

53. *Interprocedural Abstract Interpretation with PAG*

Prof. Dr. rer. nat. Uwe Aßmann

Institut für Software- und
Multimediatechnik

Lehrstuhl Softwaretechnologie

Fakultät für Informatik

TU Dresden

<http://st.inf.tu-dresden.de>

Version 12-1.0, 12.01.13

1) Interprocedural analysis

2) Ab.I. with PAG

Obligatory Literature

- ▶ Alt, Martin, Martin, Florian, Generation of efficient interprocedural analyzers with PAG. In: Mycroft, Alan, Static Analysis. Lecture Notes in Computer Science, 1995. Springer Berlin / Heidelberg
" <http://www.springerlink.com/content/y583778583740462/>
- ▶ Martin, Florian. PAG – an efficient program analyzer generator. International Journal on Software Tools for Technology Transfer (STTT), Volume 2, Number 1, 46-67, DOI: 10.1007/s100090050017, Special section on program analysis tools
" <http://www.springerlink.com/content/1pb55yv4mq4emywl/>
- ▶ Auch Technischer Bericht der U Saarbrücken:
" <http://scidok.sulb.uni-saarland.de/volltexte/2004/203/>

Ressources

- ▶ F. Martin. PAG - an efficient program analyser generator. Software Tools for Technology Transfer STTT 1998, 2:46-67, Springer
- ▶ www.absint.de (also aiSee)
- ▶ www.cs.uni-sb.de/~martin/pag
- ▶ F. Martin Generating Program Analyzers. PhD Thesis. Universität Saarbrücken.
- ▶ Martin Trapp. Optimierung Objekt-Orientierter Programme. Springer Verlag, Heidelberg, January 2001.
- ▶ Ole Agesen, Jens Palsberg, and Michael I. Schwartzbach. Type inference of SELF. In Oscar Nierstrasz, editor, ECOOP'93-Object-Oriented Programming, 7th European Conference, volume 707 of Lecture Notes in Computer Science, pages 247-267, Kaiserslautern, Germany, 26-30 July 1993. Springer.

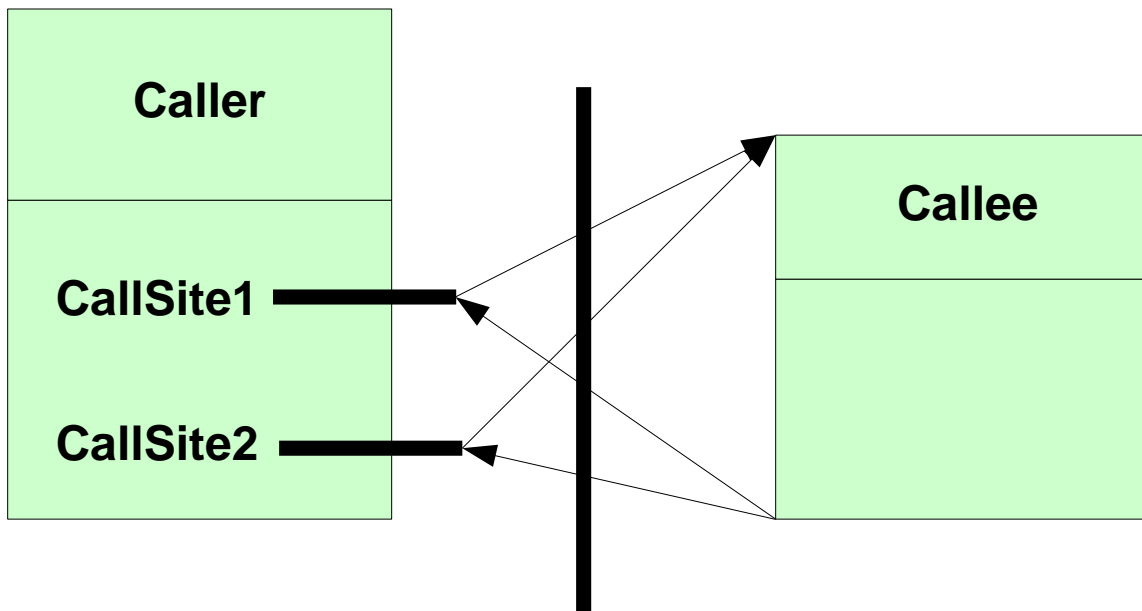


53.1 Different Approaches to Interprocedural Analysis

- Abstract interpreters can treat calls in different ways.
- From ignoring and summarizing them, to expanding them or lazily expanding them.

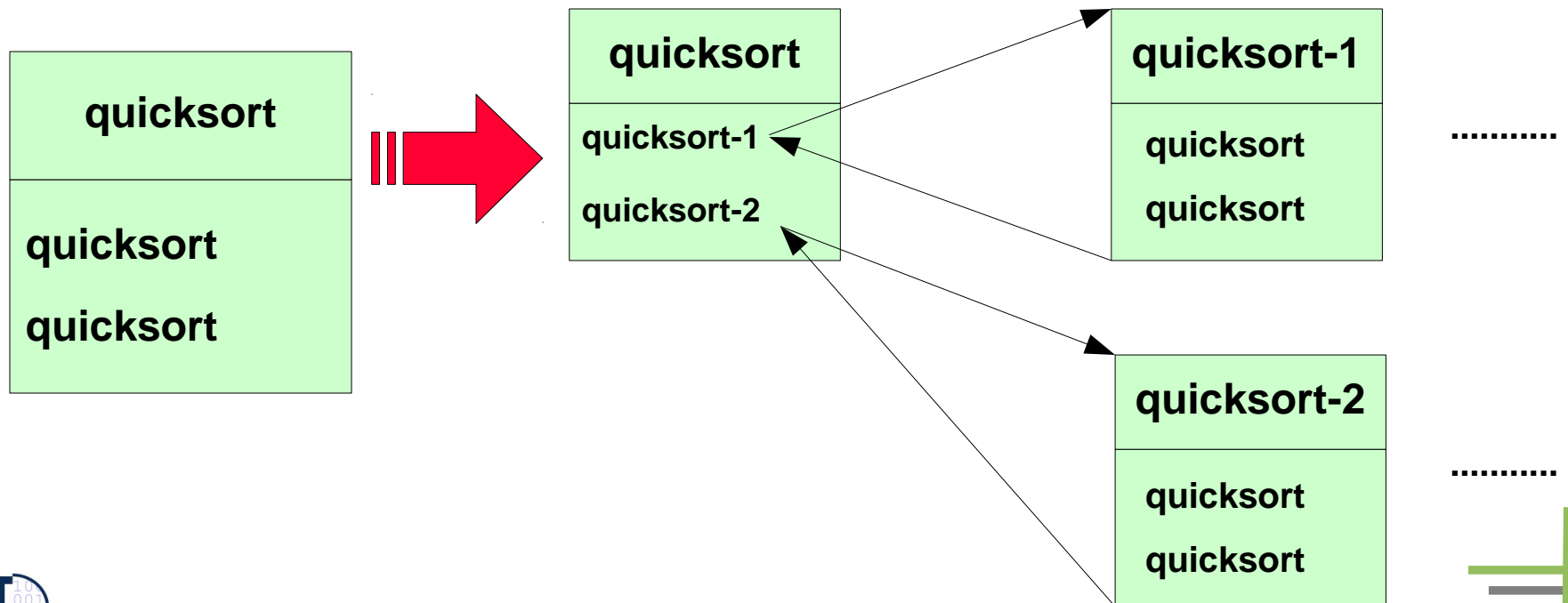
Invalidating Approach to Abstract Interpretation (Worst-Case Assumption)

- ▶ During the abstract interpretation, all information is invalidated by a call
 - " After the call, worst case value is assumed (top of lattice)
 - " Every procedure is analyzed in isolation
- ▶ Conservative (know nothing about calls)
- ▶ Improvement:
 - " Invalidate everything that might be written by the callee
 - " However then alias analysis must run before



The Cloning/Inlining Approach to Abstract Interpretation

- ▶ **Inlining abstract interpretation (interprocedural analysis)** copies a procedure's body for every call and propagate information separately in body (builds up a interprocedural control flow graph, ICFG)
- ▶ Corresponds to *inlining* into every callee
- ▶ Leads to bloat of code and analysis information
 - Is space-exponential in nesting depth of call graph



The Functional Approach to Abstract Interpretation

- ▶ Also called *effect calculation approach*
- ▶ **Functional interprocedural analysis** calculates a function/effect E_f for every procedure f
 - " Which is applied to the current input values at a caller to receive the output values after the call
 - Parametric execution with an "abstract" function E_f
- ▶ E_f is stored in a table, mapping abstract input value to abstract output value (i.e., an associative array of abstract values)
- ▶ Whenever the analysis reaches the callee, the current abstract input value is looked up
 - " If found, reuse output value
 - " Otherwise reanalyze body

The *k*-Call Context Approach to Abstract Interpretation

- ▶ The **k-contextual interprocedural analysis** maintains the calling context with a limited stack of depth k
 - Also called *k-call string approach*
 - The call history of the called procedure is incorporated in the underlying lattice D (*call strings*)
- ▶ Different bodies at different call sites are distinguished by the call strings
 - In case of $k=1$ all call sites are distinguished
 - $K=2$ all call sites, with calling context of callers
 - $K=3$: all call sites, all calling contexts of the grandfathers
 - ...

Expanded Supergraphs

- ▶ The analysis information (the abstract values) are replicated for every caller (multiplicity)
 - Procedures are not inlined, but parameter information is replicated
- ▶ Connectors connect the right incarnation of the value to a caller site
- ▶ Example
- ▶ $\text{mult}(n) = 1 \implies$ no call sites are distinguished
- ▶ $\text{mult}(P_i) = k_i$ where i is number of call sites \implies call string length 1
- ▶ $\text{mult}(P_i) = k_i * n \implies$ call string length n

The Lazy Cloning Approach to Abstract Interpretation

- ▶ [Agesen: Type inference for SELF]
- ▶ Idea: do a *lazy cloning (on-demand replication)* of the parameter values
- ▶ During propagation, store all input values of functions analyzed so far
- ▶ If an input value for a function differs from an already memoized one, clone the parameter (i.e., distinguish it)
- ▶ Cloning parameters only
- ▶ Cloning them on demand
- ▶ Cloning can be restricted
 - " Analysis works less precise but costs less memory

The Interprocedural Phi-Approach to Abstract Interpretation

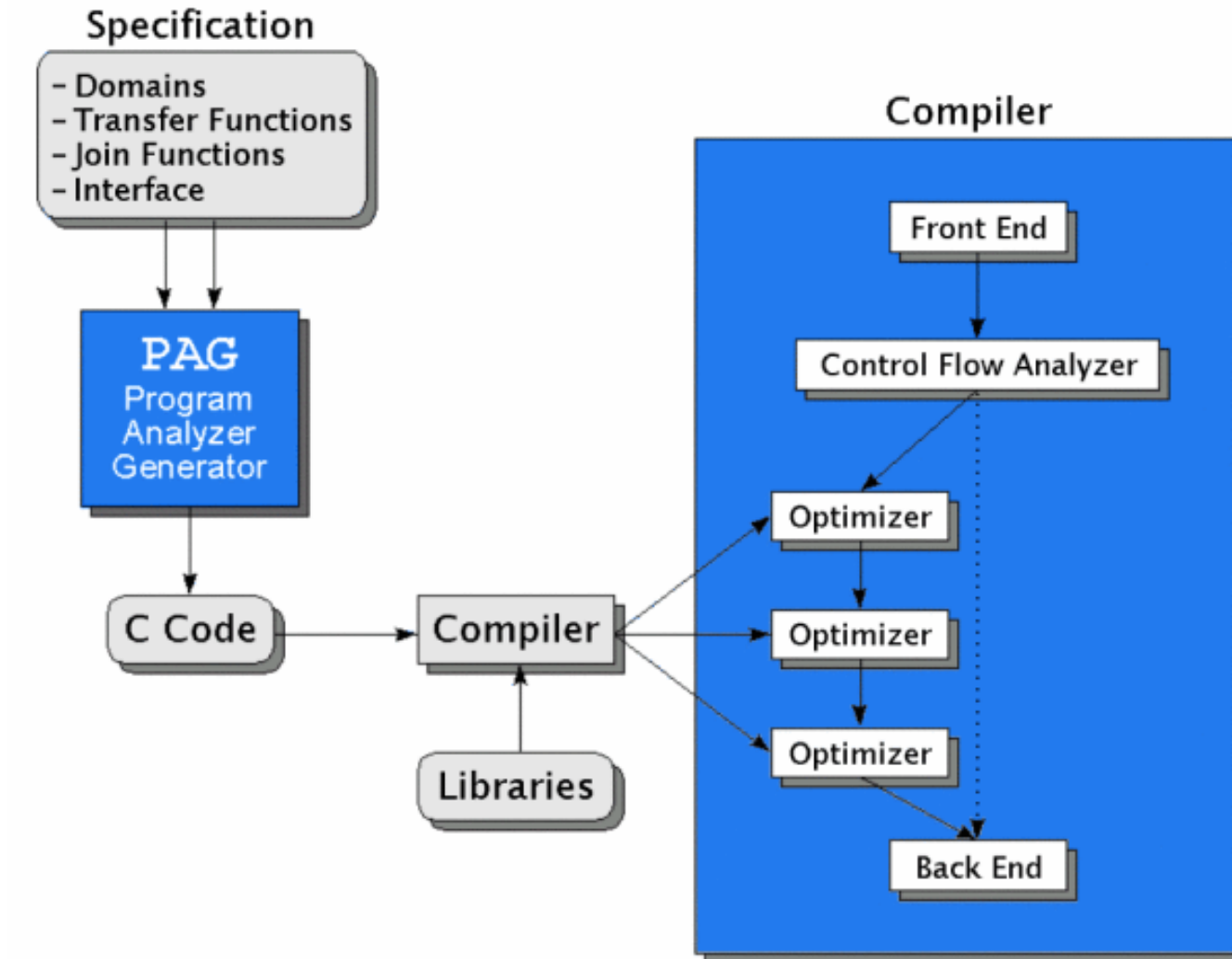
- ▶ M. Trapp (Optimization of object oriented programs) introduces interprocedural phi functions (i-phi)
- ▶ i-phs are "small-ifs" or "ifs for one value"
- ▶ Every formal parameter of a procedure gets as input an i-phi
- ▶ The i-phi depends on the control flow condition

53.2 Interprocedural Analysis with PAG



- ▶ Intra- and interprocedural analysis
- ▶ Extended super graph for interprocedural case (cloning of parameter information for call sites)
- ▶ Special Languages for:
 - " DDL for the specification of the intermediate program representation
 - " DDL for the Lattice (abstract domains)
 - " Functional language for the abstract interpreter (abstract/flow/transfer functions)

Generated Analyzer in Compiler



Node Orderings for Visits during Abstract Interpretation

- ▶ During the interprocedural abstract interpretation, instruction nodes are ordered in the worklist. Different orderings are possible, for which PAG can generate implementations:
- ▶ DFS: depth-first
- ▶ BFS: breadth-first
- ▶ SCC-D: strongly connected components in visit order depth first.
- ▶ SCC-B: same in breadth first
- ▶ WTO-D: SCCs, but ordered in weak topological ordering of Bourdoncle. Depth-first.
- ▶ WTO-B: same, but breadth-first

PAG-DDL: Data Type Specifications

- ▶ Basic sets
 - " Snum (signed numbers), unum, real, chr, string
- ▶ Basic Lattices
 - " Lsnum (lattice of signed numbers), lunum, bool, a..b, enum
- ▶ Type constructors for lattices
 - " Disjoint sum
 - " Tuple construction *
 - " Powerset operator
 - " List operator
 - " Function on $S1 \rightarrow S2$

PAG-DDL: Lattice Specifications

- ▶ Lattice operators
 - flat: Set $S \rightarrow$ Lattice
 - lift(Lattice L)
 - powerset: Set \rightarrow Lattice
 - dual(Lattice L)
 - reduce(Lattice E, reduction function f)
- ▶ Tuple space
- ▶ Function space (function lattice) $S \rightarrow L$, pointwise ordering

Example: PAG-DDL for Live Variables Analysis

```
// a simple powerset lattice for signed numbers
GLOBAL
    maxvar: snum
SET
    vars = [0..maxvar]
LATTICE
    varset = set(vars)
    var = lift(varset)
```

Example: PAG-DDL for Caches

GLOBAL

storeMin: unum

storeMax: unum

cacheSize: unum

aWays: unum<24

SET

storeLine = [storeMin..storeMax]

direct= [0..cacheSize]

LATTICE

cacheLine=[0..aWays]

age = lift(cacheLine)

assoc = storeLine -> age

cache = direct -> assoc

dfi = cache * cache

Example PAG-DDL for Intervals as Abstract Domain

LATTICE

```
upperBound = lsnum
```

```
lowerBound = dual(lsnum)
```

```
interv = lowerBound *upperBound
```

```
env = snum -> interv    // variables to intervals
```

```
dom = lift(env)
```

Example PAG-DDL for Heap Analysis

LATTICE

```
node = set(snum)           // nodes abstract vars
edge = node * snum * node
edges = set(edge)
sedge = snum * node
sedges = set(sedge)
shared = set(node)        // predicate
graph = sedges * edges * shared
dfi = lift(graph)
```

PAG-DDL: Specification of Program Representation (Metamodel of the Language)

- ▶ Types of the nodes of the CFG can be specified.
 - " Constructor based
 - " With alternatives
- ▶ In general, other DDLs can be employed (e.g., UML)

SYNTAX

```
START: Unlabstat
Unlabstat: M_Assign(var:Var, exp:Exp)
           | M_While(exp:Exp, body:Stat*)
           ...
```

Specification of Abstract Interpretation Functions

- ▶ Similar to function specification in ML
- ▶ Pattern matching on IR nodes
- ▶ Functions are annotated to control flow graph nodes
 - " Implicit parameter @ for data flow value
 - " Return a value
- ▶ Dynamic Functions (updatable)
 - " Application $f(\{!x!\})$
 - " Updating of values $f[n \rightarrow v]$
 - " Constant function $[- \rightarrow v]$

Specification of Abstract Interpretation Functions

- ▶ Lattices provide combine functions (merge, joins) for abstract values, when control flow joins
 - least-upper bound lub
 - greatest-lower bound glb
 - comparison relation $<$, $>$
- ▶ Operations for latticed and lifted lattices
 - " drop, lift
- ▶ ZF Zermelo-Fränkel Set Expressions:
- ▶ $[x \text{ !! } x \leftarrow \text{set, if } x \geq 0]$

Example: Analysis of a While Loop

```
// Source code expression:
// while(id <=exp)
// 1) pattern matching of the expression
M_While(M_Binop(M_op_leq(),
               M_Var_exp(M_simpl_var(id)),
               exp),_)
      ,true_edge):
// 2) the abstract interpretation function
let f <= @; // assignment of f to implicit data flow value
    id = val-Identifier(id);
in
    let erg = f{!id!} glb (top,(eval(exp,f))!2);
    in if is_ok(erg) then lift(f\[id->erg])=
        else bot;
    endif;
```

Other Parts of the Specification

- ▶ Direction specification: forward/backward
- ▶ Carrier graph: control-flow graph
- ▶ Init value: default initialization of values
- ▶ Init_start: init value of start node
- ▶ Equal: equality test for fixpoint detection
- ▶ Widening function
- ▶ Narrowing function

Example

PROBLEM interval

direction: forward

carrier: dom

init_start: lift([->(dual(0),0)])

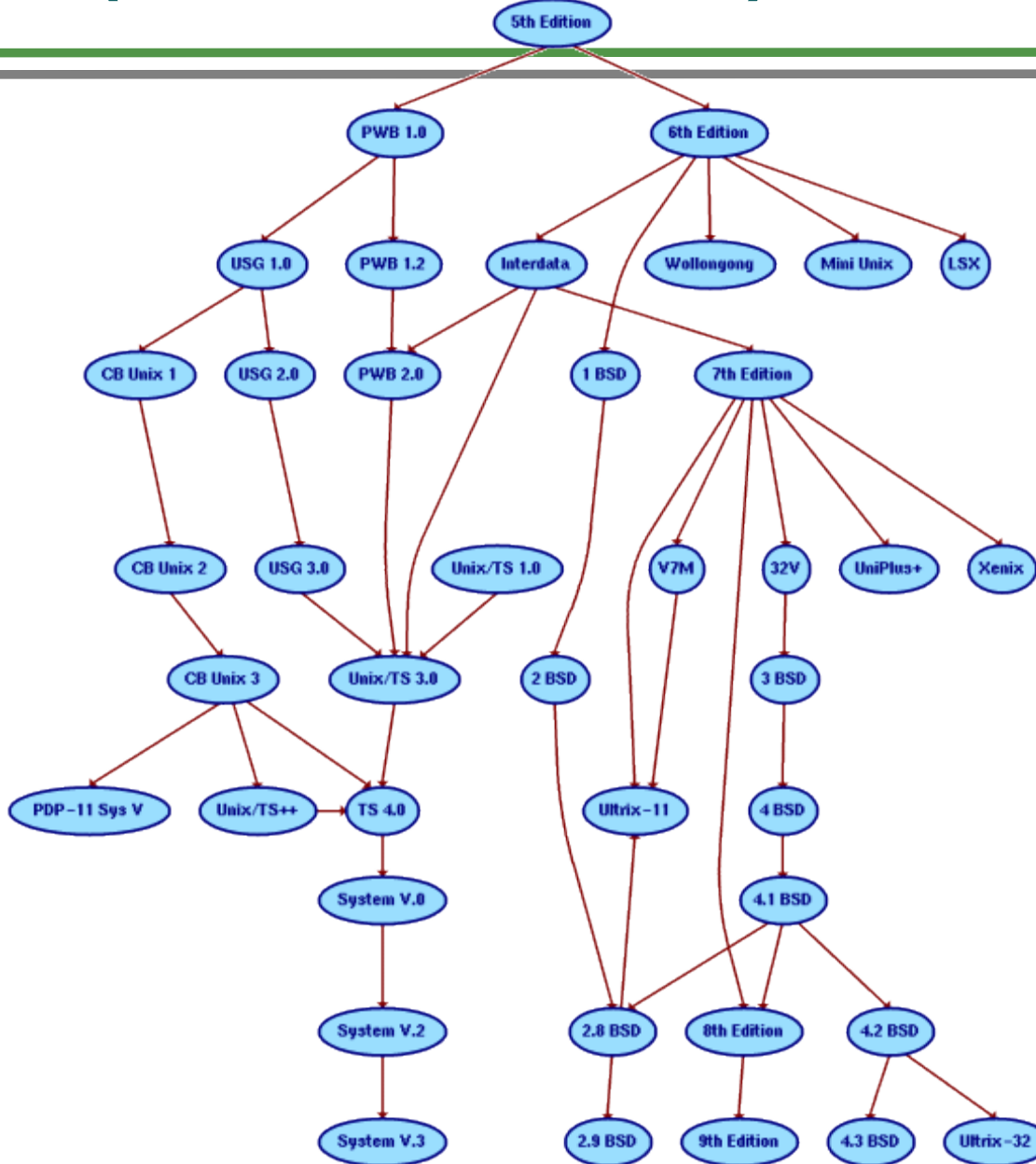
widening: wide

narrowing: narrow

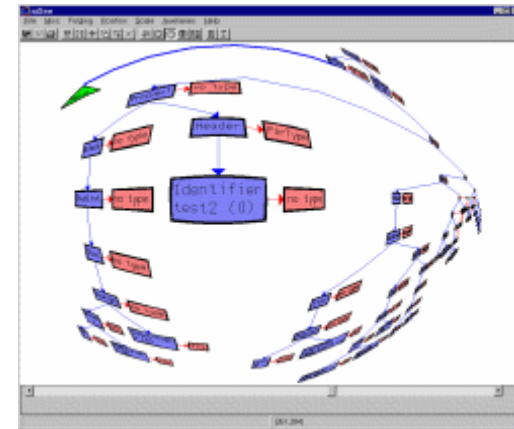
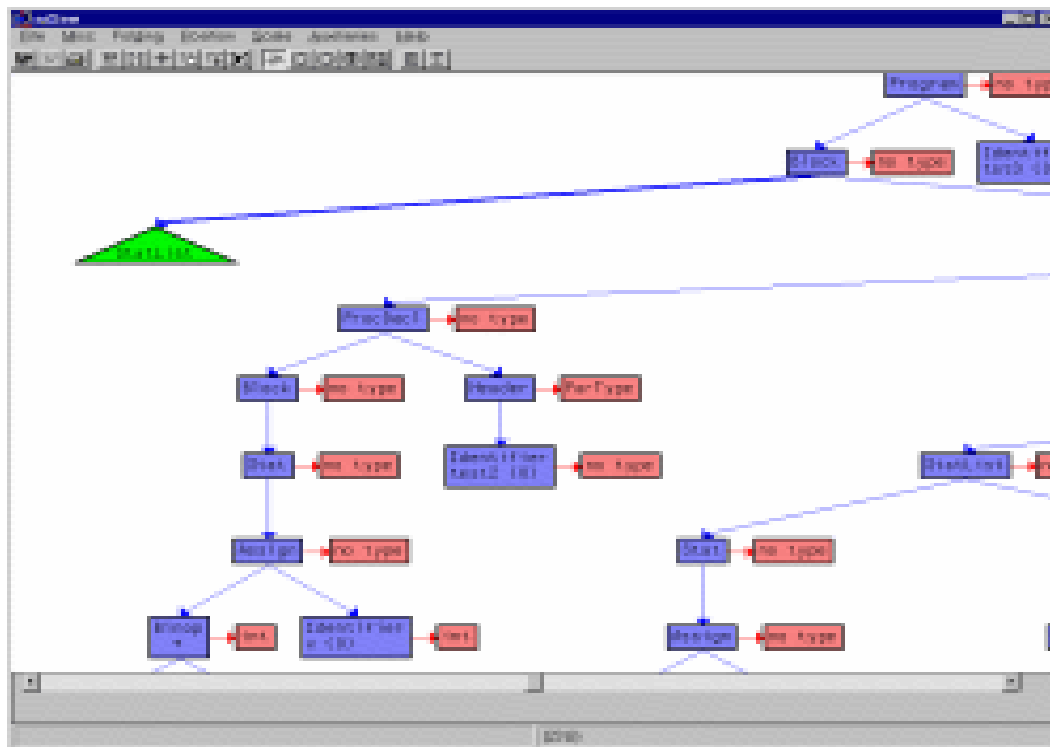
Debugging Specifications

- ▶ Export to VCG file format (or aiSee)
- ▶ Many visualizations possible
- ▶ Specific ones for flow graphs
 - " Lattice values annotated without edges to the nodes or edges of the flow graph
 - " Zoom in/out
 - " Hiding relations
 - " Blocks of nodes as regions with different color

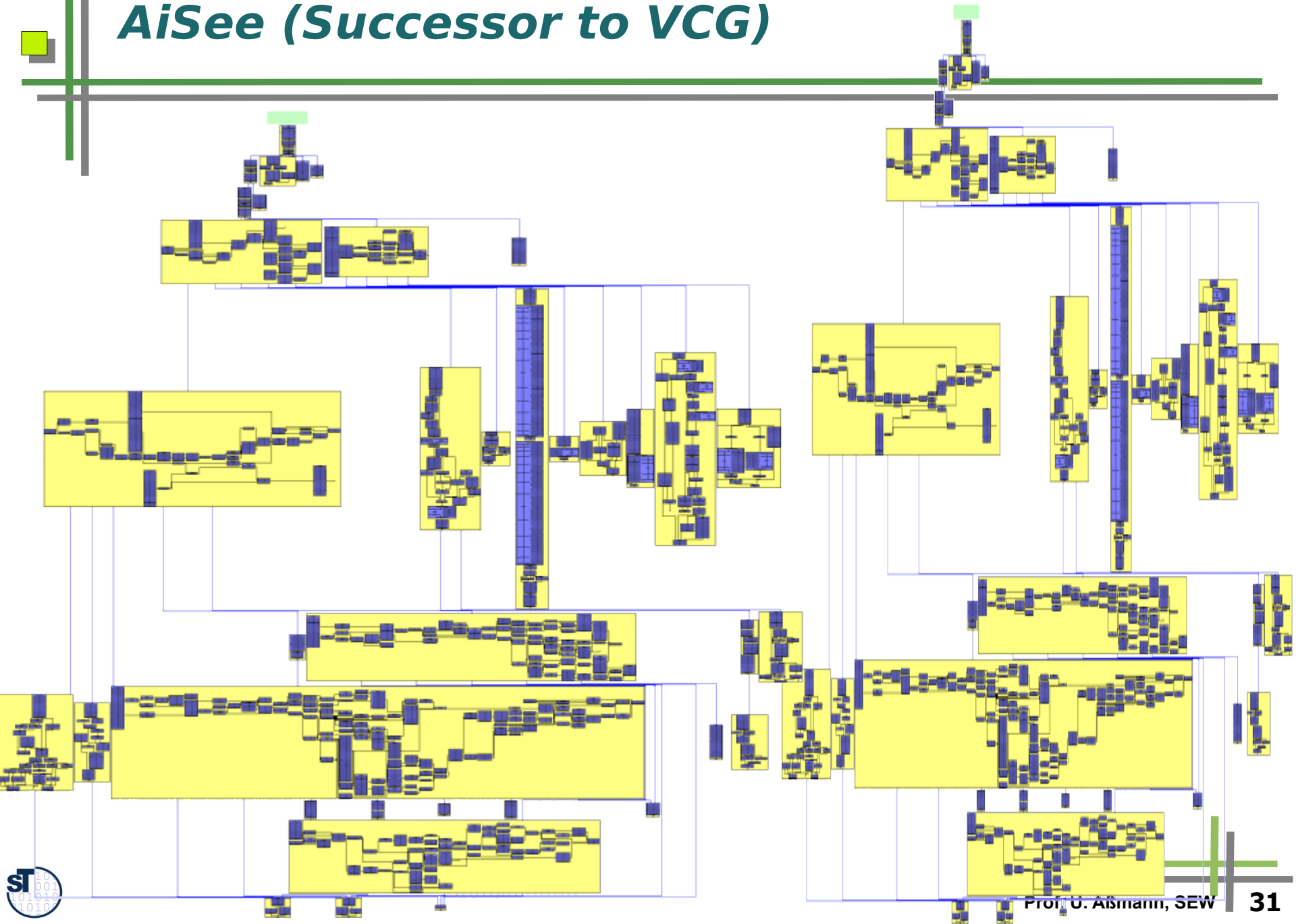
AiSee (Successor to VCG)



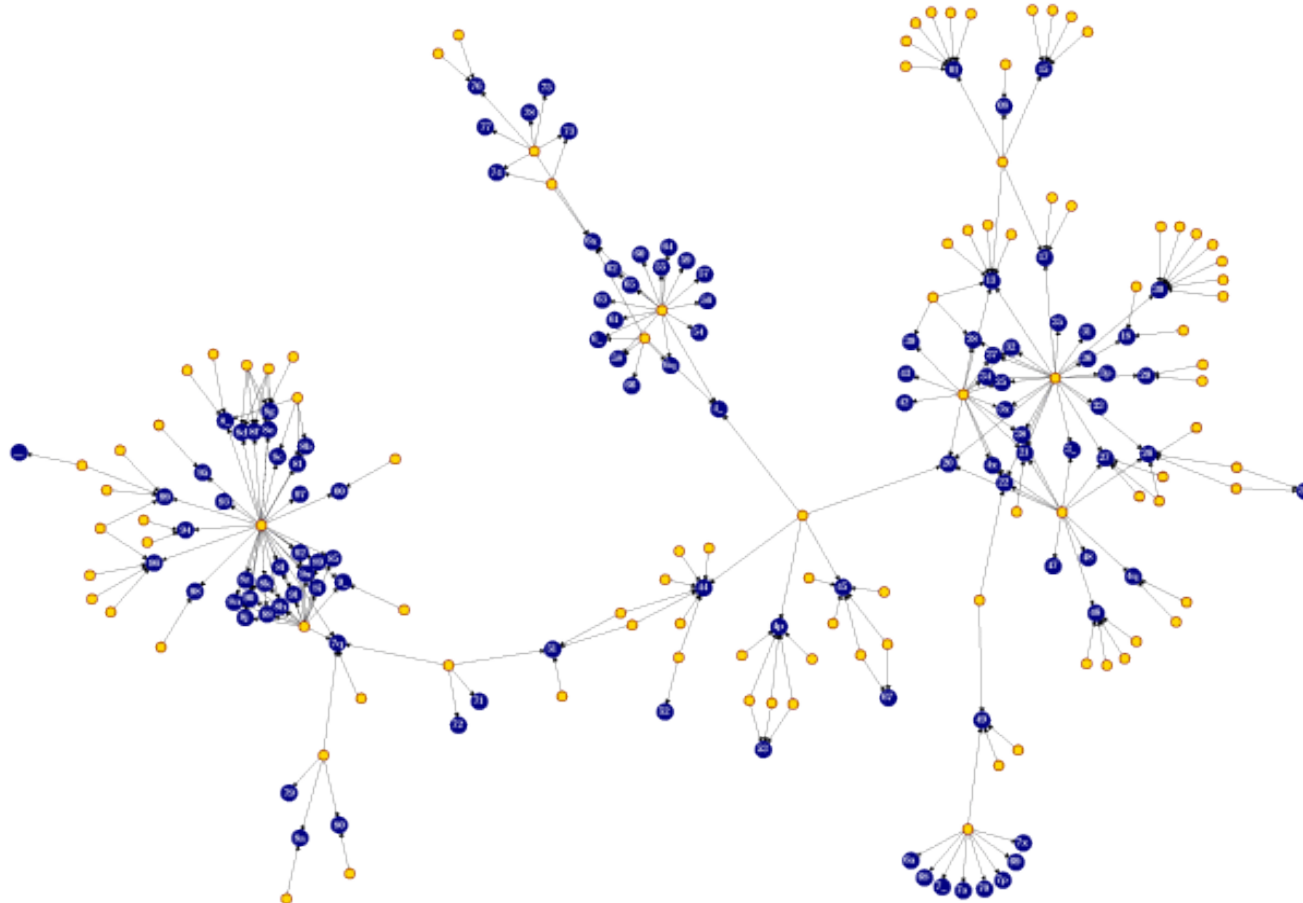
AiSee (Successor to VCG)



AiSee (Successor to VCG)



AiSee (Successor to VCG)



What have we learned?

- ▶ Interprocedural analysis can be done in several ways, spending different amount of resources, trading precision
- ▶ PAG is a tool to generate interprocedural analyzers
 - offering a specification language for lattices of abstract values
 - industrial strength
 - useful to specify many analyses, such as
 - classical data-flow analysis
 - cache analysis
 - heap analysis
 - alias analysis



The End