Control Flow as a Contour of Data Flow

Malin Altenmüller, Dan Ghica

Workshop on Diagrammatic Methods
24 January 2023

Control Flow as a Contour of Data Flow

Data Flow:

- data dependency between programs
- information states
- track definition and usage of variables

Control Flow as a Contour of Data Flow

Data Flow:

- data dependency between programs
- information states
- track definition and usage of variables

Control Flow

- order of execution of program fragments
- control flow analysis:
  important processing step for compiler optimisation, expose dead code fragments

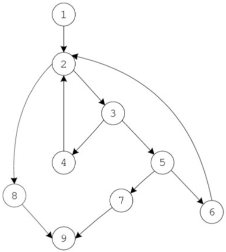
Control Flow Graphs

nodes: program blocks
edges: control flow

Source Program:

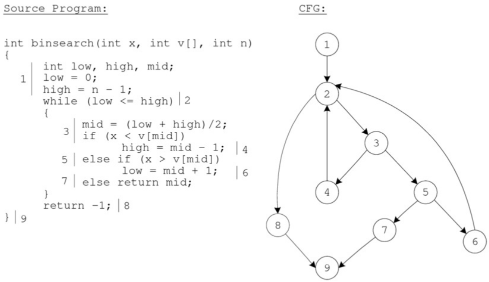
```
int binsearch(int x, int v[], int n)
{
   1 | int low, high, mid;
     | low = 0;
     | high = n - 1;
       while (low <= high) | 2
       {
          3 | mid = (low + high)/2;
            | if (x < v[mid])
                     high = mid - 1;  | 4
          5 | else if (x > v[mid])
                     low = mid + 1;   | 6
          7 | else return mid;
       }
       return -1; | 8
} | 9
```

CFG:



Control Flow Graphs

nodes: program blocks
edges: control flow

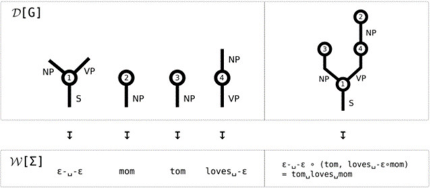


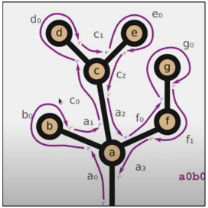
- disadvantage: graphs can be very big, algorithms on them are hard
- coming up: proposal for a different representation of control flow

Tree Contours[1]

- tree contours representing context-free languages
- trees represent derivations of a word
- linear contour around a derivation tree

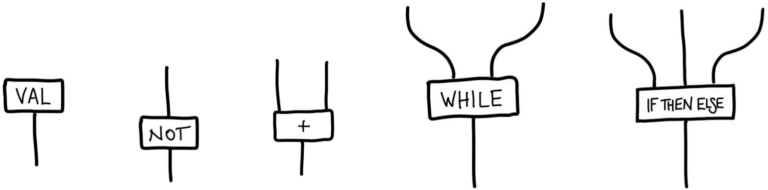
```
1 : S → NP VP
2 : NP → mom
3 : NP → tom
4 : VP → loves NP
```





[1] Melliès and Zeilberger, "Parsing as a lifting problem and the Chomsky-Schützenberger representation theorem".

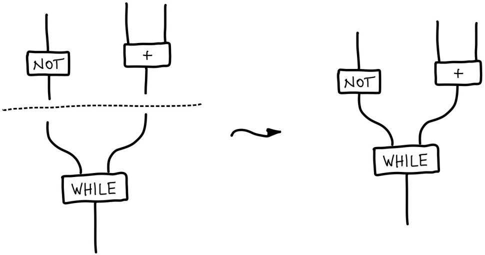
Abstract Syntax Generators



- imperative programs – variables, functions, control structures
- wires are variables
- multiple inputs, one output
- represented as a *species*: elements with multiple inputs and one output, e.g. *while* : $2 \to 1$

Composing Generators

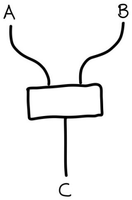
- composition is taking the free operad on the species
- amounts to building (partial) trees from the generators



Control Flow Contours(1)

given an operad, its generalised contour category consist of:

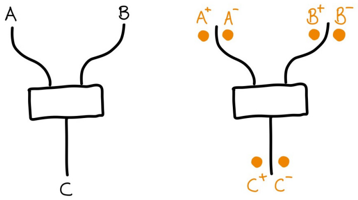
- objects: oriented colours of the operad



Control Flow Contours(1)

given an operad, its generalised contour category consist of:

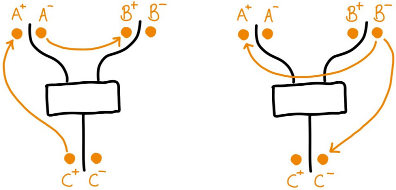
- objects: oriented colours of the operad



Control Flow Contours(1)

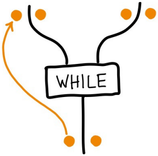
given an operad, its generalised contour category consist of:

- morphisms of the form $- \to +$, generated by colours and indices



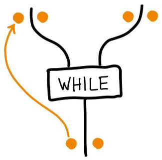
Control Flow Contours – Example 1

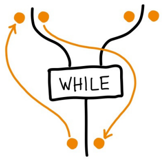
While loop:



Control Flow Contours – Example 1

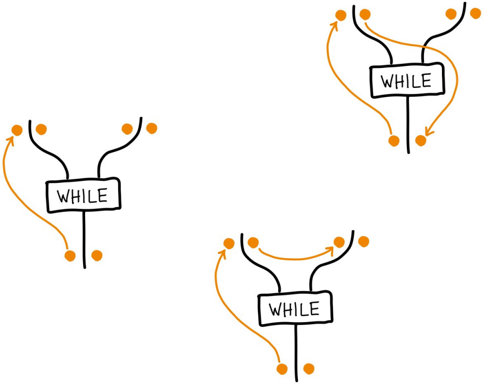
While loop:





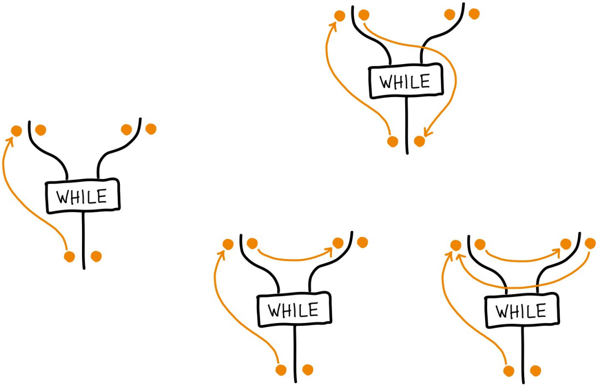
Control Flow Contours – Example 1

While loop:



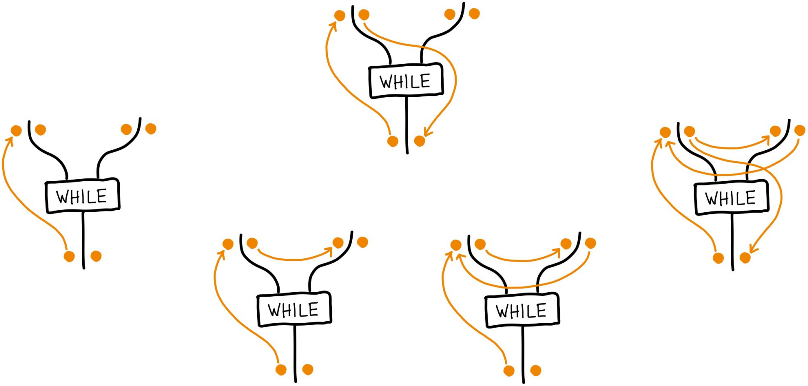
Control Flow Contours – Example 1

While loop:



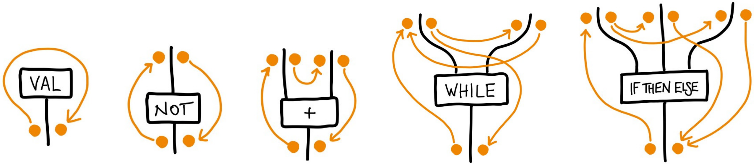
Control Flow Contours – Example 1

While loop:

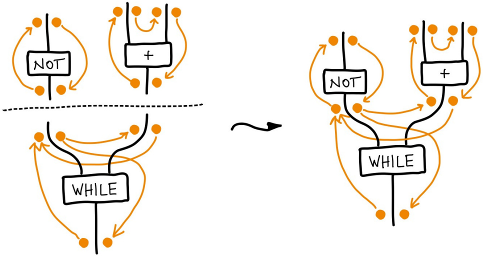


Control Flow Contours(2)

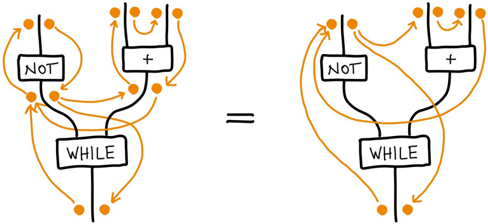
- edges represent control flow, for each of the generators
- contour is neither linear, nor ordered
- overapproximation of control flow



Contours compose!

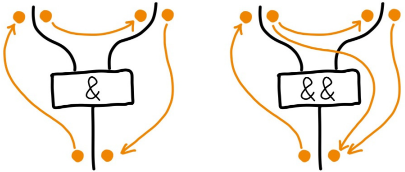


Contours compose!



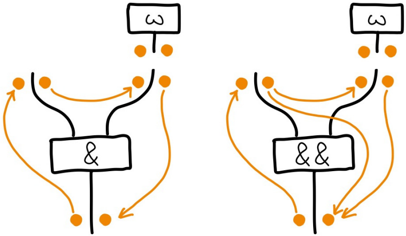
Control Flow Contours – Example 2

Boolean AND:



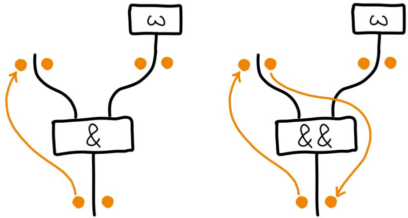
Control Flow Contours – Example 2

Boolean AND:



Control Flow Contours – Example 2

Boolean AND:



Control Flow Contours(3)

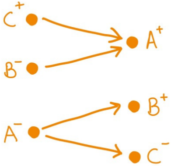
given an operad, its generalised contour category consist of:

- objects: oriented colours of the operad
- morphisms of the form $- \to +$, generated by colours and indices

This definition gives rise to a functor $\mathcal{C} : \mathbf{Operad} \to \mathbf{Cat}$.

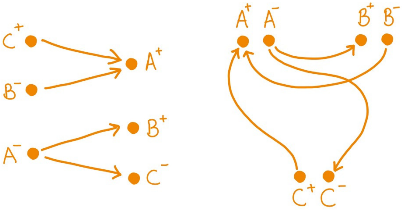
Constructing Contours and back

Here's a functor $\mathcal{J}$: **Cat** $\to$ **Operad**:



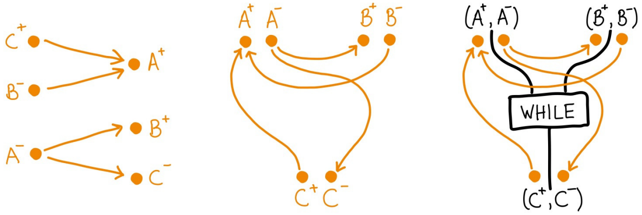
Constructing Contours and back

Here's a functor $\mathcal{J}$: **Cat** $\rightarrow$ **Operad**:



Constructing Contours and back

Here's a functor $\mathcal{J}$: **Cat** $\to$ **Operad**:



- colours are pairs of objects
- n-ary maps are sets of morphisms

Adjunction

- The contour is the left adjoint of the interior:
  $\mathbf{Operad}(\mathcal{O}, \mathcal{J}(\mathbf{C})) \cong \mathbf{Cat}(\mathcal{C}(\mathcal{O}), \mathbf{C})$

Adjunction

- The contour is the left adjoint of the interior:
  $\mathbf{Operad}(O, \mathcal{J}(\mathbf{C})) \cong \mathbf{Cat}(\mathcal{C}(O), \mathbf{C})$
- Actually, it's a *local* left adjoint:

$$\begin{tikzcd} O \arrow[d, "f"] & \mathcal{C}(O) \arrow[d, "g"] \\ \mathcal{J}(\mathbf{C}) & \mathbf{C} \end{tikzcd}$$

  for every map $f : O \to \mathcal{J}(\mathbf{C})$, there exists an object $D' \in \mathbf{Cat}$, a map $g : D' \to \mathbf{C}$ and a map $f_0 : O \to \mathcal{J} D'$, such that $f = \mathcal{J}(g) f_0$.

Adjunction

- The contour is the left adjoint of the interior: $\mathbf{Operad}(O, \mathcal{J}(\mathbf{C})) \cong \mathbf{Cat}(\mathcal{C}(O), \mathbf{C})$
- Actually, it's a *local* left adjoint:

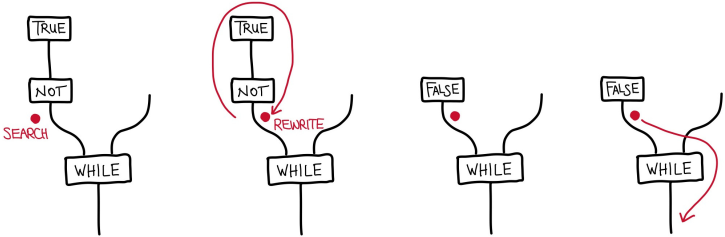
$$\begin{tikzcd} O \arrow[d, "f"] & \mathcal{C}(O) \arrow[d, "g"] \\ \mathcal{J}(\mathbf{C}) & \mathbf{C} \end{tikzcd}$$

  for every map $f : O \to \mathcal{J}(\mathbf{C})$, there exists an object $D' \in \mathbf{Cat}$, a map $g : D' \to \mathbf{C}$ and a map $f_0 : O \to \mathcal{J}D'$, such that $f = \mathcal{J}(g)f_0$.

- Take D' to be the contour of O according to f.

From Syntax to Graph Rewriting Semantics[2]

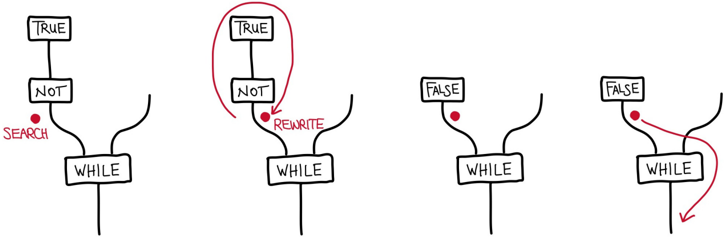
- token = current "focus"
- token moves or triggers a rewrite



[2] Muroya and Ghica, "The Dynamic Geometry of Interaction Machine: A Token-Guided Graph Rewriter".

From Syntax to Graph Rewriting Semantics[2]

- token = current "focus"
- token moves or triggers a rewrite



- objects in the contour: potential token positions
- morphisms in the contour: potential token movement

[2] Muroya and Ghica, "The Dynamic Geometry of Interaction Machine: A Token-Guided Graph Rewriter".

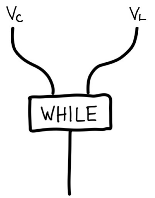
Contours for Terms

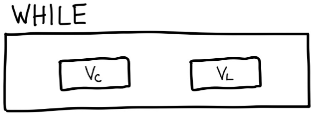
- how does control flow translate from graphs to terms?

[3] Spivak, "The operad of wiring diagrams: formalizing a graphical language for databases, recursion, and plug-and-play circuits".

Contours for Terms

- how does control flow translate from graphs to terms?
- terms have holes for variables[3]

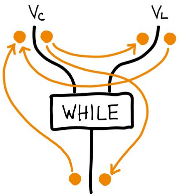


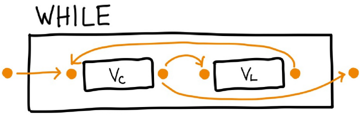


[3]Spivak, "The operad of wiring diagrams: formalizing a graphical language for databases, recursion, and plug-and-play circuits".

Contours for Terms

- how does control flow translate from graphs to terms?
- terms have holes for variables[3]





[3] Spivak, "The operad of wiring diagrams: formalizing a graphical language for databases, recursion, and plug-and-play circuits".

Summary

- new syntax for control flow, based on abstract syntax graph
- compositional, simpler analysis possible
- based on the notion of contour category

Summary

- new syntax for control flow, based on abstract syntax graph
- compositional, simpler analysis possible
- based on the notion of contour category
- translation to token-based graph rewriting semantics
- translation from trees to terms

Thoughts or questions? malin.altenmuller@huawei.com

Summary

- new syntax for control flow, based on abstract syntax graph
- compositional, simpler analysis possible
- based on the notion of contour category
- translation to token-based graph rewriting semantics
- translation from trees to terms
- what about: hypergraphs, more complex vertex/colour types?

Summary

- new syntax for control flow, based on abstract syntax graph
- compositional, simpler analysis possible
- based on the notion of contour category
- translation to token-based graph rewriting semantics
- translation from trees to terms
- what about: hypergraphs, more complex vertex/colour types?

THANK YOU FOR YOUR ATTENTION!

References

Melliès, Paul-André and Noam Zeilberger. "Parsing as a lifting problem and the Chomsky-Schützenberger representation theorem". In: *MFPS 2022 - 38th conference on Mathematical Foundations for Programming Semantics*. Ithaca, NY, United States, July 2022. URL: https://hal.archives-ouvertes.fr/hal-03702762.

Muroya, Koko and Dan R. Ghica. "The Dynamic Geometry of Interaction Machine: A Token-Guided Graph Rewriter". In: *Logical Methods in Computer Science* Volume 15, Issue 4 (Oct. 2019). DOI: 10.23638/LMCS-15(4:7)2019. URL: https://lmcs.episciences.org/5882.

Spivak, David I. "The operad of wiring diagrams: formalizing a graphical language for databases, recursion, and plug-and-play circuits". In: *CoRR* abs/1305.0297 (2013). arXiv: 1305.0297. URL: http://arxiv.org/abs/1305.0297.

Image CFG from:
Al-Ekram, R. & Kontogiannis, Kostas. (2004). Source code modularization using lattice of concept slices. 195- 203. 10.1109/CSMR.2004.1281420.