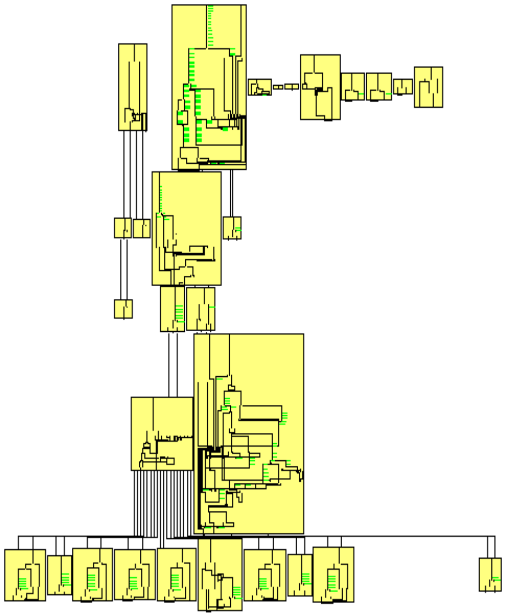
SSA FORM W.R.T. POINTER ANALYSIS PRECISION



SSA Form Seminar

France

April 27 to 30, 2009

Dr. Markus Schordan

Deputy Program Director of Game Engineering

UAS Technikum Wien

Overview

- Computation of program information with SATIrE
  - Flow-sensitivity and context-sensitivity
  - Points-to analysis
  - Shape analysis
  - program annotations for making analysis results persistent
- Representation in SSA Form
  - Memory regions and indirections
  - SSA Form for representing analysis results
  - Code pattern detection

References

- **The Language of the Visitor Design Pattern**
  Markus Schordan
  Journal of Universal Computer Science (JUCS), Vol. 12, No. 7, pp. 849-867, August 2006.
  Special Issue: Selected Papers from The 10th Brazilian Symposium on Programming Languages. Issue edited by Mariza Andrade Silva Bigonha and Alex de Vasconcellos Garcia.

- **Source Code based Component Recognition in Software Stacks for Embedded Systems**
  Dietmar Schreiner, Markus Schordan, Gergo Barany, Karl Göschka.
  In Proceedings of the 4th ASME/IEEE International Conference of Mechatronic and Embedded Systems and Applications (MESA 2008), pp. 463-468, ISBN: 978-1-4244-2368-2, Beijing, China, Oct 12-15, 2008.

SATIrE: Static Analysis Tool Integration Engine

Activities - Projects

- ALL-TIMES
  - 7. EU FP
  - Dec 2007- Feb 2010
  - European timing analysis integration
  - Partners: MDH, TU Vienna, AbsInt, Rapita, Symtavision, Gliwa
- CoSTA (timing analysis)
  - FWF, National (Austria)
  - Jul 2006 – Dec 2009
- ARTIST2
  - 6. EU FP
  - Sep 2004- Sep 2008

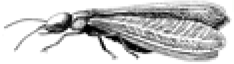
SATIrE People

SATIrE Developers

Staff: Markus Schordan, Gergö Barany, Adrian Prantl, Dietmar Schreiner, Florian Brandner, Dietmar Ebner
Students: Viktor Pavlu, Mihai Ghete, Christoph Roschger, Christoph Bonitz, Günther Khyo, Christian Biesinger

Integrated Tools - Initiators

LLNL-ROSE: Dan Quinlan (LLNL,CA,USA)
PAG: Florian Martin (AbsInt)
Termite: Adrian Prantl (TU Vienna)
Clang: LLVM/Apple Community



termite

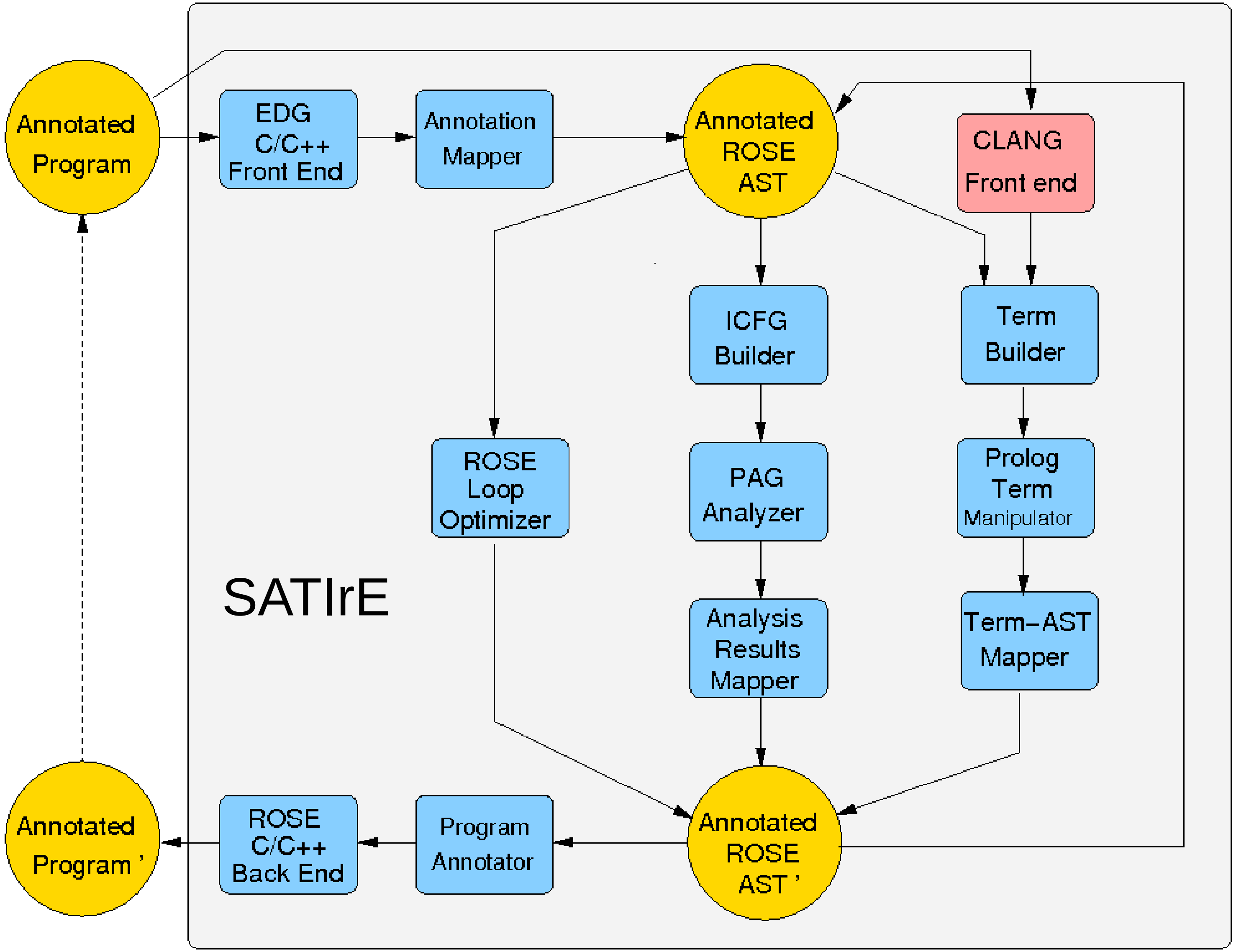
SATIrE-Based Tools – Initiators

TuBound: Adrian Prantl (TU Vienna)

SATIrE Download

http://www.complang.tuwien.ac.at/satire

SATIrE: Static Analysis Tools Integration Engine



SATIrE Analyses

Analysis Name	Implementation Language	Input	Flow Sensitive	Context Sensitive
„classic analyses“ (RD, AE, LV, CP)	FULA (PAG)	ICFG	Yes	Yes
Shape	FULA (PAG)	ICFG	yes	Yes
Points-to	C++	AST	No	Yes
Type-Based Alias	C++	AST	No	No
Interval	FULA (PAG)	ICFG	Yes	Yes
Loop-Bound	Prolog (+Constraints)	Interval	No	No

PAG – Analysis Specification

```
PROBLEM Reaching_Definitions
  direction:  forward
  carrier:    VarLabPairSetLifted
  init:       bot
  init_start: lift({})
  combine:    comb
  retfunc:    comb
  widening:   wide
  equal:      eq
```

```
SUPPORT
  comb(a.b) = a lub b;
  wide(a,b) = b;
  eq(a,b)   = (a=b);
```

Domain specific language

```
TRANSFER
...
ExprStatement(exprstmt), _:
  sl1_assignment(exprstmt,label,@);
```

PAG – Analysis Specification

```
/* handling SL1 assignments in analysis */
sl1_assignment::Expression,snum,VarLabPairSetLifted ->VarLabPairSetLifted;
sl1_assignment(exp,lab,bot)          = bot;
sl1_assignment(exp,lab,top)          = top;
sl1_assignment(exp,lab,infoLifted) =
let info <= infoLifted; in
case exp of
  /* one variable on each side of assignment */
  AssignOp(VarRefExp(cvarname1) as VarRef1,
           VarRefExp(cvarname2) as VarRef2)
     =>
      let x = varref_varid(varRef1); in
      lift(update_info(x,lab,info))  /* program variable */
      ;
endcase;

/* update the analysis information with kill and gen functions */
update_info::str,snum,VarLabPairSet -> VarLabPairSet;
update_info(x,lab,info) = union(rdkill(x,info),rdgen(x,lab));

/* kill variable */
rdkill::str,VarLabPairSet -> VarLabPairSet;
rdkill(var,varset) = { (var1,lab1) | (var1,lab1) <-- varset,
                       if var1 != var };
```

Matching

Sets

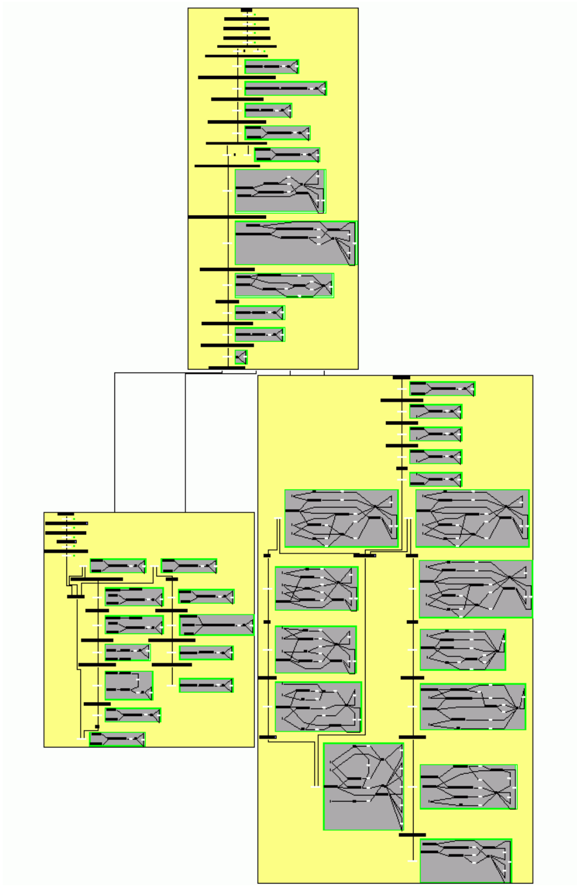
Overview of Pointer Analyses

Flow Insensitive	equality-based, subset-based
Flow Sensitive	strong/weak update
Context Insensitive	callstring length = 0
Context Senstitive	callstring length > 0
Heap Modeling	allocation site, shape, l-bounding, ...
Aggregate Modeling	elements distinguished or collapsed

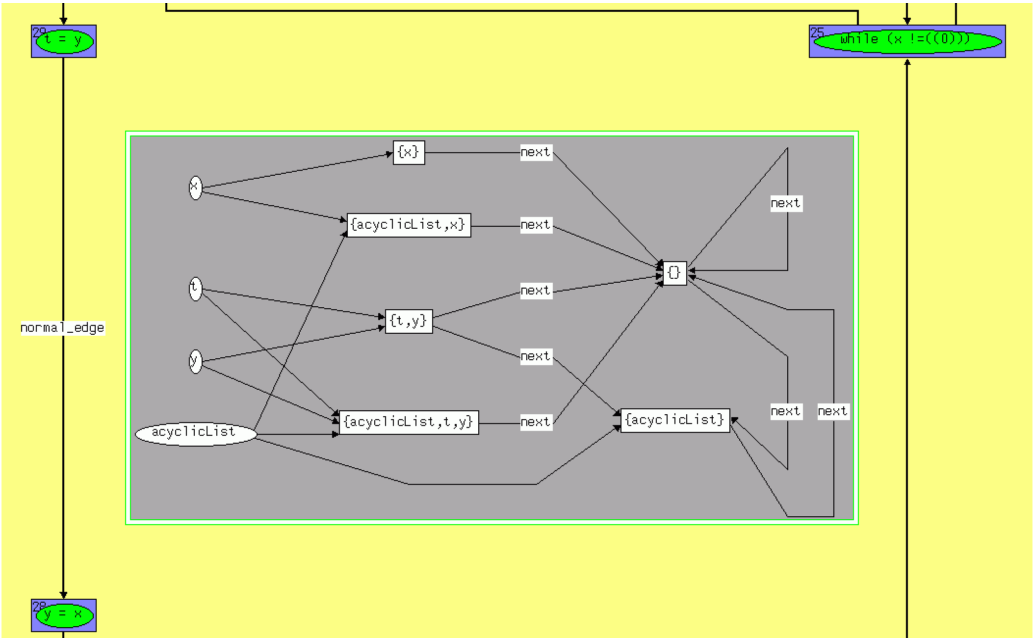
Points-To Analysis

- Variant of Steensgaard‘s algorithm
- Flow-insensitive
- Consideres type information
- Consideres function pointers
- Handles full C
- Context-sensitive version: static call strings with function summaries
- Heap allocated data structures are considered by call sites of malloc/new

Shape Analysis

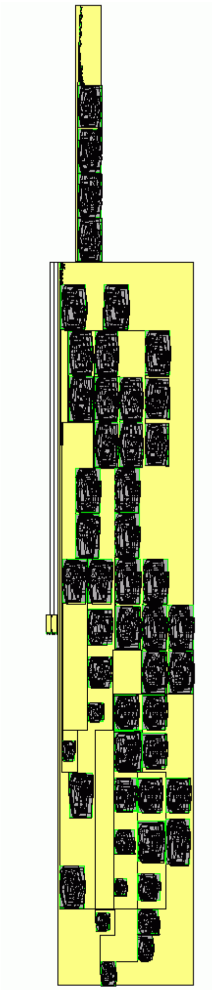


Computes the shape of heap allocated data structures for each program point

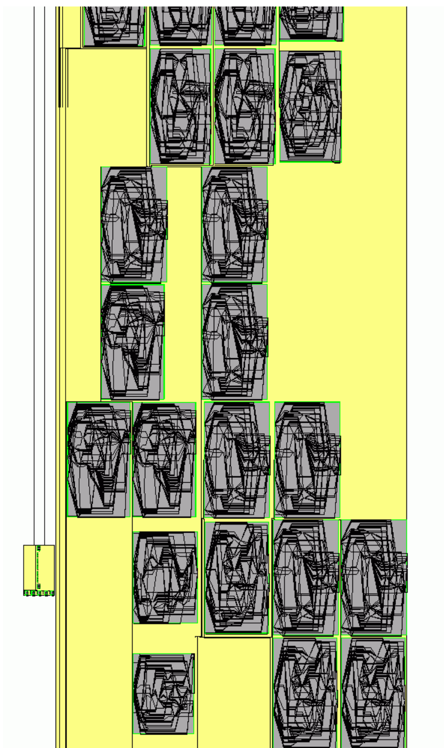


Analyzed example program: list create, list reversal

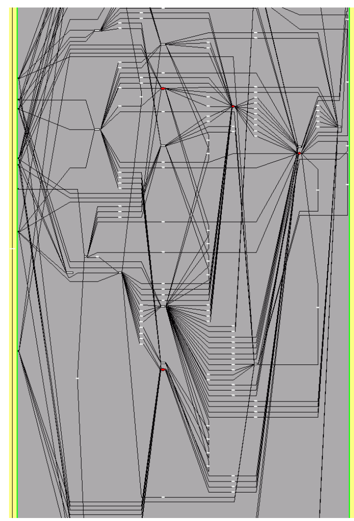
Shape Analysis
Precision and Complexity

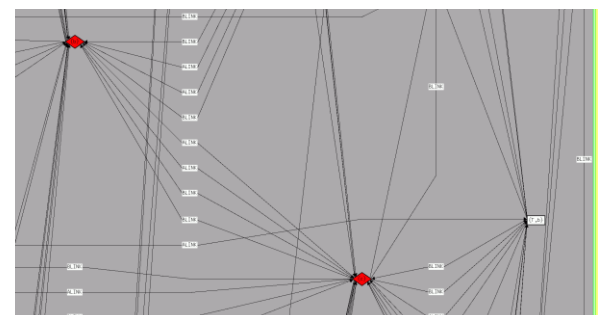


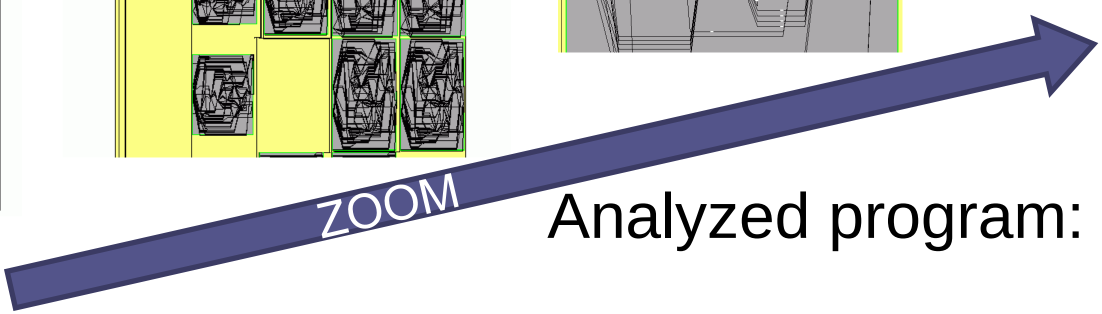
Strong update



Graph for each statement:
worst-case: 2^n Nodes







Analyzed program: DSW-Algorithm

Running Example and SSA Forms

1. Scalar variables only
2. With pointers to local variables
3. Heap allocated data structures

Artificial Sum (only scalar vars)

```
   main() {
0    int a,b,i,j,n,y;
1    n=read();
6    a=0;
7    b=0;
8    i=n;
9    j=n;
10   while (i>0) {
11     a=a+1;
12     i=i-1;
13     j=i;
14     while (j>0) {
15       b=b+1;
16       j=j-1;
       }
     };
17   y=a+b;
18   write(y);
   }
```

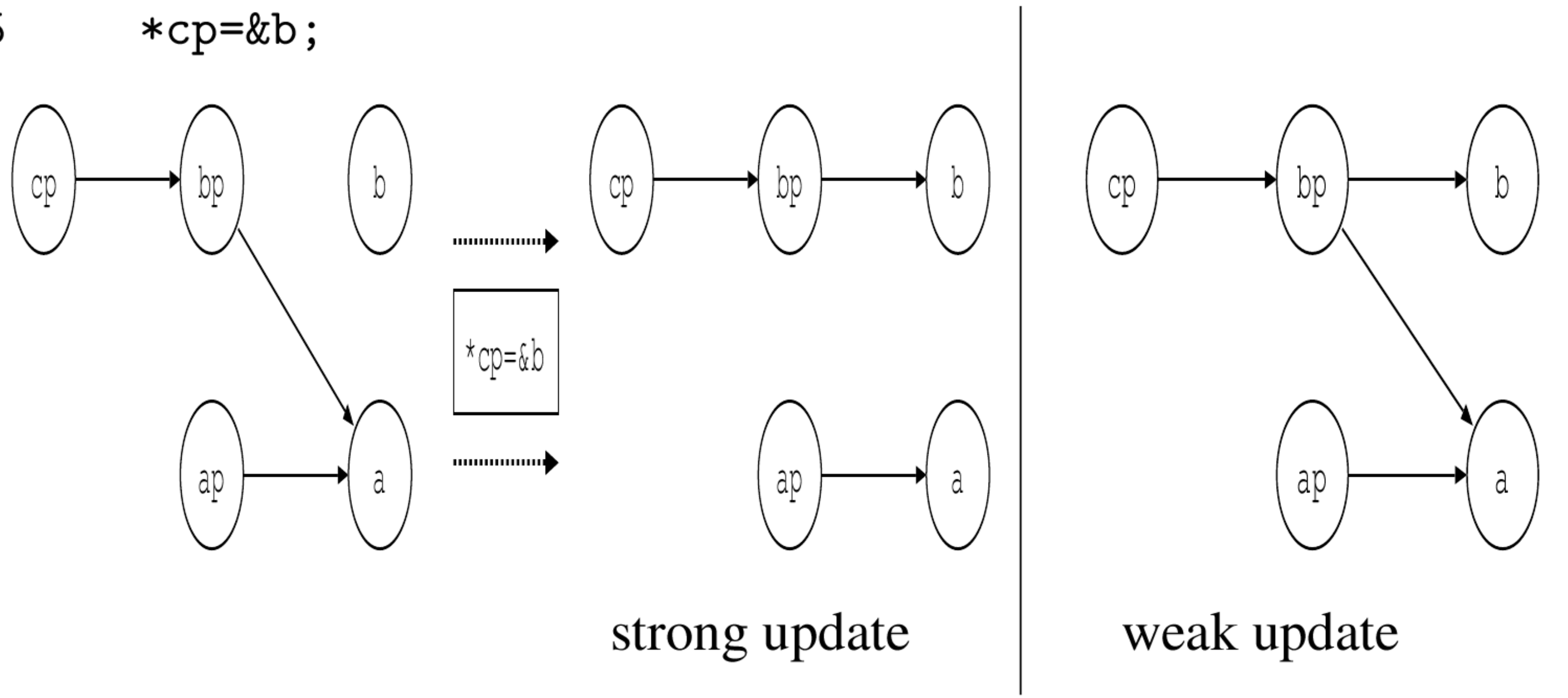
$$y = n + \sum_{i=1}^{n-1} i = \sum_{i=1}^{n} i$$

Example: With Pointers

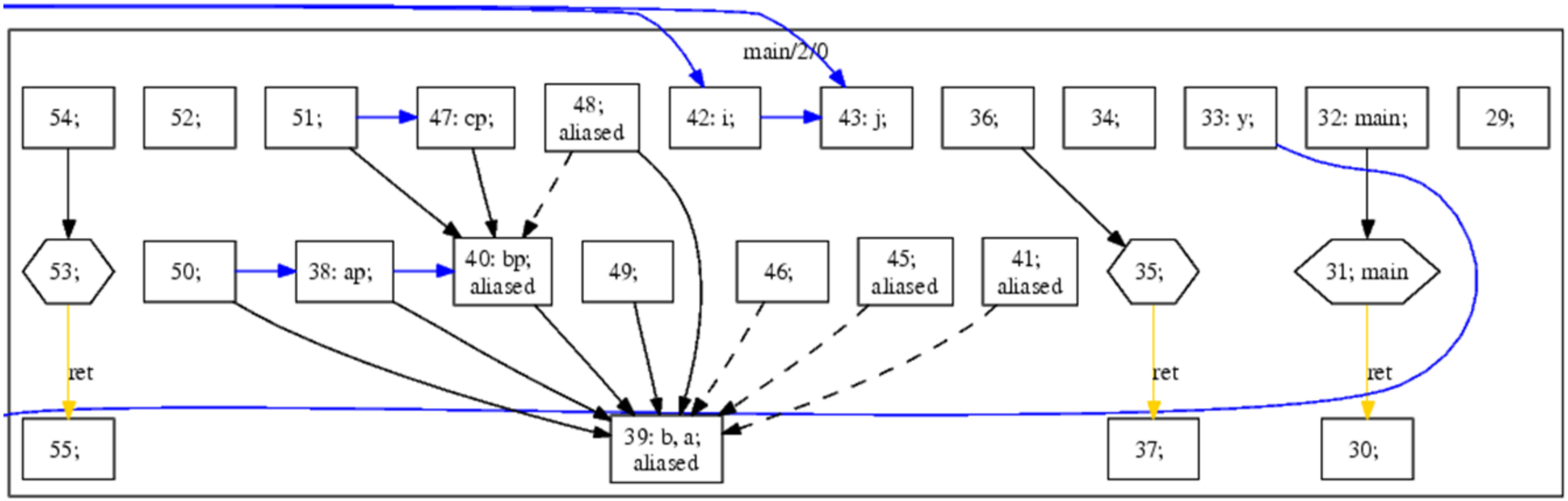
```
   main() {
0    int a,b,i,j,n, *ap,*bp,**cp;
1    n=read();
2    cp=&bp
3    ap=&a;
4    bp=ap;
5    *cp=&b;
6    a=0;
7    b=0;
8    i=n;
9    j=n;
10   while (i>0) {
11     *ap=*ap+1;
12     i=i-1;
13     j=i;
14     while (j>0) {
15       *bp=*bp+1;
16       j=j-1;
       }
     }
17   y=*ap + *bp;
18   write(y);
   }
```

Strong vs Weak Update

```
int a,b,i,j,*ap,*bp,**cp;
cp=&bp
ap=&a;
bp=ap;
*cp=&b;
```

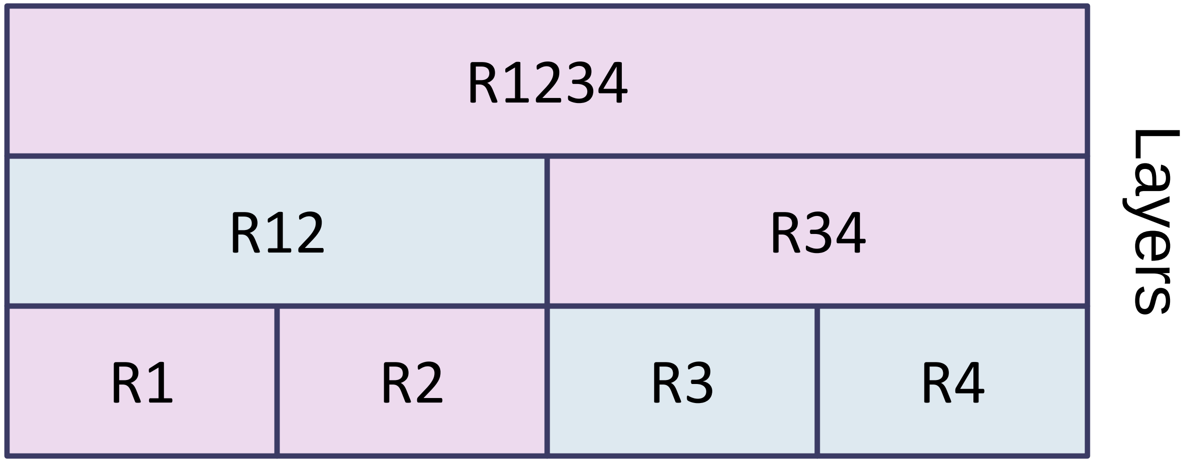


Flow-Insensitive Points-To Analysis



Memory Regions - Cases

- Partitions
- Subsets



1. R12=R3; // partitions
2. R1=R12; // sub-region is assigned a super-region
3. R12=R1; // super-region is assigned a sub-region
4. R12=R1; R12=R2; // complete region
   (e.g. initialization)

Subsets and Partition-Layers

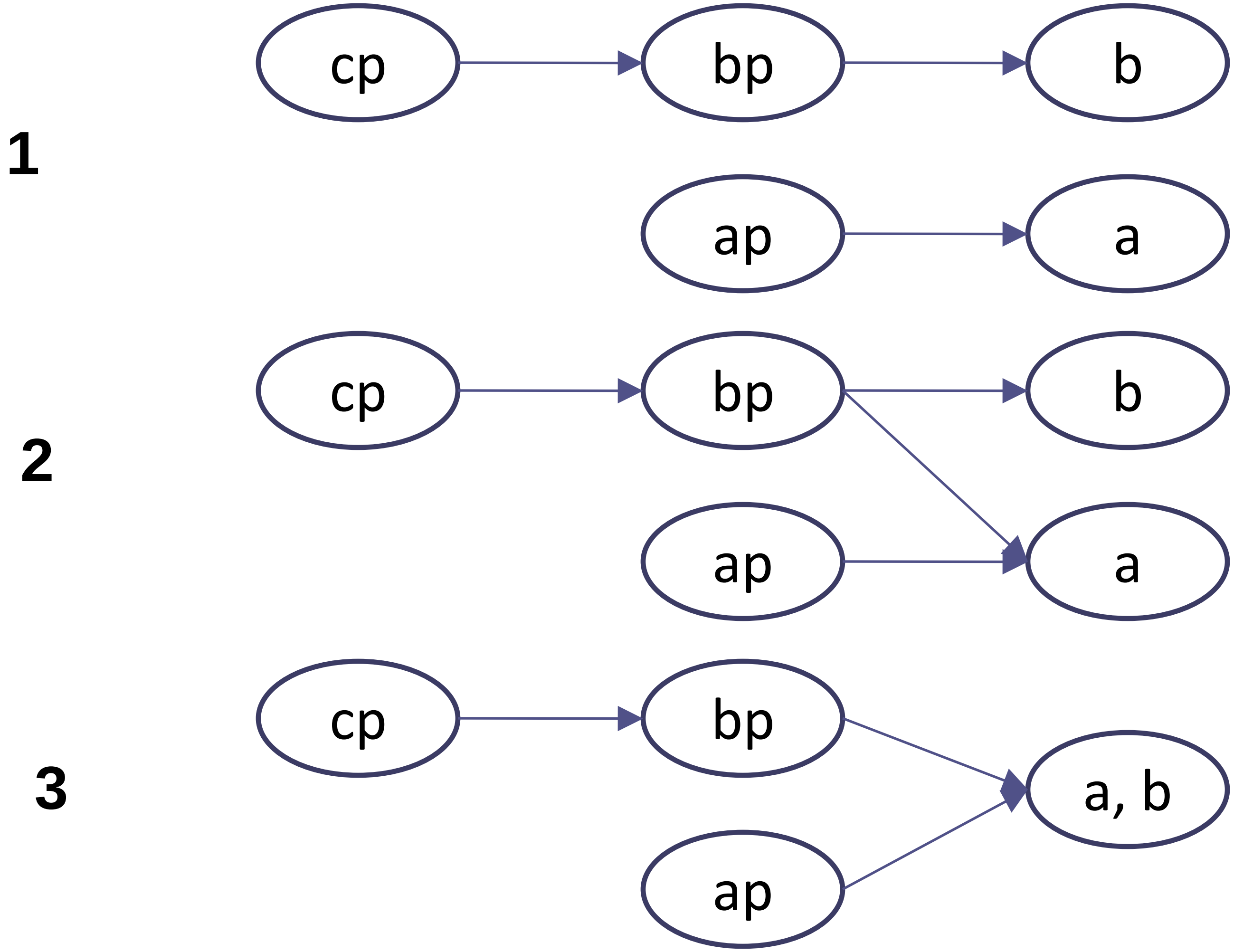
Want to have an SSA where

- each variable representing a memory region that is potentially modified, shows up on the LHS.
- each variable representing a memory region that is accessed shows up on the RHS.

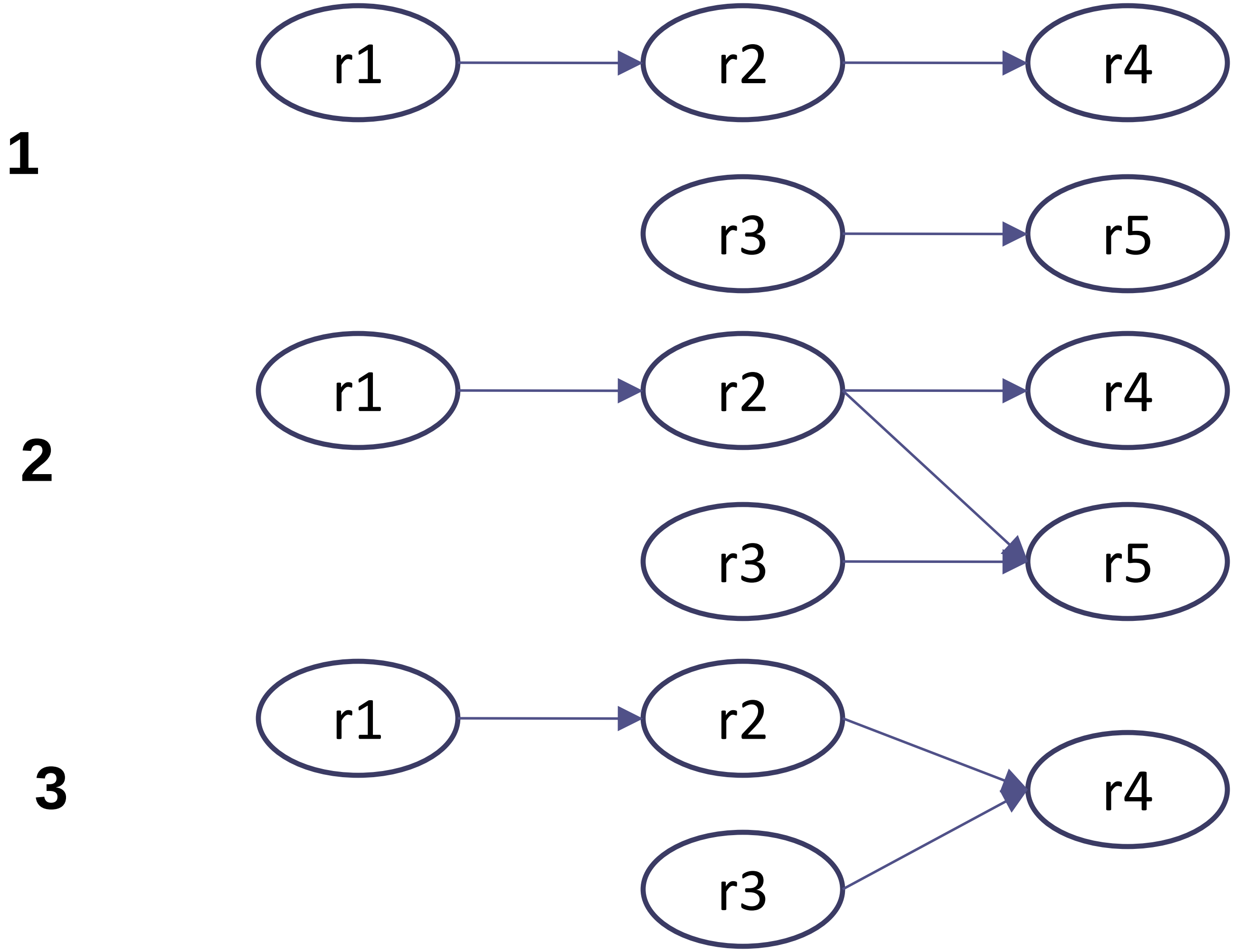
Solutions:

- Use the superset that contains all mod/ref regions and name it.
- Use multi-assignments.

Pointer Analysis Precision



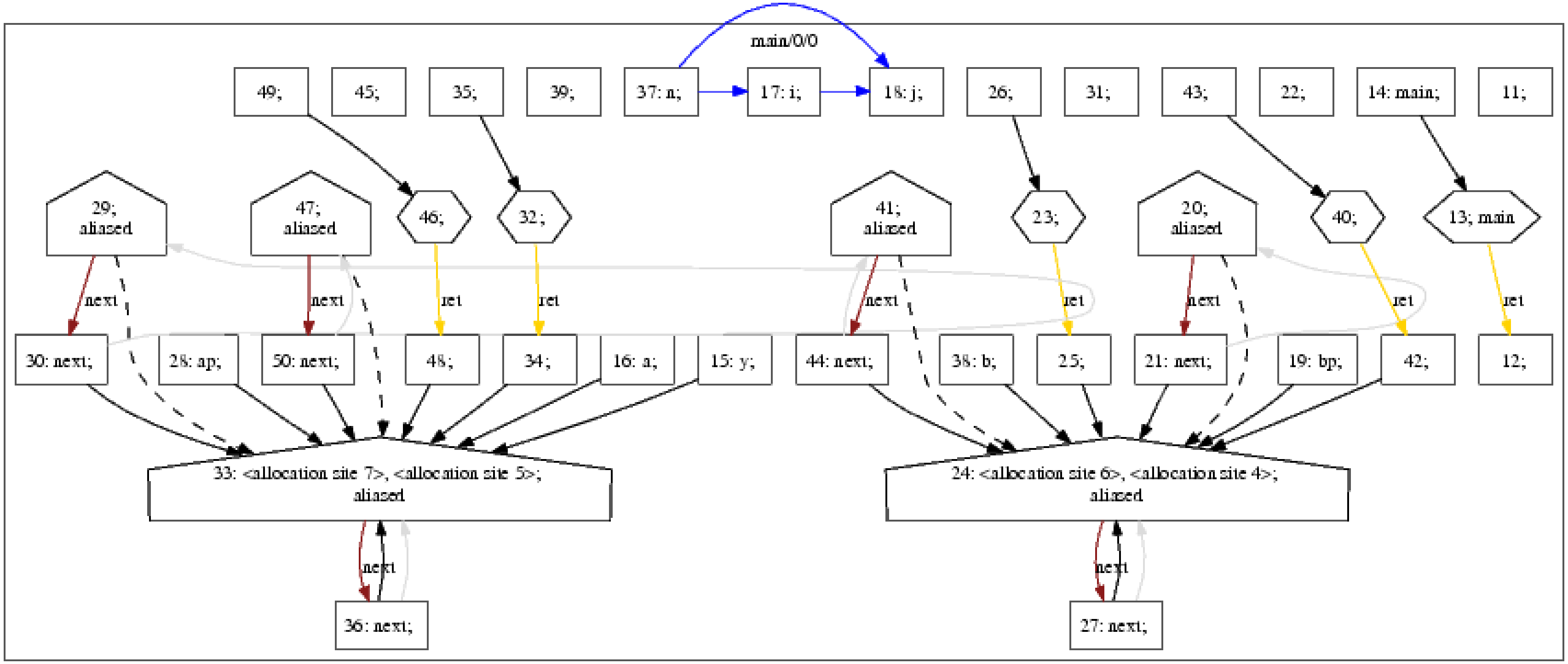
Memory Regions



With Dynamic Data Structures

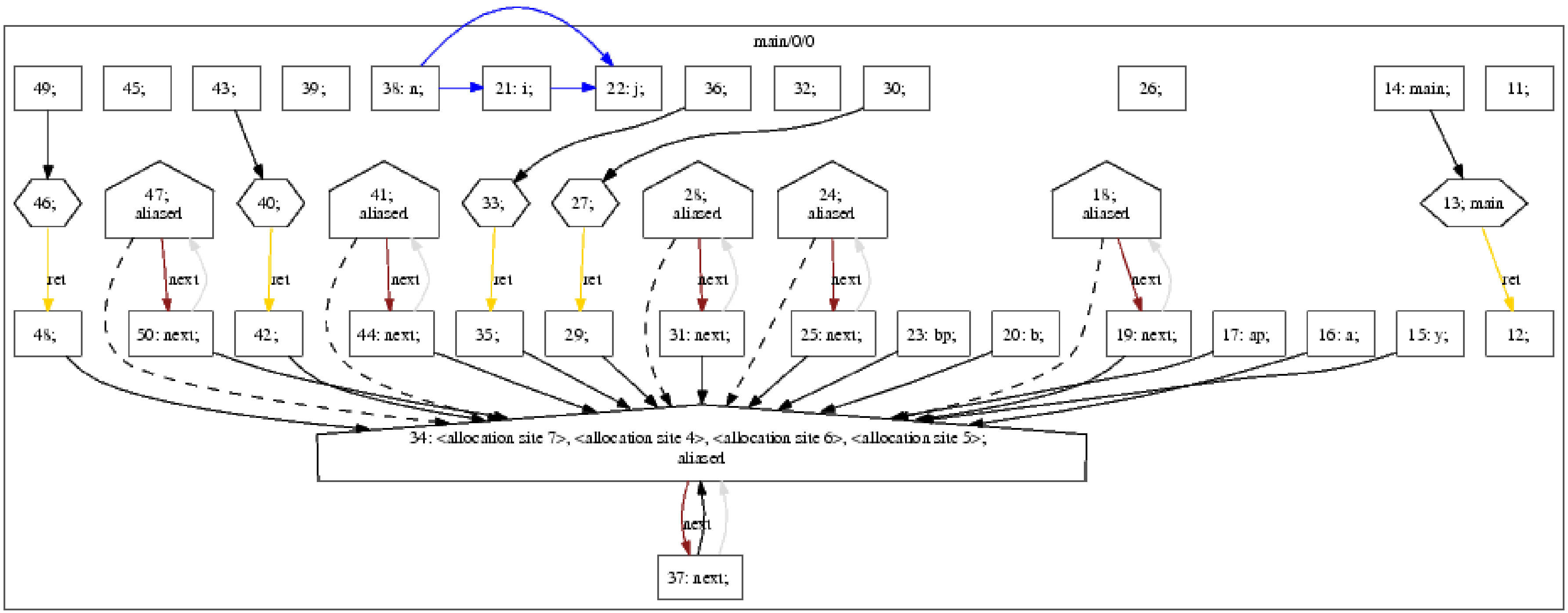
```
    main() {
    List *a,*b,*ap,*bp;
    int i,j,n;
1   n=read();
2   a=new List();
3   b=new List();
4   ap=a;
5   bp=b;
8   i=n;
9   j=n;
10    while (i>0) {
11      ap->next=new List();
11b     ap=ap->next;
12      i=i-1;
13      j=i;
14      while (j>0) {
15        bp->next=new List();
15b       bp=bp->next;
16        j=j-1;
        }
      }
17    ap->next = b; // conc
17b   y=a;
18    write(length(y)-2)
    }
```

Type-Supported Points-To [1]



Before: 17: ap->next=b

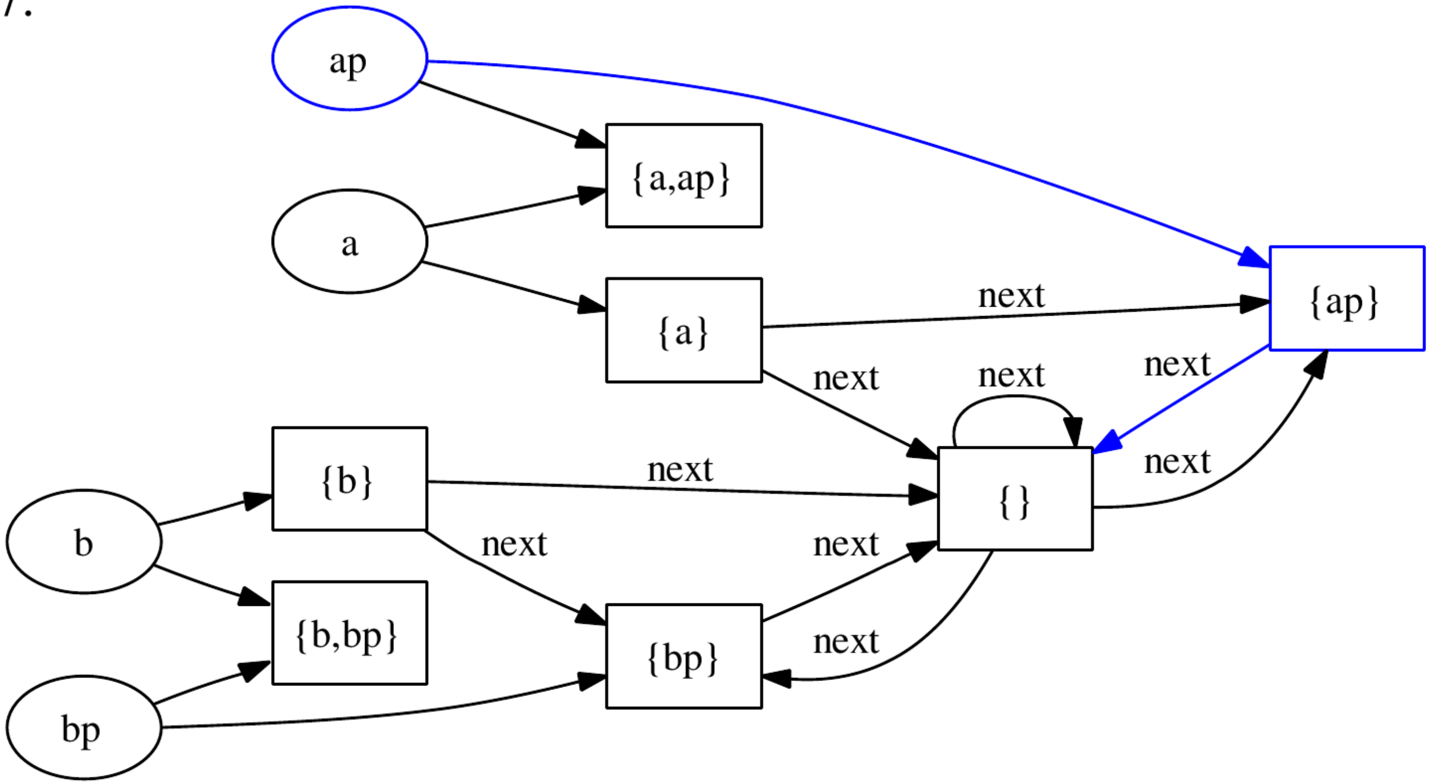
Type-Supported Points-To [2]



Collapsing after: 17: ap->next=b

Shape Analysis

17:



In general: requires cross-linking of shape graphs (ongoing work)

rSSA Form: Dynamic DS

```
a=new List();
b=new List();
ap=a;
bp=b;
i=n;
j=n;
while (i>0) {
 ap->next=new List();
 ap=ap->next;
 i=i-1;
 j=i;
 while (j>0) {
  bp->next=new List();
  bp=bp->next;
  j=j-1;
 }
}
ap->next=b;
```

Merge Sub-regions?

```
r1.1=new ;
r2.1=new ;
r1.2=r1.1;
r2.2=r2.1
i.1=n.1;
j.1=n.1;
i.3=phi(i.1,i.2)
j.5=phi(j.1,j.4)
r1.5=phi(r1.2,r1.4)
r2.6=phi(r2.2,r2.5)
while (i.3>0) {
  r1.3=r1.5 + new;
  r1.4=r1.3;
  i.2=i.3-1;
  j.2=i.2;
   j.4=phi(j.2,j.3)
   r2.5=phi(r2.6,r2.4)
  while (j.5>0) {
    r2.3=r2.5 + new;
    r2.4=r2.3;
    j.3=j.4-1;
  }
}
r1.6=r1.5 + r2.6;
```

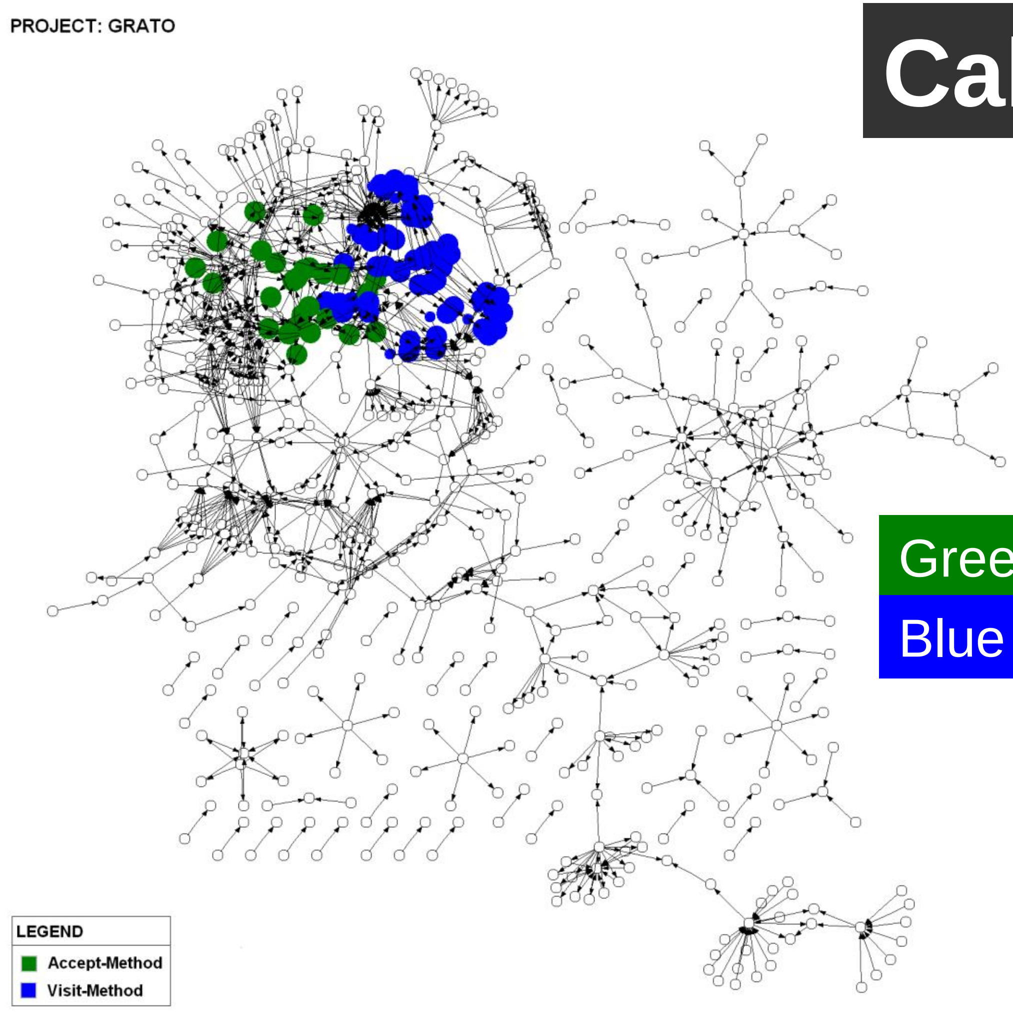
r1: a,ap,ap->next
r2: b,bp,bp->next

preserving definitions

Using SSA

- Region-based SSA Form allows to create a high-level abstraction of a program
- Design pattern detection
  - Based on reduced program dependence graph
- Component recognition
  - Based on type & field-access information (= regions)

Design Pattern Detection

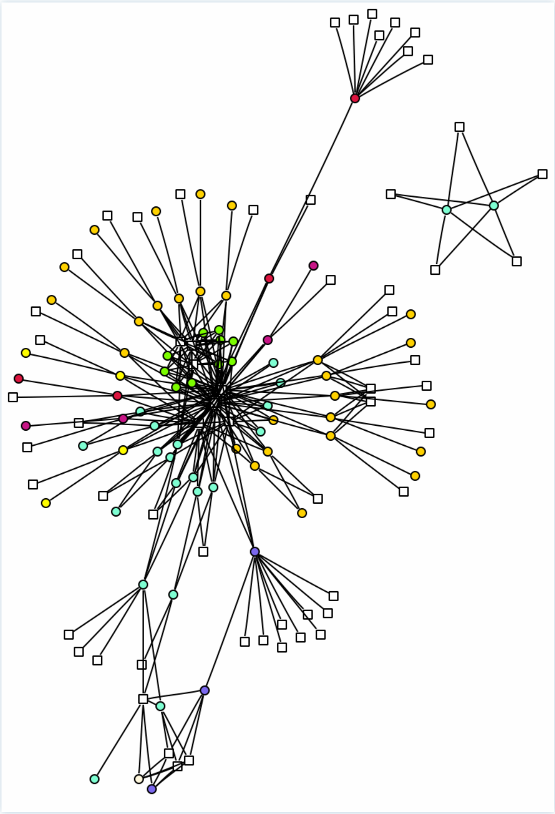


Call Graph

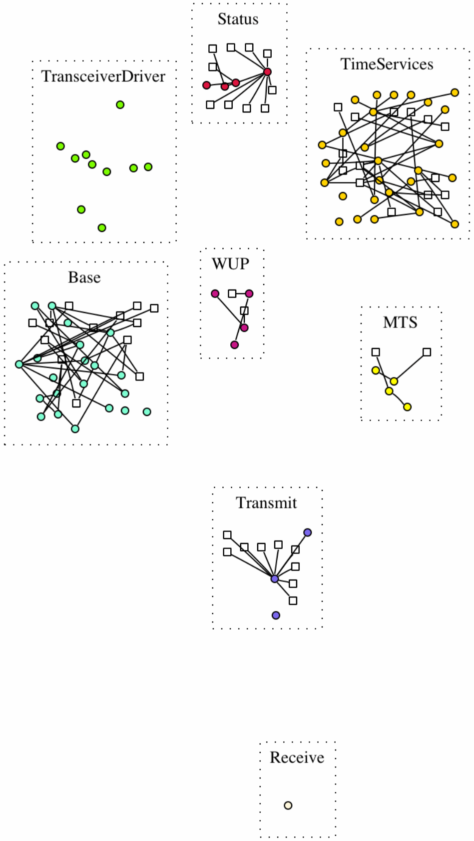
Green nodes: accept methods

Blue nodes : visit methods

Component Recognition



Unstructured Unfiltered Dependencies



Components Filtered Dependencies

Summary

- SATIrE: Static Analysis Tool Integration Engine
  - Flow-sensitive context-sensitive analysis of C/C++
  - Website: http://www.complang.tuwien.ac.at/satire
- Memory region based SSA Form
  - The more precise the pointer analysis the more memory regions
  - Scaling via memory sub-region relation
- Region-based SSA form with program information suitable for code pattern detection