DisCoRun: An implementation of the Monoidal Computer and Run language in DisCoPy

Martín Coll[1] [0009−0009−0146−7132]

Department of Computer Science, University of Buenos Aires, mcoll@dc.uba.ar, CABA, C1428EGA, Argentina

Abstract. We present DisCoRun, an implementation of the Monoidal Computer and Run language in DisCoPy, released as the `discopy-run` package. Our design follows *Programs as Diagrams* [1]: programs are typed string diagrams equipped with cartesian data services (copy and delete), a distinguished program type with decidable equality, and a universal evaluator family. In our case study, we instantiate this evaluator for the POSIX Shell Command Language [3] using `subprocess.run`. In this interpretation, sequential composition (`»`) realizes shell-style pipelines and monoidal tensor (`@`) models independent process branches. `Copy`, `Merge`, and `Trace` arise naturally from the monoidal computer representation, extending the Shell Command Language with primitive fan-out, fan-in, and feedback operators. The resulting system highlights the practical applications of categorical computability: developers can construct programs diagrammatically, analyze them compositionally, and translate them to a chosen target language.

Keywords: Categorical computability · String diagram.

1 Motivation

Graphical calculus String diagrams provide a rigorous graphical calculus for monoidal categories [2]. Sequential composition is represented by vertical wiring, while tensor product is represented by horizontal juxtaposition. Additional structural features, such as symmetries and traces, are encoded topologically. Diagram deformations that preserve connectivity correspond to sound equational reasoning.

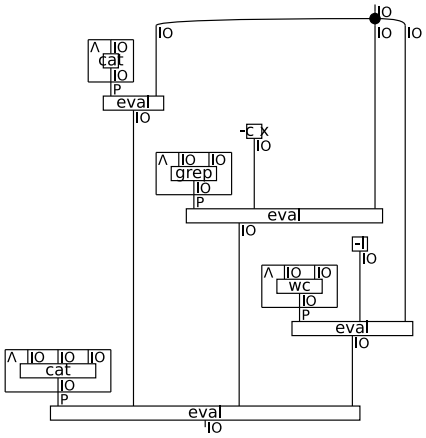
Programs as diagrams This perspective motivates treating programs as diagrams [1]. The universal evaluator is the central primitive from which sequential and parallel composition, partial evaluation, and Kleene fixed-point constructions are derived. In particular, Kleene's Second Recursion Theorem explains self-referential programs and connects naturally to supercompilation.

Computing with string diagrams DisCoRun operationalizes this approach on top of DisCoPy, the Python toolkit for monoidal categories. In this setting, diagrams are typed program objects that support composition, rewriting, and

functorial interpretation into executable backends (e.g., Python functions). We instantiate this machinery with the monoidal computer interface of *Programs as Diagrams*: `RUN` is the single primitive induced by the running surjection, while richer behaviors are derived by composing diagrams.

```
Copy(3) >> (
    Command(["cat"]),
    Command(["grep", "-c", "x"]),
    Command(["wc", "-l"]),
) >> Merge(3)
```

(a) The program equation.



(b) The program diagram.

Fig. 1: Diagrams imbue visual intuition with formal guarantees.

2 POSIX Shell Command Language Case Study

2.1 Shell Evaluator

This case study instantiates the evaluator on `IO` wires using `subprocess.run`. Each command box denotes a POSIX command invocation interpreted as an `IO` transformer.

Operators Vertical composition (`>>`) models shell pipeline composition: the stdout stream of one command becomes the stdin stream of the next command. Monoidal tensor (`@`) denotes independent command branches at the diagram level, with the `fork` command as a close translation. The language also exposes three explicit structural operators: `Copy` for fan-out, `Merge` for fan-in, and `Trace` for feedback. These are diagrammatic extensions rather than native POSIX Shell Command Language constructs.

Acknowledgments. M. Coll acknowledges public, tuition-free, and high-quality higher education, Ley 24.521.

Table 1: Mapping of Categorical Operations to POSIX Shell Command Language Concepts

Categorical Operation	POSIX Shell Command Language Concept
Vertical Composition (>>)	Command pipelining (`|`)
Monoidal Tensor (@)	Independent branches (e.g., `fork`)
Copy	Stream fan-out (e.g., `tee`)
Merge	Stream fan-in (e.g., `cat`)
Trace	Feedback/cyclic wiring (outside POSIX grammar)

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] Dusko Pavlovic. *Programs as Diagrams: From Categorical Computability to Computable Categories*. 2023. arXiv: 2208.03817 [cs.LO]. URL: https://arxiv.org/abs/2208.03817.

[2] P. Selinger. "A Survey of Graphical Languages for Monoidal Categories". In: *New Structures for Physics*. Springer Berlin Heidelberg, 2010, pp. 289–355. ISBN: 9783642128219. DOI: 10.1007/978-3-642-12821-9_4. URL: http://dx.doi.org/10.1007/978-3-642-12821-9_4.

[3] The IEEE and The Open Group. *POSIX.1-2024: The Open Group Base Specifications Issue 8*. 2024. URL: https://pubs.opengroup.org/onlinepubs/9799919799/.