

Challenge Pure PURe

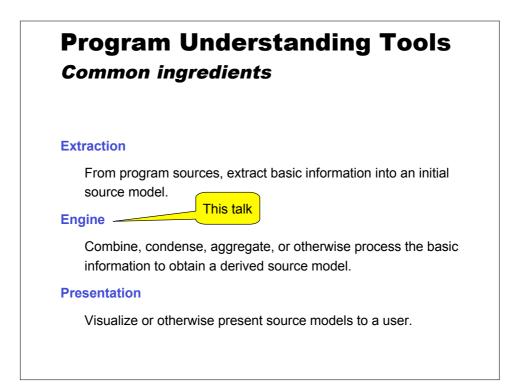
Challenge

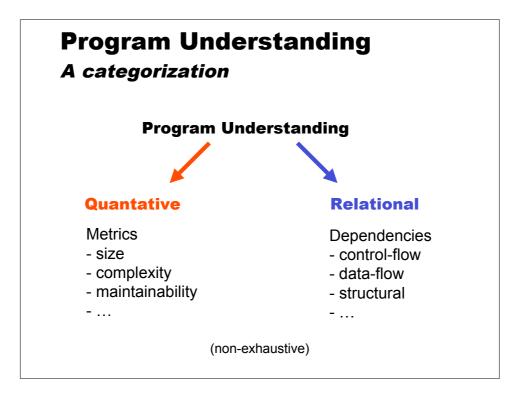
Develop 100% functional program understanding algorithms.

1

Questions

Is it possible? Is it practical? Is it useful?





Relational PU Algorithms *Overview*

Basis

Relation calculus.

Some algorithms

- Type reconstruction
- Slicing and chopping
- Formal concept analysis

Relation calculus Acpresentations & operations Relation type Rel a b = Set (a,b) set of pairs Labeled relation type LRel a b l = FM (a,b) l map from pairs Note Rel a b = Set (a,b) = FM (a,b) () = LRel a b ()

Type Reconstruction *From typeless 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 typeless legacy code pseudo code from paper* arrayIndexEquiv := arrayIndex⁻¹ ∘ arrayIndex typeEquiv := arrayIndexEquiv ∪ expression subtypeOf := assign *repeat* subtypeEquiv := equiv(subtypeOf + ∩ (subtypeOf +)⁻¹) typeEquiv := equiv(typeEquiv ∪ subtypeEquiv) subtypeOf := subtypeOf \ typeEquiv subtypeOf := subtypeOf \ typeEquiv subtypeOf := subtypeOf \ typeEquiv subtypeOf := subtypeOf \ subtypeOf ∘ typeEquiv ∪ subtypeOf until fixpoint of (typeEquiv, subtypeOf) Is directly transposed to Haskell, using Data.Relation.SetOfPairs. Online Demo (uses WASH and GraphViz)

Slicing and Chopping *Via graph reachability*

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.

Basic idea

- 1. Construct so-called Program Dependency Graph (PDG).
- 2. Apply general graph slicing algorithm.

Chop = intersection of forward and backward slice.

- Graph slicing/chopping/integration is *directly* transposed to Haskell, see Data.Relation.SetOfPairs.
- Note: these general algorithms can be applied to any kind of graph, not just PDGs.

Online Demo: chopping Java "package graphs".

Formal Concept Analysis A data analysis technique

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

- Given formal context

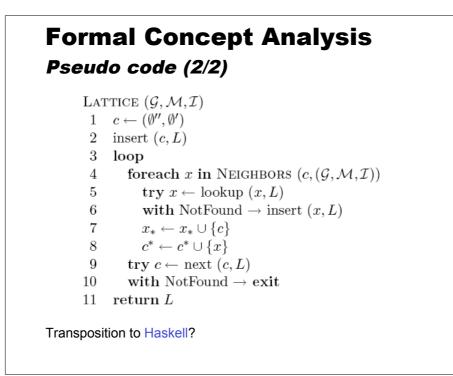
 matrix of objects vs. properties
- 2. Compute concept lattice - a concept = (extent,intent)
 - ordering is by sub/super set relation on intent/extent

Used in many fields, including program understanding.

Online Demo.

Formal Concept Analysis Pseudo code (1/2)

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 $M_1 \leftarrow (G \cup \{g\})'$ 4 $G_1 \leftarrow M'_1$ 5 $\mathbf{6}$ if $((\min \cap (G_1 \setminus G \setminus \{g\})) = \emptyset)$ then $\overline{7}$ neighbors \leftarrow neighbors $\cup \{(G_1, M_1)\}$ 8 else9 $\min \leftarrow \min \setminus \{g\}$ 10 return neighbors Note that _' operation denotes computation of intent from extent, or vice versa, implicitly given a context.



Formal Concept Analysis *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	1)

Algorithm

Given this representation, the transposition of pseudo code is straightforward.

Conclusions *Preliminary*

General

• Non-trivial program understanding techniques can be implemented straightforwardly in Haskell.

- Relation calculus is a convenient instrument here.
- Skipped over extraction, visualization, control issues. (Strafunski, GraphViz, WASH)
- Functional PU: possible!

Questions

- Performance?
- Interaction?
- Functional PU: practical? useful?