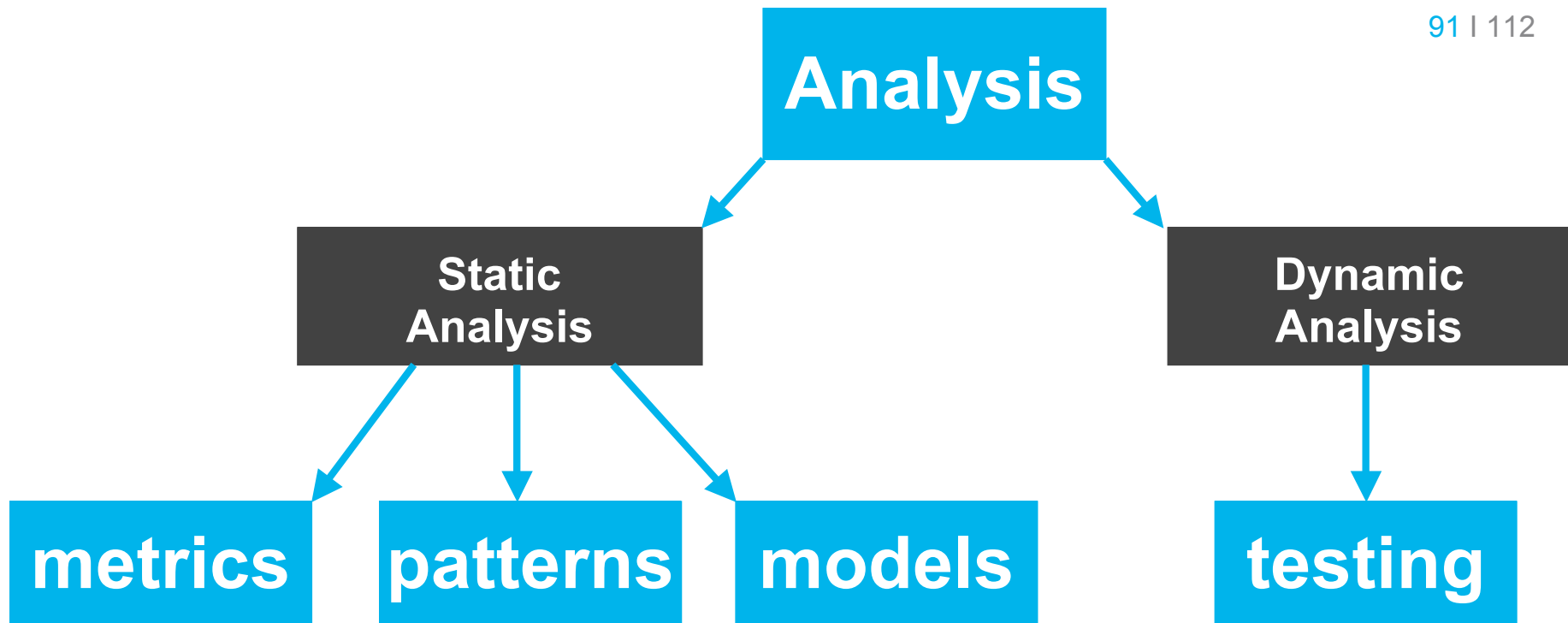


Structure of the lecture



Software Improvement Group

91 | 112

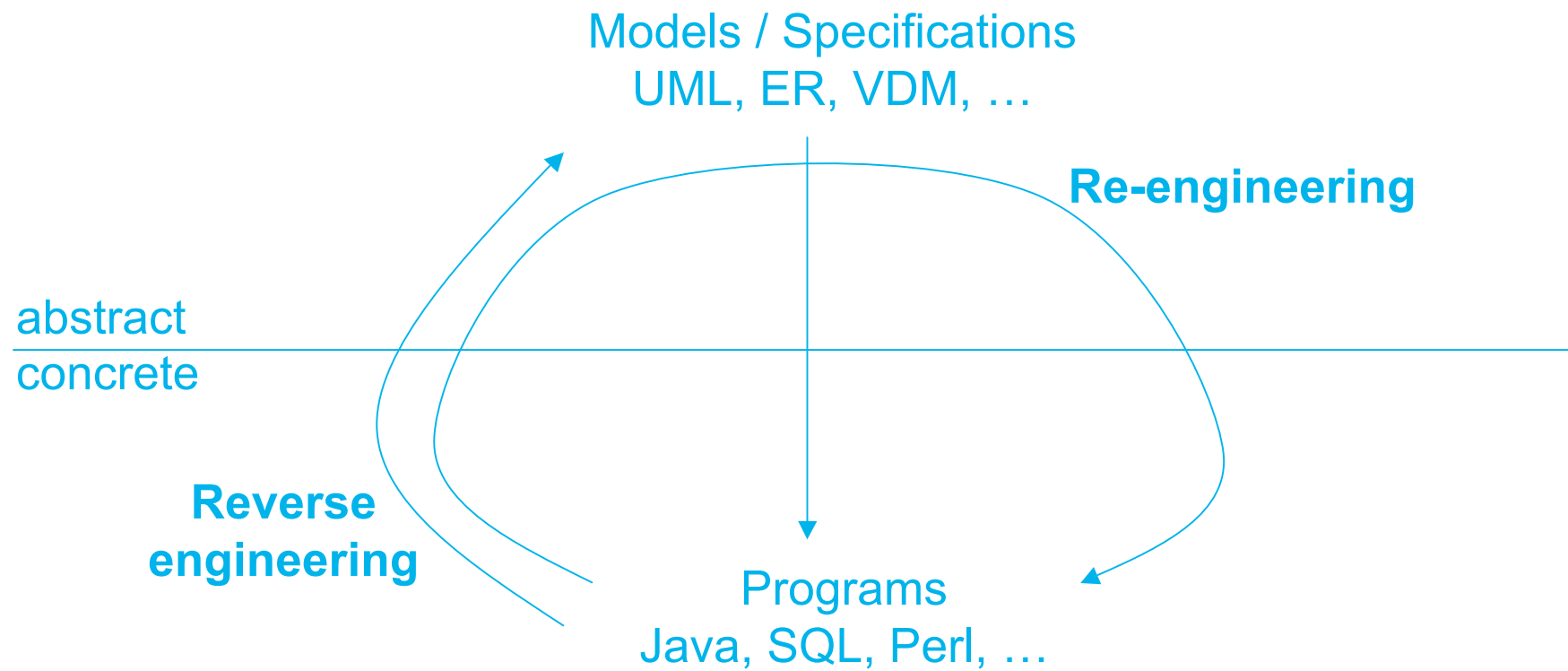




Software Improvement Group

92 | 112

REVERSE ENGINEERING



Dependencies and graphs

94 | 112

- Extraction, manipulation, presentation
- Graph metrics
- Slicing

Advanced

- Type reconstruction
- Concept analysis
- Programmatic join extraction

Extraction

From program sources, extract basic information into an initial source model.

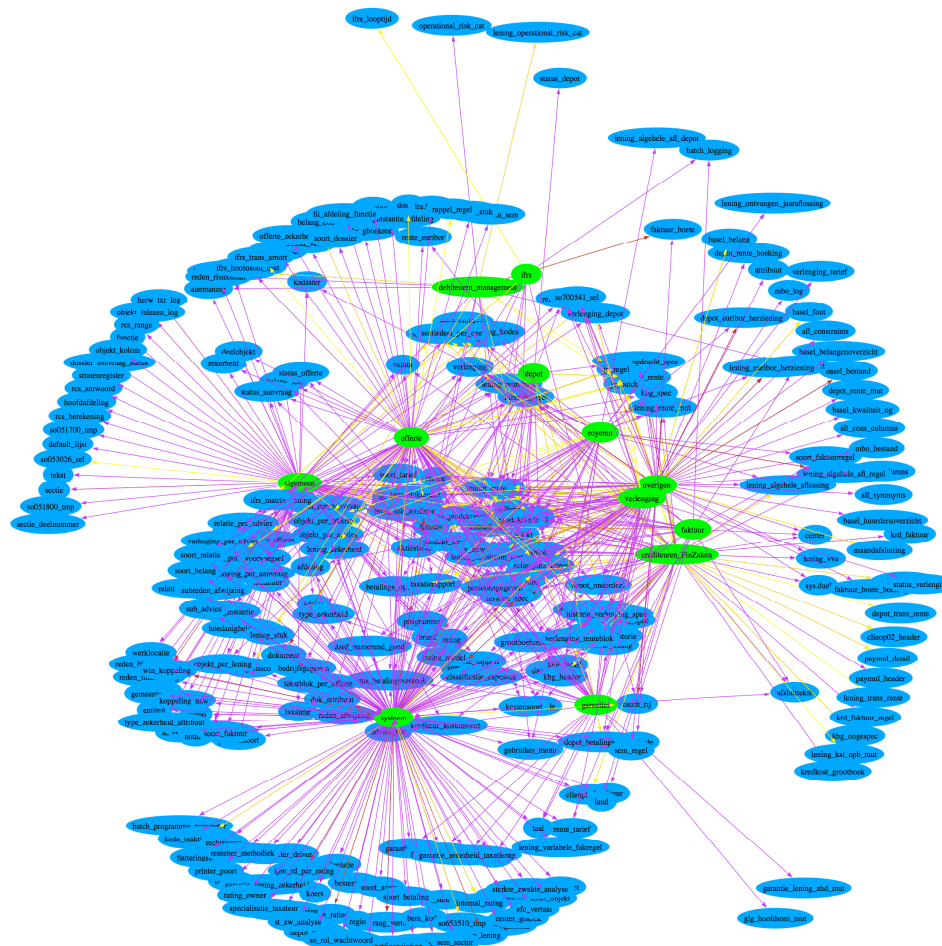
Manipulation

Combine, condense, aggregate, or otherwise process the basic information to obtain a derived source model.

Presentation

Visualize or otherwise present source models to a user.

Example



Green oval = module
Blue oval = table

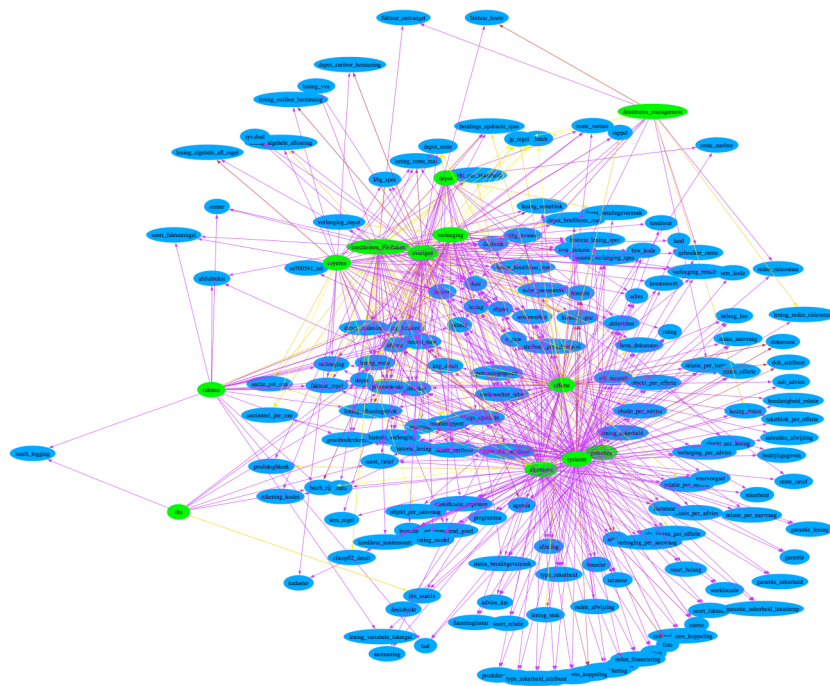
Purple arrow = select operation
Yellow arrow = insert/update operation
Brown arrow = delete operation

Example

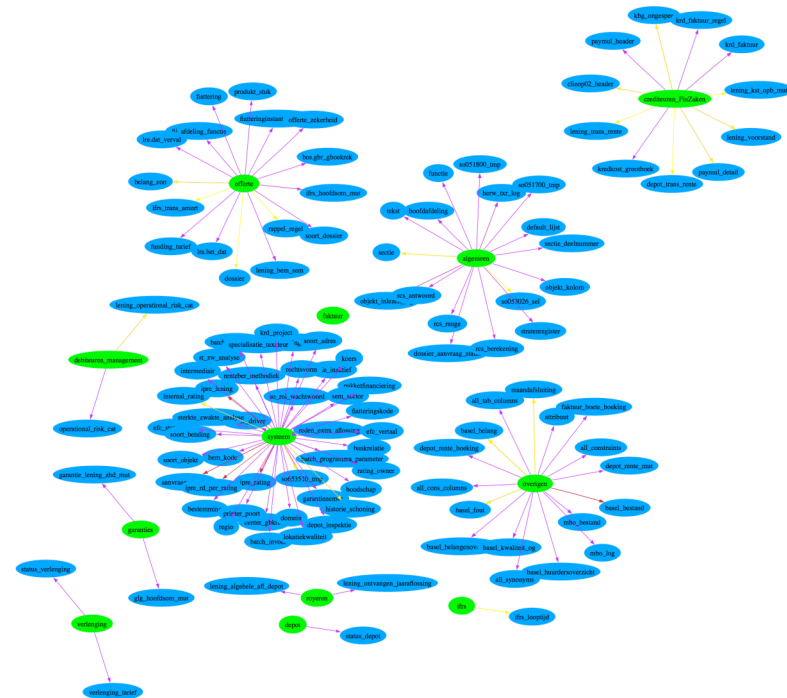


Software Improvement Group

97 | 112



Tables used by multiple modules.



Tables used by a single module.

Relation

`type Rel a b = Set (a,b)` [set of pairs](#)

Graph

`type Gph a = Rel a a` [endo-relation](#)

Labeled relation

`type LRel a b l = Map (a,b) l` [map from pairs](#)

Note

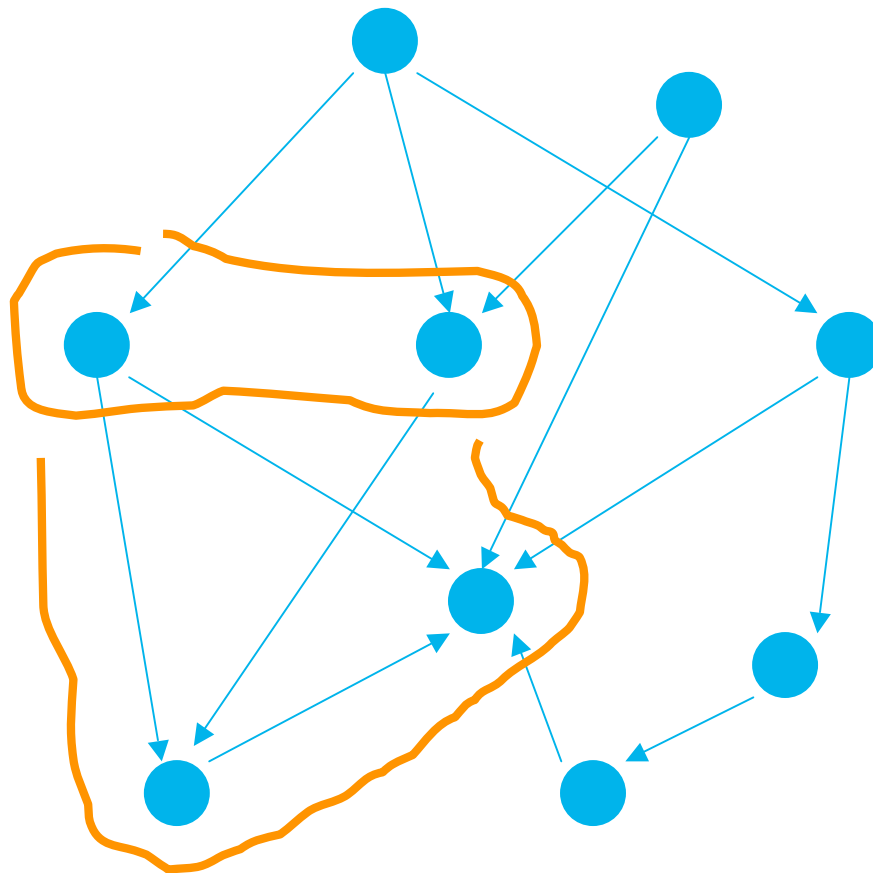
`Rel a b = Set (a,b) = Map (a,b) () = LRel a b ()`

Slicing (forward)



Software Improvement Group

99 | 112

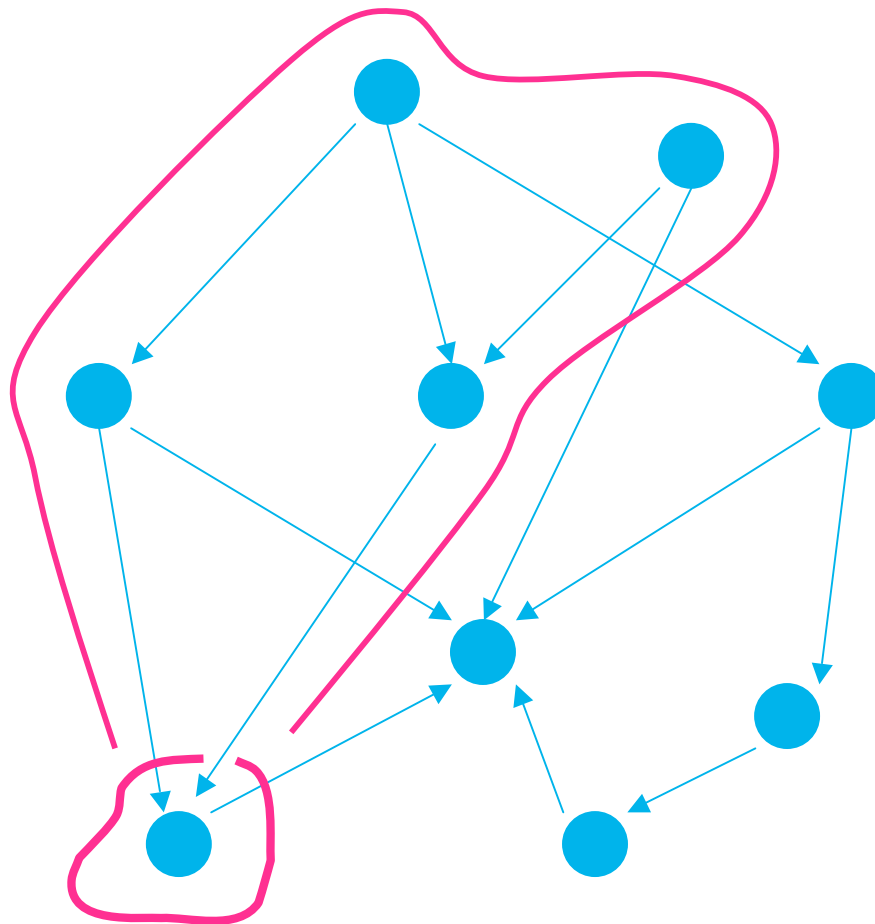


Slicing (backward)



Software Improvement Group

100 | 112

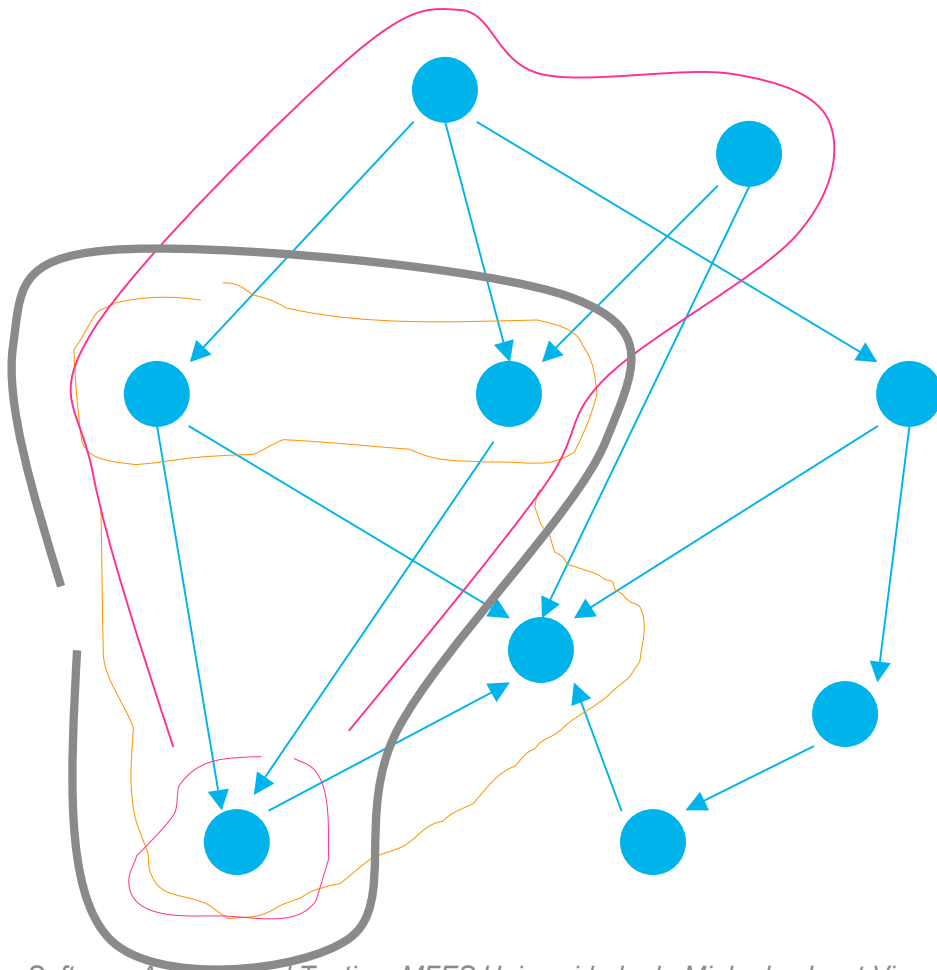


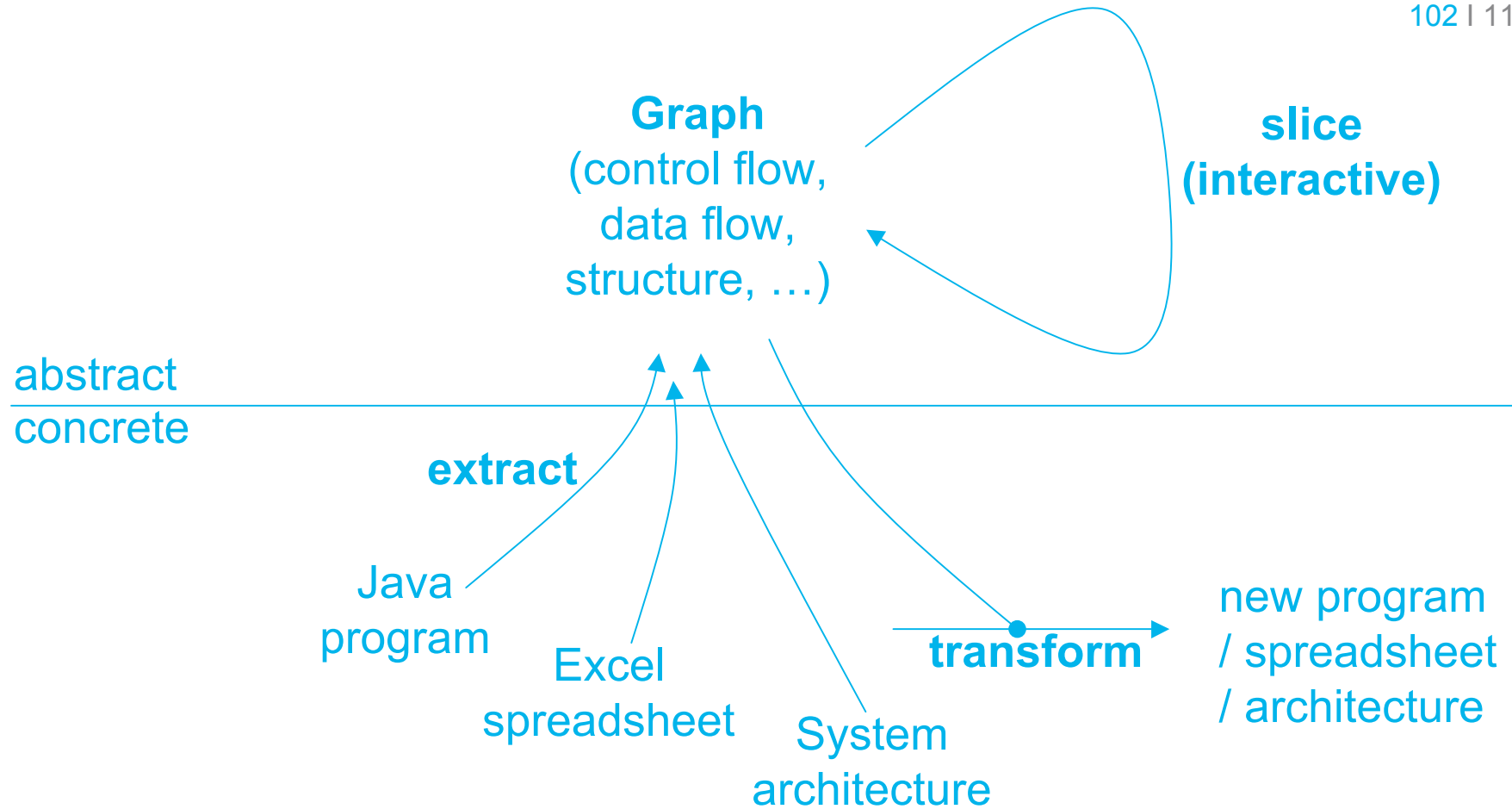
Chop = Forward \cap Backward



Software Improvement Group

101 | 112





Further reading



Software Improvement Group

103 | 112

See

Arun Lakhotia.

Graph theoretic foundations of program slicing and integration.

The Center for Advanced Computer Studies, University of Southwestern Louisiana.

Technical Report CACS TR-91-5-5, 1991.

Type reconstruction (from type-less legacy code)

See

- Arie van Deursen and Leon Moonen. *An empirical Study Into Cobol Type Inferencing*. Science of Computer Programming **40**(2-3):189-211, July 2001

Basic idea

1. Extract basic relations (entities are variables)
 - assign: ex. `a := b`
 - expression: ex. `a <= b`
 - arrayIndex: ex. `A[i]`
2. Compute derived relations
 - typeEquiv: variables belong to the same type
 - subtypeOf: variables belong to super/subtype
 - extensional notion of type: set of variables

Type reconstruction (from type-less legacy code)

Pseudo code from paper

$\text{arrayIndexEquiv} := \text{arrayIndex}^{-1} \circ \text{arrayIndex}$

$\text{typeEquiv} := \text{arrayIndexEquiv} \cup \text{expression}$

$\text{subtypeOf} := \text{assign}$

repeat

$\text{subtypeEquiv} := \text{equiv}(\text{subtypeOf} + \cap (\text{subtypeOf} +)^{-1})$

$\text{typeEquiv} := \text{equiv}(\text{typeEquiv} \cup \text{subtypeEquiv})$

$\text{subtypeOf} := \text{subtypeOf} \setminus \text{typeEquiv}$

$\text{subtypeOf} := \text{subtypeOf} \cup \text{subtypeOf} \circ \text{typeEquiv} \cup \text{typeEquiv} \circ \text{subtypeOf}$

until fixpoint of (typeEquiv, subtypeOf)

Type reconstruction (from type-less legacy code)

Data

```
type VariableGraph v array
  = (Rel v v, Rel v array, Rel v v)
```

```
type TypeGraph x
  = (Rel x x, Rel x x)  -- subtypes and type equiv
```

Operation

```
typeInference
  :: (Ord v, Ord array) =>
     VariableGraph v array -> TypeGraph v
```


See

- Christian Lindig. *Fast Concept Analysis*. In Gerhard Stumme, editors, *Working with Conceptual Structures - Contributions to ICCS 2000*, Shaker Verlag, Aachen, Germany, 2000.

Basic idea

1. Given formal context
 - matrix of objects vs. properties
2. Compute concept lattice
 - a concept = (extent,intent)
 - ordering is by sub/super set relation on intent/extent

Used in many fields, including program understanding.

Formal concept analysis pseudo-code (1/2)



Software Improvement Group

```
NEIGHBORS  $((G, M), (\mathcal{G}, \mathcal{M}, \mathcal{I}))$ 
1  min  $\leftarrow \mathcal{G} \setminus G$ 
2  neighbors  $\leftarrow \emptyset$ 
3  foreach  $g \in \mathcal{G} \setminus G$  do
4     $M_1 \leftarrow (G \cup \{g\})'$ 
5     $G_1 \leftarrow M_1'$ 
6    if  $((\text{min} \cap (G_1 \setminus G \setminus \{g\})) = \emptyset)$  then
7      neighbors  $\leftarrow$  neighbors  $\cup \{(G_1, M_1)\}$ 
8    else
9      min  $\leftarrow$  min  $\setminus \{g\}$ 
10 return neighbors
```

108 | 112

Note that $_'$ operation denotes computation of intent from extent, or vice versa, implicitly given a context.

Formal concept analysis pseudo-code (2/2)



Software Improvement Group

```
LATTICE ( $\mathcal{G}, \mathcal{M}, \mathcal{I}$ )
1   $c \leftarrow (\emptyset'', \emptyset')$ 
2  insert ( $c, L$ )
3  loop
4    foreach  $x$  in NEIGHBORS ( $c, (\mathcal{G}, \mathcal{M}, \mathcal{I})$ )
5      try  $x \leftarrow$  lookup ( $x, L$ )
6      with NotFound  $\rightarrow$  insert ( $x, L$ )
7       $x_* \leftarrow x_* \cup \{c\}$ 
8       $c^* \leftarrow c^* \cup \{x\}$ 
9      try  $c \leftarrow$  next ( $c, L$ )
10     with NotFound  $\rightarrow$  exit
11  return  $L$ 
```

109 | 112

Transposition to Haskell?

Representation

```
type Context g m = Rel g m
type Concept g m = (Set g, Set m)
type ConceptLattice g m
  = Rel (Concept g m) (Concept g m)
```

Algorithm

```
neighbors :: (Ord g, Ord m)
  => Set g           -- extent of concept
  -> Context g m    -- formal context
  -> [Concept g m]  -- list of neighbors

lattice :: (Ord g, Ord m)
  => Context g m     -- formal context
  -> ConceptLattice g m -- concept lattice
```

Repository mining

111 | 112

- Analyze relationships between code/commits/issues through time
- Clustering and decision trees
- Popularity index for programming languages, frameworks, ...

Quality and metrics

- Generalized method for derivation of quality profiles
- Metrics for architectural quality and evolution
- Quality models for BPEL, SAP, Sharepoint, ...

Large-scale source code analysis

- Detection of “events” in data streams
- Using the Amazon cloud
- Metrics warehouse

Massively parallel source code analysis

112 | 112

- Use Hadoop framework for Map-Reduce
- Put algebraic API on top
- Construct example analyses

Java library binary relational algebra

- Extend to “labeled” relations (matrix algebra)
- Extend with advanced algorithms (e.g. concept analysis)

Randomized testing for Java

- Study existing approaches
- Build / extend tool

More info? Feel free to contact...



Software Improvement Group

113 | 112

Dr. ir. Joost Visser

E: j.visser@sig.nl

W: www.sig.nl

T: +31 20 3140950