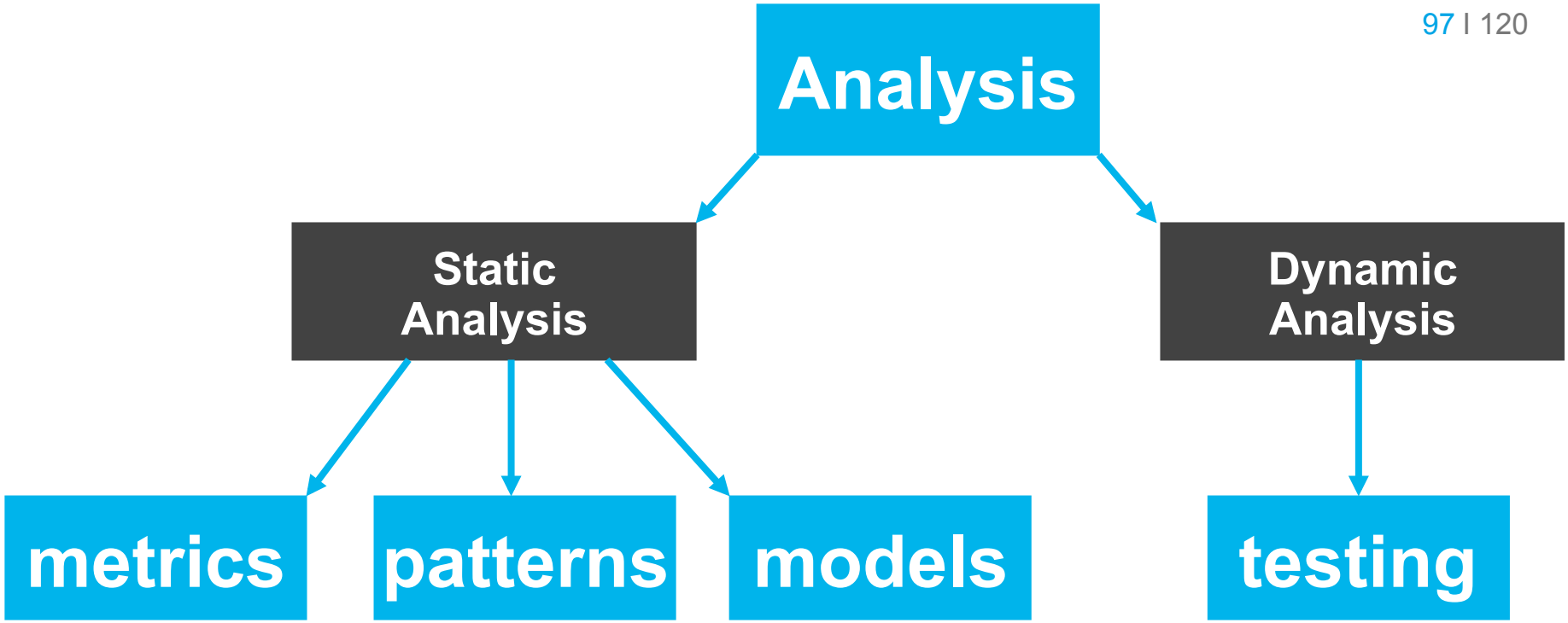
Structure of the lecture

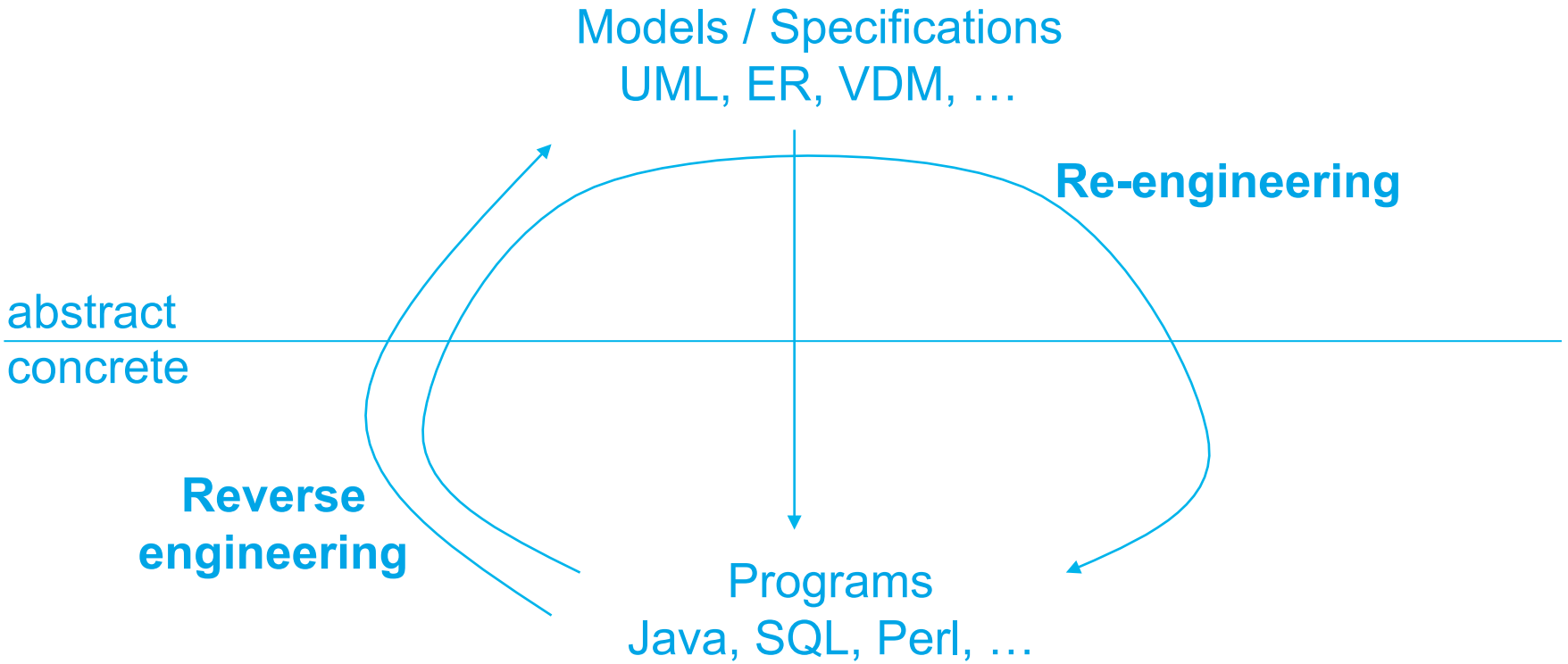
Software Improvement Group



Software Improvement Group

REVERSE ENGINEERING

Terminology



Reverse engineering

Dependencies and graphs

- Extraction, manipulation, presentation
- Graph metrics
- Slicing

Advanced

- Type reconstruction
- Concept analysis
- Programmatic join extraction

Reverse engineering trinity

Extraction

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

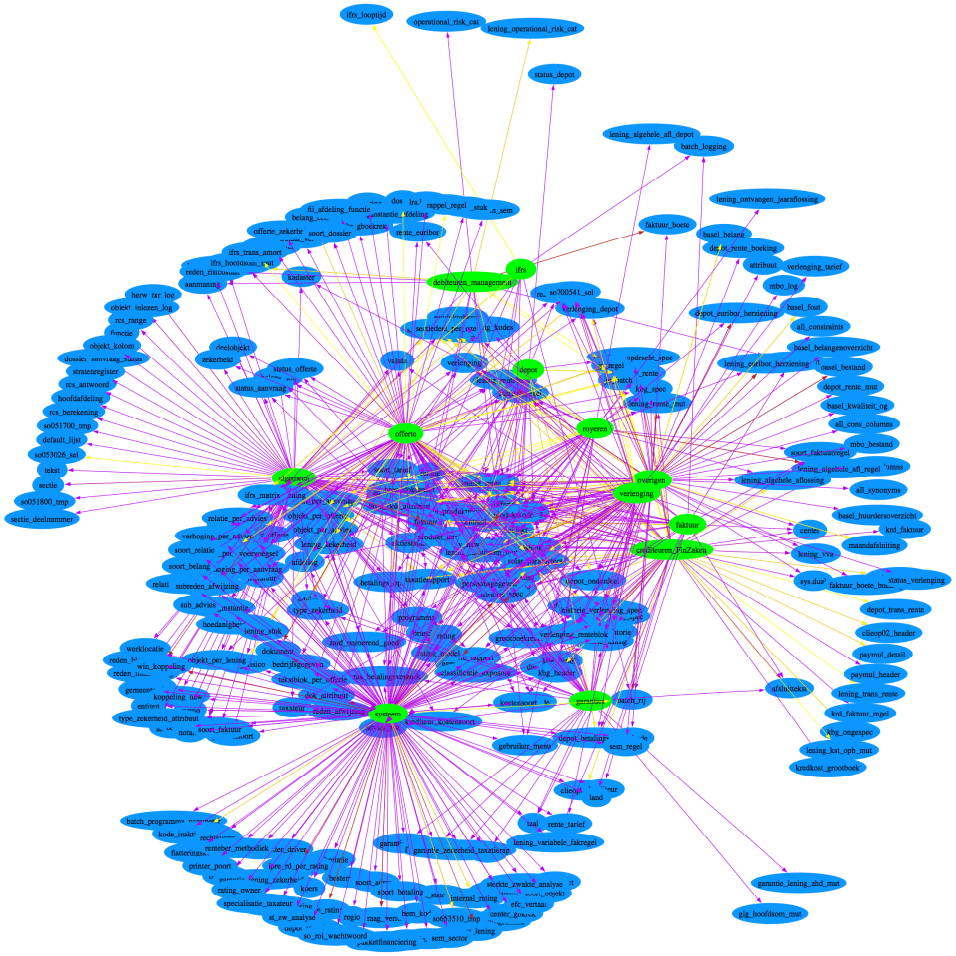
Manipulation

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

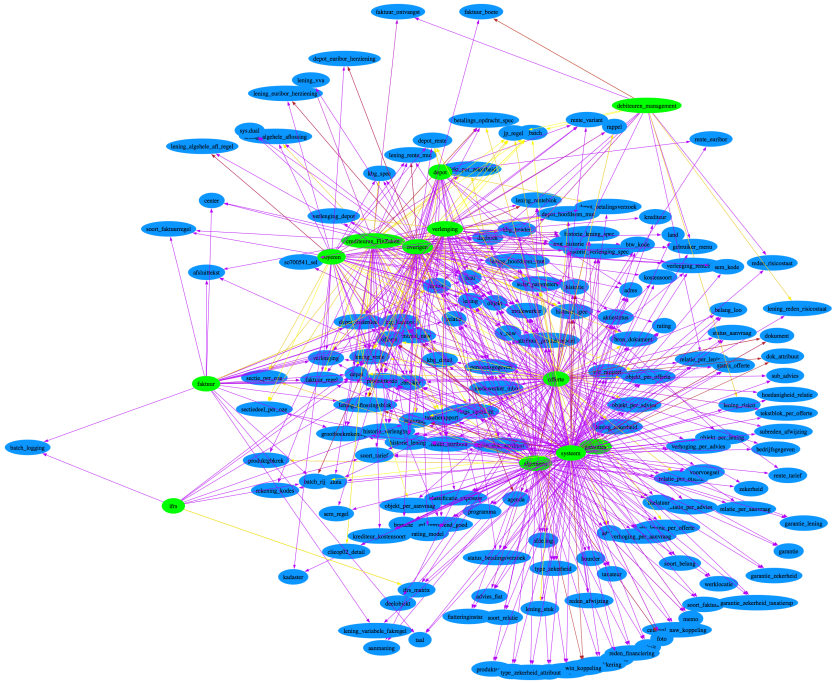
Purple arrow = select operation

Yellow arrow = insert/update operation

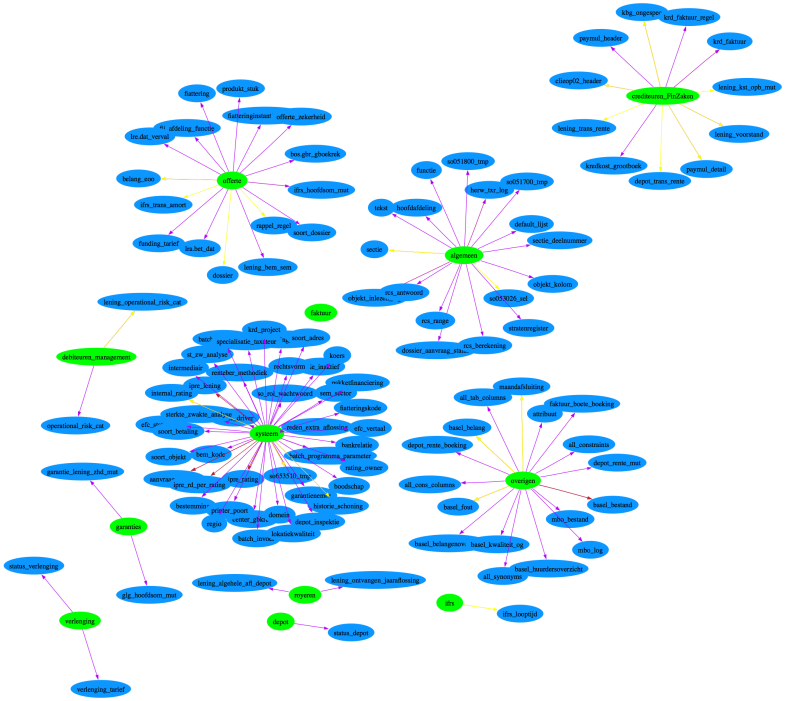
Brown arrow = delete operation

Example

Software Improvement Group



Tables used by multiple modules.



Tables used by a single module.

Relations and graphs

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
```

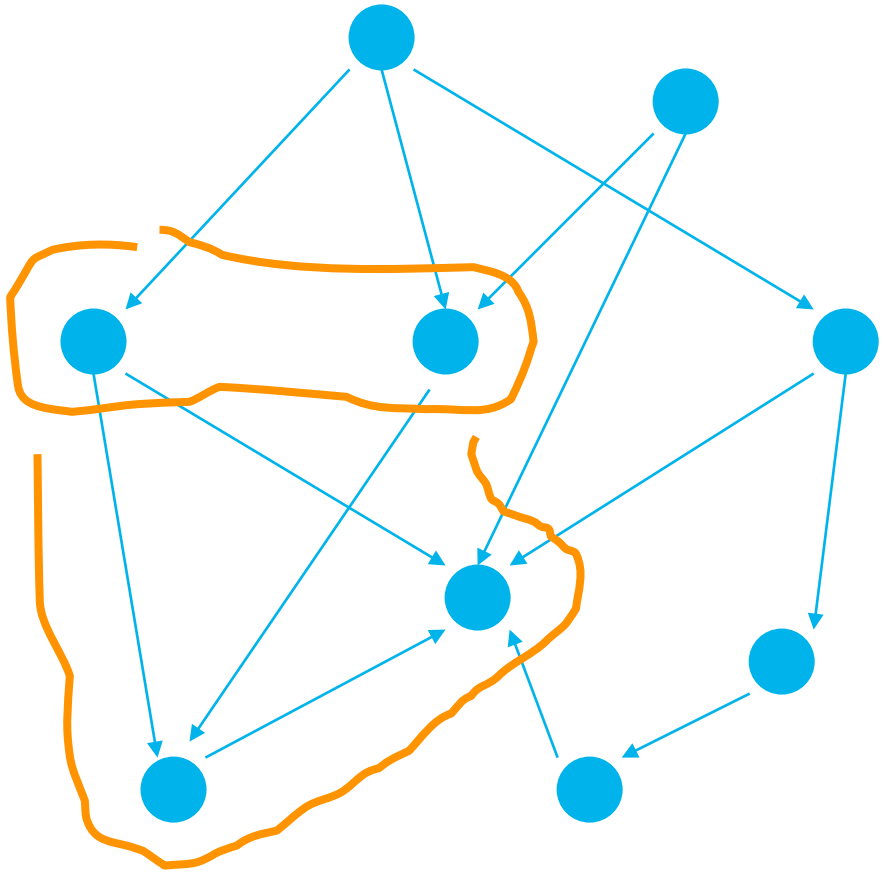
map from pairs

Note

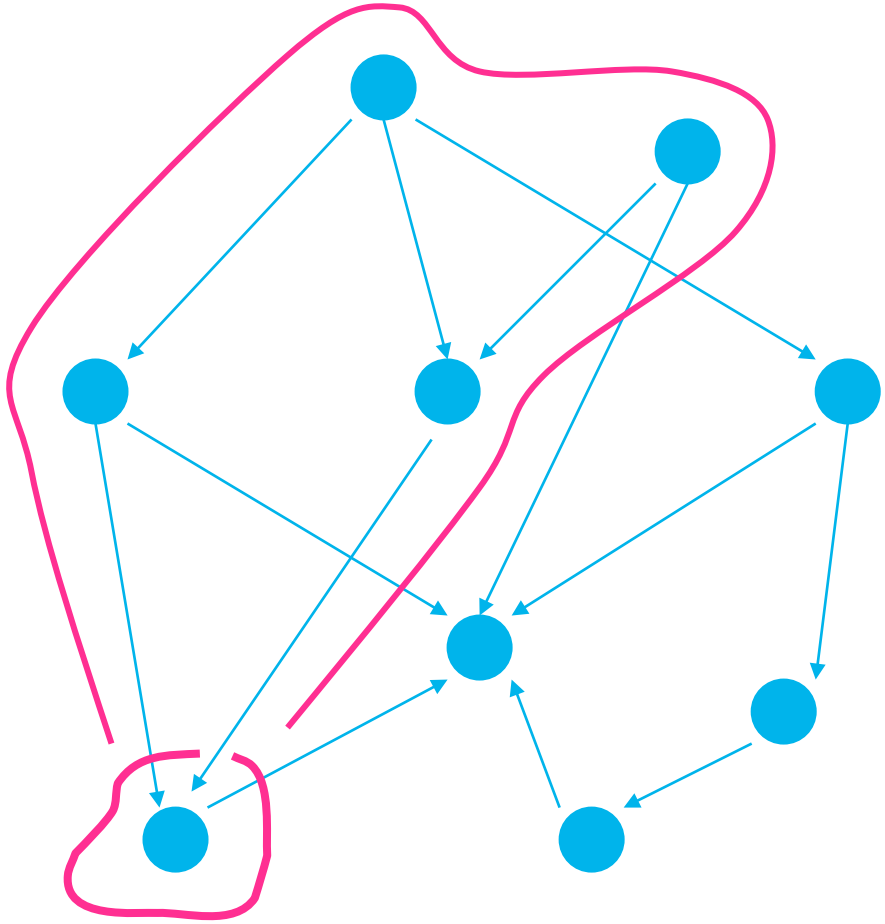
```
Rel a b = Set(a,b)= Map(a,b)()= LRel a b ()
```

Slicing (forward)

Software Improvement Group

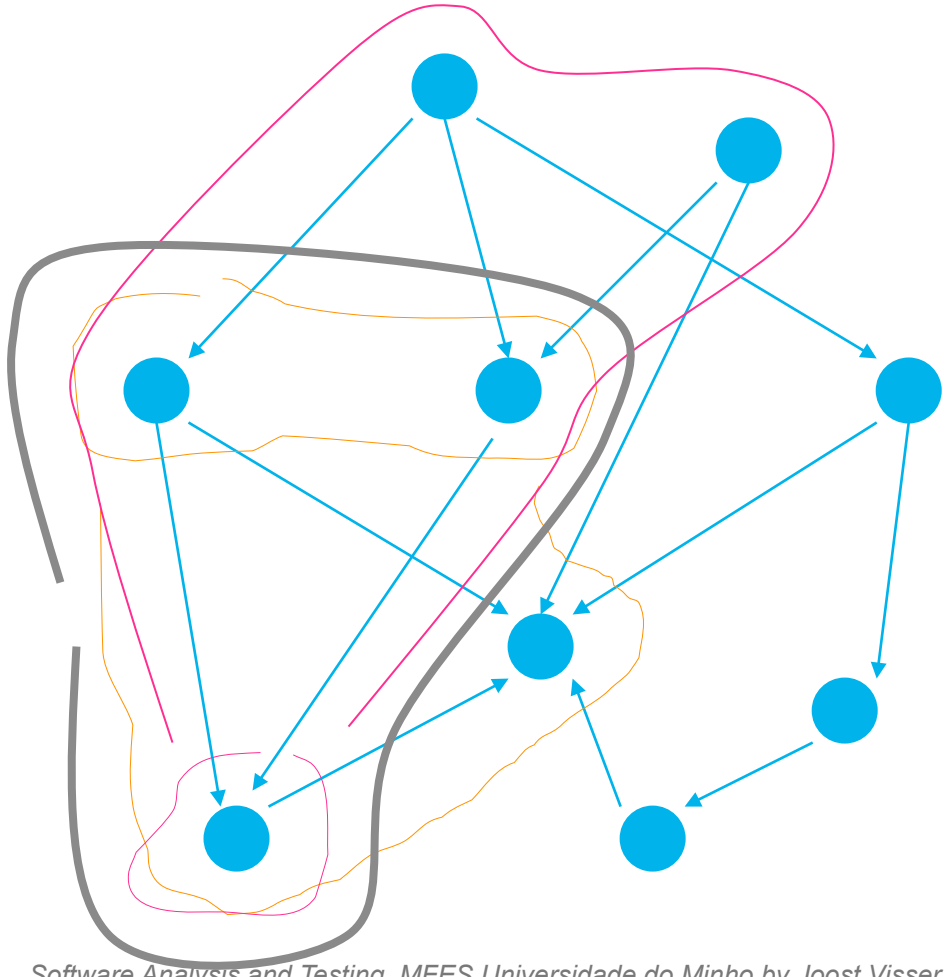


Slicing (backward)

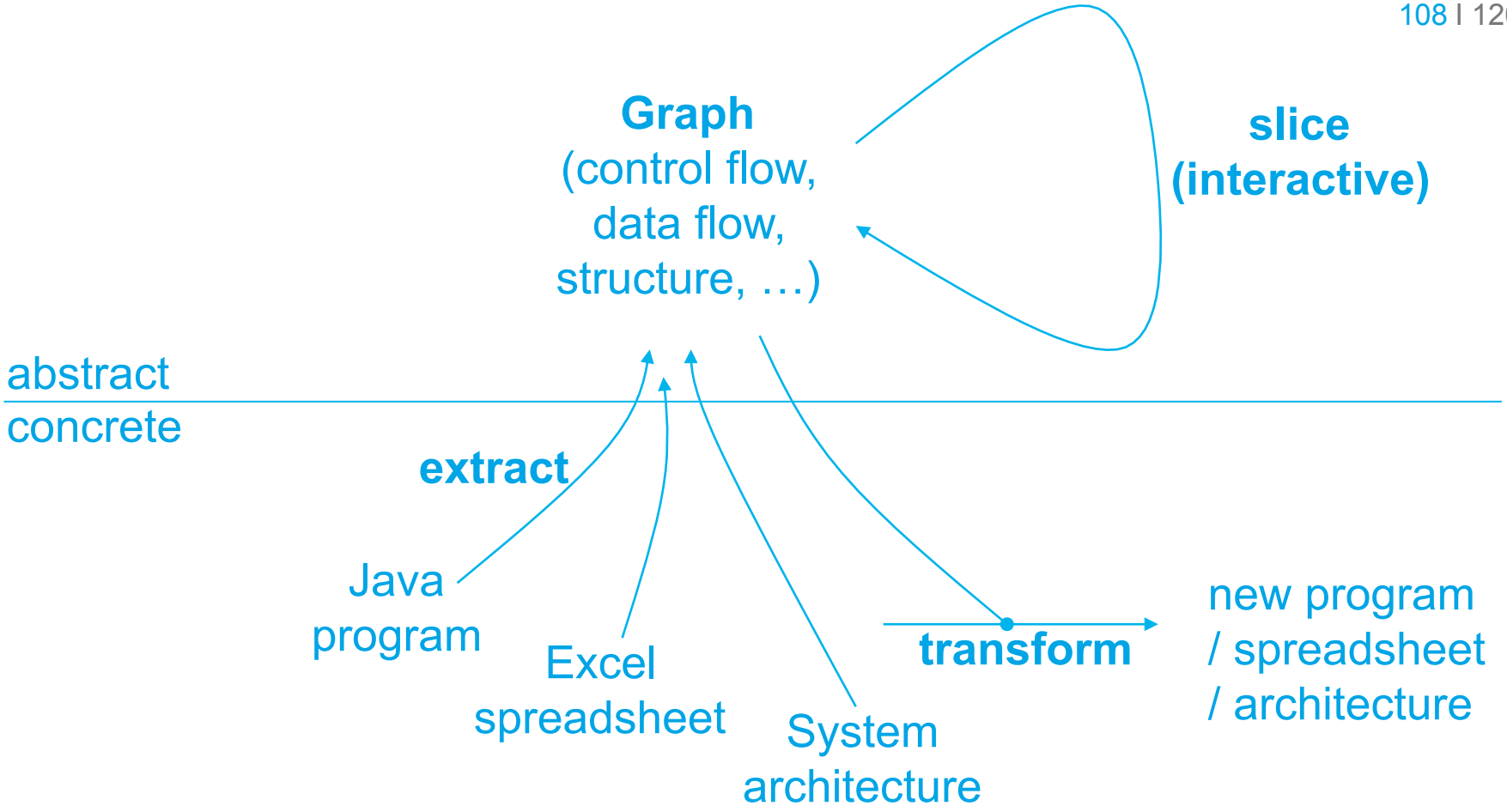


Chop
= Forward ∩ Backward

Software Improvement Group



Generic slicing



Further reading

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)

Software Improvement Group

Pseudo code from paper

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

$$typeEquiv := arrayIndexEquiv \cup expression$$

$$subtypeOf := assign$$

repeat

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

$$typeEquiv := equiv(typeEquiv \cup subtypeEquiv)$$

$$subtypeOf := subtypeOf \setminus typeEquiv$$

$$subtypeOf := subtypeOf \cup subtypeOf \circ typeEquiv \cup typeEquiv \circ subtypeOf$$

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
```

Formal concept analysis

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 ← 𝒢 \ G
2   neighbors ← ∅
3   foreach g ∈ 𝒢 \ G do
4      M1 ← (G ∪ {g})′
5      G1 ← M1′
6      if ((min ∩ (G1 \ G \ {g})) = ∅) then
7         neighbors ← neigbors ∪ {(G1, M1)}
8      else
9         min ← min \ {g}
10  return neighbors
```

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 (G, M, I)
 1  c ← (∅'', ∅')
 2  insert (c, L)
 3  loop
 4     foreach x in NEIGHBORS (c, (G, M, I))
 5        try x ← lookup (x, L)
 6        with NotFound → insert (x, L)
 7        x* ← x* ∪ {c}
 8        c* ← c* ∪ {x}
 9     try c ← next (c, L)
10     with NotFound → exit
11  return L
```

Transposition to Haskell?

Formal concept analysis

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
```

Ongoing projects at SIG

Quality and metrics

- Metrics for architectural quality and evolution
- Quality models for BPEL, SAP, Sharepoint, Spreadsheets,…
- Models for Reliability, Technical Debt

Repository mining

- Analyze relationships between code/commits/issues/tests through time
- Calibration and validation of quality indicators

Large-scale source code analysis

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

Possible MFES projects at SIG

Improvement of metrics event detection

- Evaluation of current event detection has revealed weaknesses
- Some improvements have been proposed
- Improvements must be implemented and evaluated

Analysis of OSS projects

- Select 5 high-profile OSS projects
- Analyse code, issues, tests, community structure
- Current status as well as evolution through time
- Assess each project and compare all projects

Metrics for architectural evolution

- Find different types of change (local, simultaneous, de-phased, structural, ..)
- Quantify in metrics
- Case study on two proprietary and two OSS systems

Dr. ir. Joost Visser
j.visser@sig.eu
http://twitter.com/jstvssr
www.sig.eu
+31 20 314 0950

Dr. ir. Joost Visser
j.visser@sig.eu
http://twitter.com/jstvssr
www.sig.eu
+31 20 314 0950