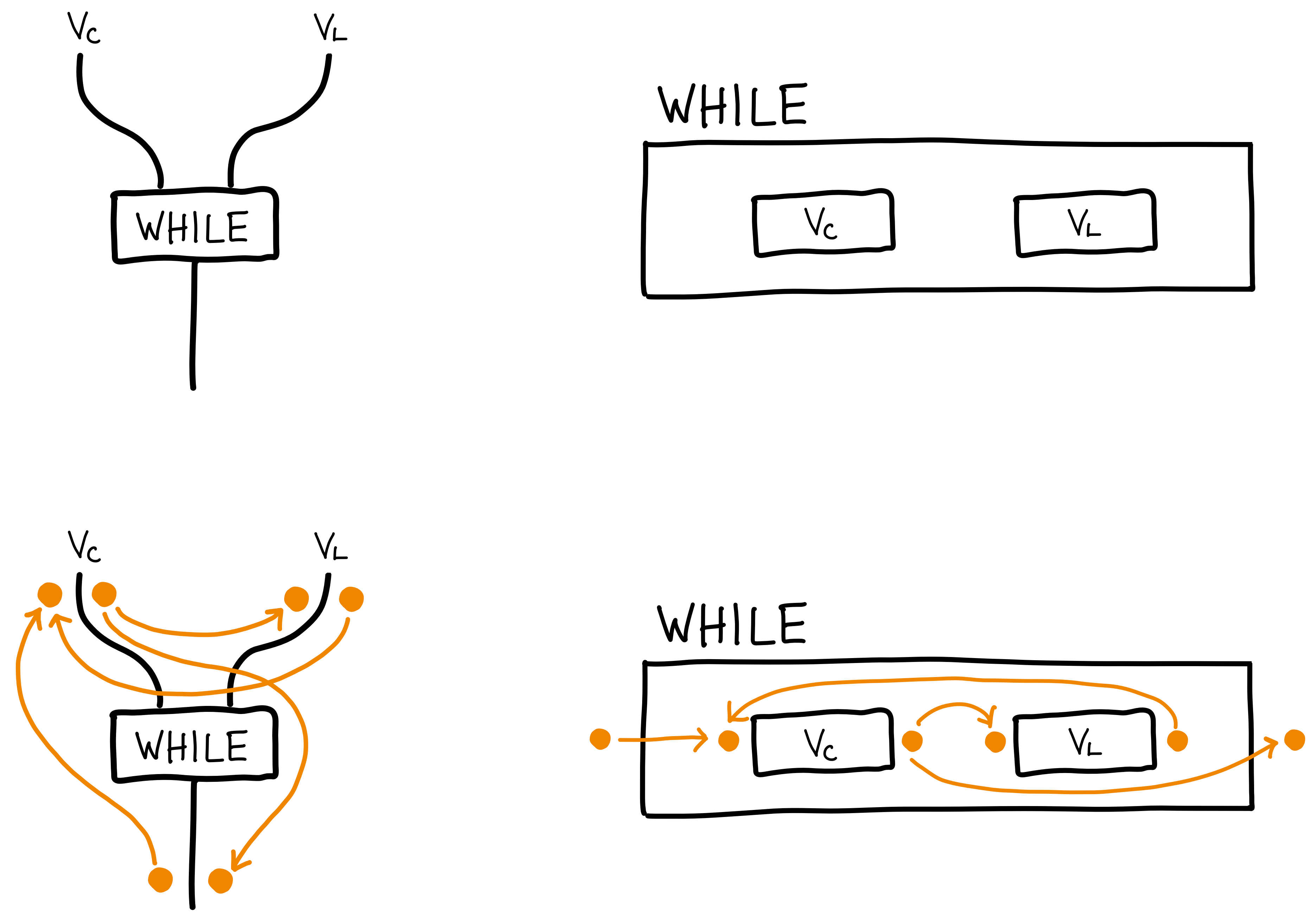




Spivak's [3] Operad of Wiring Diagrams



Control Flow for Terms



Summary

- define abstract syntax trees as free operads on its generators
- assign control flow to generators via functor into contour operads
- control flow composes according to the underlying tree structure
- can incorporate exceptional control flow
- what about: other type of events/handlers?
  translation to terms?
  more complex types in the control flow?

Thank you for listening