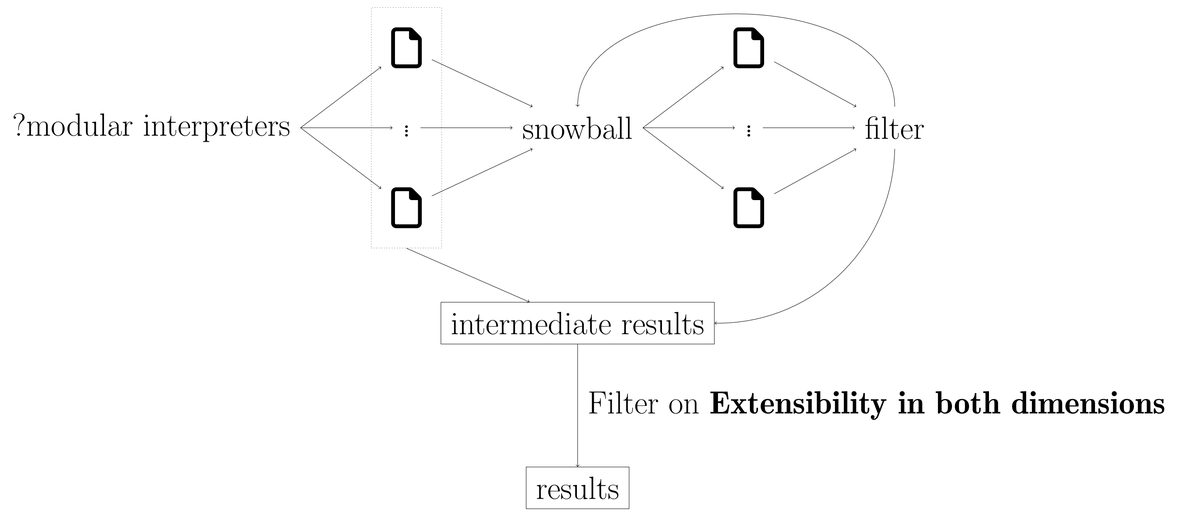
A Survey on Modular Interpreters and Language Composition

Damian Frölich
University of Amsterdam

Introduction

Programming languages provide crucial abstractions for programmers to increase productivity, maintainability and safety. Language oriented programming (LOP) is a programming paradigm that makes these abstractions a central point in the development process by including language construction in the development process instead of seeing a language as a fixed entity. However, development of a programming language is not trivial. A language often requires multiple components to be useful, like an interpreter, pretty printer, debugger; and writing these components from scratch is complex. To reduce this complexity, a modular approach to the construction of these components can be applied, which in its optimal form results in the creation of new languages by composing existing languages. **In this work, a systematic search of the current literature is performed to identify approaches suitable for the construction of modular interpreters in the context of language oriented programming**.

Search



Extensibility in both dimensions

An interpreter assigns meaning to a language

$$\text{Language} \xrightarrow{\text{interpreter}} \text{Meaning}$$

$\Longrightarrow$

Two extension points

Extend a language with a new construct

$$\text{Language+C} \xrightarrow{\text{interpreter}_{+C}} \text{Meaning}_{+C}$$

Need to modify all existing interpretations to handle the new construct.
Difficult in a functional language

Extend a language with a new interpretation

$$\text{Language} \xrightarrow{\text{interpreter}_a} \text{Meaning}_a \qquad \text{Language} \xrightarrow{\text{interpreter}_b} \text{Meaning}_b$$

Need to implement the interpretation for every construct in the language.
Difficult in an object oriented language

How suitable are the approaches achieving extensibility in both dimensions for Language Oriented Programming?

Evaluation criteria

AS. **Adaptable semantics:** An approach supports adapting semantics of existing constructs.

SSG. **Static semantic guarantees:** There is a guarantee that all semantics are defined for all syntax and semantics is not dependent on order of extension.

SC. **Separate compilation:** Compiling a new language (e.g., syntactic extension or new semantics) does not encompass recompilation of the original syntax or semantics.

IE. **Independent extensibility:** It is possible to combine language extensions independently developed, ideally without any glue code requirements.

O/C. **Open or closed**: An approach is closed when all possible languages that can be constructed with the approach are known.

Results

	AS	SSG	SC	IE	O/C
Functional					
Monad transformers (Liang, Hudak, and Jones 1995)	●	◓	○	●	Open
Monad transformers (Duponcheel 1995)	●	◓	●	●	Open
Data types à la carte (W. Swierstra 2008)	●	●	●	●	Closed
Polymorphic variants (Garrigue 1998)	●	◓	●	◓	Closed
Finally tagless (Carette, Kiselyov, and Shan 2009)	●	●	●	●	Closed
Open data types and Open functions (Löh and Hinze 2006)	○	●	○	●	Closed
Object oriented					
Object algebras (S. Oliveira and Cook 2012)	●	●	●	○	Open
Object algebras with implicit context propagation (Inostroza and Storm 2015)	●	●	●	◓	Open
Javascript modules (Kerchove, Noyé, and Südholt 2015)	●	○	N/A	●	Open
Mixin modules (Duggan and Sourelis 1996)	●	◓	○	●	Closed
Extensible algebraic data types with defaults (Zenger and Odersky 2001)	●	◓	○	○	Closed
Attribute grammars					
UUAG with AspectAG (Viera, S. D. Swierstra, and Middelkoop 2012)	●	●	○	○	Open
Composable attribute grammars (R. Farrow, T. Marlowe, and D. Yellin 1992)	●	●	○	◓	Open
Composable attribute grammars — separate evaluation (Rodney Farrow, T. J. Marlowe, and D. M. Yellin 1992)	●	●	●	◓	Closed
Modular attribute grammars (Dueck and Cormack 1990)	●	○	○	◓	Closed
Multiple attribute inheritance (Mernik et al. 2000)	●	◓	○	●	Closed

● = fully supported, ◓ = partially supported, ○ = not supported.

Conclusion

Multiple approaches are available to make LOP more feasible, and the best approach depends on the required criteria.

- Open? Object algebras with implicit context propagation
- Closed? Tagless, data types à la carte, or object algebras with implicit context propagation
- No IE but open? Object algebras, Object algebras with implicit context propagation or UUAG
- IE and open? Monad transformers (catamorphism)

More information

d.frolich@uva.nl

University of Amsterdam

Complex Cyber Infrastructure