Formal Semantics
as a Language Designer's
Toolbox

Paolo G. Giarrusso · Klaus Ostermann · Tillmann Rendel
University of Marburg · University of Tübingen

Eric Walkingshaw
University of Marburg · Oregon State University

Presentation by Tillmann Rendel
at the Second International Workshop
on Domain-Specific Language Design and Implementation
in Portland, Oregon, October 20, 2014

The great work
of this community makes

Language Implementation

easier and easier

so more and more

programmers

build their own custom

DSL

and become

programmers

and

language designers

at the same time

Language Design

is still

HARD

How can a

programmer/

language designer

learn to design languages that are

elegant and usable?

Formal Semantics

- Semanticists know a lot about languages
  (it's their job)
- Semanticists know a lot about elegance
  (they are mathematicians)
- Mathematical elegance has pragmatic advantages

  Elegant = powerful and simple, less to learn

Can formal semantics guide a programmer/language designer towards an elegant and usable design?

Problem 1

- *Problem*: Formal semantics is a lot of work.
- *Proposed Solution*: Don't actually formalize the semantics, just let the insights of formal semantics guide your design process.

Problem 2

- *Problem*: The language of the semanticists is not understandable to the working programmer/language designers
- *Proposed Solution*: Package the insights from formal semantics as **language design patterns**.

Language Design Patterns

- Patterns work for software design,
  we want to adapt them for language design
- Use terms that make sense to the working programmer/language designer

- Not in scope: language implementation patterns
- Not in scope: designing perfect languages
- In scope: language design patterns for reasonably elegant, usable languages.

name **Bound & Binding Occurrences**

problem How to structure names?

solution Distinguish bound and binding occurrences of names. Each bound occurrences refers to a binding occurrence.

effects You can reason about the naming structure of a program in terms of „this name here is bound there“

name **Bound & Binding Occurrences**

pro

solu

ef

name **Lexical Scoping**

problem Which bound occurrence refers to which binding occurrence?

solution All bound occurrences in a continuous region of the source file bind to the same binding occurrence.

effects You can reason about the binding structure statically.

name **Bound & Binding Occurrences**

name **Lexical Scoping**

name **Domain-Specific Scoping**

problem Which bound occurrence refers to which binding occurrence?

solution Use domain-specific criteria to match bound to binding occurrences.

effects Your binding structure supports your domain integration.

name **Meaning**

problem How to specify the semantics?

solution Map every program to its meaning.

effects Allows to identify programs that mean the same but work differently internally.

name **Meaning**

pro

solu

ef

name **Simple Meaning**

problem How to structure the meaning?

solution Choose the simplest thing that works.

effects Carefully choosing the meaning helps you focus your design on your domain.

name **Meaning**

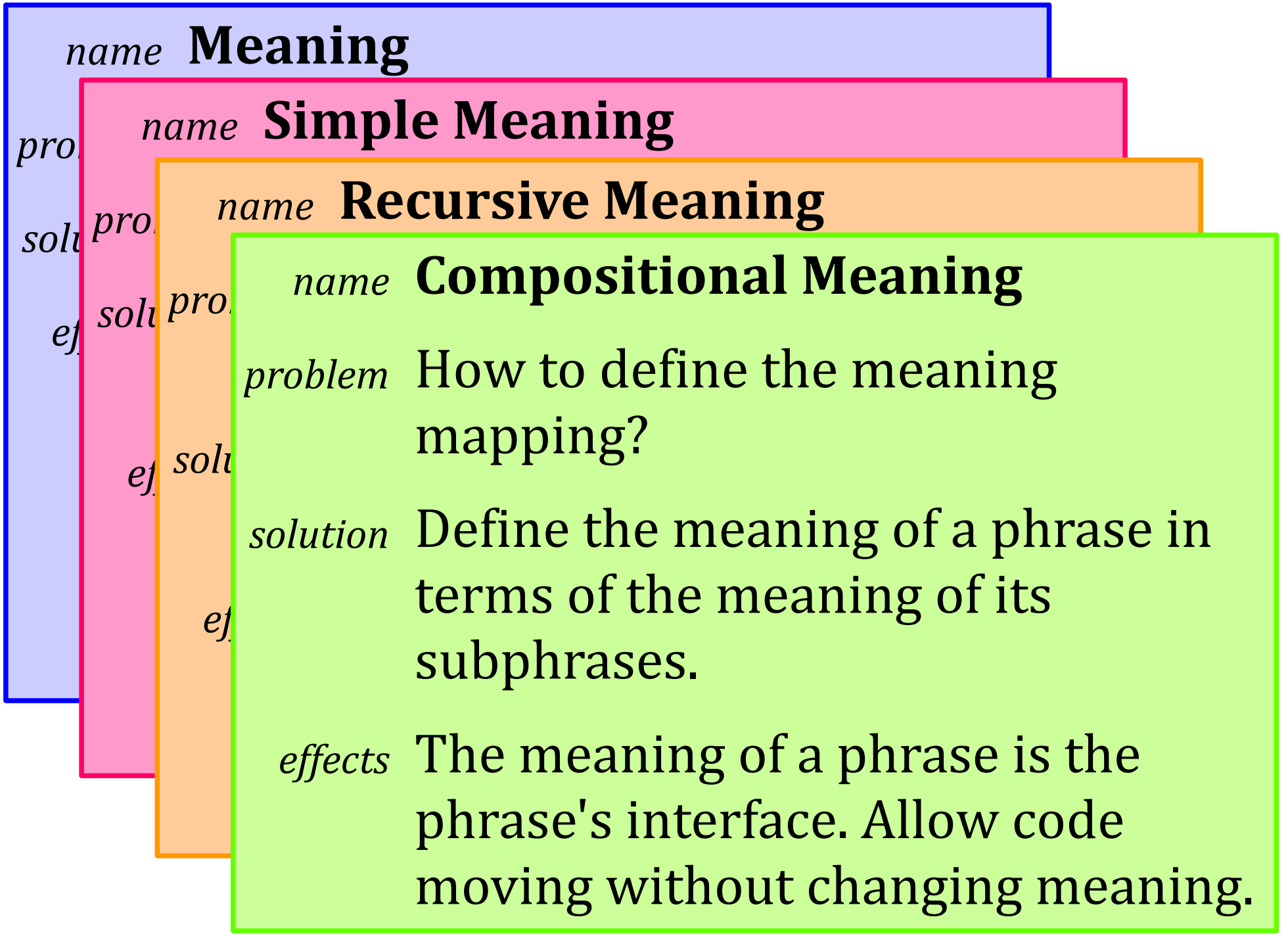
name **Simple Meaning**

name **Recursive Meaning**

problem How to define the meaning mapping?

solution Map each *phrase* of the program to its meaning.

effects You can explain what a part of a program means.


name Meaning
name Simple Meaning
name Recursive Meaning
name Compositional Meaning
problem How to define the meaning mapping?
solution Define the meaning of a phrase in terms of the meaning of its subphrases.
effects The meaning of a phrase is the phrase's interface. Allow code moving without changing meaning.

name	**Type Structure**
problem	How to structure the primitives?
solution	Structure your language design around the available types of values. Think of the primitives as the interfaces of the types.
effects	Easier to not forget primitives. Structuring principle also for documentation.

name **Type Structure**

pro

solu

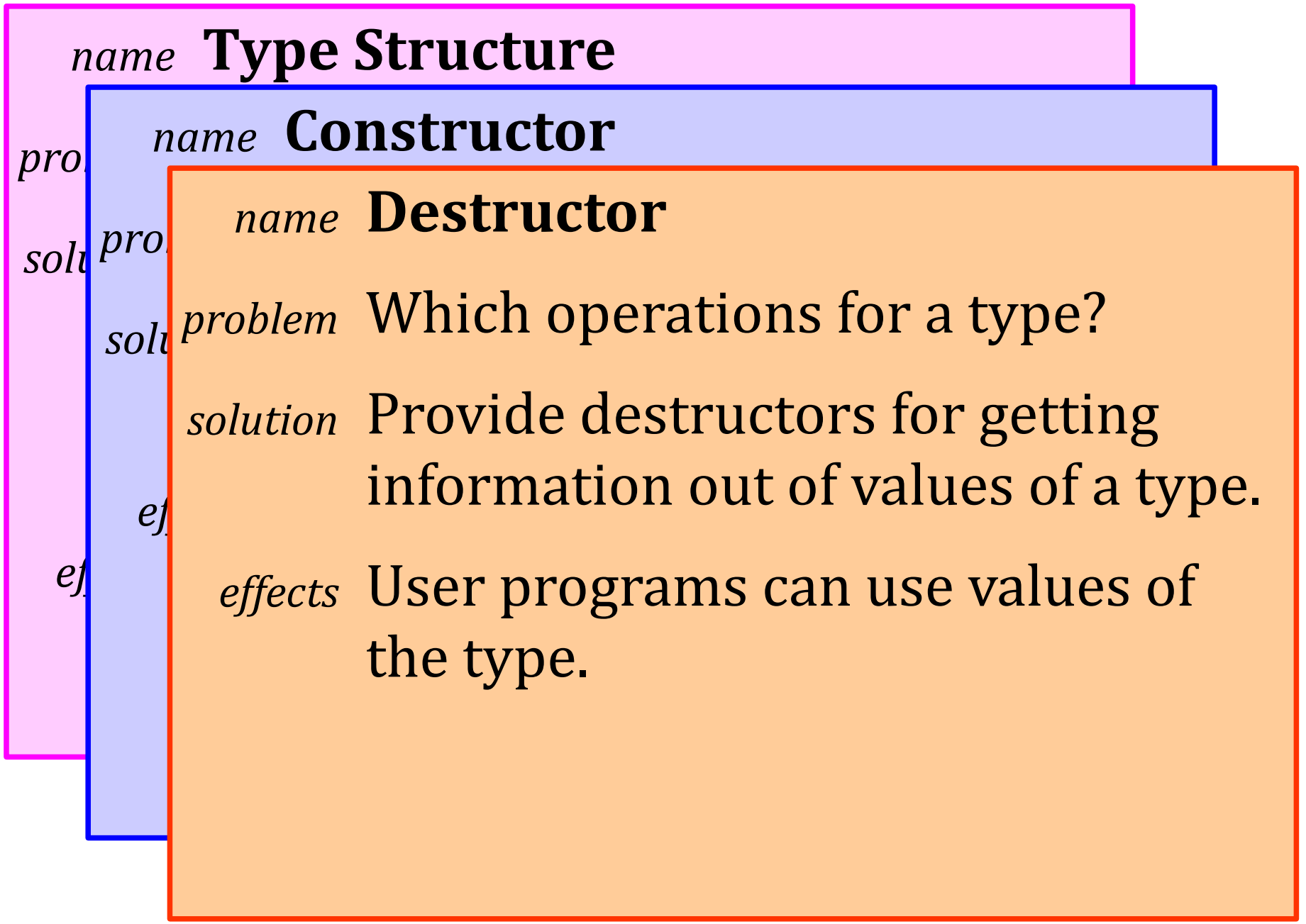
ef

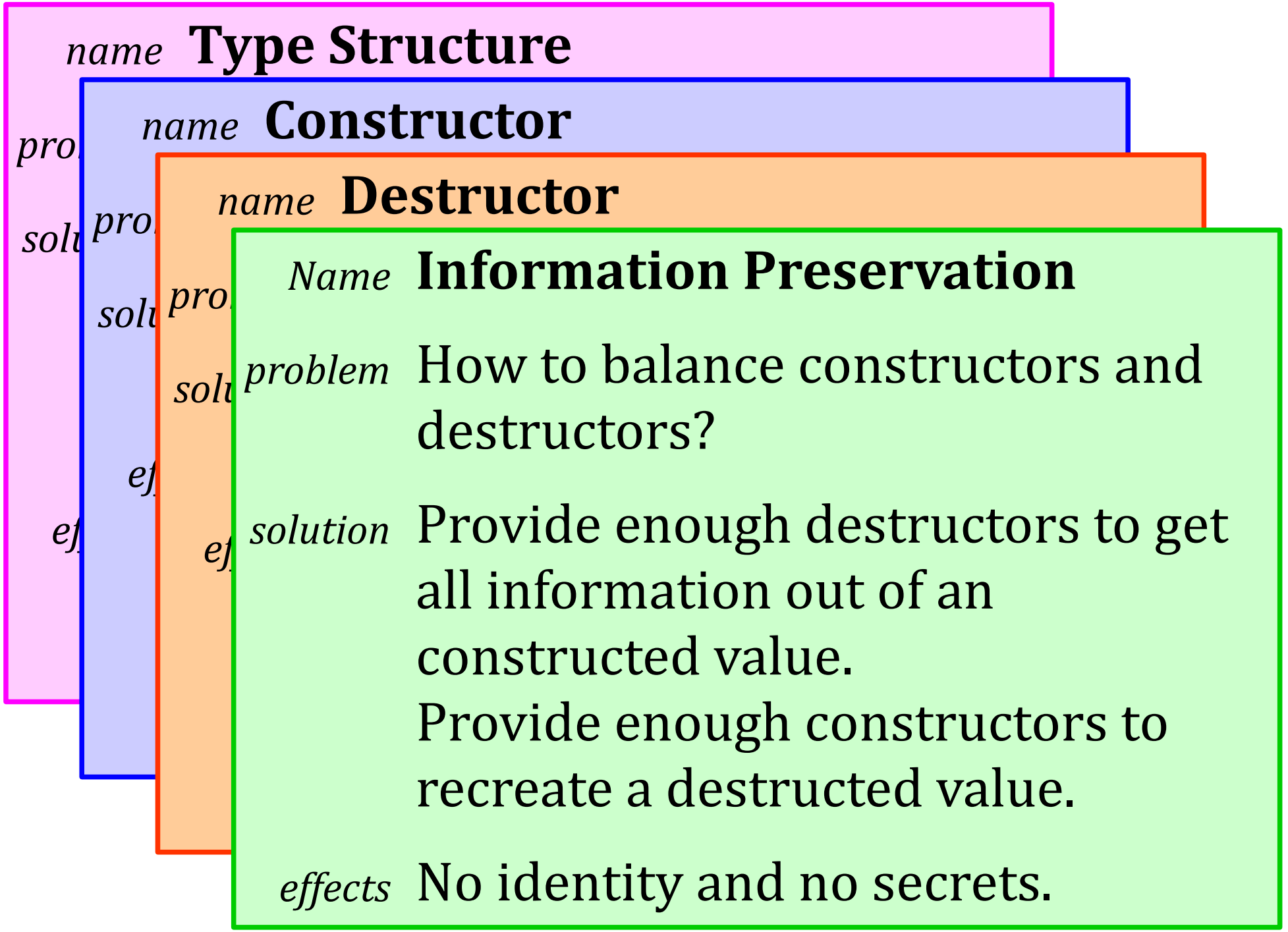
name **Constructor**

problem Which operations for a type?

solution Provide constructors for making new values of a type.

effects User programs can create values of the type.


name Type Structure
name Constructor
name Destructor
problem Which operations for a type?
solution Provide destructors for getting information out of values of a type.
effects User programs can use values of the type.


name Type Structure
name Constructor
name Destructor
Name Information Preservation
problem How to balance constructors and destructors?
solution Provide enough destructors to get all information out of an constructed value.
Provide enough constructors to recreate a destructed value.
effects No identity and no secrets.

Language Design Patterns ...

- guide the design process
  („*think of all constructors*“)
- structure the design
  („*separate constructors and destructors*“)
- highlight design choices
  („*which kind of scoping is appropriate?*“)
- explain effects („*user programs can ...*“)
- interact („*if a compositional meaning is a phrase's interface, a simple meaning is a better interface*“)

Conclusion

- We can try to phrase insights from formal semantics as language design patterns
- The language design patterns should use terms that make sense to the working programmer/language designer
- Future Work:
  Collect language design patterns and
  distill them into a coherent pattern language.