Scaling Program Analysis

CS7575

Static program analysis *is the analysis of computer software performed without actually executing programs.*

Wikipedia

http://surl.li/lboxd

Shared notebook

With material from "Principles of Program Analysis" by Nielsen[2] & Hankin

Course Structure

Part 1: Lectures

Aim: cover chapters 1-4 of PoPA (dataflow, constraint-based, AI)

Part 2: Papers & Projects

Broad overview of applications of static analysis in software engineering, and compilers, including dynamic analysis, big code and machine learning

Workload

Option 1: TestOut + Talk

The midterm test and a paper prevention are your whole grade

Option 2: Repetition

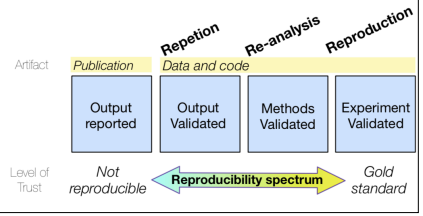
Pick a paper, re-run experiments, present and write a report

Option 3: Re-analysis

Option 2 + re-analyze the results

Option 4: Research

Work on something new of your choice



Program Analysis

Characterize possible executions behaviors of a program

Use results for:

Compiler optimizations

Program verification

Error detection

Vulnerability detection

Static Analysis

Compile-time characterization of all program executions by approximation

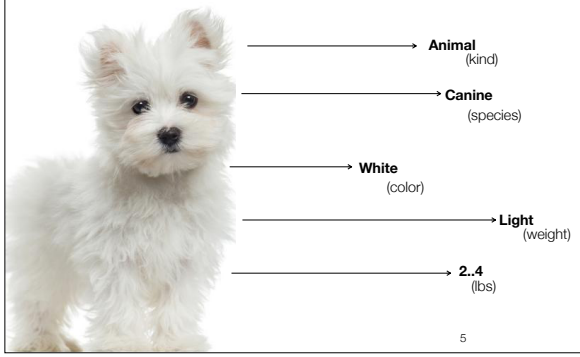
Dynamic Analysis

Run-time characterization of some program executions by observation

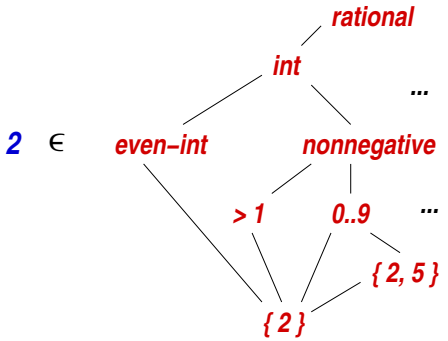
Abstraction

An *abstraction* is a property from some domain

A static analysis is determined its choice of abstraction: precise enough to capture the properties of interest, approximate enough to be computable



Value abstractions are common



All the properties listed on the right are abstractions of 2; the upwards lines denote $\sqsubseteq$, a loss of precision.

Abstractions can be relations

These Figures are from *Abstract Interpretation: Achievements and Perspectives* by Patrick Cousot, Proc. SSGRR 2000.

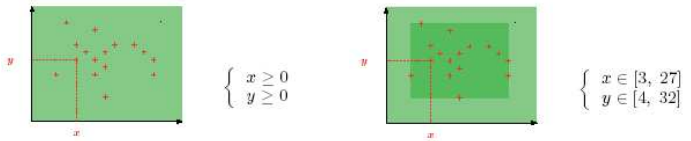


Fig. 2
SIGNS

Fig. 3
INTERVALS

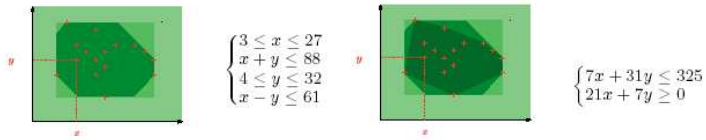


Fig. 4
OCTAGONS

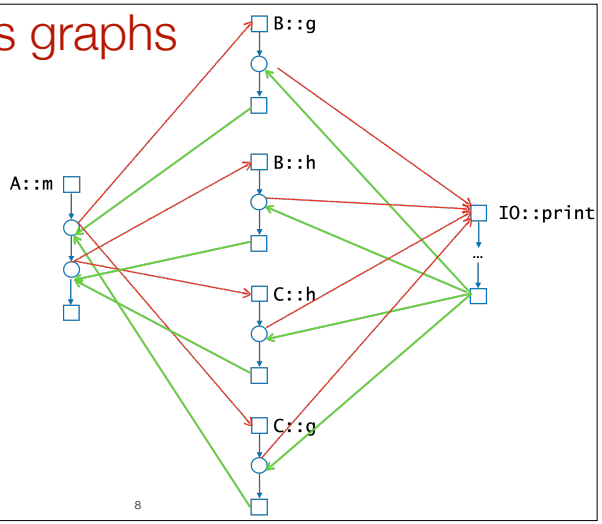
Fig. 5
POLYHEDRA

Abstractions as graphs

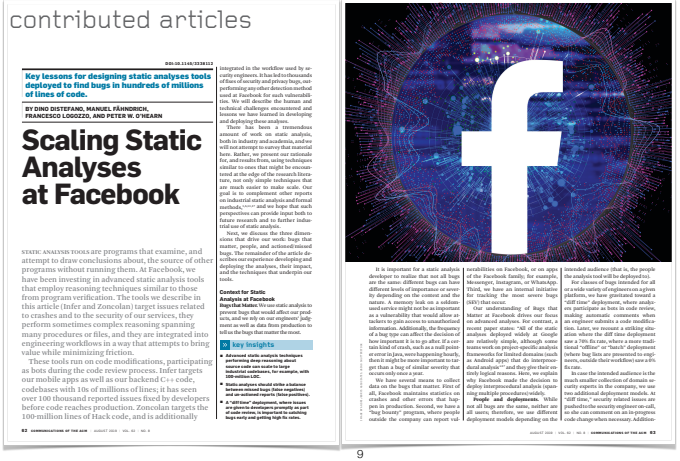
```
class A {
  m( x ) {
    x.g();
    x.h();
  }
}

class B {
   g( ) {
      IO.print("B::g");
    }
    h( ) {
      IO.print("B::h");
    }
}

class C {
   g( ) {
      IO.print("B::g");
    }
    h( ) {
      IO.print("B::h");
    }
}
```



Intermezzo Static Analysis at Facebook



contributed articles

DOI:10.1145/3338112

Key lessons for designing static analyses tools deployed to find bugs in hundreds of millions of lines of code.

BY DINO DISTEFANO, MANUEL FÄHNDRICH, FRANCESCO LOGOZZO, AND PETER W. O'HEARN

Scaling Static Analyses at Facebook

STATIC ANALYSIS TOOLS are programs that examine, and attempt to draw conclusions about, the source of other programs without running them. At Facebook, we have been investing in advanced static analysis tools that employ reasoning techniques similar to those from program verification. The tools we describe in this article (Infer and Zoncolan) target issues related to crashes and to the security of our services, they perform sometimes complex reasoning spanning many procedures or files, and they are integrated into engineering workflows in a way that attempts to bring value while minimizing friction.

These tools run on code modifications, participating as bots during the code review process. Infer targets our mobile apps as well as our backend C++ code, codebases with 10s of millions of lines; it has seen over 100 thousand reported issues fixed by developers before code reaches production. Zoncolan targets the 100-million lines of Hack code, and is additionally integrated in the workflow used by security engineers. It has led to thousands of fixes of security and privacy bugs, outperforming any other detection method used at Facebook for such vulnerabilities. We will describe the human and technical challenges encountered and lessons we have learned in developing and deploying these analyses.

There has been a tremendous resurgence of work on static analysis, both in industry and academia, and we will not attempt to survey that material here. Rather, we present our rationale for, and results from, using techniques similar to ones that might be encountered at the edge of the research literature, not only simple techniques that are much easier to make scale. Our goal is to complement other reports on industrial static analysis and formal methods,[10,13,17] and we hope that such perspectives can provide input both to future research and to further industrial use of static analysis.

Next, we discuss the three dimensions we found important in this endeavor, people, and actionable issues. The remainder of the article describes our experience developing and deploying the analyses, their impact, and the techniques that underpin our tools.

Context for Static Analysis at Facebook

Bugs that Matter. We use static analysis to prevent bugs that would affect our products, and we rely on our engineers' judgment as well as data from production to tell us the bugs that matter the most.

key insights

- **Advanced static analysis techniques performing deep reasoning about source code can be applied at industrial codebases, for instance, with 100-million LOC.**
- **Static analyzers should strike a balance between missed bugs (false negatives) and an outbound reports (false positives).**
- **A "diff time" deployment, where issues are reported directly to developers in the form of code reviews, is important to satisfying bugs early and getting high fix rates.**

It is important for a static analysis developer to realize that not all bugs are the same: different bugs can have different levels of importance or severity for different use cases. For instance, a memory leak on a seldom-used service might not be as important as a vulnerability that would allow attackers to gain access to unauthorized information. Additionally, the frequency of a bug type can affect the decision of how important it is to go after. If a certain kind of crash, such as a null pointer error in Java, were happening hourly, then it might be more important to target than a bug of similar severity that occurs only once a year.

We rely on several systems to collect data on the bugs that matter. First of all, Facebook maintains statistics on crashes and other errors that happen in production. Second, we have a "bug bounty" program, where people outside the company can report vulnerabilities on Facebook, or on apps of the Facebook family; for example, Messenger, Instagram, or WhatsApp. Third, we have an internal initiative where we "dog food" most severe bugs (SEV) that occur.

Our understanding of Bugs that Matter at Facebook drives our focus on advanced analyses. For contrast, a recent paper states: "All of the static analysis deployed widely at Google are relatively simple, although some teams work on project-specific analysis frameworks for limited domains (such as Android apps) that do interprocedural analysis"[17] and they give their entirely logical reasons. Here, we explain why we have found it important to deploy interprocedural analysis (sometimes multiple procedures) widely.

People and deployments. While not all bugs are the same, neither are all users; therefore, we use different deployment models depending on the intended audience (that is, the people the analysis tool will be deployed to).

For classes of bugs intended for all or a wide variety of engineers on a given codebase, we have implemented a "diff time" deployment, where analyzers participate as bots in code review, making automatic comments when an engineer submits a code modification. Later, we recount a striking situation where the diff time deployment saw a 70% fix rate, where a more traditional "offline" or "batch" deployment (where bug lists are presented to engineers, outside their workflow) saw a 0% fix rate.

In case the intended audience is the set of engineers who are security or security experts in the company, we use two additional deployment models. At "diff time," security related issues are pushed to the security engineer on-call, so she can comment on an in-progress code change when necessary. Addition-

Intermezzo Static Analysis at Facebook

Not all bugs are the same!

A memory leak on a seldom-used service not as important as a vulnerability granting access to unauthorized information
A crash, such as a Java NPE, that happens hourly is more important than the same bug happening yearly

Facebook built two tools: **Infer** and **Zoncolan**

Infer

Target 10**M** lines of Java, Objective-C, C++ on Android/iOS for FB, Instagram, Messenger & WhatsApp, and backend C++ or Java

Origins: Peter O'Hearn ++ (UCL⇝Monoidics) separation logic

Zoncolan

Target 100**M** lines of Hack for Facebook's website

Origins: Fahndrich (EPFLBerkeley⇝MSR) & Logozzo (Polytechnique/R. Cousot⇝MSR)

Monte Zoncolan is a mountain in the Carnic Alps, Italy, with an elevation of 1,750 metres. It is one of the most demanding climbs in professional road bicycle racing.

Distefano, Fahndrich, Logozzo, O'Hearn: *Scaling Static Analyses at Facebook,* CACM'19, doi=3351434.3338112

Intermezzo Static Analysis at Facebook

*Should **Infer** report errors, if so where?*
What are the abstractions?

```
@ThreadSafe
class R {
  int m;
  void pWOMT() {
    Utils.assertMainThread();
    synchronized(this){ m++; }
  }

  int uROMT() {
    Utils.assertMainThread();
    return m;
  }

  synchronized int pROMT() {
    Utils.assertMainThread();
    return m;
  }
```

```
  synchronized void pWOMTB() {
      m+=2;
  }

  int uROMTB() {
    return m;
  }
```


Intermezzo Static Analysis at Facebook

*Should **Infer** report errors, if so where?*
What are the abstractions?

```
r = new R()
...
x = r.m
x= r.m
```

```
1  <?hh
2  class AddMemberToGroup extends FacebookEndpoint {
3    private function getIDs (): (string, int) {
4      // User input, untrusted
5      return tuple((string) $this->getRequest('member_id'),
6                   (int) $this->getRequest('gid'));
7    }
8
9    public function render(): :xhp {
10     list($member_id, $group_id) = $this->getIDs();
11     return this->getConfirmationForm($group_id, $member_id);
12   }
13
14   public function getConfirmationForm
15    (int $group_id, string $member_id): :xhp {
16     $url = "https://facebook.com/groups/add_member/" .
17            $member_id;
18
19     return
20       <form method="post" action={$url}>
21         <input name="gid" value={$group_id}/>
22         <input name="action" value="add"/>
23       </form>;
24   }
25 }
```

⟨5, user − input⟩ → ⟨10, GenericGroupForm :: getIDs⟩ → $member_id → ⟨11, AddMemberToGroup :: getConfirmationForm⟩, 2 → ⟨20, form :: action⟩ (via stringconcat)

*Should **Zoncolan** report errors, if so where?*

What are the abstractions?

Intermezzo Static Analysis at Facebook

Takeaways

What makes a static analysis "good"?

Intermezzo Static Analysis at Facebook

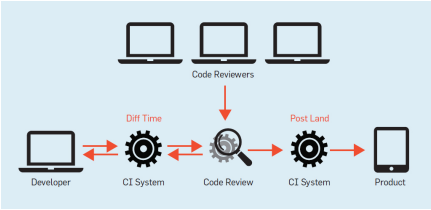
Takeaways:

Goodness = H5+H6+Scalable

Infer

70% fix rate when integrated with CI and 0% fix rate otherwise

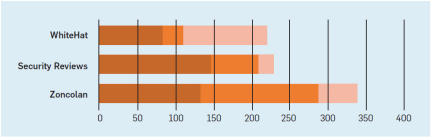
15 minutes / diff time (10M codebase)



Zoncolan

80% actioned rate and 11 "missed" bugs

100M LOC/30 minutes in batch mode



Intermezzo Static Analysis at Facebook

Takeaways

Do analyses need to scale across procedure boundaries?

	Intra-procedural Infra-file	Inter-procedural Intra-file	Inter-procedural Inter-file
Null Deref (Java)	43	9	48
Null Deref (ObjC)	73	5	24
Thread Safety	36	12	53
Allocates Memory	0	2	98
Bad Pointer Comparison	100	0	0

100 recent fixes in several bug categories

https://engineering.fb.com/developer-tools/finding-inter-procedural-bugs-at-scale-with-infer-static-analyzer/

Intermezzo Static Analysis at Facebook

Scalability

Infer is compositional; consider this bug trace

```
src/com/duckduckgo/mobile/android/activity/DuckDuckGo.java:867: error: NULL_DEREFERENCE
object `feedObject` last assigned on line 866 could be null and is dereferenced by call
to `feedItemSelected(...)` at line 867.

864.
865. public void feedItemSelected(String feedId) {
866. FeedObject feedObject = DDGApplication.getDB().selectFeedById(feedId);
867. feedItemSelected(feedObject);
```

https://engineering.fb.com/developer-tools/finding-inter-procedural-bugs-at-scale-with-infer-static-analyzer/


```
src/com/duckduckgo/mobile/android/db/DdgDB.java:483: start of procedure
selectFeedById(...)

482.
483. public FeedObject selectFeedById(String id){
484. FeedObject out = null;
485. Cursor c = null;
486. try {
487.   c = this.db.query(FEED_TABLE, null, "_id=?", new String[]{id}, null, null, null);
488.   if (c.moveToFirst()) {
489.    out = getFeedObject(c);
490.   }
491.  } finally {
492.      if(c!=null) {
493.            c.close();
494.      }
495.  }
496.  return out;
497. }
```


Intermezzo Static Analysis at Facebook

```
src/com/duckduckgo/mobile/android/activity/DuckDuckGo.java:842: start of procedure
feedItemSelected(...)

841.
842.  public void feedItemSelected(FeedObject feedObject) {
843.
844.    DDGControlVar.currentFeedObject = feedObject;
845.    DDGControlVar.mDuckDuckGoContainer.sessionType = SESSIONTYPE.SESSION_FEED;
846.
847     String url = feedObject.getUrl();
```


Intermezzo Static Analysis at Facebook

```
PRE: this.db |→ - (db is non-null and points to a valid object)

POST1: return == null (we are returning a null feed object)
POST2: return |→ - (we are returning a valid, non-null feed object)
```

```
483. public FeedObject selectFeedById(String id){
484. FeedObject out = null;
485. Cursor c = null;
486. try {
487.   c = this.db.query(FEED_TABLE, null, "_id=?", new String[]
488.   if (c.moveToFirst()) {
489.    out = getFeedObject(c);
490.   }
491.  } finally {
492.      if(c!=null) {
493.            c.close();
494.      }
495.  }
496.  return out;
```

Intermezzo Static Analysis at Facebook

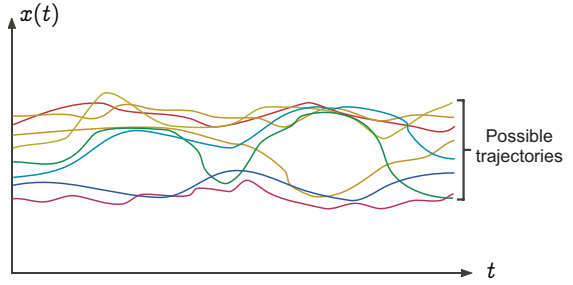
```
PRE: feedObject |→ - (feedObject is a valid, non-null object)
```

```
865. public void feedItemSelected(String feedId) {
866. FeedObject feedObject = DDGApplication.getDB().selectFeedById(feedId);
867. feedItemSelected(feedObject);
```

```
842.  public void feedItemSelected(FeedObject feedObject) {
843.
844.    DDGControlVar.currentFeedObject = feedObject;
845.    DDGControlVar.mDuckDuckGoContainer.sessionType = SESSIONTYPE.SESSI
846.
847     String url = feedObject.getUrl();
```


Principles of Program Analysis

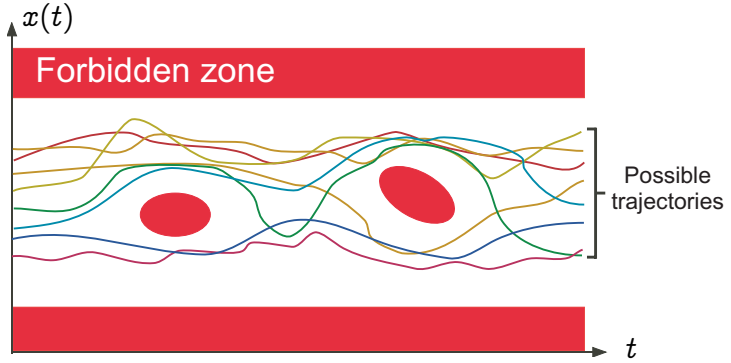
The *concrete semantics* of a program is a mathematical model of the set of possible execution in possible environments



The concrete semantics of a program is *not computable*; all non trivial questions on the concrete semantics are *undecidable*

Safety Properties

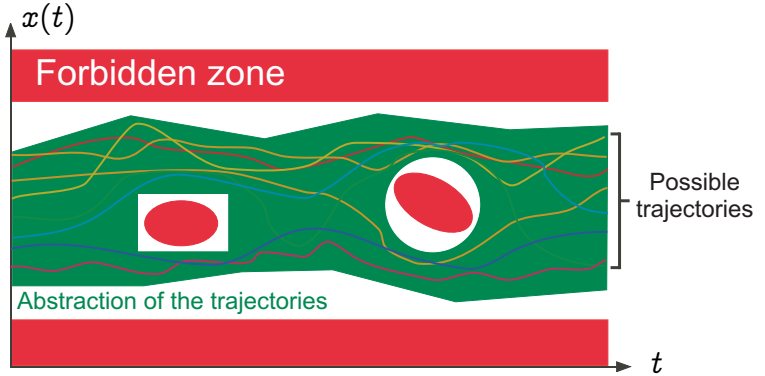
The *safety properties* of a program express that no execution in any environment can reach an erroneous state



A safety proof consists in proving that the intersection of the concrete semantics and the forbidden zone is empty

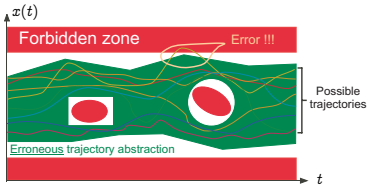
Static Program Analysis

The abstract semantics is computed automatically from the program text according to pre-defined abstractions

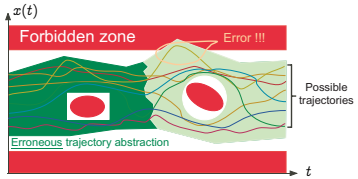


Errors

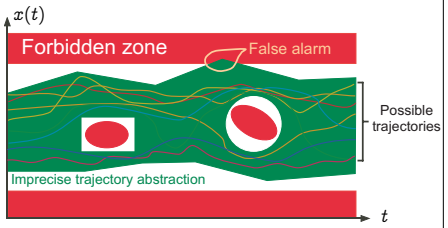
e.g.) unsound analysis



e.g.) bounded model checking



false positive



Static Analysis in Compilers

Consider the C-like program

```
i := 0;
while i <= n do
   j := 0;
   while j <= m do
      temp := Base(A) + i * (m+1) + j;
      Cont(temp) := Cont(Base(B) + i * (m+1) + j)
                  + Cont(Base(C) + i * (m+1) + j);
      j := j+1
   od;
   i := i+1
od
```

Nielsen, Nielsen, Hankin: *Principles of Program Analysis.* Chapter 1

Available Expressions analysis & Common Subexpression Elimination

```
i := 0;
while i <= n do
   j := 0;
   while j <= m do
      temp := Base(A) + i*(m+1) + j;
      Cont(temp) := Cont(Base(B) + i*(m+1) + j)
                  + Cont(Base(C) + i*(m+1) + j);
      j := j+1
   od;
   i := i+1
od
```

first computation

re-computations

```
t1 := i * (m+1) + j;
temp := Base(A) + t1;
Cont(temp) := Cont(Base(B)+t1)
            + Cont(Base(C)+t1);
```

Nielsen, Nielsen, Hankin: *Principles of Program Analysis.* Chapter 1

Detection of Loop Invariants & Invariant Code Motion

```
i := 0;
while i <= n do
   j := 0;
   while j <= m do
      t1 := i * (m+1) + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1
od
```

loop invariant

```
t2 := i * (m+1);
while j <= m do
   t1 := t2 + j;
   temp := ...
   Cont(temp) := ...
   j := ...
od
```

Nielsen, Nielsen, Hankin: *Principles of Program Analysis.* Chapter 1

Detection of Induction Variables & Reduction of Strength

```
i := 0;
while i <= n do
   j := 0;
   t2 := i * (m+1);
   while j <= m do
      t1 := t2 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1
od
```

induction variable

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   t2 := t3;
   while j <= m do ... od
   i := i + 1;
   t3 := t3 + (m+1)
od
```


Equivalent Expressions analysis & Copy Propagation

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   t2 := t3;
   while j <= m do
      t1 := t2 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1;
   t3 := t3 + (m+1)
od
```

t2 = t3

```
while j <= m do
   t1 := t3 + j;
   temp := ...;
   Cont(temp) := ...;
   j := ...
od
```


Live Variables analysis & Dead Code Elimination

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   t2 := t3;
   while j <= m do
      t1 := t3 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1;
   t3 := t3 + (m+1)
od
```

dead variable

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   while j <= m do
      t1 := t3 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1;
   t3 := t3 + (m+1)
od
```


Summary

Analysis	Transformation
Available expressions analysis	Common subexpression elimination
Detection of loop invariants	Invariant code motion
Detection of induction variables	Strength reduction
Equivalent expression analysis	Copy propagation
Live variables analysis	Dead code elimination

Approaches to Program Analysis

A family of techniques . . .

- data flow analysis
- constraint based analysis
- abstract interpretation
- type and effect systems
- . . .

. . . that differ in their focus:

- algorithmic methods
- semantic foundations
- language paradigms
  - — imperative
  - — object oriented
  - — logical
  - — functional
  - — concurrent

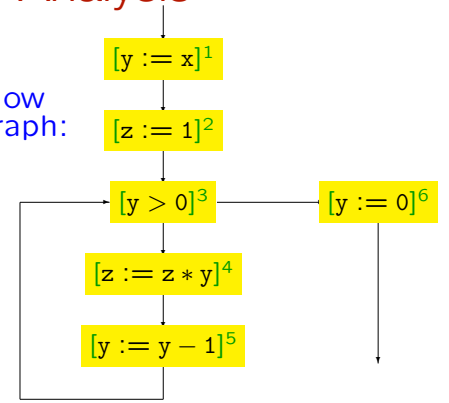
Nielsen, Nielsen, Hankin: *Principles of Program Analysis*. Chapter 1

Data Flow Analysis

Program with labels for elementary blocks:

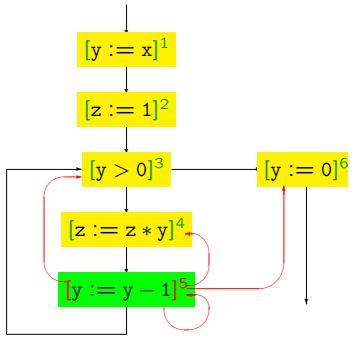
$[y := x]^1$;
$[z := 1]^2$;
while $[y > 0]^3$ do
  $[z := z * y]^4$;
  $[y := y - 1]^5$
od;
$[y := 0]^6$

Flow graph:



Reaching Definition

The assignment $[x := a]^\ell$ reaches ℓ' if there is an execution where x was last assigned at ℓ



Reaching Definition

$\{(x, ?), (y, ?), (z, ?)\}$
$[y := x]^1$;
$[z := 1]^2$;
while $[y > 0]^3$ do
  $[z := z * y]^4$;
  $[y := y - 1]^5$
od;
$[y := 0]^6$

Reaching Definition

$\{(x,?),(y,?),(z,?)\}$

$[y := x]^1;$

$\{(x,?),(y,1),(z,?)\}$

$[z := 1]^2;$

$\{(x,?),(y,1),(z,2)\}$

`while` $[y > 0]^3$ `do`

$[z := z * y]^4;$

$[y := y - 1]^5$

`od;`

$\{(x,?),(y,1),(z,2)\}$

$[y := 0]^6$

Reaching Definition

$\{(x,?),(y,?),(z,?)\}$

$[y := x]^1;$

$\{(x,?),(y,1),(z,?)\}$

$[z := 1]^2;$

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

`while` $[y > 0]^3$ `do`

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[z := z * y]^4;$

$\{(x,?),(y,1),(y,5),(z,4)\}$

$[y := y - 1]^5$

$\{(x,?),(y,5),(z,4)\}$

`od;`

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[y := 0]^6$

The "best" solution

$\{(x,?),(y,?),(z,?)\}$

$[y := x]^1;$

$\{(x,?),(y,1),(z,?)\}$

$[z := 1]^2;$

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

`while` $[y > 0]^3$ `do`

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[z := z * y]^4;$

$\{(x,?),(y,1),(y,5),(z,4)\}$

$[y := y - 1]^5$

$\{(x,?),(y,5),(z,4)\}$

`od;`

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[y := 0]^6$

$\{(x,?),(y,6),(z,2),(z,4)\}$

A solution

$\{(x,?),(y,?),(z,?)\}$

$[y := x]^1;$

$\{(x,?),(y,1),(z,?)\}$

$[z := 1]^2;$

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

`while` $[y > 0]^3$ `do`

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[z := z * y]^4;$

$\{(x,?),(y,1),(y,5),\boxed{(z,2)},(z,4)\}$

$[y := y - 1]^5$

$\{(x,?),\boxed{(y,1)},(y,5),\boxed{(z,2)},(z,4)\}$

`od;`

$\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[y := 0]^6$

$\{(x,?),(y,6),(z,2),(z,4)\}$

An unsound solution

$\{(x,?),(y,?),(z,?)\}$

$[y := x]^1;$ — $\{(x,?),(y,1),(z,?)\}$

$[z := 1]^2;$ — $\{(x,?),(y,1),(z,2),(y,5),(z,4)\}$

`while` $[y > 0]^3$ `do` — $\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[z := z * y]^4;$ — $\{(x,?),(y,1),(y,5),(z,4)\}$

$[y := y - 1]^5$ — $\{(x,?),(y,5),(z,4)\}$

`od;` — $\{(x,?),(y,1),(y,5),(z,2),(z,4)\}$

$[y := 0]^6$ — $\{(x,?),(y,6),(z,2),(z,4)\}$

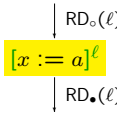
Reaching Definitions

extract equations from the program → compute the least solution to the equations

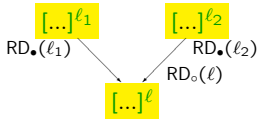
Analysis information:

- $RD_\circ(\ell)$: information available at the entry of block ℓ
- $RD_\bullet(\ell)$: information available at the exit of block ℓ

Two kinds of equations



$$RD_\circ(\ell) \setminus \{(x,\ell') \mid \ell' \in \mathbf{Lab}\} \cup \{(x,\ell)\} = RD_\bullet(\ell)$$



$$RD_\bullet(\ell_1) \cup RD_\bullet(\ell_2) = RD_\circ(\ell)$$

Flow through assignments and tests

$[y := x]^1;$ — $RD_\bullet(1) = RD_\circ(1) \setminus \{(y,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y,1)\}$

$[z := 1]^2;$ — $RD_\bullet(2) = RD_\circ(2) \setminus \{(z,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(z,2)\}$

`while` $[y > 0]^3$ `do` — $RD_\bullet(3) = RD_\circ(3)$

$[z := z * y]^4;$ — $RD_\bullet(4) = RD_\circ(4) \setminus \{(z,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(z,4)\}$

$[y := y - 1]^5$ — $RD_\bullet(5) = RD_\circ(5) \setminus \{(y,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y,5)\}$

`od;`

$[y := 0]^6$ — $RD_\bullet(6) = RD_\circ(6) \setminus \{(y,\ell) \mid \ell \in \mathbf{Lab}\} \cup \{(y,6)\}$

Lab = {1,2,3,4,5,6}

6 equations in $RD_\circ(1), \cdots, RD_\bullet(6)$

Flow along the control

$RD_\circ(1) = \{(x,?),(y,?),(z,?)\}$

$[y := x]^1;$ — $RD_\circ(2) = RD_\bullet(1)$

$[z := 1]^2;$ — $RD_\circ(3) = RD_\bullet(2) \cup RD_\bullet(5)$

`while` $[y > 0]^3$ `do` — $RD_\circ(4) = RD_\bullet(3)$

$[z := z * y]^4;$ — $RD_\circ(5) = RD_\bullet(4)$

$[y := y - 1]^5$

`od;` — $RD_\circ(6) = RD_\bullet(3)$

$[y := 0]^6$

Lab = {1,2,3,4,5,6}

6 equations in $RD_\circ(1), \cdots, RD_\bullet(6)$

Summary of equation system

$$
\begin{array}{l}
\mathsf{RD}_\bullet(1) = \mathsf{RD}_\circ(1) \setminus \{(\mathtt{y}, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{y}, 1)\} \\
\mathsf{RD}_\bullet(2) = \mathsf{RD}_\circ(2) \setminus \{(\mathtt{z}, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{z}, 2)\} \\
\mathsf{RD}_\bullet(3) = \mathsf{RD}_\circ(3) \\
\mathsf{RD}_\bullet(4) = \mathsf{RD}_\circ(4) \setminus \{(\mathtt{z}, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{z}, 4)\} \\
\mathsf{RD}_\bullet(5) = \mathsf{RD}_\circ(5) \setminus \{(\mathtt{y}, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{y}, 5)\} \\
\mathsf{RD}_\bullet(6) = \mathsf{RD}_\circ(6) \setminus \{(\mathtt{y}, \ell) \mid \ell \in \mathbf{Lab}\} \cup \{(\mathtt{y}, 6)\}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{RD}_\circ(1) = \{(\mathtt{x}, ?), (\mathtt{y}, ?), (\mathtt{z}, ?)\} \\
\mathsf{RD}_\circ(2) = \mathsf{RD}_\bullet(1) \\
\mathsf{RD}_\circ(3) = \mathsf{RD}_\bullet(2) \cup \mathsf{RD}_\bullet(5) \\
\mathsf{RD}_\circ(4) = \mathsf{RD}_\bullet(3) \\
\mathsf{RD}_\circ(5) = \mathsf{RD}_\bullet(4) \\
\mathsf{RD}_\circ(6) = \mathsf{RD}_\bullet(3)
\end{array}
$$

- 12 sets: $\mathsf{RD}_\circ(1), \cdots, \mathsf{RD}_\bullet(6)$
  all being subsets of $\mathbf{Var} \times \mathbf{Lab}$
- 12 equations:
  $\mathsf{RD}_j = \mathsf{F}_j(\mathsf{RD}_\circ(1), \cdots, \mathsf{RD}_\bullet(6))$
- one function:
  $\mathsf{F} : \mathcal{P}(\mathbf{Var} \times \mathbf{Lab})^{12} \to \mathcal{P}(\mathbf{Var} \times \mathbf{Lab})^{12}$
- we want the least fixed point of F — this is the best solution to the equation system

A simple iterative algorithm

- Initialisation
  $\mathsf{RD}_1 := \emptyset; \cdots; \mathsf{RD}_{12} := \emptyset;$
- Iteration
  `while` $\mathsf{RD}_j \neq \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$ for some j
  `do`
  $\mathsf{RD}_j := \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$

The algorithm terminates and computes the least fixed point of F.

The example equations

$\mathsf{RD}_\circ$	1	2	3	4	5	6
0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$\mathsf{RD}_\bullet$	1	2	3	4	5	6
0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

The equations:

$$
\begin{array}{l}
\mathsf{RD}_\bullet(1) = \mathsf{RD}_\circ(1) \setminus \{(\mathtt{y}, \ell) \mid \cdots\} \cup \{(\mathtt{y}, 1)\} \\
\mathsf{RD}_\bullet(2) = \mathsf{RD}_\circ(2) \setminus \{(\mathtt{z}, \ell) \mid \cdots\} \cup \{(\mathtt{z}, 2)\} \\
\mathsf{RD}_\bullet(3) = \mathsf{RD}_\circ(3) \\
\mathsf{RD}_\bullet(4) = \mathsf{RD}_\circ(4) \setminus \{(\mathtt{z}, \ell) \mid \cdots\} \cup \{(\mathtt{z}, 4)\} \\
\mathsf{RD}_\bullet(5) = \mathsf{RD}_\circ(5) \setminus \{(\mathtt{y}, \ell) \mid \cdots\} \cup \{(\mathtt{y}, 5)\} \\
\mathsf{RD}_\bullet(6) = \mathsf{RD}_\circ(6) \setminus \{(\mathtt{y}, \ell) \mid \cdots\} \cup \{(\mathtt{y}, 6)\}
\end{array}
$$

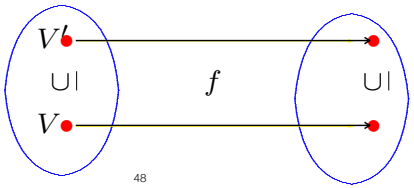
$$
\begin{array}{l}
\mathsf{RD}_\circ(1) = \{(\mathtt{x}, ?), (\mathtt{y}, ?), (\mathtt{z}, ?)\} \\
\mathsf{RD}_\circ(2) = \mathsf{RD}_\bullet(1) \\
\mathsf{RD}_\circ(3) = \mathsf{RD}_\bullet(2) \cup \mathsf{RD}_\bullet(5) \\
\mathsf{RD}_\circ(4) = \mathsf{RD}_\bullet(3) \\
\mathsf{RD}_\circ(5) = \mathsf{RD}_\bullet(4) \\
\mathsf{RD}_\circ(6) = \mathsf{RD}_\bullet(3)
\end{array}
$$

Why does it work? (1)

A function $f : \mathcal{P}(S) \to \mathcal{P}(S)$ is a monotone function if

$$V \subseteq V' \quad \Rightarrow \quad f(V) \subseteq f(V')$$

(the larger the argument – the larger the result)



Why does it work? (2)

A set L equipped with an ordering $\subseteq$ satisfies the Ascending Chain Condition if all chains

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq V_3 \subseteq \cdots$$

stabilise, that is, if there exists some n such that $V_n = V_{n+1} = V_{n+2} = \cdots$

If S is a finite set then $\mathcal{P}(S)$ equipped with the subset ordering $\subseteq$ satisfies the Ascending Chain Condition — the chains cannot grow forever since each element is a subset of a finite set.

Fact
For a given program $\mathbf{Var} \times \mathbf{Lab}$ will be a finite set so $\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})$ with the subset ordering satisfies the Ascending Chain Condition.

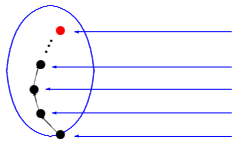
Why does it work? (3)

Let $f : \mathcal{P}(S) \to \mathcal{P}(S)$ be a monotone function. Then

$$\emptyset \subseteq f(\emptyset) \subseteq f^2(\emptyset) \subseteq f^3(\emptyset) \subseteq \cdots$$

Assume that S is a finite set; then the Ascending Chain Condition is satisfied. This means that the chain cannot be growing infinitely so there exists n such that $f^n(\emptyset) = f^{n+1}(\emptyset) = \cdots$

$f^n(\emptyset)$ is the least fixed point of f



Correctness of the algorithm

- Initialisation
  $\mathsf{RD}_1 := \emptyset; \cdots; \mathsf{RD}_{12} := \emptyset;$
  Invariant: $\vec{\mathsf{RD}} \subseteq \mathsf{F}^n(\vec{\emptyset})$ since $\vec{\mathsf{RD}} = \vec{\emptyset}$ is the least element

- Iteration
  `while` $\mathsf{RD}_j \neq \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$ for some j
  `do` assume $\vec{\mathsf{RD}}$ is $\vec{\mathsf{RD}}'$ and $\vec{\mathsf{RD}}' \subseteq \mathsf{F}^n(\vec{\emptyset})$
  $\mathsf{RD}_j := \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$
  then $\vec{\mathsf{RD}} \subseteq \mathsf{F}(\vec{\mathsf{RD}}') \subseteq \mathsf{F}^{n+1}(\vec{\emptyset}) = \mathsf{F}^n(\vec{\emptyset})$ when $\mathsf{lfp}(\mathsf{F}) = \mathsf{F}^n(\vec{\emptyset})$

If the algorithm terminates then it computes the least fixed point of F.

The algorithm terminates because $\mathsf{RD}_j \subset \mathsf{F}_j(\mathsf{RD}_1, \cdots, \mathsf{RD}_{12})$ is only possible finitely many times since $\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})^{12}$ satisfies the Ascending Chain Condition.

Intermezzo Static Analysis at Google


contributed articles

For a static analysis project to succeed, developers must feel they benefit from and enjoy using it.

BY CAITLIN SADOWSKI, EDWARD AFTANDILIAN, ALEX EAGLE, LIAM MILLER-CUSHON, AND CIERA JASPAN

Lessons from Building Static Analysis Tools at Google

Key insights

Two-billion line of code
Single Git repository
Bazel build system
C++, Java, Python, Go...

Programmers don't use static analysis because:

Not integrated into the developer's workflow
Not actionable.
Not trustworthy due to false positives
Not manifest in practice.
Too expensive to fix.
Warnings not understood.

Sadowski, Aftandilian, Eagle, Miller-Cushon, Jaspan. *Lessons from Building Static Analysis Tools at Google.* CACM'18. DOI: 10.1145/3188720

Intermezzo Static Analysis at Google

Lessons from FindBugs: a "failure" in three acts


David Hovemeyer


Bill Pugh

- Act 1: ***Bug dashboard*** (2006) FindBugs as centralized tool, nightly runs over the entire Google Java codebase, produced database of bugs. Unused because dashboard outside developers' workflow, and hard to find "new" bugs
- Act 2. ***Filing bugs.*** (2008-2009) Manually triage of issues found by nightly runs & filing bug reports for important ones. In May 2009, 100s of engineers did a "Fixit" week. Reviewed 3,954 warnings, only 16% (640) were fixed, despite that 44% (1,746) resulted in bug reports. Many bug not important enough and manual triage does not scale
- Act 3. ***Code review integration.*** (2010-2011) Run when changes sent for review, posting results as comments on code-review thread. The tooling attempted to show only new warnings but sometimes miscategorized issues as new. Discontinued when the code-review tool was replaced because: false positives caused loss of confidence

C Sadowski, E Aftandilian, A Eagle, L Miller-Cushon, C Jaspan. *Lessons from Building Static Analysis Tools at Google*. CACM'18. DOI: 10.1145/3188720

Intermezzo Static Analysis at Google

Things that worked…

- Thing #1: ***Compiler workflow.*** (1) Implement new compiler checks, (2) run updated compiler over entire Google codebase, (3) programmatically fix all problems in the codebase, (4) once the codebase was cleansed of an issue, the new diagnostic becomes a compiler error and offer an automatic patch. Survey show 57% of developers like the patches, only 2% unhappy.

- Thing #2. ***Code review checks.*** Integration into Google's code review tool, Critique; static analysis results exposed using Tricorder, Google's program-analysis platform. A code review check should: *Be understandable. Be actionable and easy to fix.* The fix may require more time than a compiler check. *Produce less than 10% effective false positives*. *Have the potential for significant impact on code quality.*

C Sadowski, E Aftandilian, A Eagle, L Miller-Cushon, C Jaspan. *Lessons from Building Static Analysis Tools at Google*. CACM'18. DOI: 10.1145/3188720

Intermezzo Static Analysis at Google

No interprocedural or whole-program analysis

The types of analyses deployed:

- ***Style checkers:*** Checkstyle, Pylint, and Golint…
- ***Bug-finding tools that extend compilers:*** Error Prone, ClangTidy, Clang Thread Safety Analysis, Govet, and the Checker Framework…, abstract-syntax-tree pattern-match tools, type-based checks, and unused variable analysis
- ***Analyzers that make calls to production services:*** e.g check if an employee mentioned in a comment is still at Google
- ***Analyzers that examine properties of build outputs:*** e.g the size of binaries

C Sadowski, E Aftandilian, A Eagle, L Miller-Cushon, C Jaspan. *Lessons from Building Static Analysis Tools at Google*. CACM'18. DOI: 10.1145/3188720

Constraint-Based Analysis

Example: Control Flow Analysis

```
let f = fn x => x 7
    g = fn y => y
    h = fn z => 3
in f g + f (g h)
   ↓       ↓
   g 7     f h
           ↓
           h 7
```

Aim: For each function application, which function abstractions may be applied?

function applications	function abstractions that may be applied
`x 7`	`g, h`
`f g`	`f`
`g h`	`g`
`f (g h)`	`f`

An application of control flow analysis

```
let f = fn x => x 7
    g = fn y => y
    h = fn z => 3
in f g + f (g h)
```

Aim: For each function application, which function abstractions may be applied?

Partial evaluation of function call:

```
let f = fn x => case x of
                  g: 7
                  h: 3
    g = fn y => y
    h = fn z => 3
in f g + f (g h)
```

function applications	function abstractions that may be applied
x 7	g, h
f g	f
g h	g
f (g h)	f

The underlying analysis problem

```
let f = fn x => x 7
    g = fn y => y
    h = fn z => 3
in f g + f (g h)
```

Aim: for each function application, which function abstractions may be applied?

The analysis will compute:

- for each subexpression, which function abstractions may it denote?
  — e.g. `(g h)` may evaluate to `h`
  introduce abstract cache C
- for each variable, which function abstractions may it denote?
  — e.g. `x` may be `g` or `h`
  introduce abstract environment R

The best solution to the analysis problem

Add labels to subexpressions:

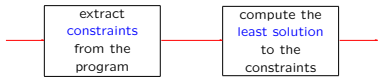
```
let f = fn x => (x^1 7^2)^3
    g = fn y => y^4
    h = fn z => 3^5
in (f^6 g^7)^8 + (f^9 (g^10 h^11)^12)^13
```

R	variable may be bound to
x	{fn y => y^4, fn z => 3^5}
y	{fn z => 3^5}
z	∅
f	{fn x => (x^1 7^2)^3}
g	{fn y => y^4}
h	{fn z => 3^5}

C	subexpression may evaluate to
1	{fn y => y^4, fn z => 3^5}
2	∅
3	∅
4	{fn z => 3^5}
5	∅
6	{fn x => (x^1 7^2)^3}
7	{fn y => y^4}
8	∅
9	{fn x => (x^1 7^2)^3}
10	{fn y => y^4}
11	{fn z => 3^5}
12	{fn z => 3^5}
13	∅

Constraint-Based Analysis

How to automate the analysis



Analysis information:

- $R(x)$: information available for the variable x
- $C(\ell)$: information available at the subexpression labelled ℓ

Three kinds of constraints

- `let`-bound variables evaluate to their abstraction
- variables evaluate to their (abstract) values
- if a function abstraction is applied to an argument then
  - the argument is a possible value of the formal parameter
  - the value of the body of the abstraction is a possible value of the application

let-bound variables

let-bound variables evaluate to their abstractions

let f = fn x => e gives rise to the constraint $\{\texttt{fn}\ x\ \texttt{=>}\ e\} \subseteq \mathsf{R}(f)$

```
let f = fn x => (x1 72)3      ⟵   {fn x => (x1 72)3} ⊆ R(f)
    g = fn y => y4            ⟵   {fn y => y4} ⊆ R(g)
in (f5 g6)7
```


Variables

Variables evaluate to their abstract values

x^ℓ gives rise to the constraint $\mathsf{R}(x) \subseteq \mathsf{C}(\ell)$

```
let f = fn x => (x1 72)3      ⟵   R(x) ⊆ C(1)
    g = fn y => y4            ⟵   R(y) ⊆ C(4)
in (f5 g6)7                   ⟵   R(f) ⊆ C(5)
                                  R(g) ⊆ C(6)
```


Function application (1)

if a function abstraction is applied to an argument then

- the argument is a possible value of the formal parameter
- the value of the body of the abstraction is a possible value of the application

```
let f = fn x => (x1 72)3      ⟵   if (fn y => y4) ∈ C(1)
    g = fn y => y4                then C(2) ⊆ R(y) and C(4) ⊆ C(3)
in (f5 g6)7                   ⟵   if (fn x => (x1 72)3) ∈ C(1)
                                  then C(2) ⊆ R(x) and C(3) ⊆ C(3)
```

Conditional constraints

```
                              ⟵   if (fn y => y4) ∈ C(5)
                                  then C(6) ⊆ R(y) and C(4) ⊆ C(7)
                              ⟵   if (fn x => (x1 72)3) ∈ C(5)
                                  then C(6) ⊆ R(x) and C(3) ⊆ C(7)
```


Summary of constraint system

```
let f = fn x => (x1 72)3
    g = fn y => y4
in (f5 g6)7
```

$\{\texttt{fn x => } (\texttt{x}^1\ 7^2)^3\} \subseteq \mathsf{R}(\texttt{f})$
$\{\texttt{fn y => } \texttt{y}^4\} \subseteq \mathsf{R}(\texttt{g})$
$\mathsf{R}(\texttt{x}) \subseteq \mathsf{C}(1)$
$\mathsf{R}(\texttt{y}) \subseteq \mathsf{C}(4)$
$\mathsf{R}(\texttt{f}) \subseteq \mathsf{C}(5)$
$\mathsf{R}(\texttt{g}) \subseteq \mathsf{C}(6)$
$(\texttt{fn y => } \texttt{y}^4) \in \mathsf{C}(1) \Rightarrow \mathsf{C}(2) \subseteq \mathsf{R}(\texttt{y})$
$(\texttt{fn y => } \texttt{y}^4) \in \mathsf{C}(1) \Rightarrow \mathsf{C}(4) \subseteq \mathsf{C}(3)$
$(\texttt{fn x => } (\texttt{x}^1\ 7^2)^3) \in \mathsf{C}(1) \Rightarrow \mathsf{C}(2) \subseteq \mathsf{R}(\texttt{x})$
$(\texttt{fn x => } (\texttt{x}^1\ 7^2)^3) \in \mathsf{C}(1) \Rightarrow \mathsf{C}(3) \subseteq \mathsf{C}(3)$
$(\texttt{fn y => } \texttt{y}^4) \in \mathsf{C}(5) \Rightarrow \mathsf{C}(6) \subseteq \mathsf{R}(\texttt{y})$
$(\texttt{fn y => } \texttt{y}^4) \in \mathsf{C}(5) \Rightarrow \mathsf{C}(4) \subseteq \mathsf{C}(7)$
$(\texttt{fn x => } (\texttt{x}^1\ 7^2)^3) \in \mathsf{C}(5) \Rightarrow \mathsf{C}(6) \subseteq \mathsf{R}(\texttt{x})$
$(\texttt{fn x => } (\texttt{x}^1\ 7^2)^3) \in \mathsf{C}(5) \Rightarrow \mathsf{C}(3) \subseteq \mathsf{C}(7)$

- 11 sets: $\mathsf{R}(\texttt{x}), \mathsf{R}(\texttt{y}), \mathsf{R}(\texttt{f}), \mathsf{R}(\texttt{g}), \mathsf{C}(1), \cdots, \mathsf{C}(7)$; all being subsets of the set **Abstr** of function abstractions
- the constraints can be reformulated as a function: $\mathsf{F} : \mathcal{P}(\mathbf{Abstr})^{11} \to \mathcal{P}(\mathbf{Abstr})^{11}$
- we want the least fixed point of F — this is the best solution to the constraint system

$\mathcal{P}(S)$ is the set of all subsets of the set S; e.g. $\mathcal{P}(\{0,1\}) = \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$.

Constraint-Based Analysis

The constraints define a function

$\{\texttt{fn x => (x}^1\ 7^2)^3\} \subseteq \mathsf{R(f)}$
$\{\texttt{fn y => y}^4\} \subseteq \mathsf{R(g)}$
$\mathsf{R(x)} \subseteq \mathsf{C(1)}$
$\mathsf{R(y)} \subseteq \mathsf{C(4)}$
$\mathsf{R(f)} \subseteq \mathsf{C(5)}$
$\mathsf{R(g)} \subseteq \mathsf{C(6)}$
$(\texttt{fn y => y}^4) \in \mathsf{C(1)} \Rightarrow \mathsf{C(2)} \subseteq \mathsf{R(y)}$
$(\texttt{fn y => y}^4) \in \mathsf{C(1)} \Rightarrow \mathsf{C(4)} \subseteq \mathsf{C(3)}$
$(\texttt{fn x => (x}^1\ 7^2)^3) \in \mathsf{C(1)} \Rightarrow \mathsf{C(2)} \subseteq \mathsf{R(x)}$
$(\texttt{fn x => (x}^1\ 7^2)^3) \in \mathsf{C(1)} \Rightarrow \mathsf{C(3)} \subseteq \mathsf{C(3)}$
$(\texttt{fn y => y}^4) \in \mathsf{C(5)} \Rightarrow \mathsf{C(6)} \subseteq \mathsf{R(y)}$
$(\texttt{fn y => y}^4) \in \mathsf{C(5)} \Rightarrow \mathsf{C(4)} \subseteq \mathsf{C(7)}$
$(\texttt{fn x => (x}^1\ 7^2)^3) \in \mathsf{C(5)} \Rightarrow \mathsf{C(6)} \subseteq \mathsf{R(x)}$
$(\texttt{fn x => (x}^1\ 7^2)^3) \in \mathsf{C(5)} \Rightarrow \mathsf{C(3)} \subseteq \mathsf{C(7)}$

$\mathsf{F} : \mathcal{P}(\mathbf{Abstr})^{11} \to \mathcal{P}(\mathbf{Abstr})^{11}$
is defined by

$$\mathsf{F}_{\mathsf{R(f)}}(\cdots, R_f, \cdots) = \vdots\ R_f \cup \{\texttt{fn x => (x}^1\ 7^2)^3\}$$
$$\mathsf{F}_{\mathsf{C(1)}}(R_x, \cdots, C_1, \cdots) = C_1 \cup R_x$$
$$\mathsf{F}_{\mathsf{R(y)}}(\cdots, R_y, \cdots, C_1, C_2, \cdots, C_5, C_6, \cdots) = R_y \cup \{a \in C_2 \mid \texttt{fn y => y}^4 \in C_1\} \cup \{a \in C_6 \mid \texttt{fn y => y}^4 \in C_5\}$$

Constraint-Based Analysis

How to solve the constraints

writing $X_1, \cdots, X_{11}$ for $\mathsf{R(x)}, \cdots, \mathsf{R(g)}, \mathsf{C(1)}, \cdots, \mathsf{C(7)}$

- Initialisation
  $X_1 := \emptyset; \cdots; X_{11} := \emptyset;$
- Iteration
  while $X_j \neq \mathsf{F}_{\mathsf{X_j}}(X_1, \cdots, X_{11})$ for some j
  do
  $X_j := \mathsf{F}_{\mathsf{X_j}}(X_1, \cdots, X_{11})$

The algorithm terminates and computes the least fixed point of F

In practice we propagate smaller contributions than $\mathsf{F}_{\mathsf{X_j}}$, e.g. a constraint at a time.

Constraint-Based Analysis

The example constraint system

R	x	y	f	g
0	∅	∅	∅	∅
1	∅	∅	fn x => .3	∅
2	∅	∅	fn x => .3	fn y => .4
3	∅	∅	fn x => .3	fn y => .4
4	∅	∅	fn x => .3	fn y => .4
5	fn y => .4	∅	fn x => .3	fn y => .4
6	fn y => .4	∅	fn x => .3	fn y => .4

C	1	2	3	4	5	6	7
0	∅	∅	∅	∅	∅	∅	∅
1	∅	∅	∅	∅	∅	∅	∅
2	∅	∅	∅	∅	∅	∅	∅
3	∅	∅	∅	∅	fn x => .3	∅	∅
4	∅	∅	∅	∅	fn x => .3	fn y => .4	∅
5	∅	∅	∅	∅	fn x => .3	fn y => .4	∅
6	fn y => .4	∅	∅	∅	fn x => .3	fn y => .4	∅

The constraints:

$\{\texttt{fn x => .3}\} \subseteq \mathsf{R(f)}$ (1)
$\{\texttt{fn y => .4}\} \subseteq \mathsf{R(g)}$ (2)
$\mathsf{R(x)} \subseteq \mathsf{C(1)}$ (6)
$\mathsf{R(y)} \subseteq \mathsf{C(4)}$
$\mathsf{R(f)} \subseteq \mathsf{C(5)}$ (3)
$\mathsf{R(g)} \subseteq \mathsf{C(6)}$ (4)
$(\texttt{fn y => .4}) \in \mathsf{C(1)} \Rightarrow \mathsf{C(2)} \subseteq \mathsf{R(y)}$
$(\texttt{fn y => .4}) \in \mathsf{C(1)} \Rightarrow \mathsf{C(4)} \subseteq \mathsf{C(3)}$
$(\texttt{fn x => .3}) \in \mathsf{C(1)} \Rightarrow \mathsf{C(2)} \subseteq \mathsf{R(x)}$
$(\texttt{fn x => .3}) \in \mathsf{C(1)} \Rightarrow \mathsf{C(3)} \subseteq \mathsf{C(3)}$
$(\texttt{fn y => .4}) \in \mathsf{C(5)} \Rightarrow \mathsf{C(6)} \subseteq \mathsf{R(y)}$
$(\texttt{fn y => .4}) \in \mathsf{C(5)} \Rightarrow \mathsf{C(4)} \subseteq \mathsf{C(7)}$
$(\texttt{fn x => .3}) \in \mathsf{C(5)} \Rightarrow \mathsf{C(6)} \subseteq \mathsf{R(x)}$ (5)
$(\texttt{fn x => .3}) \in \mathsf{C(5)} \Rightarrow \mathsf{C(3)} \subseteq \mathsf{C(7)}$

Intermezzo In Defence of Soundiness

An argument in favor of 2 + 2 = 5


viewpoints

Viewpoint
In Defense of Soundiness: A Manifesto

Soundy

Analyses are expected to be sound, i.e. their result models all possible executions of the target program, or an over-approximation to stay tractable.

In practice, soundness is commonly eschewed: all realistic whole-program analysis tool purposely make unsound choices. Most/all published whole-program analyses omit conservative handling of language features of real languages.

Some specific features are best under-approximated by pretending that perfectly possible behaviors cannot happen. E.g. highly dynamic language constructs, such as eval, may be assumed to do nothing

We term such analyses as soundy. The concept of soundiness attempts to capture the balance of over-approximated handling of most language features, yet deliberately under-approximated handling of a feature subset

Livshits, Sridharan, Smaragdakis, Lhoták, Amaral, Evan Chang, Guyer, Khedker, Møller, Vardoulakis. *In Defense of Soundiness: A Manifesto*. CACM'15. DOI: 10.1145/2644805

Intermezzo In Defence of Soundiness

An argument in favor of 2 + 2 = 5

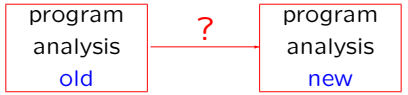
Some of the sources of unsoundness, sorted by language.

Language	Examples of commonly ignored features	Consequences of not modeling these features
C/C++	setjmp/longjmp ignored effects of pointer arithmetic "manufactured" pointers	ignores arbitrary side effects to the program heap
Java/C#	Reflection JNI	can render much of the codebase invisible for analysis "invisible" code may create invisible side effects in programs
JavaScript	eval, dynamic code loading data flow through the DOM	missing execution missing data flow in program

Read: "*A True Positives Theorem for a Static Race Detector*" POPL'19.

"We prove a True Positives Theorem stating that under certain assumptions, which reflect the way that product code uses concurrency, an idealized theoretical version of the analysis never reports a false positive.
The theorem was motivated by the desire to understand the observation that our tool was providing remarkably accurate signal to developers.
Technically, our result can be seen as saying that the analysis computes an under-approximation of an over-approximation, which is the reverse of the more usual situation in static analysis."
2+2=4 under some conditions.

Abstract Interpretation

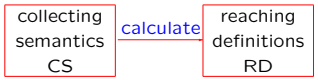


- We have the analysis old: it has already been proved correct but it is inefficient, or maybe even uncomputable
- We want the analysis new: it has to be correct as well as efficient!
- Can we develop new from old?

abstract interpretation !

Abstract Interpretation

Example: Collecting Semantics and Reaching Definitions



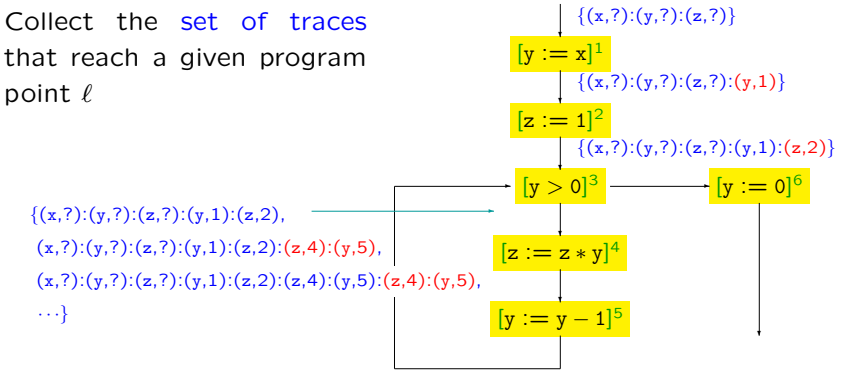
The collecting semantics CS

- collects the set of traces that can reach a given program point
- has an easy correctness proof
- is uncomputable

The reaching definitions analysis RD is as before

Example: Collecting Semantics

Collect the set of traces that reach a given program point ℓ



Abstract Interpretation

How to proceed

As before:

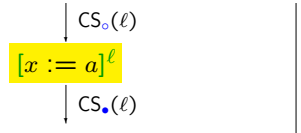
- extract a set of equations defining the possible sets of traces
- compute the least fixed point of the set of equations

And furthermore:

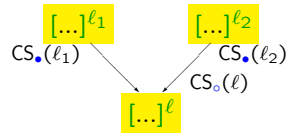
- prove the correctness: the set of traces computed by the analysis is a superset of the possible traces

Abstract Interpretation

Two kinds of equations



$$\{trace : (x, \ell) \mid trace \in \mathsf{CS}_\circ(\ell)\} = \mathsf{CS}_\bullet(\ell)$$



$$\mathsf{CS}_\bullet(\ell_1) \cup \mathsf{CS}_\bullet(\ell_2) = \mathsf{CS}_\circ(\ell)$$

Flow through assignments and tests

```
[y := x]^1;
[z := 1]^2;
while [y > 0]^3 do
   [z := z * y]^4;
   [y := y - 1]^5
od;
[y := 0]^6
```

$\mathsf{CS}_\bullet(1) = \{trace : (y, 1) \mid trace \in \mathsf{CS}_\circ(1)\}$

$\mathsf{CS}_\bullet(2) = \{trace : (z, 2) \mid trace \in \mathsf{CS}_\circ(2)\}$

$\mathsf{CS}_\bullet(3) = \mathsf{CS}_\circ(3)$

$\mathsf{CS}_\bullet(4) = \{trace : (z, 4) \mid trace \in \mathsf{CS}_\circ(4)\}$

$\mathsf{CS}_\bullet(5) = \{trace : (y, 5) \mid trace \in \mathsf{CS}_\circ(5)\}$

$\mathsf{CS}_\bullet(6) = \{trace : (y, 6) \mid trace \in \mathsf{CS}_\circ(6)\}$

6 equations in $\mathsf{CS}_\circ(1), \cdots, \mathsf{CS}_\bullet(6)$

Flow along the control

```
[y := x]^1;
[z := 1]^2;
while [y > 0]^3 do
   [z := z * y]^4;
   [y := y - 1]^5
od;
[y := 0]^6
```

$\mathsf{CS}_\circ(1) = \{(x, ?) : (y, ?) : (z, ?)\}$

$\mathsf{CS}_\circ(2) = \mathsf{CS}_\bullet(1)$

$\mathsf{CS}_\circ(3) = \mathsf{CS}_\bullet(2) \cup \mathsf{CS}_\bullet(5)$

$\mathsf{CS}_\circ(4) = \mathsf{CS}_\bullet(3)$

$\mathsf{CS}_\circ(5) = \mathsf{CS}_\bullet(4)$

$\mathsf{CS}_\circ(6) = \mathsf{CS}_\bullet(3)$

6 equations in $\mathsf{CS}_\circ(1), \cdots, \mathsf{CS}_\bullet(6)$

Summary of Collecting Semantics

$\mathsf{CS}_\bullet(1) = \{trace : (\mathsf{y}, 1) \mid trace \in \mathsf{CS}_\circ(1)\}$
$\mathsf{CS}_\bullet(2) = \{trace : (\mathsf{z}, 2) \mid trace \in \mathsf{CS}_\circ(2)\}$
$\mathsf{CS}_\bullet(3) = \mathsf{CS}_\circ(3)$
$\mathsf{CS}_\bullet(4) = \{trace : (\mathsf{z}, 4) \mid trace \in \mathsf{CS}_\circ(4)\}$
$\mathsf{CS}_\bullet(5) = \{trace : (\mathsf{y}, 5) \mid trace \in \mathsf{CS}_\circ(5)\}$
$\mathsf{CS}_\bullet(6) = \{trace : (\mathsf{y}, 6) \mid trace \in \mathsf{CS}_\circ(6)\}$

$\mathsf{CS}_\circ(1) = \{(\mathsf{x}, ?) : (\mathsf{y}, ?) : (\mathsf{z}, ?)\}$
$\mathsf{CS}_\circ(2) = \mathsf{CS}_\bullet(1)$
$\mathsf{CS}_\circ(3) = \mathsf{CS}_\bullet(2) \cup \mathsf{CS}_\bullet(5)$
$\mathsf{CS}_\circ(4) = \mathsf{CS}_\bullet(3)$
$\mathsf{CS}_\circ(5) = \mathsf{CS}_\bullet(4)$
$\mathsf{CS}_\circ(6) = \mathsf{CS}_\bullet(3)$

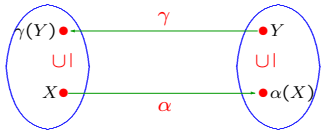
- 12 sets: $\mathsf{CS}_\circ(1), \cdots, \mathsf{CS}_\bullet(6)$
  all being subsets of $\mathbf{Trace}$
- 12 equations:
  $\mathsf{CS}_j = \mathsf{G}_j(\mathsf{CS}_\circ(1), \cdots, \mathsf{CS}_\bullet(6))$
- one function:
  $\mathsf{G} : \mathcal{P}(\mathbf{Trace})^{12} \to \mathcal{P}(\mathbf{Trace})^{12}$
- we want the least fixed point of G — but it is uncomputable!

Example: Inducing an analysis

Galois Connections

A Galois connection between two sets is a pair of (α, γ) of functions between the sets satisfying

$$X \subseteq \gamma(Y) \quad \Leftrightarrow \quad \alpha(X) \subseteq Y$$



$\mathcal{P}(\mathbf{Trace})$ collecting semantics

$\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})$ reaching definitions

α: abstraction function
γ: concretisation function

Semantically Reaching Definitions

For a single trace:

trace: (x,?):(y,?):(z,?):(y,1):(z,2)

SRD(*trace*): {(x,?),(y,1),(z,2)}

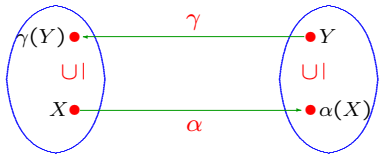
For a set of traces:

$X \in \mathcal{P}(\mathbf{Trace})$: {(x,?):(y,?):(z,?):(y,1):(z,2),
(x,?):(y,?):(z,?):(y,1):(z,2):(z,4):(y,5)}

SRD(X): {(x,?),(y,1),(z,2),(z,4),(y,5)}

Galois connection for Reaching Definitions analysis

$$\alpha(X) = \mathsf{SRD}(X)$$
$$\gamma(Y) = \{trace \mid \mathsf{SRD}(trace) \subseteq Y\}$$



$\mathcal{P}(\mathbf{Trace})$ $\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})$

Galois connection:

$$X \subseteq \gamma(Y) \quad \Leftrightarrow \quad \alpha(X) \subseteq Y$$

Abstract Interpretation

Inducing the Reaching Definitions analysis (1)

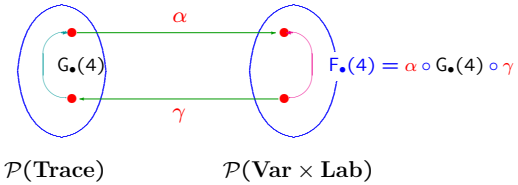
Known:

- $G_\bullet(4)$ defined on $\mathcal{P}(\mathbf{Trace})$
- the Galois connection (α, γ)

Calculate:

- $F_\bullet(4)$ defined on $\mathcal{P}(\mathbf{Var} \times \mathbf{Lab})$

  as $F_\bullet(4) = \alpha \circ G_\bullet(4) \circ \gamma$



Inducing the Reaching Definitions analysis (2)

$$
\begin{aligned}
RD_\bullet(4) &= F_\bullet(4)(\cdots, RD_\circ(4), \cdots) \\
&= \alpha(G_\bullet(4)(\gamma(\cdots, RD_\circ(4), \cdots))) \quad \text{using } F_\bullet(4) = \alpha \circ G_\bullet(4) \circ \gamma \\
&= \alpha(\{tr : (z, 4) \mid tr \in \gamma(RD_\circ(4))\}) \\
&\quad \text{using } G_\bullet(4)(\cdots, CS_\circ(4), \cdots) = \{tr : (z, 4) \mid tr \in CS_\circ(4)\} \\
&= SRD(\{tr : (z, 4) \mid tr \in \gamma(RD_\circ(4))\}) \quad \text{using } \alpha = SRD \\
&= (SRD(\{tr \mid tr \in \gamma(RD_\circ(4))\}) \backslash \{(z, \ell) \mid \ell \in \mathbf{Lab}\}) \cup \{(z, 4)\} \\
&= (\alpha(\gamma(RD_\circ(4))) \backslash \{(z, \ell) \mid \ell \in \mathbf{Lab}\}) \cup \{(z, 4)\} \quad \text{using } \alpha = SRD \\
&= (RD_\circ(4) \backslash \{(z, \ell) \mid \ell \in \mathbf{Lab}\}) \cup \{(z, 4)\} \quad \text{using } \alpha \circ \gamma = id
\end{aligned}
$$

just as before!

Scaling Program Analysis

CS7575

- Lecture 2 — Data flow analysis; PoPA Chapter 2 (part 1)

Data flow analysis

Characterizes the *flow* of analysis facts through a program

Analyses differ in aspects such as:

- dependance on statement ordering wrt control flow operations

  flow-sensitive vs. *flow-insensitive*
- direction of flow

  forward vs. *backward*
- qualification over paths

  may vs. *must*
- scope

  interprocedural vs *intraprocedural*

Intraprocedural data flow analysis

Classical analyses:

- Available Expression
- Reaching Definition
- Very Busy Expression
- Live Variables

Derived analyses:

- Use-definition
- Definition-use

While language

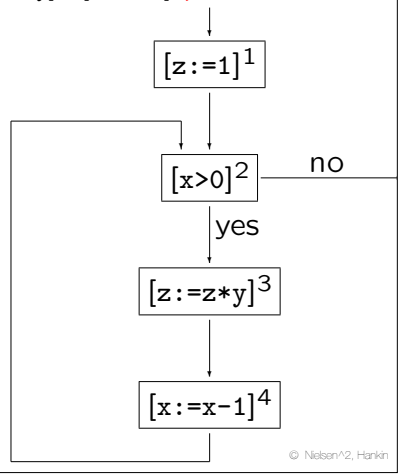
Abstract syntax

$$
\begin{array}{lcl}
a & ::= & x \mid n \mid a_1\ op_a\ a_2 \\
b & ::= & \texttt{true} \mid \texttt{false} \mid \texttt{not}\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2 \\
S & ::= & [x := a]^\ell \mid [\texttt{skip}]^\ell \mid S_1; S_2 \mid \\
 & & \texttt{if}\ [b]^\ell\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \texttt{while}\ [b]^\ell\ \texttt{do}\ S
\end{array}
$$

Example $[\texttt{z:=1}]^1; \texttt{while}\ [\texttt{x>0}]^2\ \texttt{do}\ ([\texttt{z:=z*y}]^3; [\texttt{x:=x-1}]^4)$

$[\texttt{z:=1}]^1; \texttt{while}\ [\texttt{x>0}]^2\ \texttt{do}\ ([\texttt{z:=z*y}]^3; [\texttt{x:=x-1}]^4)$

$$
\begin{array}{lcl}
init(\cdots) & = & 1 \\
final(\cdots) & = & \{2\} \\
labels(\cdots) & = & \{1,2,3,4\} \\
flow(\cdots) & = & \{(1,2),(2,3),\\ & & (3,4),(4,2)\} \\
flow^R(\cdots) & = & \{(2,1),(2,4),\\ & & (3,2),(4,3)\}
\end{array}
$$



Helper functions

$init(S)$ is the label of the first elementary block of S:

$$init : \mathbf{Stmt} \to \mathbf{Lab}$$

$$
\begin{array}{rcl}
init([x := a]^\ell) & = & \ell \\
init([\texttt{skip}]^\ell) & = & \ell \\
init(S_1; S_2) & = & init(S_1) \\
init(\texttt{if}\ [b]^\ell\ \texttt{then}\ S_1\ \texttt{else}\ S_2) & = & \ell \\
init(\texttt{while}\ [b]^\ell\ \texttt{do}\ S) & = & \ell
\end{array}
$$

$final(S)$ is the set of labels of the last elementary blocks of S:

$$final : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab})$$

$$
\begin{array}{rcl}
final([x := a]^\ell) & = & \{\ell\} \\
final([\texttt{skip}]^\ell) & = & \{\ell\} \\
final(S_1; S_2) & = & final(S_2) \\
final(\texttt{if}\ [b]^\ell\ \texttt{then}\ S_1\ \texttt{else}\ S_2) & = & final(S_1) \cup final(S_2) \\
final(\texttt{while}\ [b]^\ell\ \texttt{do}\ S) & = & \{\ell\}
\end{array}
$$

Helper functions

$\mathit{flow}(S)$ and $\mathit{flow}^R(S)$ are representations of how control flows in S:

$$\mathit{flow}, \mathit{flow}^R : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab} \times \mathbf{Lab})$$

$$\begin{aligned}
\mathit{flow}([x := a]^\ell) &= \emptyset \\
\mathit{flow}([\texttt{skip}]^\ell) &= \emptyset \\
\mathit{flow}(S_1; S_2) &= \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \\
&\quad \cup \{(\ell, \mathit{init}(S_2)) \mid \ell \in \mathit{final}(S_1)\} \\
\mathit{flow}(\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2) &= \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \\
&\quad \cup \{(\ell, \mathit{init}(S_1)), (\ell, \mathit{init}(S_2))\} \\
\mathit{flow}(\texttt{while } [b]^\ell \texttt{ do } S) &= \mathit{flow}(S) \cup \{(\ell, \mathit{init}(S))\} \\
&\quad \cup \{(\ell', \ell) \mid \ell' \in \mathit{final}(S)\} \\
\\
\mathit{flow}^R(S) &= \{(\ell, \ell') \mid (\ell', \ell) \in \mathit{flow}(S)\}
\end{aligned}$$

Helper functions

A statement consists of a set of *elementary blocks*

$$\mathit{blocks} : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Blocks})$$

$$\begin{aligned}
\mathit{blocks}([\texttt{x} := a]^\ell) &= \{[\texttt{x} := a]^\ell\} \\
\mathit{blocks}([\texttt{skip}]^\ell) &= \{[\texttt{skip}]^\ell\} \\
\mathit{blocks}(S_1; S_2) &= \mathit{blocks}(S_1) \cup \mathit{blocks}(S_2) \\
\mathit{blocks}(\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2) &= \{[b]^\ell\} \cup \mathit{blocks}(S_1) \cup \mathit{blocks}(S_2) \\
\mathit{blocks}(\texttt{while } [b]^\ell \texttt{ do } S) &= \{[b]^\ell\} \cup \mathit{blocks}(S)
\end{aligned}$$

A statement S is *label consistent* if and only if any two elementary statements $[S_1]^\ell$ and $[S_2]^\ell$ with the same label in S are equal: $S_1 = S_2$

A statement *where all labels are unique* is automatically label consistent

Available Expression Analysis

For each point, which expressions *must* be computed, and not later modified, on all paths.

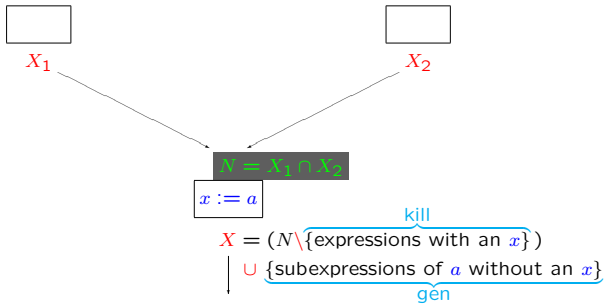
Example:

point of interest

$[\texttt{x:=a+b}]^1; [\texttt{y:=a*b}]^2; \texttt{while } [\texttt{y>a+b}]^3 \texttt{ do } ([\texttt{a:=a+1}]^4; [\texttt{x:=a+b}]^5)$

For each point, which expressions *must* be computed, and not later modified, on all paths.

$[\texttt{x:=a+b}]^1; [\texttt{y:=a*b}]^2; \texttt{while } [\texttt{y>a+b}]^3 \texttt{ do } ([\texttt{a:=a+1}]^4; [\texttt{x:=a+b}]^5)$



$N = X_1 \cap X_2$

$x := a$

kill

$X = (N \setminus \{\text{expressions with an } x\})$

$\cup$ $\{\text{subexpressions of } a \text{ without an } x\}$

gen

kill and *gen* functions

$$
\begin{array}{rcl}
kill_{\mathsf{AE}}([x := a]^\ell) & = & \{a' \in \mathbf{AExp}_\star \mid x \in FV(a')\} \\
kill_{\mathsf{AE}}([\mathtt{skip}]^\ell) & = & \emptyset \\
kill_{\mathsf{AE}}([b]^\ell) & = & \emptyset \\
gen_{\mathsf{AE}}([x := a]^\ell) & = & \{a' \in \mathbf{AExp}(a) \mid x \notin FV(a')\} \\
gen_{\mathsf{AE}}([\mathtt{skip}]^\ell) & = & \emptyset \\
gen_{\mathsf{AE}}([b]^\ell) & = & \mathbf{AExp}(b)
\end{array}
$$

data flow equations: AE$^=$

$$
\mathsf{AE}_{entry}(\ell) = \begin{cases} \emptyset & \text{if } \ell = init(S_\star) \\ \bigcap\{\mathsf{AE}_{exit}(\ell') \mid (\ell', \ell) \in flow(S_\star)\} & \text{otherwise} \end{cases}
$$

$$
\mathsf{AE}_{exit}(\ell) = (\mathsf{AE}_{entry}(\ell) \setminus kill_{\mathsf{AE}}(B^\ell)) \cup gen_{\mathsf{AE}}(B^\ell) \quad \text{where } B^\ell \in blocks(S_\star)
$$

$[\mathtt{x:=a+b}]^1; [\mathtt{y:=a*b}]^2; \mathtt{while}\ [\mathtt{y>a+b}]^3\ \mathtt{do}\ ([\mathtt{a:=a+1}]^4; [\mathtt{x:=a+b}]^5)$

ℓ	$kill_{\mathsf{AE}}(\ell)$	$gen_{\mathsf{AE}}(\ell)$
1	$\emptyset$	{a+b}
2	$\emptyset$	{a*b}
3	$\emptyset$	{a+b}
4	{a+b, a*b, a+1}	$\emptyset$
5	$\emptyset$	{a+b}

kill and *gen* functions

$$
\begin{array}{rcl}
kill_{\mathsf{AE}}([x := a]^\ell) & = & \{a' \in \mathbf{AExp}_\star \mid x \in FV(a')\} \\
kill_{\mathsf{AE}}([\mathtt{skip}]^\ell) & = & \emptyset \\
kill_{\mathsf{AE}}([b]^\ell) & = & \emptyset \\
gen_{\mathsf{AE}}([x := a]^\ell) & = & \{a' \in \mathbf{AExp}(a) \mid x \notin FV(a')\} \\
gen_{\mathsf{AE}}([\mathtt{skip}]^\ell) & = & \emptyset \\
gen_{\mathsf{AE}}([b]^\ell) & = & \mathbf{AExp}(b)
\end{array}
$$

$[\mathtt{x:=a+b}]^1; [\mathtt{y:=a*b}]^2; \mathtt{while}\ [\mathtt{y>a+b}]^3\ \mathtt{do}\ ([\mathtt{a:=a+1}]^4; [\mathtt{x:=a+b}]^5)$

$$
\begin{array}{rcl}
\mathsf{AE}_{entry}(1) & = & \emptyset \\
\mathsf{AE}_{entry}(2) & = & \mathsf{AE}_{exit}(1) \\
\mathsf{AE}_{entry}(3) & = & \mathsf{AE}_{exit}(2) \cap \mathsf{AE}_{exit}(5) \\
\mathsf{AE}_{entry}(4) & = & \mathsf{AE}_{exit}(3) \\
\mathsf{AE}_{entry}(5) & = & \mathsf{AE}_{exit}(4) \\
\mathsf{AE}_{exit}(1) & = & \mathsf{AE}_{entry}(1) \cup \{\mathtt{a+b}\} \\
\mathsf{AE}_{exit}(2) & = & \mathsf{AE}_{entry}(2) \cup \{\mathtt{a*b}\} \\
\mathsf{AE}_{exit}(3) & = & \mathsf{AE}_{entry}(3) \cup \{\mathtt{a+b}\} \\
\mathsf{AE}_{exit}(4) & = & \mathsf{AE}_{entry}(4) \setminus \{\mathtt{a+b}, \mathtt{a*b}, \mathtt{a+1}\} \\
\mathsf{AE}_{exit}(5) & = & \mathsf{AE}_{entry}(5) \cup \{\mathtt{a+b}\}
\end{array}
$$

ℓ	$kill_{\mathsf{AE}}(\ell)$	$gen_{\mathsf{AE}}(\ell)$
1	$\emptyset$	{a+b}
2	$\emptyset$	{a*b}
3	$\emptyset$	{a+b}
4	{a+b, a*b, a+1}	$\emptyset$
5	$\emptyset$	{a+b}

data flow equations: AE$^=$

$$
\mathsf{AE}_{entry}(\ell) = \begin{cases} \emptyset & \text{if } \ell = init(S_\star) \\ \bigcap\{\mathsf{AE}_{exit}(\ell') \mid (\ell', \ell) \in flow(S_\star)\} & \text{otherwise} \end{cases}
$$

$$
\mathsf{AE}_{exit}(\ell) = (\mathsf{AE}_{entry}(\ell) \setminus kill_{\mathsf{AE}}(B^\ell)) \cup gen_{\mathsf{AE}}(B^\ell) \quad \text{where } B^\ell \in blocks(S_\star)
$$

$[\mathtt{x:=a+b}]^1; [\mathtt{y:=a*b}]^2; \mathtt{while}\ [\mathtt{y>a+b}]^3\ \mathtt{do}\ ([\mathtt{a:=a+1}]^4; [\mathtt{x:=a+b}]^5)$

Largest solution:

$$
\begin{array}{rcl}
\mathsf{AE}_{entry}(1) & = & \emptyset \\
\mathsf{AE}_{entry}(2) & = & \mathsf{AE}_{exit}(1) \\
\mathsf{AE}_{entry}(3) & = & \mathsf{AE}_{exit}(2) \cap \mathsf{AE}_{exit}(5) \\
\mathsf{AE}_{entry}(4) & = & \mathsf{AE}_{exit}(3) \\
\mathsf{AE}_{entry}(5) & = & \mathsf{AE}_{exit}(4) \\
\mathsf{AE}_{exit}(1) & = & \mathsf{AE}_{entry}(1) \cup \{\mathtt{a+b}\} \\
\mathsf{AE}_{exit}(2) & = & \mathsf{AE}_{entry}(2) \cup \{\mathtt{a*b}\} \\
\mathsf{AE}_{exit}(3) & = & \mathsf{AE}_{entry}(3) \cup \{\mathtt{a+b}\} \\
\mathsf{AE}_{exit}(4) & = & \mathsf{AE}_{entry}(4) \setminus \{\mathtt{a+b}, \mathtt{a*b}, \mathtt{a+1}\} \\
\mathsf{AE}_{exit}(5) & = & \mathsf{AE}_{entry}(5) \cup \{\mathtt{a+b}\}
\end{array}
$$

ℓ	$\mathsf{AE}_{entry}(\ell)$	$\mathsf{AE}_{exit}(\ell)$
1	$\emptyset$	{a+b}
2	{a+b}	{a+b, a*b}
3	{a+b}	{a+b}
4	{a+b}	$\emptyset$
5	$\emptyset$	{a+b}

Available Expression Analysis

For each point, which expressions *must* be computed, and not later modified, on all paths.

Why the largest solution?

$$
\begin{aligned}
\mathsf{AE}_{entry}(1) &= \emptyset \\
\mathsf{AE}_{entry}(2) &= \mathsf{AE}_{exit}(1) \\
\mathsf{AE}_{entry}(3) &= \mathsf{AE}_{exit}(2) \cap \mathsf{AE}_{exit}(5) \\
\mathsf{AE}_{entry}(4) &= \mathsf{AE}_{exit}(3) \\
\mathsf{AE}_{entry}(5) &= \mathsf{AE}_{exit}(4)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{AE}_{exit}(1) &= \mathsf{AE}_{entry}(1) \cup \{\mathtt{a+b}\} \\
\mathsf{AE}_{exit}(2) &= \mathsf{AE}_{entry}(2) \cup \{\mathtt{a*b}\} \\
\mathsf{AE}_{exit}(3) &= \mathsf{AE}_{entry}(3) \cup \{\mathtt{a+b}\} \\
\mathsf{AE}_{exit}(4) &= \mathsf{AE}_{entry}(4) \setminus \{\mathtt{a+b}, \mathtt{a*b}, \mathtt{a+1}\} \\
\mathsf{AE}_{exit}(5) &= \mathsf{AE}_{entry}(5) \cup \{\mathtt{a+b}\}
\end{aligned}
$$

Available Expression Analysis

For each point, which expressions *must* be computed, and not later modified, on all paths.

Why the largest solution?

$$
\begin{aligned}
\mathsf{AE}_{entry}(1) &= \emptyset \\
\mathsf{AE}_{entry}(2) &= \{\mathtt{a}+\mathtt{b}\} \\
\mathsf{AE}_{entry}(3) &= \{\mathtt{a}+\mathtt{b}, \mathtt{a}*\mathtt{b}\} \cap \mathsf{AE}_{\mathtt{exit}}(5) \\
\mathsf{AE}_{entry}(4) &= \{\mathtt{a}+\mathtt{b}, \mathtt{a}*\mathtt{b}\} \cap \mathsf{AE}_{\mathtt{exit}}(5) \cup \{\mathtt{a}+\mathtt{b}\} \\
\mathsf{AE}_{entry}(5) &= \mathsf{AE}_{\mathtt{exit}}(5) - \{\mathtt{a}+\mathtt{b}, \mathtt{a}*\mathtt{b}, \mathtt{a}+1\}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{AE}_{exit}(1) &= \{\mathtt{a}+\mathtt{b}\} \\
\mathsf{AE}_{exit}(2) &= \{\mathtt{a}+\mathtt{b}, \mathtt{a}*\mathtt{b}\} \\
\mathsf{AE}_{exit}(3) &= \{\mathtt{a}+\mathtt{b}, \mathtt{a}*\mathtt{b}\} \cap \mathsf{AE}_{\mathtt{exit}}(5) \cup \{\mathtt{a}+\mathtt{b}\} \\
\mathsf{AE}_{exit}(4) &= \mathsf{AE}_{\mathtt{exit}}(5) - \{\mathtt{a}+\mathtt{b}, \mathtt{a}*\mathtt{b}, \mathtt{a}+1\} \\
\mathsf{AE}_{exit}(5) &= \mathsf{AE}_{\mathtt{exit}}(5) - \{\mathtt{a}*\mathtt{b}, \mathtt{a}+1\} \cup \{\mathtt{a}+\mathtt{b}\}
\end{aligned}
$$

$\mathbf{AExp} = \{\mathtt{a}+\mathtt{b}, \mathtt{a}*\mathtt{b}, \mathtt{a}+1\}$

Available Expression Analysis

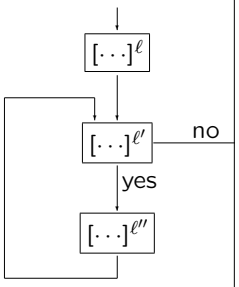
For each point, which expressions *must* be computed, and not later modified, on all paths.

$[\mathtt{z:=x+y}]^{\ell}; \mathtt{while}\ [\mathtt{true}]^{\ell'}\ \mathtt{do}\ [\mathtt{skip}]^{\ell''}$

Why the largest solution?

$$
\begin{aligned}
\mathsf{AE}_{entry}(\ell) &= \emptyset \\
\mathsf{AE}_{entry}(\ell') &= \mathsf{AE}_{exit}(\ell) \cap \mathsf{AE}_{exit}(\ell'') \\
\mathsf{AE}_{entry}(\ell'') &= \mathsf{AE}_{exit}(\ell')
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{AE}_{exit}(\ell) &= \mathsf{AE}_{entry}(\ell) \cup \{\mathtt{x+y}\} \\
\mathsf{AE}_{exit}(\ell') &= \mathsf{AE}_{entry}(\ell') \\
\mathsf{AE}_{exit}(\ell'') &= \mathsf{AE}_{entry}(\ell'')
\end{aligned}
$$



Available Expression Analysis

For each point, which expressions *must* be computed, and not later modified, on all paths.

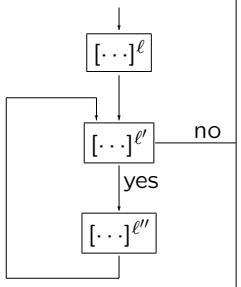
$[\mathtt{z:=x+y}]^{\ell}; \mathtt{while}\ [\mathtt{true}]^{\ell'}\ \mathtt{do}\ [\mathtt{skip}]^{\ell''}$

Why the largest solution?

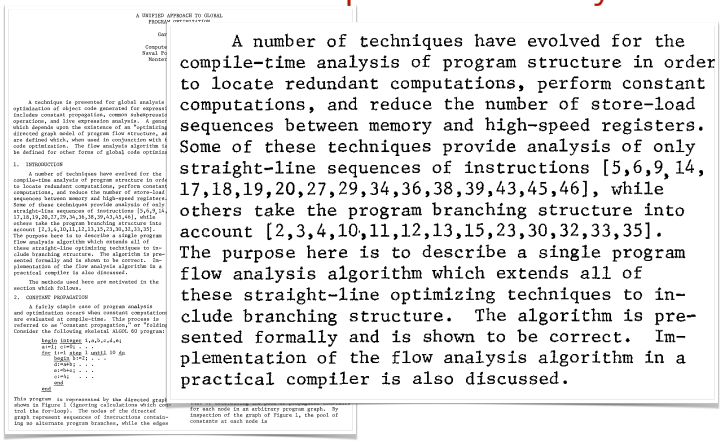
$\mathbf{AExp} = \{\mathtt{x}+\mathtt{y}\}$

$$
\begin{aligned}
\mathsf{AE}_{entry}(\ell) &= \emptyset \\
\mathsf{AE}_{entry}(\ell') &= \{\mathtt{x}+\mathtt{y}\} \cap \mathsf{AE}_{exit}(\ell'') \\
\mathsf{AE}_{entry}(\ell'') &= \{\mathtt{x}+\mathtt{y}\} \cap \mathsf{AE}_{\mathtt{exit}}(\ell'')
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{AE}_{exit}(\ell) &= \{\mathtt{x}+\mathtt{v}\} \\
\mathsf{AE}_{exit}(\ell') &= \{\mathtt{x}+\mathtt{y}\} \cap \mathsf{AE}_{\mathtt{exit}}(\ell'') \\
\mathsf{AE}_{exit}(\ell'') &= \{\mathtt{x}+\mathtt{y}\} \cap \mathsf{AE}_{\mathtt{exit}}(\ell'')
\end{aligned}
$$



Retrospective: Gary Kildall



A number of techniques have evolved for the compile-time analysis of program structure in order to locate redundant computations, perform constant computations, and reduce the number of store-load sequences between memory and high-speed registers. Some of these techniques provide analysis of only straight-line sequences of instructions [5,6,9,14, 17,18,19,20,27,29,34,36,38,39,43,45,46], while others take the program branching structure into account [2,3,4,10,11,12,13,15,23,30,32,33,35]. The purpose here is to describe a single program flow analysis algorithm which extends all of these straight-line optimizing techniques to include branching structure. The algorithm is presented formally and is shown to be correct. Implementation of the flow analysis algorithm in a practical compiler is also discussed.

Gary Kildall

G. Kildall. *A unified approach to global program optimization.* POPL73 doi=10.1145/512927.512945

Reaching Definition Analysis

For each point, which assignments *may* have been made, and not overwritten, on some paths.

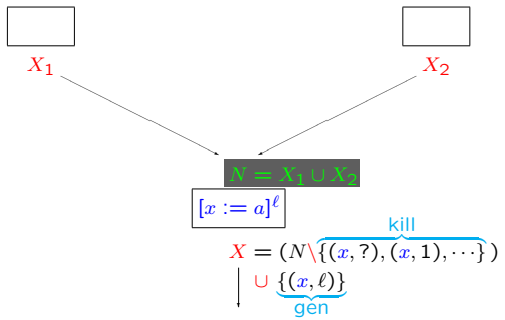
Example:

point of interest ⇓

$[\texttt{x:=5}]^1; [\texttt{y:=1}]^2; \texttt{while } [\texttt{x>1}]^3 \texttt{ do } ([\texttt{y:=x*y}]^4; [\texttt{x:=x-1}]^5)$

For each point, which assignments *may* have been made, and not overwritten, on some paths.

$[\texttt{x:=5}]^1; [\texttt{y:=1}]^2; \texttt{while } [\texttt{x>1}]^3 \texttt{ do } ([\texttt{y:=x*y}]^4; [\texttt{x:=x-1}]^5)$



$N = X_1 \cup X_2$

$[x := a]^\ell$

$X = (N \setminus \overbrace{\{(x,?),(x,1),\cdots\}}^{\text{kill}}) \cup \overbrace{\{(x,\ell)\}}^{\text{gen}}$

kill and *gen* functions

$$
\begin{aligned}
kill_{\mathsf{RD}}([x := a]^\ell) &= \{(x,?)\} \cup \{(x,\ell') \mid B^{\ell'} \text{ is an assignment to } x \text{ in } S_\star\} \\
kill_{\mathsf{RD}}([\texttt{skip}]^\ell) &= \emptyset \\
kill_{\mathsf{RD}}([b]^\ell) &= \emptyset \\
gen_{\mathsf{RD}}([x := a]^\ell) &= \{(x,\ell)\} \\
gen_{\mathsf{RD}}([\texttt{skip}]^\ell) &= \emptyset \\
gen_{\mathsf{RD}}([b]^\ell) &= \emptyset
\end{aligned}
$$

data flow equations: $\mathsf{RD}^=$

$$
\mathsf{RD}_{entry}(\ell) = \begin{cases} \{(x,?) \mid x \in FV(S_\star)\} & \text{if } \ell = init(S_\star) \\ \cup\{\mathsf{RD}_{exit}(\ell') \mid (\ell',\ell) \in flow(S_\star)\} & \text{otherwise} \end{cases}
$$

$$
\mathsf{RD}_{exit}(\ell) = (\mathsf{RD}_{entry}(\ell) \setminus kill_{\mathsf{RD}}(B^\ell)) \cup gen_{\mathsf{RD}}(B^\ell) \quad \text{where } B^\ell \in blocks(S_\star)
$$

For each point, which assignments *may* have been made, and not overwritten, on some paths.

$[x:=5]^1; [y:=1]^2; \texttt{while } [x>1]^3 \texttt{ do } ([y:=x*y]^4; [x:=x-1]^5)$

ℓ	$kill_{RD}(\ell)$	$gen_{RD}(\ell)$
1	$\{(x,?),(x,1),(x,5)\}$	$\{(x,1)\}$
2	$\{(y,?),(y,2),(y,4)\}$	$\{(y,2)\}$
3	$\emptyset$	$\emptyset$
4	$\{(y,?),(y,2),(y,4)\}$	$\{(y,4)\}$
5	$\{(x,?),(x,1),(x,5)\}$	$\{(x,5)\}$

$RD_{entry}(1) = \{(x,?),(y,?)\}$
$RD_{entry}(2) = RD_{exit}(1)$
$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$
$RD_{entry}(4) = RD_{exit}(3)$
$RD_{entry}(5) = RD_{exit}(4)$

$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(x,?),(x,1),(x,5)\}) \cup \{(x,1)\}$
$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(y,?),(y,2),(y,4)\}) \cup \{(y,2)\}$
$RD_{exit}(3) = RD_{entry}(3)$
$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(y,?),(y,2),(y,4)\}) \cup \{(y,4)\}$
$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(x,?),(x,1),(x,5)\}) \cup \{(x,5)\}$

For each point, which assignments *may* have been made, and not overwritten, on some paths.

$[x:=5]^1; [y:=1]^2; \texttt{while } [x>1]^3 \texttt{ do } ([y:=x*y]^4; [x:=x-1]^5)$

Smallest solution:

ℓ	$RD_{entry}(\ell)$	$RD_{exit}(\ell)$
1	$\{(x,?),(y,?)\}$	$\{(y,?),(x,1)\}$
2	$\{(y,?),(x,1)\}$	$\{(x,1),(y,2)\}$
3	$\{(x,1),(y,2),(y,4),(x,5)\}$	$\{(x,1),(y,2),(y,4),(x,5)\}$
4	$\{(x,1),(y,2),(y,4),(x,5)\}$	$\{(x,1),(y,4),(x,5)\}$
5	$\{(x,1),(y,4),(x,5)\}$	$\{(y,4),(x,5)\}$

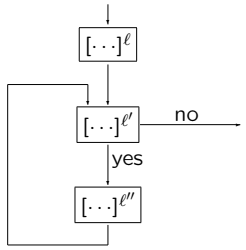
Why smallest solution?

$[z:=x+y]^\ell; \texttt{while } [\texttt{true}]^{\ell'} \texttt{ do } [\texttt{skip}]^{\ell''}$

Equations:

$RD_{entry}(\ell) = \{(x,?),(y,?),(z,?)\}$
$RD_{entry}(\ell') = RD_{exit}(\ell) \cup RD_{exit}(\ell'')$
$RD_{entry}(\ell'') = RD_{exit}(\ell')$

$RD_{exit}(\ell) = (RD_{entry}(\ell) \setminus \{(z,?)\}) \cup \{(z,\ell)\}$
$RD_{exit}(\ell') = RD_{entry}(\ell')$
$RD_{exit}(\ell'') = RD_{entry}(\ell'')$



After some simplification: $RD_{entry}(\ell') = \{(x,?),(y,?),(z,\ell)\} \cup RD_{entry}(\ell')$

Many solutions to this equation: any superset of $\{(x,?),(y,?),(z,\ell)\}$

Very Busy Expression Analysis

An expression is very busy at exit of a label, if, no matter what path is taken from the label, the expression is always used before any of the variables occurring in it are redefined.

For each point, which expression *must* be very busy at the exit from that point.

point of interest

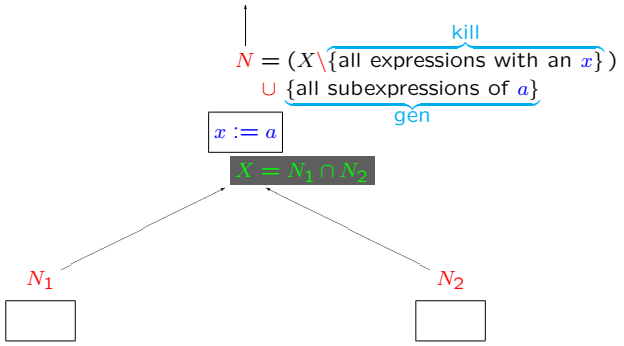
$\texttt{if } [a>b]^1 \texttt{ then } ([x:=b-a]^2; [y:=a-b]^3) \texttt{ else } ([y:=b-a]^4; [x:=a-b]^5)$

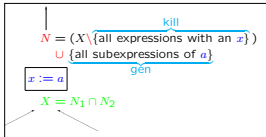
The analysis enables a transformation into

$[t1:=b-a]^A; [t2:=b-a]^B;$
$\texttt{if } [a>b]^1 \texttt{ then } ([x:=t1]^2; [y:=t2]^3) \texttt{ else } ([y:=t1]^4; [x:=t2]^5)$

For each point, which expression *must* be very busy at the exit from that point.

$\Downarrow$ `if [a>b]`^{1} `then ([x:=b-a]`^{2}`;[y:=a-b]`^{3}`) else ([y:=b-a]`^{4}`;[x:=a-b]`^{5}





kill and *gen* functions

$$\begin{aligned}
kill_{VB}([x := a]^\ell) &= \{a' \in \mathbf{AExp}_\star \mid x \in FV(a')\} \\
kill_{VB}([\texttt{skip}]^\ell) &= \emptyset \\
kill_{VB}([b]^\ell) &= \emptyset \\
gen_{VB}([x := a]^\ell) &= \mathbf{AExp}(a) \\
gen_{VB}([\texttt{skip}]^\ell) &= \emptyset \\
gen_{VB}([b]^\ell) &= \mathbf{AExp}(b)
\end{aligned}$$

data flow equations: VB$^=$

$$VB_{exit}(\ell) = \begin{cases} \emptyset & \text{if } \ell \in final(S_\star) \\ \bigcap\{VB_{entry}(\ell') \mid (\ell', \ell) \in flow^R(S_\star)\} & \text{otherwise} \end{cases}$$

$$VB_{entry}(\ell) = (VB_{exit}(\ell) \setminus kill_{VB}(B^\ell)) \cup gen_{VB}(B^\ell) \quad \text{where } B^\ell \in blocks(S_\star)$$

For each point, which expression *must* be very busy at the exit from that point.

$\Downarrow$ `if [a>b]`^{1} `then ([x:=b-a]`^{2}`;[y:=a-b]`^{3}`) else ([y:=b-a]`^{4}`;[x:=a-b]`^{5}

$$\begin{aligned}
VB_{entry}(1) &= VB_{exit}(1) \\
VB_{entry}(2) &= VB_{exit}(2) \cup \{\texttt{b-a}\} \\
VB_{entry}(3) &= \{\texttt{a-b}\} \\
VB_{entry}(4) &= VB_{exit}(4) \cup \{\texttt{b-a}\} \\
VB_{entry}(5) &= \{\texttt{a-b}\}
\end{aligned}$$

$$\begin{aligned}
VB_{exit}(1) &= VB_{entry}(2) \cap VB_{entry}(4) \\
VB_{exit}(2) &= VB_{entry}(3) \\
VB_{exit}(3) &= \emptyset \\
VB_{exit}(4) &= VB_{entry}(5) \\
VB_{exit}(5) &= \emptyset
\end{aligned}$$

ℓ	$kill_{VB}(\ell)$	$gen_{VB}(\ell)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	{b-a}
3	$\emptyset$	{a-b}
4	$\emptyset$	{b-a}
5	$\emptyset$	{a-b}

Largest solution:

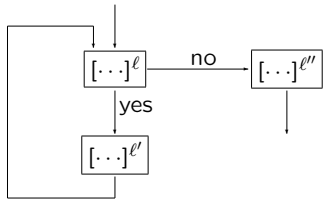
ℓ	$VB_{entry}(\ell)$	$VB_{exit}(\ell)$
1	{a-b, b-a}	{a-b, b-a}
2	{a-b, b-a}	{a-b}
3	{a-b}	$\emptyset$
4	{a-b, b-a}	{a-b}
5	{a-b}	$\emptyset$

Why largest solution?

`(while [x>1]`$^\ell$ `do [skip]`$^{\ell'}$`);[x:=x+1]`$^{\ell''}$

Equations:

$$\begin{aligned}
VB_{entry}(\ell) &= VB_{exit}(\ell) \\
VB_{entry}(\ell') &= VB_{exit}(\ell') \\
VB_{entry}(\ell'') &= \{\texttt{x+1}\} \\
VB_{exit}(\ell) &= VB_{entry}(\ell') \cap VB_{entry}(\ell'') \\
VB_{exit}(\ell') &= VB_{entry}(\ell) \\
VB_{exit}(\ell'') &= \emptyset
\end{aligned}$$



After some simplifications: $VB_{exit}(\ell) = VB_{exit}(\ell) \cap \{\texttt{x+1}\}$

Two solutions to this equation: {x+1} and $\emptyset$

Live Variable Analysis

A variable is live at the exit from a label,, if there is a path from the label to a use of the variable on which it is not overwritten.

For each point, which variables *may* be live at the exit from that point.

point of interest

$[\texttt{x}:=2]^1; [\texttt{y}:=4]^2; [\texttt{x}:=1]^3; (\texttt{if } [\texttt{y}>\texttt{x}]^4 \texttt{ then } [\texttt{z}:=\texttt{y}]^5 \texttt{ else } [\texttt{z}:=\texttt{y}*\texttt{y}]^6); [\texttt{x}:=\texttt{z}]^7$

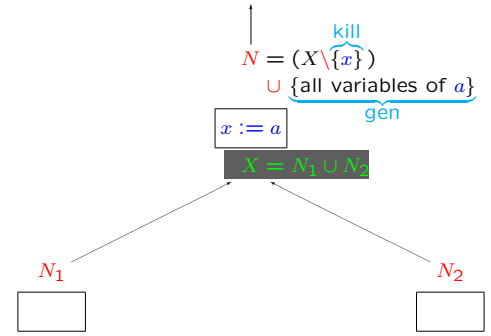
The analysis enables a transformation into

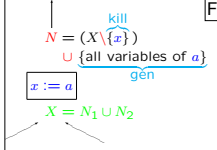
$[\texttt{y}:=4]^2; [\texttt{x}:=1]^3; (\texttt{if } [\texttt{y}>\texttt{x}]^4 \texttt{ then } [\texttt{z}:=\texttt{y}]^5 \texttt{ else } [\texttt{z}:=\texttt{y}*\texttt{y}]^6); [\texttt{x}:=\texttt{z}]^7$

Live Variable Analysis

A variable is live at the exit from a label, if there is a path from the label to a use of the variable on which it is not overwritten.

For each point, which variables *may* be live at the exit from that point.





For each point, which variables *may* be live at the exit from that point.

kill and *gen* functions

$$
\begin{aligned}
kill_{\mathsf{LV}}([x := a]^\ell) &= \{x\} \\
kill_{\mathsf{LV}}([\texttt{skip}]^\ell) &= \emptyset \\
kill_{\mathsf{LV}}([b]^\ell) &= \emptyset \\
gen_{\mathsf{LV}}([x := a]^\ell) &= FV(a) \\
gen_{\mathsf{LV}}([\texttt{skip}]^\ell) &= \emptyset \\
gen_{\mathsf{LV}}([b]^\ell) &= FV(b)
\end{aligned}
$$

data flow equations: $\mathsf{LV}^=$

$$
\mathsf{LV}_{exit}(\ell) = \begin{cases} \emptyset & \text{if } \ell \in final(S_\star) \\ \cup\{\mathsf{LV}_{entry}(\ell') \mid (\ell', \ell) \in flow^R(S_\star)\} & \text{otherwise} \end{cases}
$$

$$
\mathsf{LV}_{entry}(\ell) = (\mathsf{LV}_{exit}(\ell) \setminus kill_{\mathsf{LV}}(B^\ell)) \cup gen_{\mathsf{LV}}(B^\ell) \quad \text{where } B^\ell \in blocks(S_\star)
$$

$[\texttt{x}:=2]^1; [\texttt{y}:=4]^2; [\texttt{x}:=1]^3; (\texttt{if } [\texttt{y}>\texttt{x}]^4 \texttt{ then } [\texttt{z}:=\texttt{y}]^5 \texttt{ else } [\texttt{z}:=\texttt{y}*\texttt{y}]^6); [\texttt{x}:=\texttt{z}]^7$

$$
\begin{aligned}
\mathsf{LV}_{entry}(1) &= \mathsf{LV}_{exit}(1)\setminus\{\texttt{x}\} & \mathsf{LV}_{exit}(1) &= \mathsf{LV}_{entry}(2) \\
\mathsf{LV}_{entry}(2) &= \mathsf{LV}_{exit}(2)\setminus\{\texttt{y}\} & \mathsf{LV}_{exit}(2) &= \mathsf{LV}_{entry}(3) \\
\mathsf{LV}_{entry}(3) &= \mathsf{LV}_{exit}(3)\setminus\{\texttt{x}\} & \mathsf{LV}_{exit}(3) &= \mathsf{LV}_{entry}(4) \\
\mathsf{LV}_{entry}(4) &= \mathsf{LV}_{exit}(4)\cup\{\texttt{x},\texttt{y}\} & \mathsf{LV}_{exit}(4) &= \mathsf{LV}_{entry}(5)\cup\mathsf{LV}_{entry}(6) \\
\mathsf{LV}_{entry}(5) &= (\mathsf{LV}_{exit}(5)\setminus\{\texttt{z}\})\cup\{\texttt{y}\} & \mathsf{LV}_{exit}(5) &= \mathsf{LV}_{entry}(7) \\
\mathsf{LV}_{entry}(6) &= (\mathsf{LV}_{exit}(6)\setminus\{\texttt{z}\})\cup\{\texttt{y}\} & \mathsf{LV}_{exit}(6) &= \mathsf{LV}_{entry}(7) \\
\mathsf{LV}_{entry}(7) &= \{\texttt{z}\} & \mathsf{LV}_{exit}(7) &= \emptyset
\end{aligned}
$$

ℓ	$kill_{\mathsf{LV}}(\ell)$	$gen_{\mathsf{LV}}(\ell)$
1	{x}	$\emptyset$
2	{y}	$\emptyset$
3	{x}	$\emptyset$
4	$\emptyset$	{x, y}
5	{z}	{y}
6	{z}	{y}
7	{x}	{z}

Smallest solution:

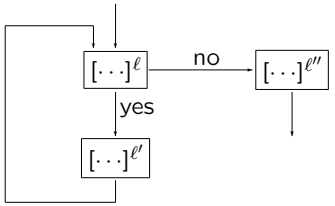
ℓ	$\mathsf{LV}_{entry}(\ell)$	$\mathsf{LV}_{exit}(\ell)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	{y}
3	{y}	{x, y}
4	{x, y}	{y}
5	{y}	{z}
6	{y}	{z}
7	{z}	$\emptyset$

Why smallest solution?

$$(\texttt{while [x>1]}^{\ell}\ \texttt{do [skip]}^{\ell'});\texttt{[x:=x+1]}^{\ell''}$$

Equations:

$$\begin{aligned}
LV_{entry}(\ell) &= LV_{exit}(\ell) \cup \{x\} \\
LV_{entry}(\ell') &= LV_{exit}(\ell') \\
LV_{entry}(\ell'') &= \{x\} \\
LV_{exit}(\ell) &= LV_{entry}(\ell') \cup LV_{entry}(\ell'') \\
LV_{exit}(\ell') &= LV_{entry}(\ell) \\
LV_{exit}(\ell'') &= \emptyset
\end{aligned}$$



After some calculations: $LV_{exit}(\ell) = LV_{exit}(\ell') \cup \{x\}$

Many solutions to this equation: any superset of $\{x\}$

Retrospective: Fran Allen

Programming Techniques — G. Manacher, S. Graham, Editors

A Program Data Flow Analysis Procedure

F.E. Allen and J. Cocke
IBM Thomas J. Watson Research Center

The global data relationships in a program can be exposed and codified by the static analysis methods described in this paper. A procedure is given which determines all the definitions which can possibly "reach" each node of the control flow graph of the program and all the definitions that are "live" on each edge of the graph. The procedure uses an "interval" ordered edge listing data structure and handles reducible and irreducible graphs indistinguishably.

Key Words and Phrases: program optimization, data flow analysis, flow graphs, algorithms, compilers
CR Categories: 4.13, 5.24

1. Introduction

Given a definition of a data item in a program, it is frequently desirable to know what uses might be affected by the particular definition. The inverse is also true: for a given use the definitions of data items which can potentially supply values to it are of interest. Such data flow relationships, or "def-use" relationships, as they are often called, can be deduced by a static, compile time analysis of a program. A procedure for determining the information required for these def-use relationships is given in this paper.

Another data flow relationship which is also of interest is the following: given a program point (instruction) what data definitions are "live" at that point, that is, what data definitions given before this point are used after this point. This information is of interest, for example, when assigning index registers: data which is live at a point in a program might profitably be held in a index register at that point. The procedure given here includes the analysis required to expose live information.

In order to more precisely define the relationships derived by the procedure and to motivate the method used, certain basic concepts and constraints must be defined. This is done in Section 2. Section 3 gives the basic data flow analysis method and concludes with a brief survey of various methods. Section 4 discusses "intervals," the control flow basis used by the procedure given in this paper. In Sections 5 and 6 the procedure is described, with the latter section giving a PL/1 implementation of the procedure. The paper concludes with a summary and an extended bibliography.

The data flow analysis procedure given here has been implemented and used in a PL/1 oriented, Experimental Compiling System. Dr. Kenneth Kennedy of Rice University is responsible for the "live" analysis algorithm, Mr. Richard Stanko of ssc contributed to some of the data structure ideas used in the implementation. Dr. Jeffrey Ullman of Princeton, Dr. Matthew Hecht of the University of Maryland, Dr. Gary Kildall of the Naval Post Graduate School, Dr. Marvin Schaefer of SDC, Dr. Jacob Schwartz of New York University, and others have also made substantial contributions to this area.

2. Context and Problem Statement

Our approach is to derive and express the data flow relationships in terms of the control flow graph [3, 7]

Copyright © 1976, Association for Computing Machinery, Inc. General permission to republish, but not for profit, all or part of this material is granted provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery.
Authors' address: IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, N.Y. 10598.

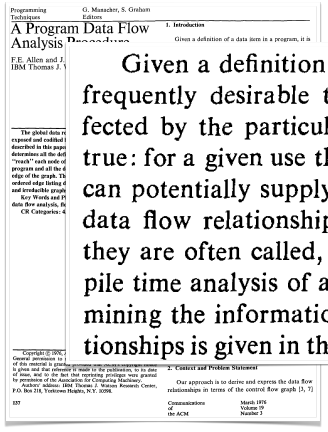
337

Communications of the ACM — March 1976, Volume 19, Number 3

Fran Allen
Turing Award (2006)

F. Allen. *A Program Data Flow Analysis Procedure.* CACM'76 doi=10.1145/360018.360025

Retr



Programming Techniques — G. Manacher, S. Graham, Editors

A Program Data Flow Analysis

F.E. Allen and J.
IBM Thomas J.

1. Introduction

Given a definition of a data item in a program, it is

The global data
exposed and codified
described in this pape
determines all the def
"reach" each node of
program and all the d
edge of the graph. Th
ordered edge listing d
and irreducible graph
Key Words and Ph
data flow analysis, fl
CR Categories: 4

Copyright © 19

2. Context and Problem Statement

Our approach is to derive and express the data flow relationships in terms of the control flow graph [3, 7]

337

Communications of the ACM — March 1976, Volume 19, Number 3

Given a definition of a data iter
frequently desirable to know wha
fected by the particular definition.
true: for a given use the definitions
can potentially supply values to it
data flow relationships, or "def-us
they are often called, can be deduc
pile time analysis of a program. A
mining the information required fo
tionships is given in this paper.

F. Allen. *A Program Data Flow Analysis P*

Derived Information

- *Use-Definition chains* or *ud* chains:

  each use of a variable is linked to all assignments that reach it

  $\texttt{[x:=0]}^1; \texttt{[x:=3]}^2; (\texttt{if [z=x]}^3 \texttt{ then [z:=0]}^4 \texttt{ else [z:=x]}^5); \texttt{[y:=x]}^6; \texttt{[x:=y+z]}^7$

- *Definition-Use chains* or *du* chains:

  each assignment to a variable is linked to all uses of it

  $\texttt{[x:=0]}^1; \texttt{[x:=3]}^2; (\texttt{if [z=x]}^3 \texttt{ then [z:=0]}^4 \texttt{ else [z:=x]}^5); \texttt{[y:=x]}^6; \texttt{[x:=y+z]}^7$

ud chains

$$ud : \mathbf{Var}_\star \times \mathbf{Lab}_\star \to \mathcal{P}(\mathbf{Lab}_\star)$$

given by

$$\begin{array}{rcl} ud(x,\ell') & = & \{\ell \mid def(x,\ell) \wedge \exists \ell'' : (\ell,\ell'') \in flow(S_\star) \wedge clear(x,\ell'',\ell')\} \\ & \cup & \{? \mid clear(x, init(S_\star), \ell')\} \end{array}$$

where

- $def(x,\ell)$ means that the block ℓ assigns a value to x
- $clear(x,\ell,\ell')$ means that none of the blocks on a path from ℓ to ℓ' contains an assignments to x but that the block ℓ' uses x (in a test or on the right hand side of an assignment)

ud chains - an alternative definition

$$UD : \mathbf{Var}_\star \times \mathbf{Lab}_\star \to \mathcal{P}(\mathbf{Lab}_\star)$$

is defined by:

$$UD(x,\ell) = \begin{cases} \{\ell' \mid (x,\ell') \in RD_{entry}(\ell)\} & \text{if } x \in gen_{LV}(B^\ell) \\ \emptyset & \text{otherwise} \end{cases}$$

One can show that:

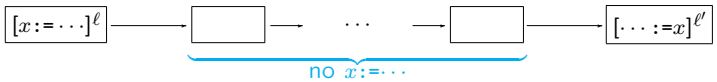
$$ud(x,\ell) = UD(x,\ell)$$

du chains

$$du : \mathbf{Var}_\star \times \mathbf{Lab}_\star \to \mathcal{P}(\mathbf{Lab}_\star)$$

given by

$$du(x,\ell) = \begin{cases} \{\ell' \mid def(x,\ell) \wedge \exists \ell'' : (\ell,\ell'') \in flow(S_\star) \wedge clear(x,\ell'',\ell')\} \\ \qquad \text{if } \ell \neq ? \\ \{\ell' \mid clear(x, init(S_\star), \ell')\} \\ \qquad \text{if } \ell = ? \end{cases}$$



One can show that:

$$du(x,\ell) = \{\ell' \mid \ell \in ud(x,\ell')\}$$

Example:

$[\texttt{x:=0}]^1; [\texttt{x:=3}]^2; (\texttt{if } [\texttt{z=x}]^3 \texttt{ then } [\texttt{z:=0}]^4 \texttt{ else } [\texttt{z:=x}]^5); [\texttt{y:=x}]^6; [\texttt{x:=y+z}]^7$

$ud(x,\ell)$	x	y	z
1	$\emptyset$	$\emptyset$	$\emptyset$
2	$\emptyset$	$\emptyset$	$\emptyset$
3	{2}	$\emptyset$	{?}
4	$\emptyset$	$\emptyset$	$\emptyset$
5	{2}	$\emptyset$	$\emptyset$
6	{2}	$\emptyset$	$\emptyset$
7	$\emptyset$	{6}	{4,5}

$du(x,\ell)$	x	y	z
1	$\emptyset$	$\emptyset$	$\emptyset$
2	{3,5,6}	$\emptyset$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$	{7}
5	$\emptyset$	$\emptyset$	{7}
6	$\emptyset$	{7}	$\emptyset$
7	$\emptyset$	$\emptyset$	$\emptyset$
?	$\emptyset$	$\emptyset$	{3}

Theoretical Properties

- Structural operational semantics
- Correctness of Live Variable Analysis

Semantics

$\langle [x := a]^\ell, \sigma\rangle \to \sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma]$

$\langle [\texttt{skip}]^\ell, \sigma\rangle \to \sigma$

$$\frac{\langle S_1, \sigma\rangle \to \langle S_1', \sigma'\rangle}{\langle S_1; S_2, \sigma\rangle \to \langle S_1'; S_2, \sigma'\rangle}$$

$$\frac{\langle S_1, \sigma\rangle \to \sigma'}{\langle S_1; S_2, \sigma\rangle \to \langle S_2, \sigma'\rangle}$$

$\langle \texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2, \sigma\rangle \to \langle S_1, \sigma\rangle$ if $\mathcal{B}[\![b]\!]\sigma = \mathit{true}$

$\langle \texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2, \sigma\rangle \to \langle S_2, \sigma\rangle$ if $\mathcal{B}[\![b]\!]\sigma = \mathit{false}$

$\langle \texttt{while } [b]^\ell \texttt{ do } S, \sigma\rangle \to \langle (S; \texttt{while } [b]^\ell \texttt{ do } S), \sigma\rangle$ if $\mathcal{B}[\![b]\!]\sigma = \mathit{true}$

$\langle \texttt{while } [b]^\ell \texttt{ do } S, \sigma\rangle \to \sigma$ if $\mathcal{B}[\![b]\!]\sigma = \mathit{false}$

A *state* is a mapping from variables to integers:

$$\sigma \in \mathbf{State} = \mathbf{Var} \to \mathbf{Z}$$

The semantics of arithmetic and boolean expressions

$\mathcal{A}: \mathbf{AExp} \to (\mathbf{State} \to \mathbf{Z})$ (no errors allowed)

$\mathcal{B}: \mathbf{BExp} \to (\mathbf{State} \to \mathbf{T})$ (no errors allowed)

The *transitions* of the semantics are of the form

$\langle S, \sigma\rangle \to \sigma'$ and $\langle S, \sigma\rangle \to \langle S', \sigma'\rangle$

Example:

$\langle [\texttt{y:=x}]^1; [\texttt{z:=1}]^2; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{300}\rangle$

$\to \langle [\texttt{z:=1}]^2; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{330}\rangle$

Equations and Constraints

Equation system $\mathsf{LV}^{=}(S_\star)$:

$$\mathsf{LV}_{exit}(\ell) = \begin{cases} \emptyset & \text{if } \ell \in \mathit{final}(S_\star) \\ \bigcup\{\mathsf{LV}_{entry}(\ell') \mid (\ell', \ell) \in \mathit{flow}^R(S_\star)\} & \text{otherwise} \end{cases}$$

$$\mathsf{LV}_{entry}(\ell) = (\mathsf{LV}_{exit}(\ell) \setminus \mathit{kill}_{\mathsf{LV}}(B^\ell)) \cup \mathit{gen}_{\mathsf{LV}}(B^\ell) \quad \text{where } B^\ell \in \mathit{blocks}(S_\star)$$

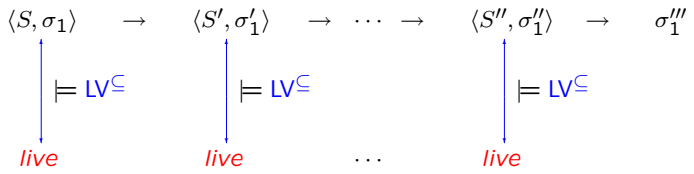
Constraint system $\mathsf{LV}^{\subseteq}(S_\star)$:

$$\mathsf{LV}_{exit}(\ell) \supseteq \begin{cases} \emptyset & \text{if } \ell \in \mathit{final}(S_\star) \\ \bigcup\{\mathsf{LV}_{entry}(\ell') \mid (\ell', \ell) \in \mathit{flow}^R(S_\star)\} & \text{otherwise} \end{cases}$$

$$\mathsf{LV}_{entry}(\ell) \supseteq (\mathsf{LV}_{exit}(\ell) \setminus \mathit{kill}_{\mathsf{LV}}(B^\ell)) \cup \mathit{gen}_{\mathsf{LV}}(B^\ell) \quad \text{where } B^\ell \in \mathit{blocks}(S_\star)$$

Lemma

A solution *live* to the constraint system is preserved during computation



Proof: requires a lot of machinery — see the book.

Correctness Relation

$$\sigma_1 \sim_V \sigma_2$$

means that for all practical purposes the two states σ_1 and σ_2 are equal: only the values of the live variables of V matters and here the two states are equal.

Example:

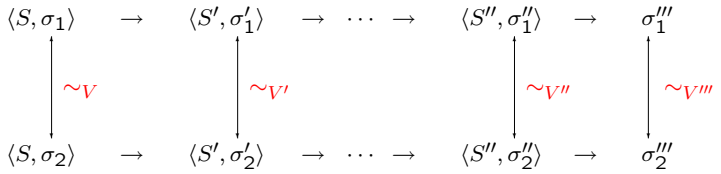
Consider the statement $[\texttt{x:=y+z}]^\ell$

Let $V_1 = \{\texttt{y}, \texttt{z}\}$. Then $\sigma_1 \sim_{V_1} \sigma_2$ means $\sigma_1(\texttt{y}) = \sigma_2(\texttt{y}) \wedge \sigma_1(\texttt{z}) = \sigma_2(\texttt{z})$

Let $V_2 = \{\texttt{x}\}$. Then $\sigma_1 \sim_{V_2} \sigma_2$ means $\sigma_1(\texttt{x}) = \sigma_2(\texttt{x})$

Correctness Theorem

The relation "$\sim$" is *invariant* under computation: the live variables for the initial configuration remain live throughout the computation.



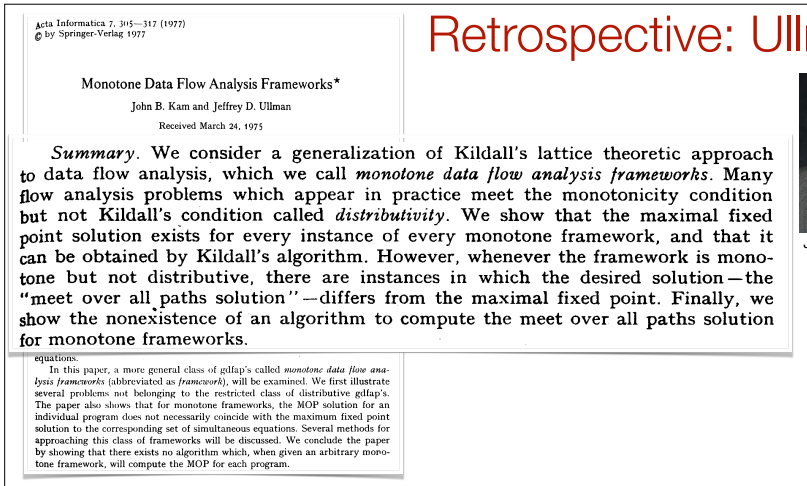
$V = \mathit{live}_{\mathit{entry}}(\mathit{init}(S))$

$V' = \mathit{live}_{\mathit{entry}}(\mathit{init}(S'))$

$V'' = \mathit{live}_{\mathit{entry}}(\mathit{init}(S''))$

$V''' = \mathit{live}_{\mathit{exit}}(\mathit{init}(S''))$
$= \mathit{live}_{\mathit{exit}}(\ell)$
for some $\ell \in \mathit{final}(S)$

Retrospective: Ullman



Acta Informatica 7, 305—317 (1977)
© by Springer-Verlag 1977

Monotone Data Flow Analysis Frameworks*

John B. Kam and Jeffrey D. Ullman

Received March 24, 1975

Summary. We consider a generalization of Kildall's lattice theoretic approach to data flow analysis, which we call *monotone data flow analysis frameworks*. Many flow analysis problems which appear in practice meet the monotonicity condition but not Kildall's condition called *distributivity*. We show that the maximal fixed point solution exists for every instance of every monotone framework, and that it can be obtained by Kildall's algorithm. However, whenever the framework is monotone but not distributive, there are instances in which the desired solution—the "meet over all paths solution"—differs from the maximal fixed point. Finally, we show the nonexistence of an algorithm to compute the meet over all paths solution for monotone frameworks.

equations.

In this paper, a more general class of gdfap's called *monotone data flow analysis frameworks* (abbreviated as *framework*), will be examined. We first illustrate several problems not belonging to the restricted class of distributive gdfap's. The paper also shows that for monotone frameworks, the MOP solution for an individual program does not necessarily coincide with the maximum fixed point solution to the corresponding set of simultaneous equations. Several methods for approaching this class of frameworks will be discussed. We conclude the paper by showing that there exists no algorithm which, when given an arbitrary monotone framework, will compute the MOP for each program.


Jeffrey D. Ullman

J Kam, J Ullman *Monotone Data Flow Analysis Frameworks.* Acta Inf'77 doi=10.1007/BF00290339

Monotone Frameworks

- Monotone and Distributive Frameworks
- Instances of Frameworks
- Constant Propagation Analysis

The Overall Pattern

Each of the four classical analyses take the form

$$\mathit{Analysis}_\circ(\ell) = \begin{cases} \iota & \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\} & \text{otherwise} \end{cases}$$

$$\mathit{Analysis}_\bullet(\ell) = f_\ell(\mathit{Analysis}_\circ(\ell))$$

where

- $\bigsqcup$ is $\bigcap$ or $\bigcup$ (and $\sqcup$ is $\cup$ or $\cap$),
- F is either $\mathit{flow}(S_\star)$ or $\mathit{flow}^R(S_\star)$,
- E is $\{\mathit{init}(S_\star)\}$ or $\mathit{final}(S_\star)$,
- ι specifies the initial or final analysis information, and
- f_ℓ is the transfer function associated with $B^\ell \in \mathit{blocks}(S_\star)$.

The Principle: forward versus backward

- The *forward analyses* have F to be $\mathit{flow}(S_\star)$ and then $\mathit{Analysis}_\circ$ concerns entry conditions and $\mathit{Analysis}_\bullet$ concerns exit conditions; the equation system presupposes that $S_\star$ has isolated entries.
- The *backward analyses* have F to be $\mathit{flow}^R(S_\star)$ and then $\mathit{Analysis}_\circ$ concerns exit conditions and $\mathit{Analysis}_\bullet$ concerns entry conditions; the equation system presupposes that $S_\star$ has isolated exits.

The Principle: union versus intersecton

- When $\bigsqcup$ is $\bigcap$ we require the greatest sets that solve the equations and we are able to detect properties satisfied by *all execution paths* reaching (or leaving) the entry (or exit) of a label; the analysis is called a must-analysis.
- When $\bigsqcup$ is $\bigcup$ we require the smallest sets that solve the equations and we are able to detect properties satisfied by *at least one execution path* to (or from) the entry (or exit) of a label; the analysis is called a may-analysis.

Property Spaces

The *property space*, L, is used to represent the data flow information, and the *combination operator*, $\bigsqcup$: $\mathcal{P}(L) \to L$, is used to combine information from different paths.

- L is a *complete lattice*, that is, a partially ordered set, $(L, \sqsubseteq)$, such that each subset, Y, has a least upper bound, $\bigsqcup Y$.
- L satisfies the *Ascending Chain Condition*; that is, each ascending chain eventually stabilises (meaning that if $(l_n)_n$ is such that $l_1 \sqsubseteq l_2 \sqsubseteq l_3 \sqsubseteq \cdots$,then there exists n such that $l_n = l_{n+1} = \cdots$).

Example: Reaching Definitions

- $L = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$ is partially ordered by subset inclusion so $\sqsubseteq$ is $\subseteq$
- the least upper bound operation $\bigsqcup$ is $\bigcup$ and the least element $\bot$ is $\emptyset$
- L satisfies the Ascending Chain Condition because $\mathbf{Var}_\star \times \mathbf{Lab}_\star$ is finite (unlike $\mathbf{Var} \times \mathbf{Lab}$)

Example: Available Expressions

- $L = \mathcal{P}(\mathbf{AExp}_\star)$ is partially ordered by superset inclusion so $\sqsubseteq$ is $\supseteq$
- the least upper bound operation $\bigsqcup$ is $\bigcap$ and the least element $\bot$ is $\mathbf{AExp}_\star$
- L satisfies the Ascending Chain Condition because $\mathbf{AExp}_\star$ is finite (unlike $\mathbf{AExp}$)

Transfer Functions

The set of transfer functions, $\mathcal{F}$, is a set of monotone functions over L, meaning that

$$l \sqsubseteq l' \text{ implies } f_\ell(l) \sqsubseteq f_\ell(l')$$

and furthermore they fulfil the following conditions:

- $\mathcal{F}$ contains *all* the transfer functions $f_\ell : L \to L$ in question (for $\ell \in \mathbf{Lab}_\star$)
- $\mathcal{F}$ contains the *identity function*
- $\mathcal{F}$ is *closed under composition* of functions

Frameworks

A *Monotone Framework* consists of:

- a complete lattice, L, that satisfies the Ascending Chain Condition; we write $\bigsqcup$ for the least upper bound operator
- a set $\mathcal{F}$ of monotone functions from L to L that contains the identity function and that is closed under function composition

A *Distributive Framework* is a Monotone Framework where additionally all functions f in $\mathcal{F}$ are required to be distributive:

$$f(l_1 \sqcup l_2) = f(l_1) \sqcup f(l_2)$$

Instances

An *instance* of a Framework consists of:

- the complete lattice, L, of the framework
- the space of functions, $\mathcal{F}$, of the framework
- a finite flow, F (typically $\mathit{flow}(S_\star)$ or $\mathit{flow}^R(S_\star)$)
- a finite set of *extremal labels*, E (typically $\{\mathit{init}(S_\star)\}$ or $\mathit{final}(S_\star)$)
- an *extremal value*, $\iota \in L$, for the extremal labels
- a mapping, $f_\cdot$, from the labels $\mathbf{Lab}_\star$ to transfer functions in $\mathcal{F}$

Equations of the Instance:

$$\mathit{Analysis}_\circ(\ell) = \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\} \sqcup \iota^\ell_E$$

$$\text{where } \iota^\ell_E = \begin{cases} \iota & \text{if } \ell \in E \\ \bot & \text{if } \ell \notin E \end{cases}$$

$$\mathit{Analysis}_\bullet(\ell) = f_\ell(\mathit{Analysis}_\circ(\ell))$$

Constraints of the Instance:

$$\mathit{Analysis}_\circ(\ell) \sqsupseteq \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\} \sqcup \iota^\ell_E$$

$$\text{where } \iota^\ell_E = \begin{cases} \iota & \text{if } \ell \in E \\ \bot & \text{if } \ell \notin E \end{cases}$$

$$\mathit{Analysis}_\bullet(\ell) \sqsupseteq f_\ell(\mathit{Analysis}_\circ(\ell))$$

The Examples Revisited

	Available Expressions	Reaching Definitions	Very Busy Expressions	Live Variables
L	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star)$
$\sqsubseteq$	$\supseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\bigsqcup$	$\bigcap$	$\bigcup$	$\bigcap$	$\bigcup$
$\bot$	$\mathbf{AExp}_\star$	$\emptyset$	$\mathbf{AExp}_\star$	$\emptyset$
ι	$\emptyset$	$\{(x,?) \mid x \in \mathit{FV}(S_\star)\}$	$\emptyset$	$\emptyset$
E	$\{\mathit{init}(S_\star)\}$	$\{\mathit{init}(S_\star)\}$	$\mathit{final}(S_\star)$	$\mathit{final}(S_\star)$
F	$\mathit{flow}(S_\star)$	$\mathit{flow}(S_\star)$	$\mathit{flow}^R(S_\star)$	$\mathit{flow}^R(S_\star)$
$\mathcal{F}$	$\{f : L \to L \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$			
f_ℓ	$f_\ell(l) = (l \setminus \mathit{kill}(B^\ell)) \cup \mathit{gen}(B^\ell)$ where $B^\ell \in \mathit{blocks}(S_\star)$			

Bit Vector Frameworks

A *Bit Vector Framework* has

- $L = \mathcal{P}(D)$ for D finite
- $\mathcal{F} = \{f \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$

Examples:

- Available Expressions
- Live Variables
- Reaching Definitions
- Very Busy Expressions

Lemma: Bit Vector Frameworks are always Distributive Frameworks

Proof

$$f(l_1 \sqcup l_2) = \begin{cases} f(l_1 \cup l_2) \\ f(l_1 \cap l_2) \end{cases} = \begin{cases} ((l_1 \cup l_2) \setminus l_k) \cup l_g \\ ((l_1 \cap l_2) \setminus l_k) \cup l_g \end{cases}$$

$$= \begin{cases} ((l_1 \setminus l_k) \cup (l_2 \setminus l_k)) \cup l_g \\ ((l_1 \setminus l_k) \cap (l_2 \setminus l_k)) \cup l_g \end{cases} = \begin{cases} ((l_1 \setminus l_k) \cup l_g) \cup ((l_2 \setminus l_k) \cup l_g) \\ ((l_1 \setminus l_k) \cup l_g) \cap ((l_2 \setminus l_k) \cup l_g) \end{cases}$$

$$= \begin{cases} f(l_1) \cup f(l_2) \\ f(l_1) \cap f(l_2) \end{cases} = f(l_1) \sqcup f(l_2)$$

- $id(l) = (l \setminus \emptyset) \cup \emptyset$
- $f_2(f_1(l)) = (((l \setminus l_k^1) \cup l_g^1) \setminus l_k^2) \cup l_g^2 = (l \setminus (l_k^1 \cup l_k^2)) \cup ((l_g^1 \setminus l_k^2) \cup l_g^2)$
- monotonicity follows from distributivity
- $\mathcal{P}(D)$ satisfies the Ascending Chain Condition because D is finite

The Constant Propagation Framework

An example of a Monotone Framework that is not a Distributive Framework

The aim of the *Constant Propagation Analysis* is to determine

> For each program point, whether or not a variable has a constant value whenever execution reaches that point.

Example:

$$[\texttt{x:=6}]^1; [\texttt{y:=3}]^2; \texttt{while } [\texttt{x} > \texttt{y}]^3 \texttt{ do } ([\texttt{x:=x} - 1]^4; [\texttt{z:=y} * \texttt{y}]^6)$$

The analysis enables a transformation into

$$[\texttt{x:=6}]^1; [\texttt{y:=3}]^2; \texttt{while } [\texttt{x} > 3]^3 \texttt{ do } ([\texttt{x:=x} - 1]^4; [\texttt{z:=9}]^6)$$

Elements of L

$$\widehat{\mathbf{State}}_{\mathsf{CP}} = ((\mathbf{Var}_\star \to \mathbf{Z}^\top)_\perp, \sqsubseteq)$$

Idea:

- $\perp$ is the least element: no information is available
- $\widehat{\sigma} \in \mathbf{Var}_\star \to \mathbf{Z}^\top$ specifies for each variable whether it is constant:
  - $\widehat{\sigma}(x) \in \mathbf{Z}$: x is constant and the value is $\widehat{\sigma}(x)$
  - $\widehat{\sigma}(x) = \top$: x might not be constant

Partial Ordering on L

The partial ordering $\sqsubseteq$ on $(\mathbf{Var}_\star \to \mathbf{Z}^\top)_\perp$ is defined by

$$\forall \hat{\sigma} \in (\mathbf{Var}_\star \to \mathbf{Z}^\top)_\perp : \quad \perp \sqsubseteq \hat{\sigma}$$

$$\forall \hat{\sigma}_1, \hat{\sigma}_2 \in \mathbf{Var}_\star \to \mathbf{Z}^\top : \quad \hat{\sigma}_1 \sqsubseteq \hat{\sigma}_2 \quad \underline{\text{iff}} \quad \forall x : \hat{\sigma}_1(x) \sqsubseteq \hat{\sigma}_2(x)$$

where $\mathbf{Z}^\top = \mathbf{Z} \cup \{\top\}$ is partially ordered as follows:

$$\forall z \in \mathbf{Z}^\top : z \sqsubseteq \top$$

$$\forall z_1, z_2 \in \mathbf{Z} : (z_1 \sqsubseteq z_2) \Leftrightarrow (z_1 = z_2)$$

Transfer Functions in $\mathcal{F}$

$$\mathcal{F}_{\mathsf{CP}} = \{f \mid f \text{ is a monotone function on } \widehat{\mathbf{State}}_{\mathsf{CP}}\}$$

Lemma

Constant Propagation as defined by $\widehat{\mathbf{State}}_{\mathsf{CP}}$ and $\mathcal{F}_{\mathsf{CP}}$ is a Monotone Framework

Instances

Constant Propagation is a forward analysis, so for the program $S_\star$:

- the flow, F, is $\mathit{flow}(S_\star)$,
- the extremal labels, E, is $\{\mathit{init}(S_\star)\}$,
- the extremal value, ι_{CP}, is $\lambda x.\top$, and
- the mapping, $f^{\mathsf{CP}}_\cdot$, of labels to transfer functions is as shown next

Constant Propagation Analysis

$$\mathcal{A}_{\mathsf{CP}} : \mathbf{AExp} \to (\widehat{\mathbf{State}}_{\mathsf{CP}} \to \mathbf{Z}^\top_\perp)$$

$$\mathcal{A}_{\mathsf{CP}}[\![x]\!]\hat{\sigma} = \begin{cases} \perp & \text{if } \hat{\sigma} = \perp \\ \hat{\sigma}(x) & \text{otherwise} \end{cases}$$

$$\mathcal{A}_{\mathsf{CP}}[\![n]\!]\hat{\sigma} = \begin{cases} \perp & \text{if } \hat{\sigma} = \perp \\ n & \text{otherwise} \end{cases}$$

$$\mathcal{A}_{\mathsf{CP}}[\![a_1\ op_a\ a_2]\!]\hat{\sigma} = \mathcal{A}_{\mathsf{CP}}[\![a_1]\!]\hat{\sigma}\ \widehat{op}_a\ \mathcal{A}_{\mathsf{CP}}[\![a_2]\!]\hat{\sigma}$$

transfer functions: f^{CP}_ℓ

$$[x := a]^\ell : \quad f^{\mathsf{CP}}_\ell(\hat{\sigma}) = \begin{cases} \perp & \text{if } \hat{\sigma} = \perp \\ \hat{\sigma}[x \mapsto \mathcal{A}_{\mathsf{CP}}[\![a]\!]\hat{\sigma}] & \text{otherwise} \end{cases}$$

$$[\mathtt{skip}]^\ell : \quad f^{\mathsf{CP}}_\ell(\hat{\sigma}) = \hat{\sigma}$$

$$[b]^\ell : \quad f^{\mathsf{CP}}_\ell(\hat{\sigma}) = \hat{\sigma}$$

Lemma

Constant Propagation is not a Distributive Framework

Proof

Consider the transfer function f_ℓ^{CP} for $[\mathtt{y:=x*x}]^\ell$

Let $\hat{\sigma}_1$ and $\hat{\sigma}_2$ be such that $\hat{\sigma}_1(\mathtt{x}) = 1$ and $\hat{\sigma}_2(\mathtt{x}) = -1$

Then $\hat{\sigma}_1 \sqcup \hat{\sigma}_2$ maps $\mathtt{x}$ to $\top$ — $f_\ell^{\mathsf{CP}}(\hat{\sigma}_1 \sqcup \hat{\sigma}_2)$ maps $\mathtt{y}$ to $\top$

Both $f_\ell^{\mathsf{CP}}(\hat{\sigma}_1)$ and $f_\ell^{\mathsf{CP}}(\hat{\sigma}_2)$ map $\mathtt{y}$ to 1 — $f_\ell^{\mathsf{CP}}(\hat{\sigma}_1) \sqcup f_\ell^{\mathsf{CP}}(\hat{\sigma}_2)$ maps $\mathtt{y}$ to 1

Equation Solving

- The MFP solution — "Maximum" (actually least) Fixed Point
  - Worklist algorithm for Monotone Frameworks
- The MOP solution — "Meet" (actually join) Over all Paths

The MFP Solution

– Idea: iterate until stabilisation.

Worklist Algorithm

Input: An instance $(L, \mathcal{F}, F, E, \iota, f.)$ of a Monotone Framework

Output: The MFP Solution: $MFP_\circ$, $MFP_\bullet$

Data structures:

- Analysis: the current analysis result for block entries (or exits)
- The worklist W: a list of pairs (ℓ, ℓ') indicating that the current analysis result has changed at the entry (or exit) to the block ℓ and hence the entry (or exit) information must be recomputed for ℓ'

Worklist Algorithm

Step 1 Initialisation (of W and Analysis)

```
W := nil;
for all (ℓ,ℓ') in F do W := cons((ℓ,ℓ'),W);
for all ℓ in F or E do
  if ℓ ∈ E then Analysis[ℓ] := ι else Analysis[ℓ] := ⊥_L;
```

Step 2 Iteration (updating W and Analysis)

```
while W ≠ nil do
  ℓ := fst(head(W)); ℓ' = snd(head(W)); W := tail(W);
  if f_ℓ(Analysis[ℓ]) ⋢ Analysis[ℓ'] then
    Analysis[ℓ'] := Analysis[ℓ'] ⊔ f_ℓ(Analysis[ℓ]);
    for all ℓ'' with (ℓ',ℓ'') in F do W := cons((ℓ',ℓ''),W);
```

Step 3 Presenting the result ($MFP_\circ$ and $MFP_\bullet$)

```
for all ℓ in F or E do
    MFP_∘(ℓ) := Analysis[ℓ];
    MFP_•(ℓ) := f_ℓ(Analysis[ℓ])
```

Correctness

The worklist algorithm always terminates and it computes the least (or MFP) solution to the instance given as input.

Complexity

Suppose that E and F contain at most $b \geq 1$ distinct labels, that F contains at most $e \geq b$ pairs, and that L has finite height at most $h \geq 1$.

Count as basic operations the applications of f_ℓ, applications of $\sqcup$, or updates of Analysis.

Then there will be at most $O(e \cdot h)$ basic operations.

Example: Reaching Definitions (assuming unique labels):

$O(b^2)$ where b is size of program: $O(h) = O(b)$ and $O(e) = O(b)$.

The MOP Solution

– Idea: propagate analysis information along paths.

Paths

The paths up to but not including ℓ:

$$path_\circ(\ell) = \{[\ell_1, \cdots, \ell_{n-1}] \mid n \geq 1 \wedge \forall i < n : (\ell_i, \ell_{i+1}) \in F \wedge \ell_n = \ell \wedge \ell_1 \in E\}$$

The paths up to and including ℓ:

$$path_\bullet(\ell) = \{[\ell_1, \cdots, \ell_n] \mid n \geq 1 \wedge \forall i < n : (\ell_i, \ell_{i+1}) \in F \wedge \ell_n = \ell \wedge \ell_1 \in E\}$$

Transfer functions for a path $\vec{\ell} = [\ell_1, \cdots, \ell_n]$:

$$f_{\vec{\ell}} = f_{\ell_n} \circ \cdots \circ f_{\ell_1} \circ id$$

The MOP Solution

The solution up to but not including ℓ:

$$MOP_\circ(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in path_\circ(\ell)\}$$

The solution up to and including ℓ:

$$MOP_\bullet(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in path_\bullet(\ell)\}$$

Precision of the MOP versus MFP solutions

The MFP solution safely approximates the MOP solution: $MFP \sqsupseteq MOP$ ("because" $f(x \sqcup y) \sqsupseteq f(x) \sqcup f(y)$ when f is monotone).

For Distributive Frameworks the MFP and MOP solutions are equal: $MFP = MOP$ ("because" $f(x \sqcup y) = f(x) \sqcup f(y)$ when f is distributive).

Lemma

Consider the MFP and MOP solutions to an instance $(L, \mathcal{F}, F, B, \iota, f.)$ of a Monotone Framework; then:

$$MFP_\circ \sqsupseteq MOP_\circ \text{ and } MFP_\bullet \sqsupseteq MOP_\bullet$$

If the framework is distributive and if $path_\circ(\ell) \neq \emptyset$ for all ℓ in E and F then:

$$MFP_\circ = MOP_\circ \text{ and } MFP_\bullet = MOP_\bullet$$

Decidability of MOP and MFP

The MFP solution is always computable (meaning that it is decidable) because of the Ascending Chain Condition.

The MOP solution is often uncomputable (meaning that it is undecidable): the existence of a general algorithm for the MOP solution would imply the decidability of the *Modified Post Correspondence Problem*, which is known to be undecidable.

Lemma

The MOP solution for Constant Propagation is undecidable.

Proof: Let $u_1, \cdots, u_n$ and $v_1, \cdots, v_n$ be strings over the alphabet {1,⋯,9}; let $\mid u \mid$ denote the length of u; let $[\![u]\!]$ be the natural number denoted.

The Modified Post Correspondence Problem is to determine whether or not $u_{i_1} \cdots u_{i_m} = v_{i_1} \cdots v_{i_n}$ for some sequence $i_1, \cdots, i_m$ with $i_1 = 1$.

```
x:=⟦u1⟧; y:=⟦v1⟧;
while [···] do
   (if [···] then x:=x * 10^|u1| + ⟦u1⟧; y:=y * 10^|v1| + ⟦v1⟧ else
    ⋮
    if [···] then x:=x * 10^|un| + ⟦un⟧; y:=y * 10^|vn| + ⟦vn⟧ else skip)
[z:=abs((x-y)*(x-y))]^ℓ
```

Then $MOP_\bullet(\ell)$ will map z to 1 if and only if the Modified Post Correspondence Problem has no solution. This is undecidable.

Scaling Program Analysis

CS7575

- Lecture 3 — Data flow analysis; PoPA Chapter 2 (part 2) - Interprocedural + Shape

Interprocedural Analysis

- The problem
- MVP: "Meet" over Valid Paths
- Making context explicit
- Context based on call-strings
- Context based on assumption sets

```
proc add(val x, y, res z) is
    if (x is Integer) then
        z := x + y;
    else if (x is String) then
        z := concatenate(x,y);
end

add(40, 2, a);
b := a+1;
add("4", "2", s);
println(s)
```

Syntax for procedures

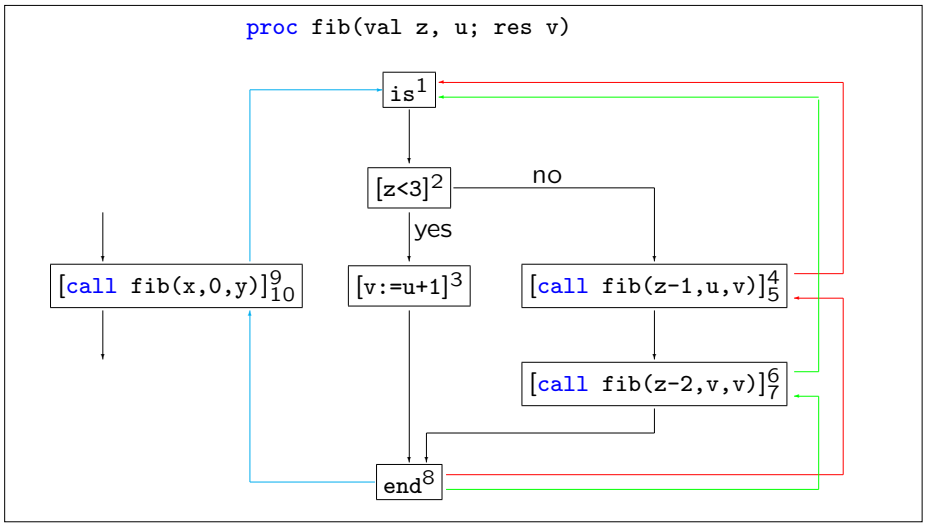
Programs: $P_\star = \texttt{begin}\ D_\star\ S_\star\ \texttt{end}$

Declarations: $D ::= D; D \mid \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}$

Statements: $S ::= \cdots \mid [\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}$

Example:

```
begin  proc fib(val z, u; res v) is1
         if [z<3]2 then [v:=u+1]3
         else ([call fib(z-1,u,v)]45; [call fib(z-2,v,v)]67)
       end8;
       [call fib(x,0,y)]910
end
```

Flow graphs for procedure calls

$$
\begin{aligned}
\mathit{init}([\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}) &= \ell_c \\
\mathit{final}([\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}) &= \{\ell_r\} \\
\mathit{blocks}([\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}) &= \{[\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}\} \\
\mathit{labels}([\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}) &= \{\ell_c,\ell_r\} \\
\mathit{flow}([\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}) &= \{(\ell_c;\ell_n),(\ell_x;\ell_r)\} \\
&\quad \text{if}\ \ \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}\ \text{is in}\ D_\star
\end{aligned}
$$

- $(\ell_c;\ell_n)$ is the flow corresponding to *calling* a procedure at ℓ_c and entering the procedure body at ℓ_n, and
- $(\ell_x;\ell_r)$ is the flow corresponding to exiting a procedure body at ℓ_x and *returning* to the call at ℓ_r.

Flow graphs for procedure declarations

For each procedure declaration $\texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}$ of $D_\star$:

$$
\begin{aligned}
\mathit{init}(p) &= \ell_n \\
\mathit{final}(p) &= \{\ell_x\} \\
\mathit{blocks}(p) &= \{\texttt{is}^{\ell_n}, \texttt{end}^{\ell_x}\} \cup \mathit{blocks}(S) \\
\mathit{labels}(p) &= \{\ell_n,\ell_x\} \cup \mathit{labels}(S) \\
\mathit{flow}(p) &= \{(\ell_n, \mathit{init}(S))\} \cup \mathit{flow}(S) \cup \{(\ell,\ell_x) \mid \ell \in \mathit{final}(S)\}
\end{aligned}
$$

Flow graphs for programs

For the program $P_\star = \texttt{begin}\ D_\star\ S_\star\ \texttt{end}$:

$$
\begin{array}{rcl}
\mathit{init}_\star & = & \mathit{init}(S_\star) \\
\mathit{final}_\star & = & \mathit{final}(S_\star) \\
\mathit{blocks}_\star & = & \bigcup\{\mathit{blocks}(p) \mid \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x} \text{ is in } D_\star\} \\
 & & \cup \mathit{blocks}(S_\star) \\
\mathit{labels}_\star & = & \bigcup\{\mathit{labels}(p) \mid \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x} \text{ is in } D_\star\} \\
 & & \cup \mathit{labels}(S_\star) \\
\mathit{flow}_\star & = & \bigcup\{\mathit{flow}(p) \mid \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x} \text{ is in } D_\star\} \\
 & & \cup \mathit{flow}(S_\star) \\
\mathit{interflow}_\star & = & \{(\ell_c, \ell_n, \ell_x, \ell_r) \mid \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x} \text{ is in } D_\star \\
 & & \quad \text{and } [\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r} \text{ is in } S_\star\}
\end{array}
$$

Example:

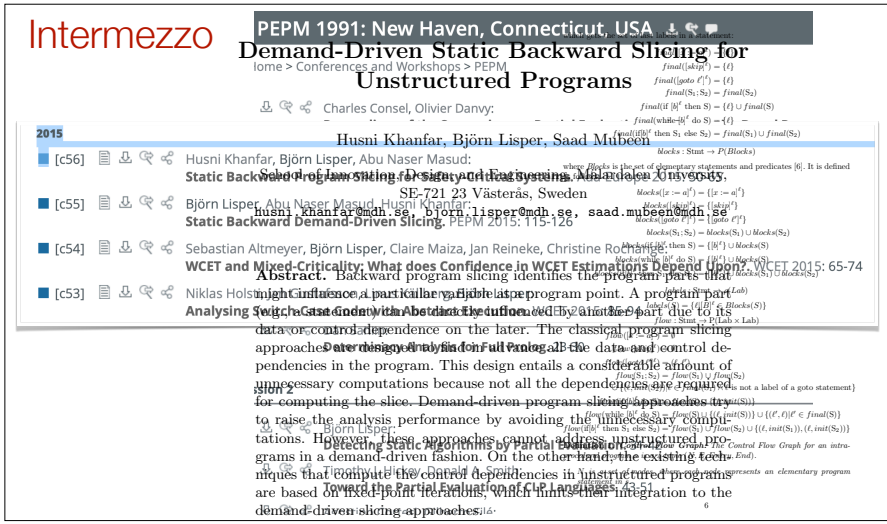
```
begin  proc fib(val z, u; res v) is[1]
           if [z<3][2] then [v:=u+1][3]
           else ([call fib(z-1,u,v)][4]_[5]; [call fib(z-2,v,v)][6]_[7])
       end[8];
       [call fib(x,0,y)][9]_[10]
end
```

We have

$$
\begin{array}{rcl}
\mathit{flow}_\star & = & \{(1,2),(2,3),(3,8), \\
 & & (2,4),(4;1),(8;5),(5,6),(6;1),(8;7),(7,8), \\
 & & (9;1),(8;10)\} \\
\mathit{interflow}_\star & = & \{(9,1,8,10),\ (4,1,8,5),\ (6,1,8,7)\}
\end{array}
$$

and $\mathit{init}_\star = 9$ and $\mathit{final}_\star = \{10\}$.

Intermezzo



A naive formulation

Treat the three kinds of flow in the same way:

flow	treat as
(ℓ_1, ℓ_2)	(ℓ_1, ℓ_2)
$(\ell_c; \ell_n)$	(ℓ_c, ℓ_n)
$(\ell_x; \ell_r)$	(ℓ_x, ℓ_r)

Equation system:

$$
\begin{array}{rcl}
A_\bullet(\ell) & = & f_\ell(A_\circ(\ell)) \\
A_\circ(\ell) & = & \bigsqcup\{A_\bullet(\ell') \mid (\ell', \ell) \in F \text{ or } (\ell', \ell) \in F \text{ or } (\ell', \ell) \in F\} \sqcup \iota^\ell_E
\end{array}
$$

But there is no matching between entries and exits.

MVP: "Meet" over Valid Paths

Complete Paths

We need to match procedure entries and exits:

A *complete path* from ℓ_1 to ℓ_2 in $P_\star$ has proper nesting of procedure entries and exits; and a procedure returns to the point where it was called:

$CP_{\ell_1,\ell_2} \longrightarrow \ell_1$ whenever $\ell_1 = \ell_2$

$CP_{\ell_1,\ell_3} \longrightarrow \ell_1, CP_{\ell_2,\ell_3}$ whenever $(\ell_1,\ell_2) \in flow_\star$

$CP_{\ell_c,\ell} \longrightarrow \ell_c, CP_{\ell_n,\ell_x}, CP_{\ell_r,\ell}$ whenever $P_\star$ contains $[\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}$ and $\texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}$

More generally: whenever $(\ell_c,\ell_n,\ell_x,\ell_r)$ is an element of $interflow_\star$ (or $interflow^R_\star$ for backward analyses); see the book.

Valid Paths

A *valid path* starts at the entry node $init_\star$ of $P_\star$, all the procedure exits match the procedure entries but some procedures might be entered but not yet exited:

$VP_\star \longrightarrow VP_{init_\star,\ell}$ whenever $\ell \in \mathbf{Lab}_\star$

$VP_{\ell_1,\ell_2} \longrightarrow \ell_1$ whenever $\ell_1 = \ell_2$

$VP_{\ell_1,\ell_3} \longrightarrow \ell_1, VP_{\ell_2,\ell_3}$ whenever $(\ell_1,\ell_2) \in flow_\star$

$VP_{\ell_c,\ell} \longrightarrow \ell_c, CP_{\ell_n,\ell_x}, VP_{\ell_r,\ell}$ whenever $P_\star$ contains $[\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}$ and $\texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}$

$VP_{\ell_c,\ell} \longrightarrow \ell_c, VP_{\ell_n,\ell}$ whenever $P_\star$ contains $[\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}$ and $\texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}$

The MVP solution

$$MVP_\circ(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in vpath_\circ(\ell)\}$$

where

$$vpath_\circ(\ell) = \{[\ell_1,\cdots,\ell_{n-1}] \mid n \geq 1 \wedge \ell_n = \ell \wedge [\ell_1,\cdots,\ell_n] \text{ is a valid path}\}$$

The MVP solution may be undecidable for lattices satisfying the Ascending Chain Condition, just as was the case for the MOP solution.

The MVP solution

$$MVP_\circ(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in vpath_\circ(\ell)\}$$

where

$$vpath_\circ(\ell) = \{[\ell_1,\cdots,\ell_{n-1}] \mid n \geq 1 \wedge \ell_n = \ell \wedge [\ell_1,\cdots,\ell_n] \text{ is a valid path}\}$$

QUESTIONS?

The MVP solution may be undecidable for lattices satisfying the Ascending Chain Condition, just as was the case for the MOP solution.

Making Context Explicit

Starting point: an instance $(L, \mathcal{F}, F, E, \iota, f.)$ of a Monotone Framework

- the analysis is forwards, i.e. $F = \mathit{flow}_\star$ and $E = \{\mathit{init}_\star\}$;
- the complete lattice is a powerset, i.e. $L = \mathcal{P}(D)$;
- the transfer functions in $\mathcal{F}$ are completely additive; and
- each f_ℓ is given by $f_\ell(Y) = \bigcup\{\phi_\ell(d) \mid d \in Y\}$ where $\phi_\ell : D \to \mathcal{P}(D)$.

An embellished monotone framework

- $L' = \mathcal{P}(\Delta \times D)$;
- the transfer functions in $\mathcal{F}'$ are completely additive; and
- each f'_ℓ is given by $f'_\ell(Z) = \bigcup\{\{\delta\} \times \phi_\ell(d) \mid (\delta, d) \in Z\}$.

Ignoring procedures, the data flow equations will take the form:

$$A_\bullet(\ell) = f'_\ell(A_\circ(\ell))$$
for all labels that do not label a procedure call

$$A_\circ(\ell) = \bigsqcup\{A_\bullet(\ell') \mid (\ell', \ell) \in F \text{ or } (\ell'; \ell) \in F\} \sqcup \iota^\ell_E$$
for all labels (including those that label procedure calls)

Example:

Detection of Signs Analysis as a Monotone Framework:

$(L_{\mathsf{sign}}, \mathcal{F}_{\mathsf{sign}}, F, E, \iota_{\mathsf{sign}}, f.^{\mathsf{sign}})$ where $\mathbf{Sign} = \{-, 0, +\}$ and

$$L_{\mathsf{sign}} = \mathcal{P}(\mathbf{Var}_\star \to \mathbf{Sign})$$

The transfer function f^{sign}_ℓ associated with the assignment $[x := a]^\ell$ is

$$f^{\mathsf{sign}}_\ell(Y) = \bigcup\{\phi^{\mathsf{sign}}_\ell(\sigma^{\mathsf{sign}}) \mid \sigma^{\mathsf{sign}} \in Y\}$$

where $Y \subseteq \mathbf{Var}_\star \to \mathbf{Sign}$ and

$$\phi^{\mathsf{sign}}_\ell(\sigma^{\mathsf{sign}}) = \{\sigma^{\mathsf{sign}}[x \mapsto s] \mid s \in \mathcal{A}_{\mathsf{sign}}[\![a]\!](\sigma^{\mathsf{sign}})\}$$

Example (cont.):

Detection of Signs Analysis as an embellished monotone framework

$$L'_{\mathsf{sign}} = \mathcal{P}(\Delta \times (\mathbf{Var}_\star \to \mathbf{Sign}))$$

The transfer function associated with $[x := a]^\ell$ will now be:

$$f^{\mathsf{sign}\prime}_\ell(Z) = \bigcup\{\{\delta\} \times \phi^{\mathsf{sign}}_\ell(\sigma^{\mathsf{sign}}) \mid (\delta, \sigma^{\mathsf{sign}}) \in Z\}$$

Transfer functions for procedure declarations

Procedure declarations

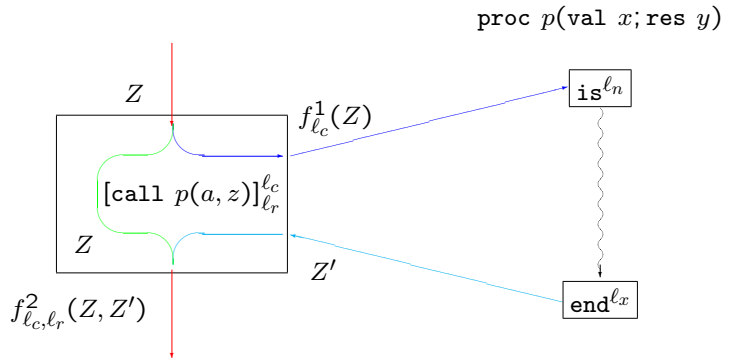
$$\texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}$$

have two transfer functions, one for entry and one for exit:

$$f_{\ell_n}, f_{\ell_x} : \mathcal{P}(\Delta \times D) \to \mathcal{P}(\Delta \times D)$$

For simplicity we take both to be the identity function (thus incorporating procedure entry as part of procedure call, and procedure exit as part of procedure return).

Procedure calls and returns



Transfer functions for procedure calls

Procedure calls $[\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}$ have two transfer functions:

For the *procedure call*

$$f^1_{\ell_c} : \mathcal{P}(\Delta \times D) \to \mathcal{P}(\Delta \times D)$$

and it is used in the equation:

$$A_\bullet(\ell_c) \ = \ f^1_{\ell_c}(A_\circ(\ell_c)) \quad \text{for all procedure calls } [\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}$$

For the *procedure return*

$$f^2_{\ell_c,\ell_r} : \boxed{\mathcal{P}(\Delta \times D)} \times \mathcal{P}(\Delta \times D) \to \mathcal{P}(\Delta \times D)$$

and it is used in the equation:

$$A_\bullet(\ell_r) \ = \ f^2_{\ell_c,\ell_r}(\boxed{A_\circ(\ell_c)}, A_\circ(\ell_r)) \quad \text{for all procedure calls } [\texttt{call}\ p(a,z)]^{\ell_c}_{\ell_r}$$

(Note that $A_\circ(\ell_r)$ will equal $A_\bullet(\ell_x)$ for the relevant procedure exit.)

Different Kinds of Context

- Call Strings — contexts based on control
  - Call strings of unbounded length
  - Call strings of bounded length (k)
- Assumption Sets — contexts based on data
  - Large assumption sets ($k = 1$)

Call Strings of Unbounded Length

$$\Delta = \mathbf{Lab}^*$$

Transfer functions for procedure call

$$f^1_{\ell_c}(Z) = \bigcup\{\{\delta'\} \times \phi^1_{\ell_c}(d) \mid (\delta, d) \in Z \ \wedge \ \delta' = [\delta, \ell_c]\}$$

$$f^2_{\ell_c,\ell_r}(Z, Z') = \bigcup\{\{\delta\} \times \phi^2_{\ell_c,\ell_r}(d, d') \mid (\delta, d) \in Z \ \wedge \ (\delta', d') \in Z' \ \wedge \ \delta' = [\delta, \ell_c]\}$$

Example:

Recalling the statements:

$$\texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x} \qquad [\texttt{call}\ p(a, z)]^{\ell_c}_{\ell_r}$$

Detection of Signs Analysis:

$$\phi^{\mathsf{sign}1}_{\ell_c}(\sigma^{\mathsf{sign}}) = \{\sigma^{\mathsf{sign}}\overbrace{[x \mapsto s][y \mapsto s']}^{\text{initialise formals}} \mid s \in \mathcal{A}_{\mathsf{sign}}[\![a]\!](\sigma^{\mathsf{sign}}), s' \in \{-, 0, +\}\}$$

$$\phi^{\mathsf{sign}2}_{\ell_c,\ell_r}(\sigma^{\mathsf{sign}}_1, \sigma^{\mathsf{sign}}_2) = \{\sigma^{\mathsf{sign}}_2\underbrace{[x \mapsto \sigma^{\mathsf{sign}}_1(x)][y \mapsto \sigma^{\mathsf{sign}}_1(y)]}_{\text{restore formals}}\underbrace{[z \mapsto \sigma^{\mathsf{sign}}_2(y)]}_{\text{return result}}\}$$

Example:

Recalling the statements:

$$\texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x} \qquad [\texttt{call}\ p(a, z)]^{\ell_c}_{\ell_r}$$

Detection of Signs Analysis:

$$\phi^{\mathsf{sign}1}_{\ell_c}(\sigma^{\mathsf{sign}}) = \{\sigma^{\mathsf{sign}}\overbrace{[x \mapsto s][y \mapsto s']}^{\text{initialise formals}} \mid s \in \mathcal{A}_{\mathsf{sign}}[\![a]\!](\sigma^{\mathsf{sign}}), s' \in \{-, 0, +\}\}$$

$$\phi^{\mathsf{sign}2}_{\ell_c,\ell_r}(\sigma^{\mathsf{sign}}_1, \sigma^{\mathsf{sign}}_2) = \{\sigma^{\mathsf{sign}}_2\underbrace{[x \mapsto \sigma^{\mathsf{sign}}_1(x)][y \mapsto \sigma^{\mathsf{sign}}_1(y)]}_{\text{restore formals}}\underbrace{[z \mapsto \sigma^{\mathsf{sign}}_2(y)]}_{\text{return result}}\}$$

QUESTIONS?

Call Strings of Bounded Length

$$\Delta = \mathbf{Lab}^{\leq k}$$

Transfer functions for procedure call

$$f^1_{\ell_c}(Z) = \bigcup\{\{\delta'\} \times \phi^1_{\ell_c}(d) \mid (\delta, d) \in Z \ \wedge \ \delta' = \lceil \delta, \ell_c \rceil_k\}$$

$$f^2_{\ell_c,\ell_r}(Z, Z') = \bigcup\{\{\delta\} \times \phi^2_{\ell_c,\ell_r}(d, d') \mid (\delta, d) \in Z \ \wedge \ (\delta', d') \in Z' \ \wedge \ \delta' = \lceil \delta, \ell_c \rceil_k\}$$

A special case: call strings of length $k = 0$

$$\Delta = \{\Lambda\}$$

Note: this is equivalent to having no context information!

Specialising the transfer functions:

$$f^1_{\ell_c}(Y) = \bigcup\{\phi^1_{\ell_c}(d) \mid d \in Y\}$$

$$f^2_{\ell_c,\ell_r}(Y,Y') = \bigcup\{\phi^2_{\ell_c,\ell_r}(d,d') \mid d \in Y \ \wedge \ d' \in Y'\}$$

(We use that $\mathcal{P}(\Delta \times D)$ isomorphic to $\mathcal{P}(D)$.)

A special case: call strings of length $k = 1$

$$\Delta = \mathbf{Lab} \cup \{\Lambda\}$$

Specialising the transfer functions:

$$f^1_{\ell_c}(Z) = \bigcup\{\{\ell_c\} \times \phi^1_{\ell_c}(d) \mid (\delta,d) \in Z\}$$

$$f^2_{\ell_c,\ell_r}(Z,Z') = \bigcup\{\{\delta\} \times \phi^2_{\ell_c,\ell_r}(d,d') \mid (\delta,d) \in Z \ \wedge \ (\ell_c,d') \in Z'\}$$

Large Assumption Sets $(k = 1)$

$\Delta = \mathcal{P}(D)$

Transfer functions for procedure call

$$f^1_{\ell_c}(Z) = \bigcup\{\{\delta'\} \times \phi^1_{\ell_c}(d) \mid \ (\delta,d) \in Z \ \wedge \ \delta' = \{d'' \mid (\delta, d'') \in Z\}\}$$

$$f^2_{\ell_c,\ell_r}(Z,Z') = \bigcup\{\{\delta\} \times \phi^2_{\ell_c,\ell_r}(d,d') \mid \ (\delta,d) \in Z \ \wedge \ (\delta',d') \in Z' \wedge \delta' = \{d'' \mid (\delta, d'') \in Z\}\}$$

Intermezzo: PEPM'91

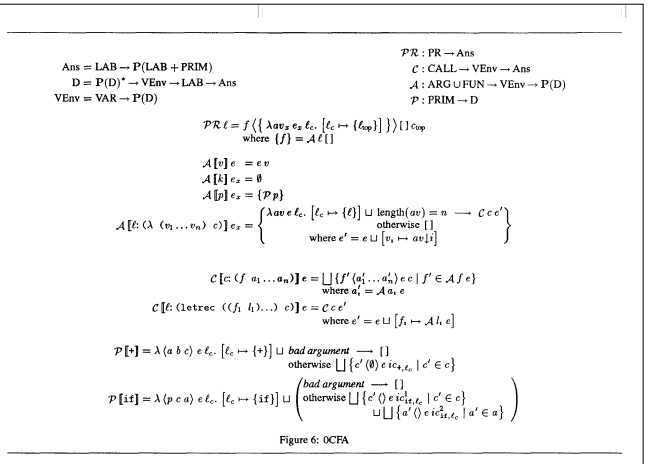


Figure 6: 0CFA

Shivers. *The Semantics of Scheme Control-Flow Analysis*, PEPM'91 https://doi.org/10.1145/115865.115884

Shape Analysis

Goal: to obtain a finite representation of the shape of the heap of a language with pointers.

The analysis result can be used for

- detection of pointer aliasing
- detection of sharing between structures
- software development tools
  – detection of errors like dereferences of `nil`-pointers
- program verification
  – `reverse` transforms a non-cyclic list to a non-cyclic list

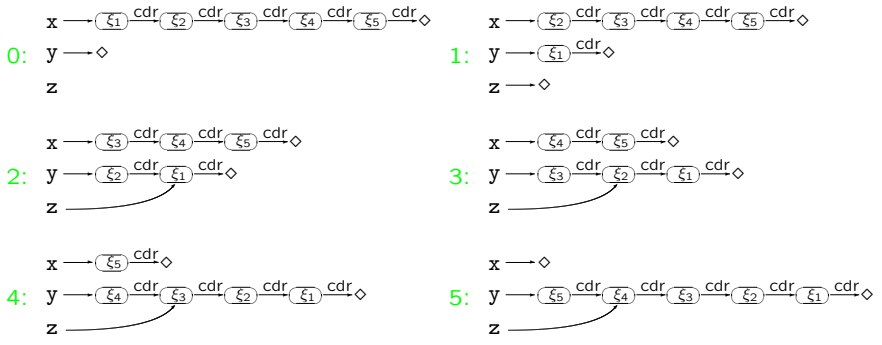
Syntax of the pointer language

$$
\begin{array}{rcl}
a & ::= & p \mid n \mid a_1\ op_a\ a_2 \mid \texttt{nil} \\
p & ::= & x \mid x.sel \\
b & ::= & \texttt{true} \mid \texttt{false} \mid \texttt{not}\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2 \mid op_p\ p \\
S & ::= & [p\texttt{:=}a]^\ell \mid [\texttt{skip}]^\ell \mid S_1;\ S_2 \mid \\
 & & \texttt{if}\ [b]^\ell\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \texttt{while}\ [b]^\ell\ \texttt{do}\ S \mid \\
 & & [\texttt{malloc}\ p]^\ell
\end{array}
$$

Example

```
[y:=nil]^1;
while [not is-nil(x)]^2 do
   ([z:=y]^3; [y:=x]^4; [x:=x.cdr]^5; [y.cdr:=z]^6);
[z:=nil]^7
```


Reversal of a list



Intermezzo: meaning of car/cdr

The IBM 704 had 36-bit words and a 15-bit address space. … The first 15-bit field was the operand address and the second held a decrement. … Indexing was a subtractive process on the 704, hence the value to be loaded into an index register was called a "decrement". The 704 hardware had special instructions for accessing the address and decrement fields in a word. As a result it was efficient to use those two fields to store within a single word the two pointers needed for a list.

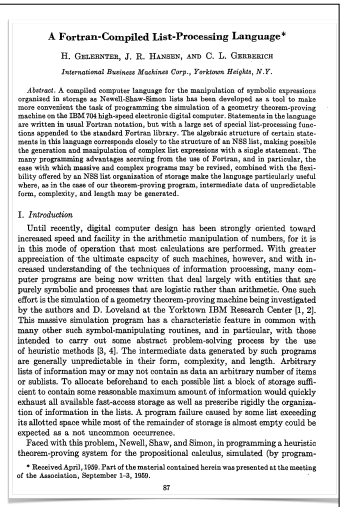
Thus, "CAR" is "Contents of the Address *part* of the Register". The term "register" in this context refers to "memory location".

Precursors to Lisp included functions:
- `car` ("contents of the address part of register number"),
- `cdr` ("contents of the decrement part of register number"),

each of which took a machine address as an argument, loaded the corresponding word from memory, and extracted the appropriate bits.

Wikipedia

H Gelernter, J. R. Hansen, C. L. Gerberich. "*A Fortran-Compiled List-Processing Language*" J. ACM 1960 doi=10.1145/321021.321022



A Fortran-Compiled List-Processing Language*

H. Gelernter, J. R. Hansen, and C. L. Gerberich

International Business Machines Corp., Yorktown Heights, N.Y.

Abstract. A compiled computer language for the manipulation of symbolic expressions organized in storage as Newell-Shaw-Simon lists has been developed as a tool to make more convenient the task of programming the simulation of a geometry theorem-proving machine on the IBM 704 high-speed electronic digital computer. Statements in the language are written in usual Fortran notation, but with a large set of special list-processing functions appended to the standard Fortran library. The algebraic structure of certain statements in this language corresponds closely to the structure of an NSS list, making possible the generation and manipulation of complex list expressions with a single statement. The many programming advantages accruing from the use of Fortran, and in particular, the ease with which massive and complex programs may be revised, combined with the flexibility offered by an NSS list organization of storage make the language particularly useful where, as in the case of our theorem-proving program, intermediate data of unpredictable form, complexity, and length may be generated.

I. *Introduction*

Until recently, digital computer design has been strongly oriented toward increased speed and facility in the arithmetic manipulation of numbers, for it is in this mode of operation that most calculations are performed. With greater appreciation of the ultimate capacity of such machines, however, and with increased understanding of the techniques of information processing, many computer programs are being now written that deal largely with entities that are purely symbolic and processes that are logistic rather than arithmetic. One such effort is the simulation of a geometry theorem-proving machine being investigated by the authors and D. Loveland at the Yorktown IBM Research Center [1, 2]. This massive simulation program has a characteristic feature in common with many other such symbol-manipulating routines, and in particular, with those intended to carry out some abstract problem-solving process by the use of heuristic methods [3, 4]. The intermediate data generated by such programs are generally unpredictable in their form, complexity, and length. Arbitrary lists of information may or may not contain as data an arbitrary number of items or sublists. To allocate beforehand to each possible list a block of storage sufficient to contain some reasonable maximum amount of information would quickly exhaust all available fast-access storage as well as prescribe rigidly the organization of information in the lists. A program failure caused by some list exceeding its allotted space while most of the remainder of storage is almost empty could be expected as a not uncommon occurrence.

Faced with this problem, Newell, Shaw, and Simon, in programming a heuristic theorem-proving system for the propositional calculus, simulated (by program-

* Received April, 1959. Part of the material contained herein was presented at the meeting of the Association, September 1-3, 1959.

87

Structural Operational Semantics

A configurations consists of

- a state $\sigma \in \mathbf{State} = \mathbf{Var}_\star \to (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$

  mapping variables to values, locations (in the heap) or the nil-value

- a heap $\mathcal{H} \in \mathbf{Heap} = (\mathbf{Loc} \times \mathbf{Sel}) \to_{\text{fin}} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$

  mapping pairs of locations and selectors to values, locations in the heap or the nil-value

Pointer expressions

$$\wp \;:\; \mathbf{PExp} \to (\mathbf{State} \times \mathbf{Heap}) \to_{\text{fin}} (\mathbf{Z} + \{\diamond\} + \mathbf{Loc})$$

is defined by

$$\wp[\![x]\!](\sigma,\mathcal{H}) \;=\; \sigma(x)$$

$$\wp[\![x.sel]\!](\sigma,\mathcal{H}) \;=\; \begin{cases} \mathcal{H}(\sigma(x), sel) & \text{if } \sigma(x) \in \mathbf{Loc} \text{ and } \mathcal{H} \text{ is defined on } (\sigma(x), sel) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Arithmetic and boolean expressions

$$\mathcal{A} \;:\; \mathbf{AExp} \to (\mathbf{State} \times \mathbf{Heap}) \to_{\text{fin}} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$$

$$\mathcal{B} \;:\; \mathbf{BExp} \to (\mathbf{State} \times \mathbf{Heap}) \to_{\text{fin}} \mathbf{T}$$

Statements

Clauses for assignments:

$$\langle [x\texttt{:=}a]^\ell, \sigma, \mathcal{H}\rangle \to \langle \sigma[x \mapsto \mathcal{A}[\![a]\!](\sigma,\mathcal{H})], \mathcal{H}\rangle$$

if $\mathcal{A}[\![a]\!](\sigma,\mathcal{H})$ is defined

$$\langle [x.sel\texttt{:=}a]^\ell, \sigma, \mathcal{H}\rangle \to \langle \sigma, \mathcal{H}[(\sigma(x), sel) \mapsto \mathcal{A}[\![a]\!](\sigma,\mathcal{H})]\rangle$$

if $\sigma(x) \in \mathbf{Loc}$ and $\mathcal{A}[\![a]\!](\sigma,\mathcal{H})$ is defined

Clauses for malloc:

$$\langle [\texttt{malloc } x]^\ell, \sigma, \mathcal{H}\rangle \to \langle \sigma[x \mapsto \xi], \mathcal{H}\rangle$$

where ξ does not occur in σ or $\mathcal{H}$

$$\langle [\texttt{malloc } (x.sel)]^\ell, \sigma, \mathcal{H}\rangle \to \langle \sigma, \mathcal{H}[(\sigma(x), sel) \mapsto \xi]\rangle$$

where ξ does not occur in σ or $\mathcal{H}$ and $\sigma(x) \in \mathbf{Loc}$

Shape graphs

The analysis will operate on *shape graphs* (S, H, is) consisting of

- an abstract state, S,
- an abstract heap, H, and
- sharing information, is, for the abstract locations.

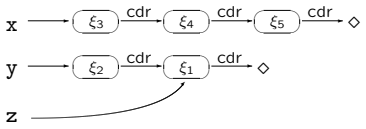
The nodes of the shape graphs are abstract locations:

$$\mathbf{ALoc} = \{n_X \mid X \subseteq \mathbf{Var}_\star\}$$

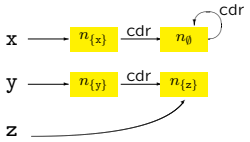
Note: there will only be *finitely* many abstract locations

Example

In the semantics:



In the analysis:



Abstract Locations

The abstract location n_X represents the location $\sigma(x)$ if $x \in X$

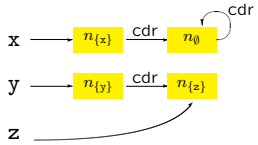
The abstract location $n_\emptyset$ is called the *abstract summary location*: $n_\emptyset$ represents all the locations that cannot be reached directly from the state without consulting the heap

Invariant 1 If two abstract locations n_X and n_Y occur in the same shape graph then either $X = Y$ or $X \cap Y = \emptyset$

Abstract states and heaps

$$S \in \mathbf{AState} = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{ALoc}) \quad \text{abstract states}$$

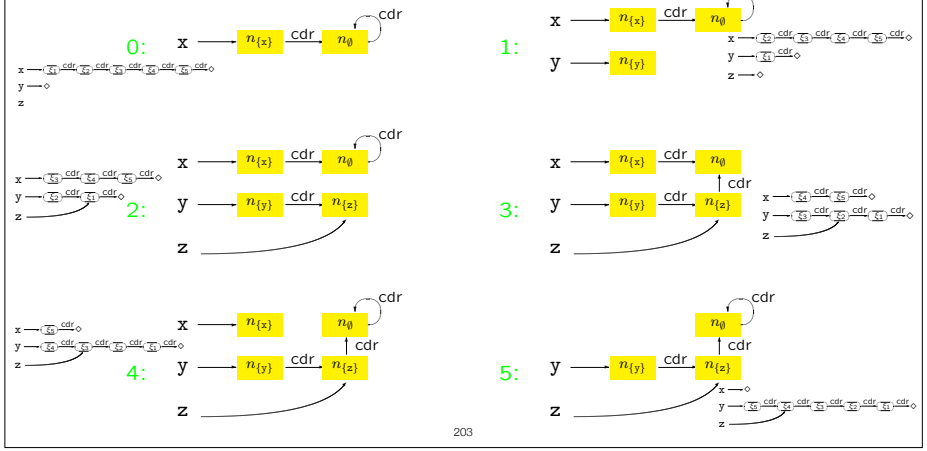
$$H \in \mathbf{AHeap} = \mathcal{P}(\mathbf{ALoc} \times \mathbf{Sel} \times \mathbf{ALoc}) \quad \text{abstract heap}$$



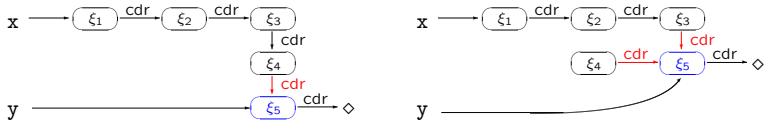
Invariant 2 If x is mapped to n_X by the abstract state S then $x \in X$

Invariant 3 Whenever (n_V, sel, n_W) and $(n_V, sel, n_{W'})$ are in the abstract heap H then either $V = \emptyset$ or $W = W'$

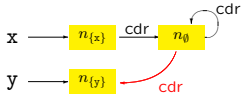
Reversal of a list



Sharing in the heap



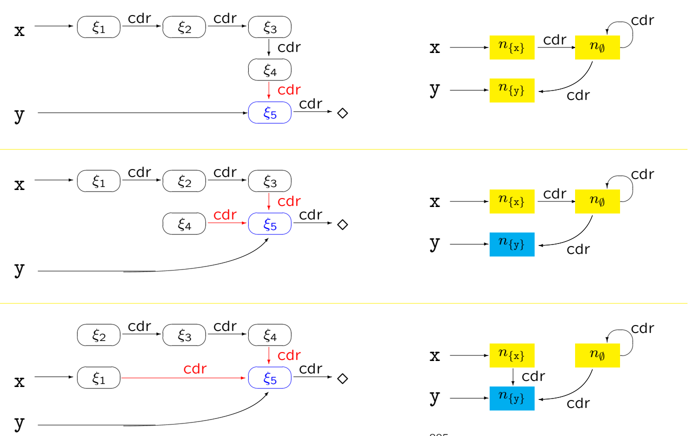
Give rise to the same shape graph:



is: the abstract locations that *might* be shared due to pointers in the heap:

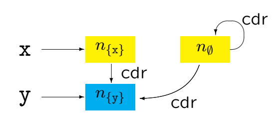
n_X is included in is if it might represents a location that is the target of more than one pointer in the heap

Examples: sharing in the heap



Sharing information

The implicit sharing information of the abstract heap must be consistent with the explicit sharing information:



Invariant 4 If $n_X \in$ is then either

- $(n_\emptyset, sel, n_X)$ is in the abstract heap for some *sel*, or
- there are two distinct triples (n_V, sel_1, n_X) and (n_W, sel_2, n_X) in the abstract heap

Invariant 5 Whenever there are two distinct triples (n_V, sel_1, n_X) and (n_W, sel_2, n_X) in the abstract heap and $X \neq \emptyset$ then $n_X \in$ is

The complete lattice of shape graphs

A *shape graph* is a triple (S,H,is) where

$$\begin{array}{lcll} \mathsf{S} & \in & \mathbf{AState} & = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{ALoc}) \\ \mathsf{H} & \in & \mathbf{AHeap} & = \mathcal{P}(\mathbf{ALoc} \times \mathbf{Sel} \times \mathbf{ALoc}) \\ \mathsf{is} & \in & \mathbf{IsShared} & = \mathcal{P}(\mathbf{ALoc}) \end{array}$$

and $\mathbf{ALoc} = \{n_Z \mid Z \subseteq \mathbf{Var}_\star\}$.

A shape graph (S, H, is) is *compatible* if it fulfils the five invariants.

The analysis computes over *sets of compatible shape graphs*

$$\mathbf{SG} = \{(\mathsf{S}, \mathsf{H}, \mathsf{is}) \mid (\mathsf{S}, \mathsf{H}, \mathsf{is}) \text{ is compatible}\}$$

The analysis

An instance of a *forward* Monotone Framework with the complete lattice of interest being $\mathcal{P}(\mathbf{SG})$

A *may analysis*: each of the sets of shape graphs computed by the analysis may contain shape graphs that cannot really arrise

Aspects of a *must analysis*: each of the individual shape graphs (in a set of shape graphs computed by the analysis) will be the best possible description of some $(\sigma, \mathcal{H})$

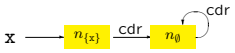
The analysis

Equations:

$$Shape_{\circ}(\ell) = \begin{cases} \iota & \text{if } \ell = init(S_\star) \\ \bigcup\{Shape_{\bullet}(\ell') \mid (\ell', \ell) \in flow(S_\star)\} & \text{otherwise} \end{cases}$$

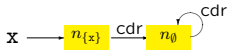
$$Shape_{\bullet}(\ell) = f^{SA}_{\ell}(Shape_{\circ}(\ell))$$

Example: The extremal value ι for the list reversal program



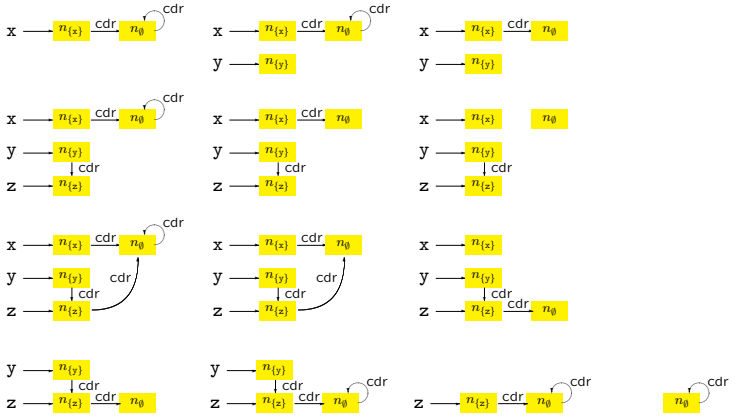
– `x` points to a non-cyclic list with at least three elements

$Shape_{\bullet}(1)$ for [y:=nil]^1

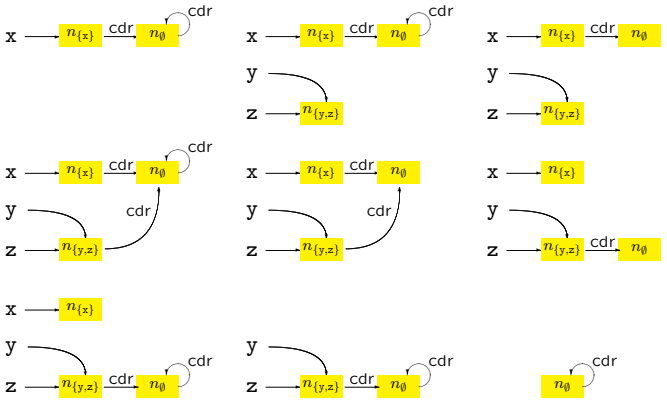


Note: we do not record `nil`-values in the analysis

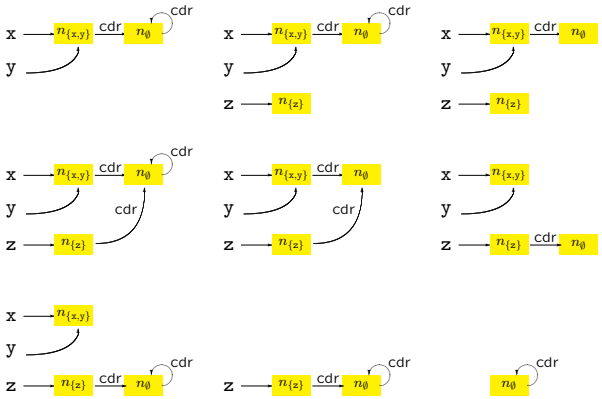
$Shape_{\bullet}(2)$ for [not is-nil(x)]^2



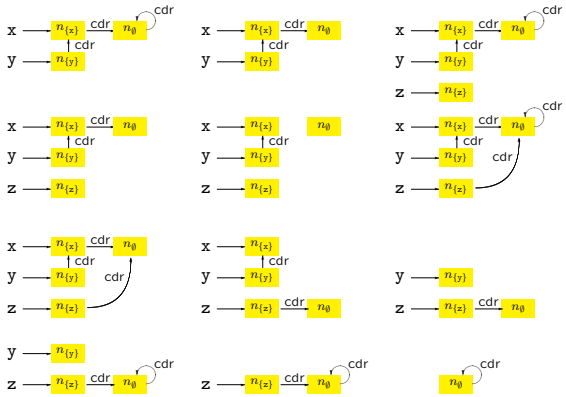
$Shape_{\bullet}(3)$ for [z:=y]^3



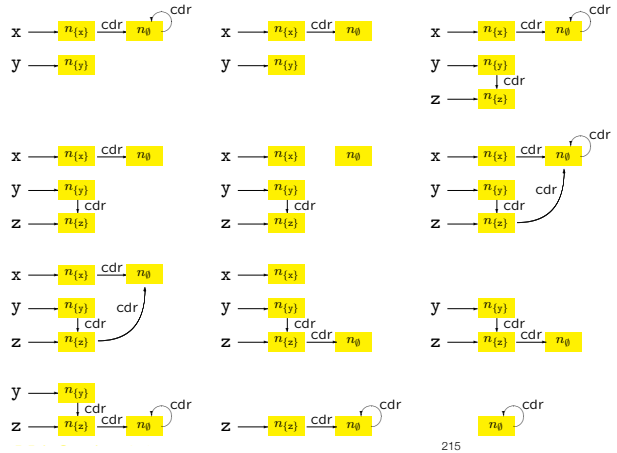
$Shape_{\bullet}(4)$ for [y:=x]^4



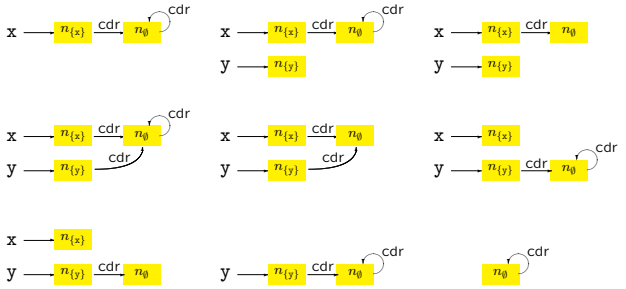
$Shape_{\bullet}(5)$ for [x:=x.cdr]^5



$Shape_{\bullet}(6)$ for [y.cdr:=z]^6



$Shape_{\bullet}(7)$ for [z:=nil]^7



– upon termination y points to a non-circular list

– a more precise analysis taking tests into account will know that x is nil upon termination

Transfer functions

$$f_\ell^{\mathsf{SA}} : \mathcal{P}(\mathbf{SG}) \to \mathcal{P}(\mathbf{SG})$$

has the form:

$$f_\ell^{\mathsf{SA}}(SG) = \bigcup\{\phi_\ell^{\mathsf{SA}}((\mathsf{S},\mathsf{H},\mathsf{is})) \mid (\mathsf{S},\mathsf{H},\mathsf{is}) \in SG\}$$

where

$$\phi_\ell^{\mathsf{SA}} : \mathbf{SG} \to \mathcal{P}(\mathbf{SG})$$

specifies how a *single* shape graph (in $Shape_\circ(\ell)$) may be transformed into a *set* of shape graphs (in $Shape_\bullet(\ell)$) by the elementary block.

Transfer function for $[b]^\ell$ and $[\mathtt{skip}]^\ell$

We are only interested in the shape of the heap – and it is not changed by these elementary blocks:

$$\phi_\ell^{\mathsf{SA}}((\mathsf{S},\mathsf{H},\mathsf{is})) = \{(\mathsf{S},\mathsf{H},\mathsf{is})\}$$

Transfer function for $[x\mathtt{:=}a]^\ell$

— where a is of the form n, a_1 op_a a_2 or $\mathtt{nil}$

$$\phi_\ell^{\mathsf{SA}}((\mathsf{S},\mathsf{H},\mathsf{is})) = \{kill_x((\mathsf{S},\mathsf{H},\mathsf{is}))\}$$

where $kill_x((\mathsf{S},\mathsf{H},\mathsf{is})) = (\mathsf{S}',\mathsf{H}',\mathsf{is}')$ is

$$\begin{aligned}
\mathsf{S}' &= \{(z, k_x(n_Z)) \mid (z, n_Z) \in S \ \wedge\ z \neq x\}\\
\mathsf{H}' &= \{(k_x(n_V), \mathit{sel}, k_x(n_W)) \mid (n_V, \mathit{sel}, n_W) \in \mathsf{H}\}\\
\mathsf{is}' &= \{k_x(n_X) \mid n_X \in \mathsf{is}\}
\end{aligned}$$

and

$$k_x(n_Z) = n_{Z\setminus\{x\}}$$

Idea: all abstract locations are renamed to not having x in their name set

Transfer function for $[x\mathtt{:=}a]^\ell$

— where a is of the form n, a_1 op_a a_2 or $\mathtt{nil}$

QUESTIONS?

$$\phi_\ell^{\mathsf{SA}}((\mathsf{S},\mathsf{H},\mathsf{is})) = \{kill_x((\mathsf{S},\mathsf{H},\mathsf{is}))\}$$

where $kill_x((\mathsf{S},\mathsf{H},\mathsf{is})) = (\mathsf{S}',\mathsf{H}',\mathsf{is}')$ is

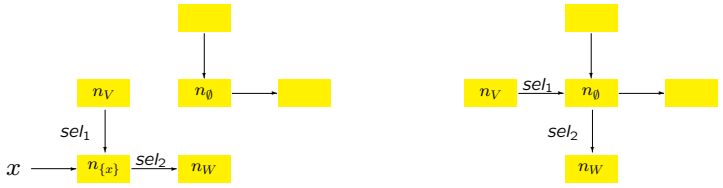
$$\begin{aligned}
\mathsf{S}' &= \{(z, k_x(n_Z)) \mid (z, n_Z) \in S \ \wedge\ z \neq x\}\\
\mathsf{H}' &= \{(k_x(n_V), \mathit{sel}, k_x(n_W)) \mid (n_V, \mathit{sel}, n_W) \in \mathsf{H}\}\\
\mathsf{is}' &= \{k_x(n_X) \mid n_X \in \mathsf{is}\}
\end{aligned}$$

and

$$k_x(n_Z) = n_{Z\setminus\{x\}}$$

Idea: all abstract locations are renamed to not having x in their name set

The effect of $[x\texttt{:=nil}]^\ell$



$(\mathsf{S}, \mathsf{H}, \mathsf{is})$ $(\mathsf{S}', \mathsf{H}', \mathsf{is}')$

Transfer function for $[x\texttt{:=}y]^\ell$ when $x \neq y$

$$\phi^{\mathsf{SA}}_\ell((\mathsf{S}, \mathsf{H}, \mathsf{is})) = \{(\mathsf{S}'', \mathsf{H}'', \mathsf{is}'')\}$$

where $(\mathsf{S}', \mathsf{H}', \mathsf{is}') = kill_x((\mathsf{S}, \mathsf{H}, \mathsf{is}))$ and

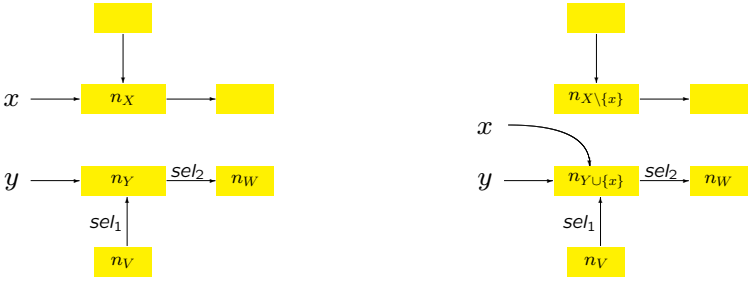
$$\begin{aligned}
\mathsf{S}'' &= \{(z, g^y_x(n_Z)) \mid (z, n_Z) \in \mathsf{S}'\} \\
&\quad \cup \{(x, g^y_x(n_Y)) \mid (y', n_Y) \in \mathsf{S}' \wedge y' = y\} \\
\mathsf{H}'' &= \{(g^y_x(n_V), sel, g^y_x(n_W)) \mid (n_V, sel, n_W) \in \mathsf{H}'\} \\
\mathsf{is}'' &= \{g^y_x(n_Z) \mid n_Z \in \mathsf{is}'\}
\end{aligned}$$

and

$$g^y_x(n_Z) = \begin{cases} n_{Z\cup\{x\}} & \text{if } y \in Z \\ n_Z & \text{otherwise} \end{cases}$$

Idea: all abstract locations are renamed to also have x in their name set if they already have y

The effect of $[x\texttt{:=}y]^\ell$ when $x \neq y$



$(\mathsf{S}, \mathsf{H}, \mathsf{is})$ $(\mathsf{S}'', \mathsf{H}'', \mathsf{is}'')$

Transfer function for $[x\texttt{:=}y.sel]^\ell$ when $x \neq y$

Remove the old binding for x: strong nullification

$$(\mathsf{S}', \mathsf{H}', \mathsf{is}') = kill_x((\mathsf{S}, \mathsf{H}, \mathsf{is}))$$

Establish the new binding for x:

1. There is no abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ – or there is an abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ but no n_Z such that $(n_Y, sel, n_Z) \in \mathsf{H}'$
2. There is an abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ and there is an abstract location $n_U \neq n_\emptyset$ such that $(n_Y, sel, n_U) \in \mathsf{H}'$
3. There is an abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ and $(n_Y, sel, n_\emptyset) \in \mathsf{H}'$

Case 1 for $[x{:=}y.sel]^\ell$

Assume there is no abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$

$$\phi^{\mathsf{SA}}_\ell((\mathsf{S}, \mathsf{H}, \mathsf{is})) = \{(\mathsf{S}', \mathsf{H}', \mathsf{is}')\}$$

OBS: dereference of a `nil`-pointer

Assume there is an abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ but there is no abstract location n such that $(n_Y, sel, n) \in \mathsf{H}'$

$$\phi^{\mathsf{SA}}_\ell((\mathsf{S}, \mathsf{H}, \mathsf{is})) = \{(\mathsf{S}', \mathsf{H}', \mathsf{is}')\}$$

OBS: dereference of a non-existing *sel*-field

Case 2 for $[x{:=}y.sel]^\ell$

Assume there is an abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ and there is an abstract location $n_U \neq n_\emptyset$ such that $(n_Y, sel, n_U) \in \mathsf{H}'$.

The abstract location n_U will be renamed to include the variable x using the function:

$$h^U_x(n_Z) = \begin{cases} n_{U \cup \{x\}} & \text{if } Z = U \\ n_Z & \text{otherwise} \end{cases}$$

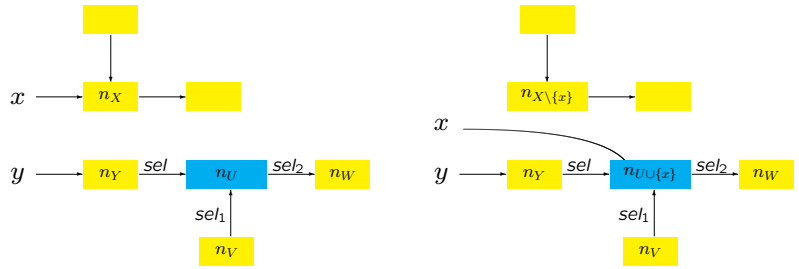
We take

$$\phi^{\mathsf{SA}}_\ell((\mathsf{S}, \mathsf{H}, \mathsf{is})) = \{(\mathsf{S}'', \mathsf{H}'', \mathsf{is}'')\}$$

where $(\mathsf{S}', \mathsf{H}', \mathsf{is}') = kill_x((\mathsf{S}, \mathsf{H}, \mathsf{is}))$ and

$$\begin{aligned} \mathsf{S}'' &= \{(z, h^U_x(n_Z)) \mid (z, n_Z) \in \mathsf{S}'\} \cup \{(x, h^U_x(n_U))\} \\ \mathsf{H}'' &= \{(h^U_x(n_V), sel', h^U_x(n_W)) \mid (n_V, sel', n_W) \in \mathsf{H}'\} \\ \mathsf{is}'' &= \{h^U_x(n_Z) \mid n_Z \in \mathsf{is}'\} \end{aligned}$$

The effect of $[x{:=}y.sel]^\ell$ in Case 2

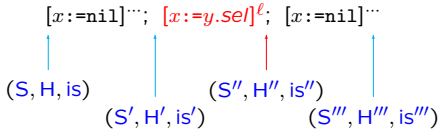


$(\mathsf{S}, \mathsf{H}, \mathsf{is})$ $(\mathsf{S}'', \mathsf{H}'', \mathsf{is}'')$

Case 3 for $[x{:=}y.sel]^\ell$ (1)

Assume that there is an abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ and furthermore $(n_Y, sel, n_\emptyset) \in \mathsf{H}'$.

We have to *materialise* a new abstract location $n_{\{x\}}$ from $n_\emptyset$.



Idea:

$$(\mathsf{S}', \mathsf{H}', \mathsf{is}') = (\mathsf{S}''', \mathsf{H}''', \mathsf{is}''') = kill_x((\mathsf{S}'', \mathsf{H}'', \mathsf{is}''))$$

Case 3 for $[x{:=}y.sel]^\ell$ (2)

Transfer function:

$$\phi^{SA}_\ell((S,H,is)) = \{(S'',H'',is'') \mid (S'',H'',is'') \text{ is compatible} \wedge kill_x((S'',H'',is'')) = (S',H',is') \wedge (x,n_{\{x\}}) \in S'' \wedge (n_Y,sel,n_{\{x\}}) \in H''\}$$

where $(S',H',is') = kill_x((S,H,is))$.

Case 3 for $[x{:=}y.sel]^\ell$ (2)

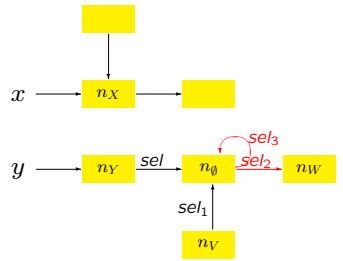
Transfer function:

$$\phi^{SA}_\ell((S,H,is)) = \{(S'',H'',is'') \mid (S'',H'',is'') \text{ is compatible} \wedge kill_x((S'',H'',is'')) = (S',H',is') \wedge (x,n_{\{x\}}) \in S'' \wedge (n_Y,sel,n_{\{x\}}) \in H''\}$$

where $(S',H',is') = kill_x((S,H,is))$.

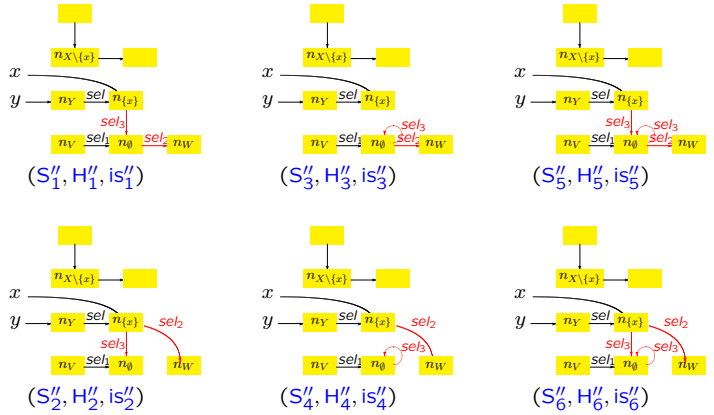
QUESTIONS?

The effect of $[x{:=}y.sel]^\ell$ in Case 3 (1)



(S,H,is)

The effect of $[x{:=}y.sel]^\ell$ in Case 3 (2)



Transfer function for $[x.sel := a]^\ell$

— where a is of the form n, a_1 op_a a_2 or `nil`.

If there is no n_X such that $(x, n_X) \in \mathsf{S}$ then f_ℓ^{SA} is the identity.

If there is n_X such that $(x, n_X) \in \mathsf{S}$ but that there is no n_U such that $(n_X, sel, n_U) \in \mathsf{H}$ then f_ℓ^{SA} is the identity.

If there are abstract locations n_X and n_U such that $(x, n_X) \in \mathsf{S}$ and $(n_X, sel, n_U) \in \mathsf{H}$ then

$$\phi_\ell^{\mathsf{SA}}((\mathsf{S}, \mathsf{H}, \mathsf{is})) = \{kill_{x.sel}((\mathsf{S}, \mathsf{H}, \mathsf{is}))\}$$

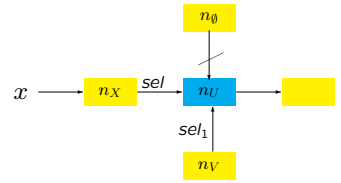
where $kill_{x.sel}((\mathsf{S}, \mathsf{H}, \mathsf{is})) = (\mathsf{S}', \mathsf{H}', \mathsf{is}')$ is given by

$$\mathsf{S}' = \mathsf{S}$$

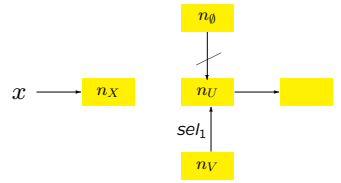
$$\mathsf{H}' = \{(n_V, sel', n_W) \mid (n_V, sel', n_W) \in \mathsf{H} \ \wedge \ \neg(X = V \ \wedge \ sel = sel')\}$$

$$\mathsf{is}' = \begin{cases} \mathsf{is} \backslash \{n_U\} & \text{if } n_U \in \mathsf{is} \ \wedge \ \#into(n_U, \mathsf{H}') \leq 1 \ \wedge \ \neg\exists(n_\emptyset, sel', n_U) \in \mathsf{H}' \\ \mathsf{is} & \text{otherwise} \end{cases}$$

The effect of $[x.sel := \texttt{nil}]^\ell$ when $\#into(n_U, \mathsf{H}') \leq 1$



$(\mathsf{S}, \mathsf{H}, \mathsf{is})$



$(\mathsf{S}', \mathsf{H}', \mathsf{is}')$

Transfer function for $[\texttt{malloc}\ x]^\ell$

$$\phi_\ell^{\mathsf{SA}}((\mathsf{S}, \mathsf{H}, \mathsf{is})) = \{(\mathsf{S}' \cup \{(x, n_{\{x\}})\}, \mathsf{H}', \mathsf{is}')\}$$

where $(\mathsf{S}', \mathsf{H}', \mathsf{is}') = kill_x(\mathsf{S}, \mathsf{H}, \mathsf{is})$.

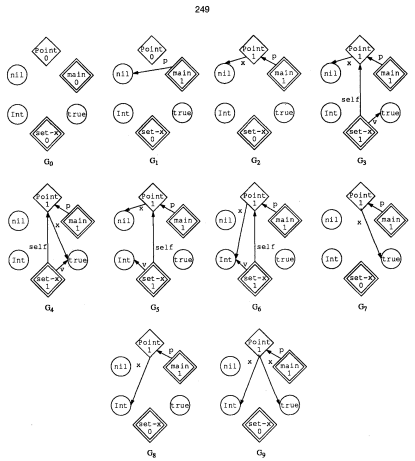
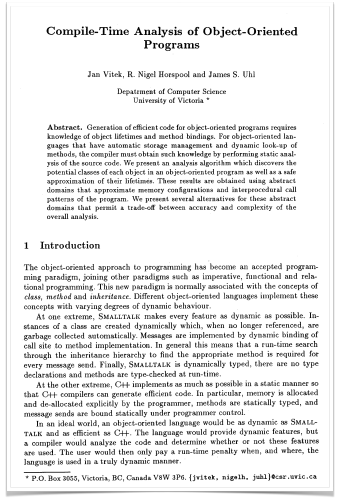


Fig. 6: Abstract Object Graphs. Creation counts are displayed under node names.



Compile-Time Analysis of Object-Oriented Programs

Jan Vitek, R. Nigel Horspool and James S. Uhl

Department of Computer Science
University of Victoria *

Abstract. Generation of efficient code for object-oriented programs requires knowledge of object lifetimes and method bindings. For object-oriented languages that have automatic storage management and dynamic look-up of methods, the compiler must obtain such knowledge by performing static analysis of the source code. We present an analysis algorithm which discovers the potential classes of each object in an object-oriented program as well as a safe approximation of their lifetimes. Those results are obtained using abstract domains that approximate memory configurations and interprocedural call patterns of the program. We present several alternatives for these abstract domains that permit a trade-off between accuracy and complexity of the overall analysis.

1 Introduction

The object-oriented approach to programming has become an accepted programming paradigm, joining other paradigms such as imperative, functional and relational programming. This new paradigm is normally associated with the concepts of *class*, *method* and *inheritance*. Different object-oriented languages implement these concepts with varying degrees of dynamic behaviour.

At one extreme, SMALLTALK makes every feature as dynamic as possible. Instances of a class are created dynamically which, when no longer referenced, are garbage collected automatically. Messages are implemented by dynamic binding of call site to method implementation. In general this means that a run-time search through the inheritance hierarchy to find the appropriate method is required for every message send. Finally, SMALLTALK is dynamically typed, there are no type declarations and methods are type-checked at run-time.

At the other extreme, C++ implements as much as possible in a static manner so that C++ compilers can generate efficient code. In particular, memory is allocated and de-allocated explicitly by the programmer, methods are statically typed, and message sends are bound statically under programmer control.

In an ideal world, an object-oriented language would be as dynamic as SMALLTALK and as efficient as C++. The language would provide dynamic features, but a compiler would analyze the code and determine whether or not these features are used. The user would then only pay a run-time penalty when, and where, the language is used in a truly dynamic manner.

* P.O. Box 3055, Victoria, BC, Canada V8W 3P6. {jvitek, nigelh, jshl}@csr.uvic.ca

Scaling Program Analysis

CS7575

- Lecture 4— Constraint Based Analysis - PoPA Chapter 3

The Dynamic Dispatch Problem

$\vdots$
$[\texttt{call p(p1,1,v)}]^{\ell^1_c}_{\ell^1_r}$
$\vdots$
$[\texttt{call p(p2,2,v)}]^{\ell^2_c}_{\ell^2_r}$
$\vdots$

$\texttt{proc p(procval q, val x, res y) is}^{\ell_n}$
$\vdots$
$[\texttt{call q (x,y)}]^{\ell^p_c}_{\ell^p_r}$
$\vdots$
$\texttt{end}^{\ell_x}$

which procedure is called?

These problems arise for:

- imperative languages with procedures as parameters
- object oriented languages
- functional languages


```
let f = fn x => x 1;
    g = fn y => y+2;
    h = fn z => z+3
in (f g) + (f h)
```

The aim of Control Flow Analysis:

For each function application, which functions may be applied?

Control Flow Analysis computes the interprocedural flow relation used when formulating interprocedural Data Flow Analysis.

Syntax of the Fun Language

Syntactic categories:

e	$\in$	**Exp**	expressions (or labelled terms)
t	$\in$	**Term**	terms (or unlabelled expressions)
f, x	$\in$	**Var**	variables
c	$\in$	**Const**	constants
op	$\in$	**Op**	binary operators
ℓ	$\in$	**Lab**	labels

Syntax:

$$e ::= t^\ell$$

$$t ::= c \mid x \mid \texttt{fn}\ x\ \texttt{=>}\ e_0 \mid \texttt{fun}\ f\ x\ \texttt{=>}\ e_0 \mid e_1\ e_2$$
$$\mid \texttt{if}\ e_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \texttt{let}\ x\ \texttt{=}\ e_1\ \texttt{in}\ e_2 \mid e_1\ op\ e_2$$

(Labels correspond to program points or nodes in the parse tree.)

```
((fn x => x1)2 (fn y => y3)4)5
```

```
(let f = (fn x => (x1 12)3)4;
 in (let g = (fn y => y5)6;
     in (let h = (fn z => z7)8
         in ((f9 g10)11 + (f12 h13)14)15)16)17)18
```

```
(let g = (fun f x => (f1 (fn y => y2)3)4)5
 in (g6 (fn z => z7)8)9)10
```


Abstract 0-CFA Analysis

- Abstract domains
- Specification of the analysis
- Well-definedness of the analysis

Towards defining the Abstract Domains

The *result* of a 0-CFA analysis is a pair $(\widehat{C}, \widehat{\rho})$:

- $\widehat{C}$ is the *abstract cache* associating abstract values with each labelled program point
- $\widehat{\rho}$ is the *abstract environment* associating abstract values with each variable

((fn x => x1)2 (fn y => y3)4)5

Three guesses of a 0-CFA analysis result:

	$(\widehat{C}_e, \widehat{\rho}_e)$	$(\widehat{C}'_e, \widehat{\rho}'_e)$	$(\widehat{C}''_e, \widehat{\rho}''_e)$
1	{fn y => y3}	{fn y => y3}	{fn x => x1, fn y => y3}
2	{fn x => x1}	{fn x => x1}	{fn x => x1, fn y => y3}
3	$\emptyset$	$\emptyset$	{fn x => x1, fn y => y3}
4	{fn y => y3}	{fn y => y3}	{fn x => x1, fn y => y3}
5	{fn y => y3}	{fn y => y3}	{fn x => x1, fn y => y3}
x	{fn y => y3}	$\emptyset$	{fn x => x1, fn y => y3}
y	$\emptyset$	$\emptyset$	{fn x => x1, fn y => y3}

```
(let g = (fun f x => (f^1 (fn y => y^2)^3)^4)^5
 in (g^6 (fn z => z^7)^8)^9)^10
```

Abbreviations:

$$\begin{array}{rcl} \texttt{f} & = & \texttt{fun f x => (f}^1\ \texttt{(fn y => y}^2\texttt{)}^3\texttt{)}^4 \\ \mathsf{id}_y & = & \texttt{fn y => y}^2 \\ \mathsf{id}_z & = & \texttt{fn z => z}^7 \end{array}$$

One guess of a 0-CFA analysis result:

$$\begin{array}{lcllcllcl} \widehat{\mathsf{C}}_{\mathsf{lp}}(1) & = & \{\mathsf{f}\} & \widehat{\mathsf{C}}_{\mathsf{lp}}(6) & = & \{\mathsf{f}\} & \widehat{\rho}_{\mathsf{lp}}(\mathsf{f}) & = & \{\mathsf{f}\} \\ \widehat{\mathsf{C}}_{\mathsf{lp}}(2) & = & \emptyset & \widehat{\mathsf{C}}_{\mathsf{lp}}(7) & = & \emptyset & \widehat{\rho}_{\mathsf{lp}}(\mathsf{g}) & = & \{\mathsf{f}\} \\ \widehat{\mathsf{C}}_{\mathsf{lp}}(3) & = & \{\mathsf{id}_y\} & \widehat{\mathsf{C}}_{\mathsf{lp}}(8) & = & \{\mathsf{id}_z\} & \widehat{\rho}_{\mathsf{lp}}(\mathsf{x}) & = & \{\mathsf{id}_y, \mathsf{id}_z\} \\ \widehat{\mathsf{C}}_{\mathsf{lp}}(4) & = & \emptyset & \widehat{\mathsf{C}}_{\mathsf{lp}}(9) & = & \emptyset & \widehat{\rho}_{\mathsf{lp}}(\mathsf{y}) & = & \emptyset \\ \widehat{\mathsf{C}}_{\mathsf{lp}}(5) & = & \{\mathsf{f}\} & \widehat{\mathsf{C}}_{\mathsf{lp}}(10) & = & \emptyset & \widehat{\rho}_{\mathsf{lp}}(\mathsf{z}) & = & \emptyset \end{array}$$

Abstract Domains

Formally:

$$\begin{array}{rcll} \widehat{v} & \in & \widehat{\mathbf{Val}} = \mathcal{P}(\mathbf{Term}) & \textit{abstract values} \\ \widehat{\rho} & \in & \widehat{\mathbf{Env}} = \mathbf{Var} \to \widehat{\mathbf{Val}} & \textit{abstract environments} \\ \widehat{\mathsf{C}} & \in & \widehat{\mathbf{Cache}} = \mathbf{Lab} \to \widehat{\mathbf{Val}} & \textit{abstract caches} \end{array}$$

An abstract value $\widehat{v}$ is a set of terms of the forms

- `fn` x `=>` e_0
- `fun` f x `=>` e_0

Specification of the 0-CFA

When is a proposed guess $(\widehat{\mathsf{C}}, \widehat{\rho})$ of an analysis results an *acceptable 0-CFA analysis* for the program?

Different approaches:

- abstract specification
- syntax-directed and constraint-based specifications
- algorithms for computing the *best* result

Specification of the Abstract 0-CFA

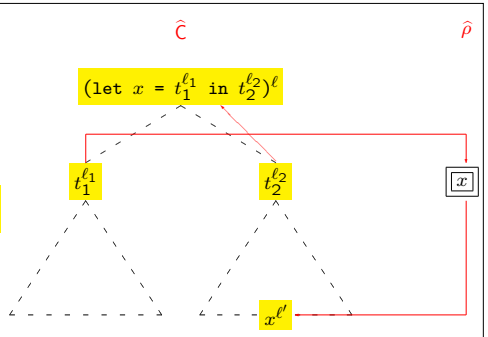
$(\widehat{\mathsf{C}}, \widehat{\rho}) \models e$ means that $(\widehat{\mathsf{C}}, \widehat{\rho})$ is an *acceptable Control Flow Analysis* of the expression e

The relation $\models$ has functionality:

$$\models : (\widehat{\mathbf{Cache}} \times \widehat{\mathbf{Env}} \times \mathbf{Exp}) \to \{\mathit{true}, \mathit{false}\}$$

$(\hat{C}, \hat{\rho}) \models c^\ell$ always

$(\hat{C}, \hat{\rho}) \models x^\ell$ iff $\hat{\rho}(x) \subseteq \hat{C}(\ell)$



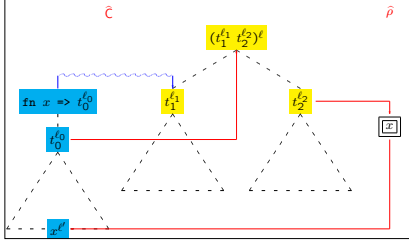
$(\hat{C}, \hat{\rho}) \models (\texttt{let}\ x = t_1^{\ell_1}\ \texttt{in}\ t_2^{\ell_2})^\ell$
iff $(\hat{C}, \hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C}, \hat{\rho}) \models t_2^{\ell_2} \wedge$
$\hat{C}(\ell_1) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell)$

$(\hat{C}, \hat{\rho}) \models (\texttt{if}\ t_0^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell$
iff $(\hat{C}, \hat{\rho}) \models t_0^{\ell_0} \wedge$
$(\hat{C}, \hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C}, \hat{\rho}) \models t_2^{\ell_2} \wedge$
$\hat{C}(\ell_1) \subseteq \hat{C}(\ell) \wedge \hat{C}(\ell_2) \subseteq \hat{C}(\ell)$

$(\hat{C}, \hat{\rho}) \models (t_1^{\ell_1}\ op\ t_2^{\ell_2})^\ell$
iff $(\hat{C}, \hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C}, \hat{\rho}) \models t_2^{\ell_2}$

$(\hat{C}, \hat{\rho}) \models (\texttt{fn}\ x \Rightarrow t_0^{\ell_0})^\ell$ iff $\{\texttt{fn}\ x \Rightarrow t_0^{\ell_0}\} \subseteq \hat{C}(\ell)$

$(\hat{C}, \hat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^\ell$
iff $(\hat{C}, \hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C}, \hat{\rho}) \models t_2^{\ell_2} \wedge$
$(\forall(\texttt{fn}\ x \Rightarrow t_0^{\ell_0}) \in \hat{C}(\ell_1) : (\hat{C}, \hat{\rho}) \models t_0^{\ell_0} \wedge$
$\hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell))$



$(\hat{C}, \hat{\rho}) \models (\texttt{fun}\ f\ x \Rightarrow e_0)^\ell$ iff $\{\texttt{fun}\ f\ x \Rightarrow e_0\} \subseteq \hat{C}(\ell)$

$(\hat{C}, \hat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^\ell$
iff $(\hat{C}, \hat{\rho}) \models t_1^{\ell_1} \wedge (\hat{C}, \hat{\rho}) \models t_2^{\ell_2} \wedge$
$(\forall(\texttt{fn}\ x \Rightarrow t_0^{\ell_0}) \in \hat{C}(\ell_1) : (\hat{C}, \hat{\rho}) \models t_0^{\ell_0} \wedge$
$\hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell)) \wedge$
$(\forall(\texttt{fun}\ f\ x \Rightarrow t_0^{\ell_0}) \in \hat{C}(\ell_1) : (\hat{C}, \hat{\rho}) \models t_0^{\ell_0} \wedge$
$\hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell) \wedge$
$\{\texttt{fun}\ f\ x \Rightarrow t_0^{\ell_0}\} \subseteq \hat{\rho}(f))$

Two guesses for $((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5$

	$(\widehat{C}_e, \widehat{\rho}_e)$	$(\widehat{C}'_e, \widehat{\rho}'_e)$
1	$\{\texttt{fn y => y}^3\}$	$\{\texttt{fn y => y}^3\}$
2	$\{\texttt{fn x => x}^1\}$	$\{\texttt{fn x => x}^1\}$
3	$\emptyset$	$\emptyset$
4	$\{\texttt{fn y => y}^3\}$	$\{\texttt{fn y => y}^3\}$
5	$\{\texttt{fn y => y}^3\}$	$\{\texttt{fn y => y}^3\}$
x	$\{\texttt{fn y => y}^3\}$	$\emptyset$
y	$\emptyset$	$\emptyset$

Checking the guesses:

$$(\widehat{C}_e, \widehat{\rho}_e) \models ((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5$$

$$(\widehat{C}'_e, \widehat{\rho}'_e) \not\models ((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5$$

Well-definedness of the Abstract 0-CFA

Difficulty: The clause for function application is *not* of a form that allows us to define $(\widehat{C}, \widehat{\rho}) \models e$ by Structural Induction in the expression e

$$(\widehat{C}, \widehat{\rho}) \models (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell}$$

iff $\quad (\widehat{C}, \widehat{\rho}) \models t_1^{\ell_1} \ \wedge\ (\widehat{C}, \widehat{\rho}) \models t_2^{\ell_2} \ \wedge$

$$(\forall(\texttt{fn}\ x \texttt{ => } t_0^{\ell_0}) \in \widehat{C}(\ell_1) : \quad (\widehat{C}, \widehat{\rho}) \models t_0^{\ell_0} \ \wedge$$

$$\widehat{C}(\ell_2) \subseteq \widehat{\rho}(x) \ \wedge\ \widehat{C}(\ell_0) \subseteq \widehat{C}(\ell))$$

Solution: The relation $\models$ is defined by coinduction, that is, as the greatest fixed point of a functional.

Theoretical Properties:

- structural operational semantics
- semantic correctness
- the existence of least solutions

Choice of Semantics

- operational or denotational semantics?
  - an operational semantics more easily models intensional properties
- small-step or big-step operational semantics?
  - a small-step semantics allows us to reason about looping programs
- operational semantics based on environments or substitutions?
  - an environment based semantics preserves the identity of functions

Configurations and Transitions

Semantic categories:

$v \in \mathbf{Val}$ *values*

$\rho \in \mathbf{Env}$ *environments*

defined by:

$v ::= c \mid \texttt{close}\ t\ \texttt{in}\ \rho$ *closures*

$\rho ::= [\,] \mid \rho[x \mapsto v]$

Transitions have the form

$$\rho \vdash e_1 \to e_2$$

meaning that *one step* of computation of the expression e_1 in the environment ρ will transform it into e_2.

Transitions

$\rho \vdash x^\ell \to v^\ell$ if $x \in dom(\rho)$ and $v = \rho(x)$

$\rho \vdash (\texttt{fn}\ x\ \texttt{=>}\ e_0)^\ell \to (\texttt{close}\ (\texttt{fn}\ x\ \texttt{=>}\ e_0)\ \texttt{in}\ \rho_0)^\ell$
where $\rho_0 = \rho \mid FV(\texttt{fn}\ x\ \texttt{=>}\ e_0)$

$\rho \vdash (\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)^\ell \to (\texttt{close}\ (\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)\ \texttt{in}\ \rho_0)^\ell$
where $\rho_0 = \rho \mid FV(\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)$

static scope!

Intermediate Expressions and Terms

$ie \in \mathbf{IExp}$ *intermediate expressions*

$it \in \mathbf{ITerm}$ *intermediate terms*

extending the syntax:

$ie ::= it^\ell$

$it ::= c \mid x \mid \texttt{fn}\ x\ \texttt{=>}\ e_0 \mid \texttt{fun}\ f\ x\ \texttt{=>}\ e_0 \mid ie_1\ ie_2$
$\mid \texttt{if}\ ie_0\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \texttt{let}\ x\ \texttt{=}\ ie_1\ \texttt{in}\ e_2 \mid ie_1\ op\ ie_2$
$\mid \texttt{close}\ t\ \texttt{in}\ \rho \mid \texttt{bind}\ \rho\ \texttt{in}\ ie$

The correct form of transitions

$$\rho \vdash ie_1 \to ie_2$$

Transitions

$$\frac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (ie_1\ ie_2)^\ell \to (ie_1'\ ie_2)^\ell} \qquad \frac{\rho \vdash ie_2 \to ie_2'}{\rho \vdash (v_1^{\ell_1}\ ie_2)^\ell \to (v_1^{\ell_1}\ ie_2')^\ell}$$

$\rho \vdash ((\texttt{close}\ (\texttt{fn}\ x\ \texttt{=>}\ e_1)\ \texttt{in}\ \rho_1)^{\ell_1}\ v_2^{\ell_2})^\ell \to (\texttt{bind}\ \rho_1[x \mapsto v_2]\ \texttt{in}\ e_1)^\ell$

$\rho \vdash ((\texttt{close}\ (\texttt{fun}\ f\ x\ \texttt{=>}\ e_1)\ \texttt{in}\ \rho_1)^{\ell_1}\ v_2^{\ell_2})^\ell \to (\texttt{bind}\ \rho_2[x \mapsto v_2]\ \texttt{in}\ e_1)^\ell$
where $\rho_2 = \rho_1[f \mapsto \texttt{close}\ (\texttt{fun}\ f\ x\ \texttt{=>}\ e_1)\ \texttt{in}\ \rho_1]$

$$\frac{\rho_1 \vdash ie_1 \to ie_1'}{\rho \vdash (\texttt{bind}\ \rho_1\ \texttt{in}\ ie_1)^\ell \to (\texttt{bind}\ \rho_1\ \texttt{in}\ ie_1')^\ell}$$

$\rho \vdash (\texttt{bind}\ \rho_1\ \texttt{in}\ v_1^{\ell_1})^\ell \to v_1{}^\ell$ the outermost label remains the same

$[\,] \vdash ((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5$

$\rightarrow \quad ((\texttt{close (fn x => x}^1\texttt{) in [ ]})^2\ (\texttt{fn y => y}^3)^4)^5$

$\rightarrow \quad ((\texttt{close (fn x => x}^1\texttt{) in [ ]})^2\ (\texttt{close (fn y => y}^3\texttt{) in [ ]})^4)^5$

$\rightarrow \quad (\texttt{bind [x} \mapsto \texttt{(close (fn y => y}^3\texttt{) in [ ])] in x}^1)^5$

$\rightarrow \quad (\texttt{bind [x} \mapsto \texttt{(close (fn y => y}^3\texttt{) in [ ])] in}$
$\qquad\qquad \texttt{(close (fn y => y}^3\texttt{) in [ ])}^1)^5$

$\rightarrow \quad \texttt{(close (fn y => y}^3\texttt{) in [ ])}^5$

Transitions

$$\frac{\rho \vdash ie_0 \rightarrow ie_0'}{\rho \vdash (\texttt{if } ie_0 \texttt{ then } e_1 \texttt{ else } e_2)^\ell \rightarrow (\texttt{if } ie_0' \texttt{ then } e_1 \texttt{ else } e_2)^\ell}$$

$$\rho \vdash (\texttt{if true}^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^\ell \rightarrow t_1^\ell$$

$$\rho \vdash (\texttt{if false}^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^\ell \rightarrow t_2^\ell$$

$$\frac{\rho \vdash ie_1 \rightarrow ie_1'}{\rho \vdash (\texttt{let } x \texttt{ = } ie_1 \texttt{ in } e_2)^\ell \rightarrow (\texttt{let } x \texttt{ = } ie_1' \texttt{ in } e_2)^\ell}$$

$$\rho \vdash (\texttt{let } x \texttt{ = } v^{\ell_1} \texttt{ in } e_2)^\ell \rightarrow (\texttt{bind } [x \mapsto v] \texttt{ in } e_2)^\ell$$

$$\frac{\rho \vdash ie_1 \rightarrow ie_1'}{\rho \vdash (ie_1\ \mathit{op}\ ie_2)^\ell \rightarrow (ie_1'\ \mathit{op}\ ie_2)^\ell} \qquad \frac{\rho \vdash ie_2 \rightarrow ie_2'}{\rho \vdash (v_1^{\ell_1}\ \mathit{op}\ ie_2)^\ell \rightarrow (v_1^{\ell_1}\ \mathit{op}\ ie_2')^\ell}$$

$$\rho \vdash (v_1^{\ell_1}\ \mathit{op}\ v_2^{\ell_2})^\ell \rightarrow v^\ell \quad \text{if } v = v_1\ \mathbf{op}\ v_2$$

$[\,] \vdash \texttt{(let g = (fun f x => (f}^1\ \texttt{(fn y => y}^2\texttt{)}^3\texttt{)}^4\texttt{)}^5$
$\quad \texttt{in (g}^6\ \texttt{(fn z => z}^7\texttt{)}^8\texttt{)}^9\texttt{)}^{10}$
$\rightarrow \quad \texttt{(let g =}\ \mathbf{f}^5\ \texttt{in (g}^6\ \texttt{(fn z => z}^7\texttt{)}^8\texttt{)}^9\texttt{)}^{10}$
$\rightarrow \quad \texttt{(bind [g} \mapsto \mathbf{f}\texttt{] in (g}^6\ \texttt{(fn z => z}^7\texttt{)}^8\texttt{)}^9\texttt{)}^{10}$
$\rightarrow \quad \texttt{(bind [g} \mapsto \mathbf{f}\texttt{] in (}\mathbf{f}^6\ \texttt{(fn z => z}^7\texttt{)}^8\texttt{)}^9\texttt{)}^{10}$
$\rightarrow \quad \texttt{(bind [g} \mapsto \mathbf{f}\texttt{] in (}\mathbf{f}^6\ \mathbf{id}_z^8\texttt{)}^9\texttt{)}^{10}$
$\rightarrow \quad \texttt{(bind [g} \mapsto \mathbf{f}\texttt{] in (bind [f} \mapsto \mathbf{f}\texttt{][x} \mapsto \mathbf{id}_z\texttt{] in (f}^1\ \texttt{(fn y => y}^2\texttt{)}^3\texttt{)}^4\texttt{)}^9\texttt{)}^{10}$
$\rightarrow^* \quad \texttt{(bind [g} \mapsto \mathbf{f}\texttt{] in (bind [f} \mapsto \mathbf{f}\texttt{][x} \mapsto \mathbf{id}_z\texttt{] in}$
$\qquad \texttt{(bind [f} \mapsto \mathbf{f}\texttt{][x} \mapsto \mathbf{id}_y\texttt{] in (f}^1\ \texttt{(fn y => y}^2\texttt{)}^3\texttt{)}^4\texttt{)}^4\texttt{)}^9\texttt{)}^{10}$
$\rightarrow^* \quad \ldots$

Abbreviations:

$$\begin{aligned} \mathbf{f} &= \texttt{close (fun f x => (f}^1\ \texttt{(fn y => y}^2\texttt{)}^3\texttt{)}^4\texttt{) in [ ]} \\ \mathbf{id}_y &= \texttt{close (fn y => y}^2\texttt{) in [ ]} \\ \mathbf{id}_z &= \texttt{close (fn z => z}^7\texttt{) in [ ]} \end{aligned}$$

Semantic Correctness

A *subject reduction result:* an acceptable result of the analysis remains acceptable under evaluation

Analysis of intermediate expressions

$(\hat{C}, \hat{\rho}) \models (\texttt{bind } \rho \texttt{ in } it_0^{\ell_0})^\ell$
iff $\quad (\hat{C}, \hat{\rho}) \models it_0^{\ell_0} \ \wedge\ \hat{C}(\ell_0) \subseteq \hat{C}(\ell) \ \wedge\ \rho\ \mathcal{R}\ \hat{\rho}$

$(\hat{C}, \hat{\rho}) \models (\texttt{close } t_0 \texttt{ in } \rho)^\ell$
iff $\quad \{t_0\} \subseteq \hat{C}(\ell) \ \wedge\ \rho\ \mathcal{R}\ \hat{\rho}$

Correctness Relation

The global abstract environment, $\hat{\rho}$ models *all* the local environments of the semantics

Correctness relation

$$\mathcal{R} : (\mathbf{Env} \times \widehat{\mathbf{Env}}) \to \{true, false\}$$

We demand that $\rho\ \mathcal{R}\ \hat{\rho}$ for all local environments, ρ, occurring in the intermediate expressions

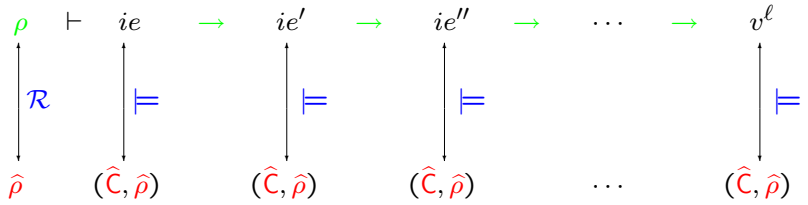
Define

$\rho\ \mathcal{R}\ \hat{\rho}$ iff $\forall x \in dom(\rho) \subseteq dom(\hat{\rho})\ \forall t_x\ \forall \rho_x$:

$(\rho(x) = \texttt{close}\ t_x\ \texttt{in}\ \rho_x) \Rightarrow (t_x \in \hat{\rho}(x) \wedge \rho_x\ \mathcal{R}\ \hat{\rho})$

(Well-defined by induction in the size of ρ.)

Correctness Result



Formal details of Correctness Result

Theorem:

If $\rho\ \mathcal{R}\ \hat{\rho}$ and $\rho \vdash ie \to ie'$ then $(\hat{C}, \hat{\rho}) \models ie$ implies $(\hat{C}, \hat{\rho}) \models ie'$.

Intuitively:

If there is a possible evaluation of the program such that the function at a call point evaluates to some abstraction, then this abstraction has to be in the set of possible abstractions computed by the analysis.

Observe: the theorem expresses that *all* acceptable analysis results remain acceptable under evaluation!

Thus we do *not* rely on the existence of a least or "best" solution.

Semantics:

$[\,] \vdash ((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5 \to^* (\texttt{close (fn y => y}^3\texttt{) in [ ]})^5$

	$(\hat{C}_e, \hat{\rho}_e)$
1	$\{\texttt{fn y => y}^3\}$
2	$\{\texttt{fn x => x}^1\}$
3	$\emptyset$
4	$\{\texttt{fn y => y}^3\}$
5	$\{\texttt{fn y => y}^3\}$
x	$\{\texttt{fn y => y}^3\}$
y	$\emptyset$

Analysis:

$(\hat{C}_e, \hat{\rho}_e) \models ((\texttt{fn x => x}^1)^2\ (\texttt{fn y => y}^3)^4)^5$

Correctness relation:

$[\,]\ \mathcal{R}\ \hat{\rho}_e$

Correctness theorem: $(\hat{C}_e, \hat{\rho}_e) \models (\texttt{close (fn y => y}^3\texttt{) in [ ]})^5$

The History of T

Olin Shivers

This brings us to the summer of 1984. The mission was to build the world's most highly-optimizing Scheme compiler. We wanted to compete with C and Fortran. … I had passed the previous semester at CMU becoming an expert on data-flow analysis, a topic on which I completely grooved. All hot compilers do DFA. It is necessary for all the really cool optimisations, like loop-invariant hoisting, global register allocation, global common subexpression elimination, copy propagation, induction-variable elimination. I knew that no Scheme or Lisp compiler had ever provided these hot optimisations. I was burning to make it happen. … So when we divided up the compiler, I told everyone else to back off and loudly claimed DFA for my own. Fine, everyone said.

Lamping and I spent the rest of the summer failing. Taking trips to the Stanford library to look up papers. Hashing things out on white boards. Staring into space. Writing little bits of experimental code. Failing. Finding out **why** no one had ever provided DFA optimization for Scheme. In short, the fundamental item the classical data-flow analysis algorithms need to operate is not available in a Scheme program. It was really depressing. I was making more money than I'd ever made in my life ($600/wk). … It was **not** a good summer. I slunk back to CMU with my tail between my legs, having contributed not one line of code.

About three years after the summer, I **finally** figured out how to do data-flow analysis for Scheme, which ended a long, pretty unhappy period in my life. I officially switched from being an AI student to being a PL student…

Olin Shivers. *The History of T.* http://www.paulgraham.com/thist.html

4.1 Handling Lambda

Defs can be given with a recursive definition: In a call $c : (f \ldots a_i \ldots)$, $\forall l \in \text{Defs}(f)$ of the form `(lambda (`$\ldots v_i \ldots$`) ...)`,

$$\text{Defs}(a_i) \subset \text{Defs}(v_i) \qquad \text{[LAM]}$$

I.e. if lambda l can be called from call site c, then l's ith variable can evaluate to any of the lambdas that c's ith argument can evaluate to.

Olin Shivers. *Control Flow Analysis in Scheme.* PLDI 1988

Control Flow Analysis in Scheme

Olin Shivers
Carnegie Mellon University
shivers@cs.cmu.edu

Abstract

Traditional flow analysis techniques, such as the ones typically employed by optimising Fortran compilers, do not work for Scheme-like languages. This paper presents a flow analysis technique — *control flow analysis* — which is applicable to Scheme-like languages. As a demonstration application, the information gathered by control flow analysis is used to perform a traditional flow analysis problem, induction variable elimination. Extensions and limitations are discussed.

The techniques presented in this paper are backed up by working code. They are applicable not only to Scheme, but also to related languages, such as Common Lisp and ML.

1 The Task

Flow analysis is a traditional optimising compiler technique for determining useful information about a program at compile time. Flow analysis determines path invariant facts about points in a program. A flow analysis problem is a question of the form:

> "What is true at a given point p in my program, independent of the execution path taken to p from the start of the program?"

Example domains of interest might be the following:

- Range analysis: What is the range of values that a given reference to an integer variable is constrained to lie within? Range analysis can be used, for instance, to do array bounds checking at compile time.
- Loop invariant detection: Do all possible prior assignments to a given variable reference lie outside its containing loop?

Over the last thirty years, standard techniques have been developed to answer these questions for the standard imperative, Algol-like languages (*e.g.*, Pascal, C, Ada, Bliss, and chiefly Fortran). Representative texts describing these techniques are [Dragon], and in more detail, [Hecht]. Flow analysis is perhaps the chief tool in the optimising compiler writer's bag of tricks; an incomplete list of the problems that can be addressed with flow analysis includes global constant subexpression elimination, loop invariant detection, redundant assignment detection, dead code elimination, constant propagation, range analysis, code hoisting, induction variable elimination, copy propagation, live variable analysis, loop unrolling, and loop jamming.

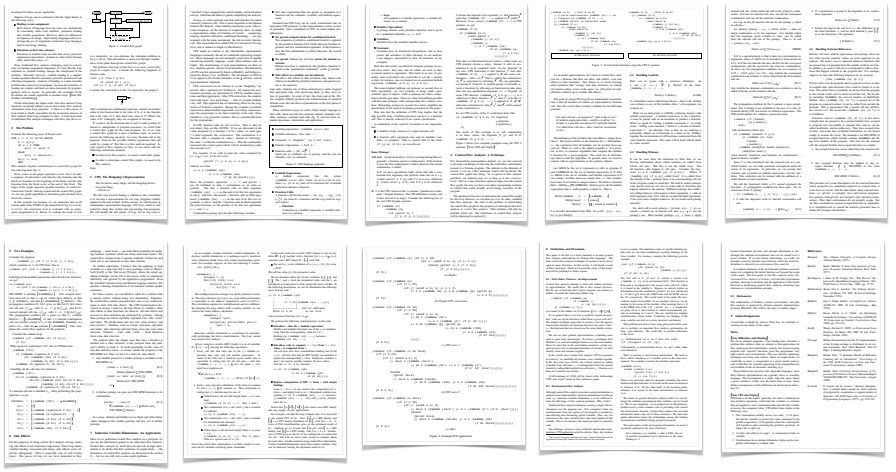
However, these traditional flow analysis techniques have never successfully been applied to the Lisp family of computer languages. This is a curious omission. The Lisp community has had sufficient time to consider the problem. Flow analysis dates back at least to 1960. ([Dragon], pp. 516), and Lisp is one of the oldest computer programming languages currently in use, rivalled only by Fortran and COBOL.

Indeed, the Lisp community has long been concerned with the execution speed of their programs. Typical Lisp programs, such as large AI systems, are both interactive and cycle-intensive. AI researchers often find their research efforts frustrated by the necessity of waiting several hours for their enormous Lisp-based production system or back-propagation network to produce a single run. Since Lisp users are willing to pay premium prices for special computer architectures specially designed to execute Lisp rapidly, we can safely assume they are even more willing to consider compiler techniques that apply generally to target machines of any nature.

Finally, the problems addressed by flow analysis are relevant to Lisp. None of the flow analysis problems listed above are restricted to arithmetic computation; they apply just as well to symbolic processing. Furthermore, Lisp opens up new domains of interest. For example, Lisp permits weak typing and run time type checking. Type information is not statically apparent, as it is in the Algol family of languages. Type information is nonetheless extremely important to the efficient execution of Lisp programs, both to remove run time safety checks of function arguments, and to span code functions that operate on a variety of argument types. Thus flow analysis offers a tempting opportunity to perform type inference from the occurence of calls to type predicates in Lisp programs.

So it is clear that flow analysis has much potential with respect to compiling Lisp programs. Unfortunately, this potential has not been realised because Lisp is sufficiently different from the Algol family of languages that the traditional techniques

To appear at the ACM SIGPLAN '88 Conference on Programming Language Design and Implementation, Atlanta, Georgia, June 22-24, 1988.



- Peter Lee, my thesis advisor and good friend, taught me how to do research, taught me how to write, helped me publish papers, and kept me up-to-date on the latest developments in Formula-1 Grand Prix. I suppose that I recursively owe a debt to Uwe Pleban for teaching Peter all the things he taught me.
- Allen Newell was my research advisor for my first four years at CMU, and co-advised me with Peter after I picked up my thesis topic. He taught me about having "a bright blue flame for science." Allen sheltered me during the period when I was drifting in research limbo; I'm reasonably certain that had I chosen a different advisor, I'd have become an ex-Ph.D.-candidate pretty rapidly. Allen believed in me -- certainly more than I did.
- John Reynolds taught (drilled/hammered/beat into) me the importance of formal rigour in my work. Doing mathematics with John is an exhilarating experience, if you have lots of stamina and don't mind chalk dust or feelings of inadequacy. John's emphasis on formal methods rescued my dissertation from being just another pile of hacks.
- Jeannette Wing was my contact with the programming-systems world while I was officially an AI student. She supported my control-flow analysis research in its early stages. She also has an eagle eye for poor writing and a deep appreciation for edged weapons.
- Paul Hudak's work influenced me a great deal. His papers on abstract semantic interpretations caused me to consider applying the whole approach to my problem.

Control-Flow Analysis of Higher-Order Languages
or
Taming Lambda

Olin Shivers

May 1991
CMU-CS-91-145

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

$$\begin{array}{rcl@{\qquad}rcl}
\widehat{\text{Proc}} & = & \text{LAM} + \text{PRIM} + \{stop\} & \widehat{\mathcal{PR}} & : & \text{PR} \to \widehat{\text{CCache}} \\
\widehat{\text{D}} & = & \mathcal{P}(\widehat{\text{Proc}}) & \widehat{\mathcal{A}} & : & \text{ARG} \cup \text{FUN} \to \widehat{\text{VEnv}} \to \widehat{\text{D}} \\
\widehat{\text{CCache}} & = & \text{LAB} \to \widehat{\text{D}} & \widehat{\mathcal{C}} & : & \text{CALL} \to \widehat{\text{VEnv}} \to \widehat{\text{CCache}} \\
\widehat{\text{VEnv}} & = & \text{VAR} \to \widehat{\text{D}} & \widehat{\mathcal{F}} & : & \widehat{\text{Proc}} \to \widehat{\text{D}}^* \to \widehat{\text{VEnv}} \to \widehat{\text{CCache}}
\end{array}$$

Figure 3.6: 0CFA Domains and Functionalities

$$\widehat{\mathcal{PR}}\,\ell = \left(\widehat{\mathcal{F}}\; f\; \langle\{stop\}\rangle\; [\,]\right) \sqcup [c_{\text{top}} \mapsto \{f\}] \quad \text{where } \{f\} = \widehat{\mathcal{A}}\,\ell\,[\,]$$

$$\widehat{\mathcal{A}}\,[\![k]\!]\,\widehat{ve} = \emptyset$$
$$\widehat{\mathcal{A}}\,[\![prim]\!]\,\widehat{ve} = \{prim\}$$
$$\widehat{\mathcal{A}}\,[\![\ell]\!]\,\widehat{ve} = \{\ell\}$$
$$\widehat{\mathcal{A}}\,[\![v]\!]\,\widehat{ve} = \widehat{ve}\,v$$

$$\widehat{\mathcal{C}}\,[\![c{:}(f\ a_1 \ldots a_n)]\!]\,\widehat{ve} = \left(\bigsqcup_{f' \in F} \widehat{\mathcal{F}}\; f'\,\widehat{a}\widehat{w}\,\widehat{ve}\right) \sqcup [c \mapsto F] \quad \text{where } F = \widehat{\mathcal{A}}\,f\,\widehat{ve},\ \widehat{a}\widehat{w}\downarrow i = \widehat{\mathcal{A}}\,a_i\,\widehat{ve}$$

$$\widehat{\mathcal{C}}\,[\![c{:}(\texttt{letrec}\ ((f_1\ l_1)\ldots)\ c)]\!]\,\widehat{ve} = \widehat{\mathcal{C}}\,c\,\widehat{ve}' \quad \text{where } \widehat{ve}' = \widehat{ve} \sqcup \left[f_i \mapsto \widehat{\mathcal{A}}\,l_i\,\widehat{ve}\right]$$

$$\widehat{\mathcal{F}}\,[\![\ell{:}(\lambda\ (v_1 \ldots v_n)\ c)]\!]\,\widehat{a}\widehat{v}\,\widehat{ve} = \text{length}\,\widehat{a}\widehat{v} \neq n \longrightarrow [\,] \quad \text{otherwise } \widehat{\mathcal{C}}\,c\,(\widehat{ve} \sqcup [v_i \mapsto \widehat{a}\widehat{v}\downarrow i])$$

$$\widehat{\mathcal{F}}\,[\![p{+}]\!]\,\left\langle \hat{x}, \hat{y}, \hat{k} \right\rangle\,\widehat{ve} = \left(\bigsqcup_{f \in \hat{k}} \widehat{\mathcal{F}}\; f\; \langle\emptyset\rangle\; \widehat{ve}\right) \sqcup \left[ic_{p,+} \mapsto \hat{k}\right]$$

$$\widehat{\mathcal{F}}\,[\![p\texttt{if}]\!]\,\left\langle \hat{x}, \hat{k}_0, \hat{k}_1 \right\rangle\,\widehat{ve} = \left(\bigsqcup_{f \in \hat{k}_0} \widehat{\mathcal{F}}\; f\; \langle\rangle\; \widehat{ve}\right) \sqcup \left(\bigsqcup_{f \in \hat{k}_1} \widehat{\mathcal{F}}\; f\; \langle\rangle\; \widehat{ve}\right) \sqcup \left[ic^0_{p\texttt{if}} \mapsto \hat{k}_0,\ ic^1_{p\texttt{if}} \mapsto \hat{k}_1\right]$$

Syntax Directed 0-CFA Analysis

Reformulate the abstract specification:

(i) Syntax directed specification

(ii) Constructing a finite set of constraints

(iii) Compute the least solution of the set of constraints

Common Phenomenon

A specification $\models_A$ is reformulated into a specification $\models_B$ ensuring that

$$(\hat{C}, \hat{\rho}) \models_A e_\star \Leftarrow (\hat{C}, \hat{\rho}) \models_B e_\star$$

so that "$\models_B$" is a *safe approximation* to "$\models_A$" and hence the best (i.e. least) solution to "$\models_B e_\star$" will also be a solution to "$\models_A e_\star$".

If additionally

$$(\hat{C}, \hat{\rho}) \models_A e_\star \Rightarrow (\hat{C}, \hat{\rho}) \models_B e_\star$$

then we can be assured that *no solutions are lost* and hence the best (i.e. least) solution to "$\models_B e_\star$" will also be the best (i.e. least) solution to "$\models_A e_\star$".

Syntax Directed Specification (1)

$(\hat{\mathsf{C}}, \hat{\rho}) \models_s (\texttt{fn}\ x\ \texttt{=>}\ e_0)^\ell$
iff $\{\texttt{fn}\ x\ \texttt{=>}\ e_0\} \subseteq \hat{\mathsf{C}}(\ell) \wedge$
$(\hat{\mathsf{C}}, \hat{\rho}) \models_s e_0$

$(\hat{\mathsf{C}}, \hat{\rho}) \models_s (\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)^\ell$
iff $\{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq \hat{\mathsf{C}}(\ell) \wedge$
$(\hat{\mathsf{C}}, \hat{\rho}) \models_s e_0 \wedge \{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq \hat{\rho}(f)$

$(\hat{\mathsf{C}}, \hat{\rho}) \models_s (t_1^{\ell_1}\ t_2^{\ell_2})^\ell$
iff $(\hat{\mathsf{C}}, \hat{\rho}) \models_s t_1^{\ell_1} \wedge (\hat{\mathsf{C}}, \hat{\rho}) \models_s t_2^{\ell_2} \wedge$
$(\forall(\texttt{fn}\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \hat{\mathsf{C}}(\ell_1) :$
$\hat{\mathsf{C}}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{\mathsf{C}}(\ell_0) \subseteq \hat{\mathsf{C}}(\ell) \quad \wedge$
$(\forall(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \hat{\mathsf{C}}(\ell_1) :$
$\hat{\mathsf{C}}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{\mathsf{C}}(\ell_0) \subseteq \hat{\mathsf{C}}(\ell)$

$(\hat{\mathsf{C}}, \hat{\rho}) \models_s c^\ell$ always

$(\hat{\mathsf{C}}, \hat{\rho}) \models_s x^\ell$ iff $\hat{\rho}(x) \subseteq \hat{\mathsf{C}}(\ell)$

$(\hat{\mathsf{C}}, \hat{\rho}) \models_s (\texttt{if}\ t_0^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell$
iff $(\hat{\mathsf{C}}, \hat{\rho}) \models_s t_0^{\ell_0} \wedge$
$(\hat{\mathsf{C}}, \hat{\rho}) \models_s t_1^{\ell_1} \wedge (\hat{\mathsf{C}}, \hat{\rho}) \models_s t_2^{\ell_2} \wedge$
$\hat{\mathsf{C}}(\ell_1) \subseteq \hat{\mathsf{C}}(\ell) \wedge \hat{\mathsf{C}}(\ell_2) \subseteq \hat{\mathsf{C}}(\ell)$

$(\hat{\mathsf{C}}, \hat{\rho}) \models_s (\texttt{let}\ x\ \texttt{=}\ t_1^{\ell_1}\ \texttt{in}\ t_2^{\ell_2})^\ell$
iff $(\hat{\mathsf{C}}, \hat{\rho}) \models_s t_1^{\ell_1} \wedge (\hat{\mathsf{C}}, \hat{\rho}) \models_s t_2^{\ell_2} \wedge$
$\hat{\mathsf{C}}(\ell_1) \subseteq \hat{\rho}(x) \wedge \hat{\mathsf{C}}(\ell_2) \subseteq \hat{\mathsf{C}}(\ell)$

$(\hat{\mathsf{C}}, \hat{\rho}) \models_s (t_1^{\ell_1}\ op\ t_2^{\ell_2})^\ell$ iff $(\hat{\mathsf{C}}, \hat{\rho}) \models_s t_1^{\ell_1} \wedge (\hat{\mathsf{C}}, \hat{\rho}) \models_s t_2^{\ell_2}$

Constraint Based 0-CFA Analysis

$\mathcal{C}_\star[\![e_\star]\!]$ is a set of constraints of the form

$$lhs \subseteq rhs$$

$$\{t\} \subseteq rhs' \Rightarrow lhs \subseteq rhs$$

where

$$rhs ::= \mathsf{C}(\ell) \mid \mathsf{r}(x)$$

$$lhs ::= \mathsf{C}(\ell) \mid \mathsf{r}(x) \mid \{t\}$$

and all occurrences of t are of the form $\texttt{fn}\ x\ \texttt{=>}\ e_0$ or $\texttt{fun}\ f\ x\ \texttt{=>}\ e_0$

Constraint Based Control Flow Analysis (1)

$\mathcal{C}_\star[\![(\texttt{fn}\ x\ \texttt{=>}\ e_0)^\ell]\!] = \{\{\texttt{fn}\ x\ \texttt{=>}\ e_0\} \subseteq \mathsf{C}(\ell)\} \cup \mathcal{C}_\star[\![e_0]\!]$

$\mathcal{C}_\star[\![(\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)^\ell]\!] = \{\{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq \mathsf{C}(\ell)\} \cup \mathcal{C}_\star[\![e_0]\!]$
$\cup \{\{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq \mathsf{r}(f)\}$

$\mathcal{C}_\star[\![(t_1^{\ell_1}\ t_2^{\ell_2})^\ell]\!] = \mathcal{C}_\star[\![t_1^{\ell_1}]\!] \cup \mathcal{C}_\star[\![t_2^{\ell_2}]\!]$
$\cup \{\{t\} \subseteq \mathsf{C}(\ell_1) \Rightarrow \mathsf{C}(\ell_2) \subseteq \mathsf{r}(x) \mid t = (\texttt{fn}\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \mathbf{Term}_\star\}$
$\cup \{\{t\} \subseteq \mathsf{C}(\ell_1) \Rightarrow \mathsf{C}(\ell_0) \subseteq \mathsf{C}(\ell) \mid t = (\texttt{fn}\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \mathbf{Term}_\star\}$
$\cup \{\{t\} \subseteq \mathsf{C}(\ell_1) \Rightarrow \mathsf{C}(\ell_2) \subseteq \mathsf{r}(x) \mid t = (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \mathbf{Term}_\star\}$
$\cup \{\{t\} \subseteq \mathsf{C}(\ell_1) \Rightarrow \mathsf{C}(\ell_0) \subseteq \mathsf{C}(\ell) \mid t = (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \mathbf{Term}_\star\}$

(Eager rather than lazy unfolding – easy but costly.)

Constraint Based Control Flow Analysis (2)

$\mathcal{C}_\star[\![c^\ell]\!] = \emptyset$

$\mathcal{C}_\star[\![x^\ell]\!] = \{\, \mathsf{r}(x) \subseteq \mathsf{C}(\ell) \,\}$

$$\begin{aligned}\mathcal{C}_\star[\![(\texttt{if } t_0^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^\ell]\!] &= \mathcal{C}_\star[\![t_0^{\ell_0}]\!] \cup \mathcal{C}_\star[\![t_1^{\ell_1}]\!] \cup \mathcal{C}_\star[\![t_2^{\ell_2}]\!] \\ &\cup \{\, \mathsf{C}(\ell_1) \subseteq \mathsf{C}(\ell) \,\} \\ &\cup \{\, \mathsf{C}(\ell_2) \subseteq \mathsf{C}(\ell) \,\}\end{aligned}$$

$$\begin{aligned}\mathcal{C}_\star[\![(\texttt{let } x \texttt{ = } t_1^{\ell_1} \texttt{ in } t_2^{\ell_2})^\ell]\!] &= \mathcal{C}_\star[\![t_1^{\ell_1}]\!] \cup \mathcal{C}_\star[\![t_2^{\ell_2}]\!] \\ &\cup \{\, \mathsf{C}(\ell_1) \subseteq \mathsf{r}(x) \,\} \cup \{\, \mathsf{C}(\ell_2) \subseteq \mathsf{C}(\ell) \,\}\end{aligned}$$

$$\mathcal{C}_\star[\![(t_1^{\ell_1} \ op \ t_2^{\ell_2})^\ell]\!] = \mathcal{C}_\star[\![t_1^{\ell_1}]\!] \cup \mathcal{C}_\star[\![t_2^{\ell_2}]\!]$$

$$\begin{aligned}&\mathcal{C}_\star[\![(\texttt{(fn x => x}^1\texttt{)}^2\ \texttt{(fn y => y}^3\texttt{)}^4\texttt{)}^5]\!] = \\ &\quad \{\ \{\texttt{fn x => x}^1\} \subseteq \mathsf{C}(2), \\ &\quad\ \mathsf{r}(\texttt{x}) \subseteq \mathsf{C}(1), \\ &\quad\ \{\texttt{fn y => y}^3\} \subseteq \mathsf{C}(4), \\ &\quad\ \mathsf{r}(\texttt{y}) \subseteq \mathsf{C}(3), \\ &\quad\ \{\texttt{fn x => x}^1\} \subseteq \mathsf{C}(2) \Rightarrow \mathsf{C}(4) \subseteq \mathsf{r}(\texttt{x}), \\ &\quad\ \{\texttt{fn x => x}^1\} \subseteq \mathsf{C}(2) \Rightarrow \mathsf{C}(1) \subseteq \mathsf{C}(5), \\ &\quad\ \{\texttt{fn y => y}^3\} \subseteq \mathsf{C}(2) \Rightarrow \mathsf{C}(4) \subseteq \mathsf{r}(\texttt{y}), \\ &\quad\ \{\texttt{fn y => y}^3\} \subseteq \mathsf{C}(2) \Rightarrow \mathsf{C}(3) \subseteq \mathsf{C}(5)\ \}\end{aligned}$$

Preservation of Solutions

Translating syntactic entities to sets of terms:

$$\begin{aligned}(\hat{\mathsf{C}}, \hat{\rho})[\![\mathsf{C}(\ell)]\!] &= \hat{\mathsf{C}}(\ell) \\ (\hat{\mathsf{C}}, \hat{\rho})[\![\mathsf{r}(x)]\!] &= \hat{\rho}(x) \\ (\hat{\mathsf{C}}, \hat{\rho})[\![\{t\}]\!] &= \{t\}\end{aligned}$$

Satisfaction relation for constraints: $(\hat{\mathsf{C}}, \hat{\rho}) \models_c (lhs \subseteq rhs)$

$$\begin{aligned}&(\hat{\mathsf{C}}, \hat{\rho}) \models_c (lhs \subseteq rhs) \\ &\quad \underline{\text{iff}} \quad (\hat{\mathsf{C}}, \hat{\rho})[\![lhs]\!] \subseteq (\hat{\mathsf{C}}, \hat{\rho})[\![rhs]\!]\end{aligned}$$

$$\begin{aligned}&(\hat{\mathsf{C}}, \hat{\rho}) \models_c (\{t\} \subseteq rhs' \Rightarrow lhs \subseteq rhs) \\ &\quad \underline{\text{iff}} \quad (\{t\} \subseteq (\hat{\mathsf{C}}, \hat{\rho})[\![rhs']\!] \wedge (\hat{\mathsf{C}}, \hat{\rho})[\![lhs]\!] \subseteq (\hat{\mathsf{C}}, \hat{\rho})[\![rhs]\!]) \\ &\quad \vee \quad (\{t\} \not\subseteq (\hat{\mathsf{C}}, \hat{\rho})[\![rhs']\!])\end{aligned}$$

Proposition: $(\hat{\mathsf{C}}, \hat{\rho}) \models_s e_\star$ if and only if $(\hat{\mathsf{C}}, \hat{\rho}) \models_c \mathcal{C}_\star[\![e_\star]\!]$.

Solving the Constraints (1)

Input: a set of constraints $\mathcal{C}_\star[\![e_\star]\!]$

Output: the least solution $(\hat{\mathsf{C}}, \hat{\rho})$ to the constraints

Data structures: a graph with one node for each $\mathsf{C}(\ell)$ and $\mathsf{r}(x)$ (where $\ell \in \mathbf{Lab}_\star$ and $x \in \mathbf{Var}_\star$) and zero, one or two edges for each constraint in $\mathcal{C}_\star[\![e_\star]\!]$

- W: the worklist of the nodes whose outgoing edges should be traversed
- D: an array that for each node gives an element of $\widehat{\mathbf{Val}}_\star$
- E: an array that for each node gives a list of constraints influenced (and outgoing edges)

Auxiliary procedure:
procedure add(q,d) is if $\neg$ ($d \subseteq$ D[q]) then D[q] := D[q] $\cup$ d;
W := cons(q,W);

Solving the Constraints (2)

Step 1 Initialisation
W := nil;
for q in Nodes do D[q] := $\emptyset$; E[q] := nil;

Step 2 Building the graph
for cc in $\mathcal{C}_\star[\![e_\star]\!]$ do
case cc of $\{t\} \subseteq p$: add(p,{t});
$p_1 \subseteq p_2$: E[p_1] := cons(cc,E[p_1]);
$\{t\} \subseteq p \Rightarrow p_1 \subseteq p_2$: E[$p_1$] := cons($cc$,E[$p_1$]);
E[p] := cons(cc,E[p]);

Step 3 Iteration
while W $\neq$ nil do
q := head(W); W := tail(W);
for cc in E[q] do
case cc of $p_1 \subseteq p_2$: add(p_2, D[p_1]);
$\{t\} \subseteq p \Rightarrow p_1 \subseteq p_2$: if $t \in$ D[p] then add(p_2, D[p_1]);

Step 4 Recording the solution
for ℓ in $\mathbf{Lab}_\star$ do $\widehat{\mathsf{C}}(\ell)$:= D[C(ℓ)]; for x in $\mathbf{Var}_\star$ do $\widehat{\rho}(x)$:= D[r(x)];

Example:

Initialisation of data structures

p	D[p]	E[p]
C(1)	$\emptyset$	[$\mathsf{id}_x \subseteq$ C(2) $\Rightarrow$ C(1) $\subseteq$ C(5)]
C(2)	id_x	[$\mathsf{id}_y \subseteq$ C(2) $\Rightarrow$ C(3) $\subseteq$ C(5), $\mathsf{id}_y \subseteq$ C(2) $\Rightarrow$ C(4) $\subseteq$ r(y), $\mathsf{id}_x \subseteq$ C(2) $\Rightarrow$ C(1) $\subseteq$ C(5), $\mathsf{id}_x \subseteq$ C(2) $\Rightarrow$ C(4) $\subseteq$ r(x)]
C(3)	$\emptyset$	[$\mathsf{id}_y \subseteq$ C(2) $\Rightarrow$ C(3) $\subseteq$ C(5)]
C(4)	id_y	[$\mathsf{id}_y \subseteq$ C(2) $\Rightarrow$ C(4) $\subseteq$ r(y), $\mathsf{id}_x \subseteq$ C(2) $\Rightarrow$ C(4) $\subseteq$ r(x)]
C(5)	$\emptyset$	[ ]
r(x)	$\emptyset$	[r(x) $\subseteq$ C(1)]
r(y)	$\emptyset$	[r(y) $\subseteq$ C(3)]

Example:

Iteration steps

W	[C(4),C(2)]	[r(x),C(2)]	[C(1),C(2)]	[C(5),C(2)]	[C(2)]	[ ]
p	D[p]	D[p]	D[p]	D[p]	D[p]	D[p]
C(1)	$\emptyset$	$\emptyset$	id_y	id_y	id_y	id_y
C(2)	id_x	id_x	id_x	id_x	id_x	id_x
C(3)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
C(4)	id_y	id_y	id_y	id_y	id_y	id_y
C(5)	$\emptyset$	$\emptyset$	$\emptyset$	id_y	id_y	id_y
r(x)	$\emptyset$	id_y	id_y	id_y	id_y	id_y
r(y)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Correctness:

Given input $\mathcal{C}_\star[\![e_\star]\!]$ the worklist algorithm terminates and the result $(\widehat{\mathsf{C}}, \widehat{\rho})$ produced by the algorithm satisfies

$$(\widehat{\mathsf{C}}, \widehat{\rho}) = \bigsqcap \{(\widehat{\mathsf{C}}', \widehat{\rho}') \mid (\widehat{\mathsf{C}}', \widehat{\rho}') \models_c \mathcal{C}_\star[\![e_\star]\!]\}$$

and hence it is the least solution to $\mathcal{C}_\star[\![e_\star]\!]$.

Complexity:

The algorithm takes at most $O(n^3)$ steps if the original expression $e_\star$ has size n.

Adding Data Flow Analysis

Idea: extend the set $\widehat{\mathbf{Val}}$ to contain other abstract values than just abstractions

- powerset (possibly finite)
- complete lattice (possibly satisfying Ascending Chain Condition)

Abstract Values as Powersets

Let **Data** be a set of *abstract data values* (i.e. abstract properties of booleans and integers)

$$\hat{v} \in \widehat{\mathbf{Val}}_d = \mathcal{P}(\mathbf{Term} \cup \mathbf{Data}) \quad \text{abstract values}$$

For each constant $c \in \mathbf{Const}$ we need an element $d_c \in \mathbf{Data}$

For each operator $op \in \mathbf{Op}$ we need a total function

$$\widehat{\mathsf{op}} : \widehat{\mathbf{Val}}_d \times \widehat{\mathbf{Val}}_d \to \widehat{\mathbf{Val}}_d$$

typically

$$\hat{v}_1 \ \widehat{\mathsf{op}} \ \hat{v}_2 = \bigcup\{d_{op}(d_1, d_2) \mid d_1 \in \hat{v}_1 \cap \mathbf{Data}, d_2 \in \hat{v}_2 \cap \mathbf{Data}\}$$

for some $d_{op} : \mathbf{Data} \times \mathbf{Data} \to \mathcal{P}(\mathbf{Data})$

Example: *Detection of Signs Analysis*

$\mathbf{Data}_{\mathsf{sign}} = \{\mathtt{tt}, \mathtt{ff}, \mathtt{-}, \mathtt{0}, \mathtt{+}\}$

$d_{\mathtt{true}} = \mathtt{tt}$

$d_7 = \mathtt{+}$

$\widehat{+}$ is defined from

d_+	tt	ff	-	0	+
tt	∅	∅	∅	∅	∅
ff	∅	∅	∅	∅	∅
-	∅	∅	{-}	{-}	{-, 0, +}
0	∅	∅	{-}	{0}	{+}
+	∅	∅	{-, 0, +}	{+}	{+}

Abstract Values as Powersets (1)

$(\hat{C}, \hat{\rho}) \models_d (\mathtt{fn}\ x\ \mathtt{=>}\ e_0)^\ell$ iff $\{\mathtt{fn}\ x\ \mathtt{=>}\ e_0\} \subseteq \hat{C}(\ell)$

$(\hat{C}, \hat{\rho}) \models_d (\mathtt{fun}\ f\ x\ \mathtt{=>}\ e_0)^\ell$ iff $\{\mathtt{fun}\ f\ x\ \mathtt{=>}\ e_0\} \subseteq \hat{C}(\ell)$

$(\hat{C}, \hat{\rho}) \models_d (t_1^{\ell_1}\ t_2^{\ell_2})^\ell$

iff $(\hat{C}, \hat{\rho}) \models_d t_1^{\ell_1} \wedge (\hat{C}, \hat{\rho}) \models_d t_2^{\ell_2} \wedge$
$(\forall(\mathtt{fn}\ x\ \mathtt{=>}\ t_0^{\ell_0}) \in \hat{C}(\ell_1) :$
$(\hat{C}, \hat{\rho}) \models_d t_0^{\ell_0} \wedge$
$\hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell)) \wedge$
$(\forall(\mathtt{fun}\ f\ x\ \mathtt{=>}\ t_0^{\ell_0}) \in \hat{C}(\ell_1) :$
$(\hat{C}, \hat{\rho}) \models_d t_0^{\ell_0} \wedge$
$\hat{C}(\ell_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(\ell_0) \subseteq \hat{C}(\ell) \wedge$
$\{\mathtt{fun}\ f\ x\ \mathtt{=>}\ t_0^{\ell_0}\} \subseteq \hat{\rho}(f))$

$(\widehat{C},\widehat{\rho}) \models_d c^\ell$ iff $\{d_c\} \subseteq \widehat{C}(\ell)$

$(\widehat{C},\widehat{\rho}) \models_d x^\ell$ iff $\widehat{\rho}(x) \subseteq \widehat{C}(\ell)$

$(\widehat{C},\widehat{\rho}) \models_d (\texttt{if } t_0^{\ell_0} \texttt{ then } t_1^{\ell_1} \texttt{ else } t_2^{\ell_2})^\ell$
iff $(\widehat{C},\widehat{\rho}) \models_d t_0^{\ell_0} \wedge$
$(d_{\texttt{true}} \in \widehat{C}(\ell_0) \Rightarrow ((\widehat{C},\widehat{\rho}) \models_d t_1^{\ell_1} \wedge \widehat{C}(\ell_1) \subseteq \widehat{C}(\ell))) \wedge$
$(d_{\texttt{false}} \in \widehat{C}(\ell_0) \Rightarrow ((\widehat{C},\widehat{\rho}) \models_d t_2^{\ell_2} \wedge \widehat{C}(\ell_2) \subseteq \widehat{C}(\ell)))$

$(\widehat{C},\widehat{\rho}) \models_d (\texttt{let } x = t_1^{\ell_1} \texttt{ in } t_2^{\ell_2})^\ell$
iff $(\widehat{C},\widehat{\rho}) \models_d t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models_d t_2^{\ell_2} \wedge \widehat{C}(\ell_1) \subseteq \widehat{\rho}(x) \wedge \widehat{C}(\ell_2) \subseteq \widehat{C}(\ell)$

$(\widehat{C},\widehat{\rho}) \models_d (t_1^{\ell_1} \; op \; t_2^{\ell_2})^\ell$
iff $(\widehat{C},\widehat{\rho}) \models_d t_1^{\ell_1} \wedge (\widehat{C},\widehat{\rho}) \models_d t_2^{\ell_2} \wedge \widehat{C}(\ell_1) \; \widehat{op} \; \widehat{C}(\ell_2) \subseteq \widehat{C}(\ell)$

Example:

$$(\texttt{let f = (fn x => (if } (\texttt{x}^1 > 0^2)^3 \texttt{ then (fn y => y}^4)^5 \texttt{ else (fn z => } 25^6)^7)^8)^9 \texttt{ in } ((\texttt{f}^{10}\ 3^{11})^{12}\ 0^{13})^{14})^{15}$$

A pure 0-CFA analysis will not be able to discover that the `else`-branch of the conditional will never be executed.

When we combine the analysis with a Detection of Signs Analysis then the analysis can determine that only $\texttt{fn y => y}^4$ is a possible abstraction at label 12.

	$(\widehat{C},\widehat{\rho})$	$(\widehat{C},\widehat{\rho})$
1	$\emptyset$	{+}
2	$\emptyset$	{0}
3	$\emptyset$	{tt}
4	$\emptyset$	{0}
5	$\{\texttt{fn y => y}^4\}$	$\{\texttt{fn y => y}^4\}$
6	$\emptyset$	$\emptyset$
7	$\{\texttt{fn z => } 25^6\}$	$\emptyset$
8	$\{\texttt{fn y => y}^4, \texttt{fn z => } 25^6\}$	$\{\texttt{fn y => y}^4\}$
9	$\{\texttt{fn x => } (\cdots)^8\}$	$\{\texttt{fn x => } (\cdots)^8\}$
10	$\{\texttt{fn x => } (\cdots)^8\}$	$\{\texttt{fn x => } (\cdots)^8\}$
11	$\emptyset$	{+}
12	$\{\texttt{fn y => y}^4, \texttt{fn z => } 25^6\}$	$\{\texttt{fn y => y}^4\}$
13	$\emptyset$	{0}
14	$\emptyset$	{0}
15	$\emptyset$	{0}
f	$\{\texttt{fn x => } (\cdots)^8\}$	$\{\texttt{fn x => } (\cdots)^8\}$
x	$\emptyset$	{+}
y	$\emptyset$	{0}
z	$\emptyset$	$\emptyset$

Abstract Values as Complete Lattices

A *monotone structure* consists of:

- a complete lattice L, and
- a set $\mathcal{F}$ of monotone functions of $L \times L \to L$.

An *instance* of a monotone structure consists of the structure $(L, \mathcal{F})$ and

- a mapping ι. from the constants $c \in \mathbf{Const}$ to values in L, and
- a mapping f. from the binary operators $op \in \mathbf{Op}$ to functions of $\mathcal{F}$.

Abstract Domains

For the Control Flow Analysis:

$$\begin{array}{rcrcll} \widehat{v} & \in & \widehat{\mathbf{Val}} & = & \mathcal{P}(\mathbf{Term}) & \text{abstract values} \\ \widehat{\rho} & \in & \widehat{\mathbf{Env}} & = & \mathbf{Var} \to \widehat{\mathbf{Val}} & \text{abstract environments} \\ \widehat{\mathsf{C}} & \in & \widehat{\mathbf{Cache}} & = & \mathbf{Lab} \to \widehat{\mathbf{Val}} & \text{abstract caches} \end{array}$$

For the Data Flow Analysis:

$$\begin{array}{rcrcll} \widehat{d} & \in & \widehat{\mathbf{Data}} & = & L & \text{abstract data values} \\ \widehat{\delta} & \in & \widehat{\mathbf{DEnv}} & = & \mathbf{Var} \to \widehat{\mathbf{Data}} & \text{abstract data environments} \\ \widehat{\mathsf{D}} & \in & \widehat{\mathbf{DCache}} & = & \mathbf{Lab} \to \widehat{\mathbf{Data}} & \text{abstract data caches} \end{array}$$

Abstract Values as Complete Lattices (1)

$$(\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D (\texttt{fn}\ x\ \texttt{=>}\ e_0)^\ell \quad \text{iff} \quad \{\texttt{fn}\ x\ \texttt{=>}\ e_0\} \subseteq \widehat{\mathsf{C}}(\ell)$$

$$(\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D (\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)^\ell \quad \text{iff} \quad \{\texttt{fun}\ f\ x\ \texttt{=>}\ e_0\} \subseteq \widehat{\mathsf{C}}(\ell)$$

$$\begin{array}{l} (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D (t_1^{\ell_1}\ t_2^{\ell_2})^\ell \\ \quad \text{iff} \quad (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_1^{\ell_1} \wedge (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_2^{\ell_2} \wedge \\ \qquad (\forall(\texttt{fn}\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1) : \ (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_0^{\ell_0} \wedge \\ \qquad\qquad \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\rho}(x) \ \wedge\ \widehat{\mathsf{D}}(\ell_2) \sqsubseteq \widehat{\delta}(x) \ \wedge \\ \qquad\qquad \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell) \ \wedge\ \widehat{\mathsf{D}}(\ell_0) \sqsubseteq \widehat{\mathsf{D}}(\ell)\ ) \wedge \\ \qquad (\forall(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}) \in \widehat{\mathsf{C}}(\ell_1) : \ (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_0^{\ell_0} \wedge \\ \qquad\qquad \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\rho}(x) \ \wedge\ \widehat{\mathsf{D}}(\ell_2) \sqsubseteq \widehat{\delta}(x) \ \wedge \\ \qquad\qquad \widehat{\mathsf{C}}(\ell_0) \subseteq \widehat{\mathsf{C}}(\ell) \ \wedge\ \widehat{\mathsf{D}}(\ell_0) \sqsubseteq \widehat{\mathsf{D}}(\ell) \ \wedge \\ \qquad\qquad \{\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}\} \subseteq \widehat{\rho}(f)) \end{array}$$

Abstract Values as Complete Lattices (2)

$$(\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D c^\ell \quad \text{iff} \quad \iota_C \sqsubseteq \widehat{\mathsf{D}}(\ell)$$

$$(\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D x^\ell \quad \text{iff} \quad \widehat{\rho}(x) \subseteq \widehat{\mathsf{C}}(\ell) \ \wedge\ \widehat{\delta}(x) \sqsubseteq \widehat{\mathsf{D}}(\ell)$$

$$\begin{array}{l} (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D (\texttt{if}\ t_0^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^\ell \\ \quad \text{iff} \quad (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_0^{\ell_0} \wedge \\ \qquad (\iota_{\texttt{true}} \sqsubseteq \widehat{\mathsf{D}}(\ell_0) \Rightarrow (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_1^{\ell_1} \wedge \\ \qquad\qquad \widehat{\mathsf{C}}(\ell_1) \subseteq \widehat{\mathsf{C}}(\ell) \wedge \widehat{\mathsf{D}}(\ell_1) \sqsubseteq \widehat{\mathsf{D}}(\ell)\ ) \wedge \\ \qquad (\iota_{\texttt{false}} \sqsubseteq \widehat{\mathsf{D}}(\ell_0) \Rightarrow (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_2^{\ell_2} \wedge \\ \qquad\qquad \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(\ell) \wedge \widehat{\mathsf{D}}(\ell_2) \sqsubseteq \widehat{\mathsf{D}}(\ell)\ ) \end{array}$$

Abstract Values as Complete Lattices (3)

$$\begin{array}{l} (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D (\texttt{let}\ x\ \texttt{=}\ t_1^{\ell_1}\ \texttt{in}\ t_2^{\ell_2})^\ell \\ \quad \text{iff} \quad (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_1^{\ell_1} \wedge \\ \qquad (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_2^{\ell_2} \wedge \\ \qquad \widehat{\mathsf{C}}(\ell_1) \subseteq \widehat{\rho}(x) \wedge \widehat{\mathsf{D}}(\ell_1) \sqsubseteq \widehat{\delta}(x) \ \wedge\ \widehat{\mathsf{C}}(\ell_2) \subseteq \widehat{\mathsf{C}}(\ell) \wedge \widehat{\mathsf{D}}(\ell_2) \sqsubseteq \widehat{\mathsf{D}}(\ell) \end{array}$$

$$\begin{array}{l} (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D (t_1^{\ell_1}\ op\ t_2^{\ell_2})^\ell \\ \quad \text{iff} \quad (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_1^{\ell_1} \ \wedge\ (\widehat{\mathsf{C}},\widehat{\mathsf{D}},\widehat{\rho},\widehat{\delta}) \models_D t_2^{\ell_2} \ \wedge \\ \qquad f_{op}(\widehat{\mathsf{D}}(\ell_1),\widehat{\mathsf{D}}(\ell_2)) \sqsubseteq \widehat{\mathsf{D}}(\ell) \end{array}$$

	$(\widehat{C},\widehat{\rho})$	$(\widehat{C},\widehat{\rho})$	$(\widehat{C},\widehat{\rho})$	$(\widehat{D},\widehat{\delta})$
1	$\emptyset$	{+}	$\emptyset$	{+}
2	$\emptyset$	{0}	$\emptyset$	{0}
3	$\emptyset$	{tt}	$\emptyset$	{tt}
4	$\emptyset$	{0}	$\emptyset$	{0}
5	{fn y => y^4}	{fn y => y^4}	{fn y => y^4}	$\emptyset$
6	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
7	{fn z => 25^6}	$\emptyset$	$\emptyset$	$\emptyset$
8	{ fn y => y^4, fn z => 25^6 }	{fn y => y^4}	{fn y => y^4}	$\emptyset$
9	{fn x => $(\cdots)^8$}	{fn x => $(\cdots)^8$}	{fn x => $(\cdots)^8$}	$\emptyset$
10	{fn x => $(\cdots)^8$}	{fn x => $(\cdots)^8$}	{fn x => $(\cdots)^8$}	$\emptyset$
11	$\emptyset$	{+}	$\emptyset$	{+}
12	{ fn y => y^4, fn z => 25^6 }	{fn y => y^4}	{fn y => y^4}	$\emptyset$
13	$\emptyset$	{0}	$\emptyset$	{0}
14	$\emptyset$	{0}	$\emptyset$	{0}
15	$\emptyset$	{0}	$\emptyset$	{0}
f	{fn x => $(\cdots)^8$}	{fn x => $(\cdots)^8$}	{fn x => $(\cdots)^8$}	$\emptyset$
x	$\emptyset$	{+}	$\emptyset$	{+}
y	$\emptyset$	{0}	$\emptyset$	{0}
z	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Adding Context Information

Mono-variant analysis: does not distinguish the various instances of variables and program points from one another. (Compare with context-insensitive interprocedural analysis.) 0-CFA is a typical example.

Poly-variant analysis: distinguishes between the various instances of variables and program points. (Compare with context-sensitive interprocedural analysis.)

A purely syntactic solution:

Expand

```
(let f = (fn x => x) in ((f f) (fn y => y)))
```

into

```
let f1 = (fn x1 => x1)
in  let f2 = (fn x2 => x2) in (f1 f2) (fn y => y)
```

and analyse the expanded expression.

The 0-CFA analysis is now able to deduce that the overall expression will evaluate to `fn y => y` only.

A purely semantic solution: Uniform k-CFA

Idea: extend the set $\widehat{\mathbf{Val}}$ to include context information

In a (uniform) *k-CFA* a context δ records the last k dynamic call points; hence contexts will be sequences of labels of length at most k and they will be updated whenever a function application is analysed. (Compare call strings of length at most k.)

Abstract Domains

$$
\begin{array}{rcll}
\delta & \in & \boxed{\Delta} & = \mathbf{Lab}^{\leq k} \quad \text{context information} \\
ce & \in & \mathbf{CEnv} & = \mathbf{Var} \to \boxed{\Delta} \quad \text{context environments} \\
\hat{v} & \in & \widehat{\mathbf{Val}} & = \mathcal{P}(\mathbf{Term} \times \mathbf{CEnv}) \quad \text{abstract values} \\
\hat{\rho} & \in & \widehat{\mathbf{Env}} & = (\mathbf{Var} \times \boxed{\Delta}) \to \widehat{\mathbf{Val}} \quad \text{abstract environments} \\
\hat{\mathsf{C}} & \in & \widehat{\mathbf{Cache}} & = (\mathbf{Lab} \times \boxed{\Delta}) \to \widehat{\mathbf{Val}} \quad \text{abstract caches}
\end{array}
$$

(Uniform because Δ used both for $\widehat{\mathbf{Env}}$ and $\widehat{\mathbf{Cache}}$.)

Acceptability Relation

$$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} e$$

where

- ce is the current context environment – will be changed when new bindings are made
- δ is the current context – will be changed when functions are called

Idea: The formula expresses that $(\hat{\mathsf{C}}, \hat{\rho})$ is an acceptable analysis of e in the *context* specified by ce and δ.

Control Flow Analysis with Context (1)

$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} (\texttt{fn}\ x\ \texttt{=>}\ e_0)^{\ell}$ iff $\{(\texttt{fn}\ x\ \texttt{=>}\ e_0, \boxed{ce})\} \subseteq \hat{\mathsf{C}}(\ell, \delta)$

$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} (\texttt{fun}\ f\ x\ \texttt{=>}\ e_0)^{\ell}$ iff $\{(\texttt{fun}\ f\ x\ \texttt{=>}\ e_0, \boxed{ce})\} \subseteq \hat{\mathsf{C}}(\ell, \delta)$

$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} (t_1^{\ell_1}\ t_2^{\ell_2})^{\ell}$ iff
$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} t_1^{\ell_1} \wedge (\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} t_2^{\ell_2} \wedge$
$(\forall (\texttt{fn}\ x\ \texttt{=>}\ t_0^{\ell_0}, \boxed{ce_0}) \in \hat{\mathsf{C}}(\ell_1, \delta)$:
$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce'_0}_{\delta_0} t_0^{\ell_0} \wedge \hat{\mathsf{C}}(\ell_2, \delta) \subseteq \hat{\rho}(x, \delta_0) \wedge \hat{\mathsf{C}}(\ell_0, \delta_0) \subseteq \hat{\mathsf{C}}(\ell, \delta)$
where $\delta_0 = \lceil \delta, \ell \rceil_k$ and $ce'_0 = ce_0[x \mapsto \delta_0]$) $\wedge$
$(\forall (\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}, \boxed{ce_0}) \in \hat{\mathsf{C}}(\ell_1, \delta)$:
$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce'_0}_{\delta_0} t_0^{\ell_0} \wedge \hat{\mathsf{C}}(\ell_2, \delta) \subseteq \hat{\rho}(x, \delta_0) \wedge \hat{\mathsf{C}}(\ell_0, \delta_0) \subseteq \hat{\mathsf{C}}(\ell, \delta) \wedge$
$\{(\texttt{fun}\ f\ x\ \texttt{=>}\ t_0^{\ell_0}, \boxed{ce_0})\} \subseteq \hat{\rho}(f, \delta_0)$
where $\delta_0 = \lceil \delta, \ell \rceil_k$ and $ce'_0 = ce_0[f \mapsto \delta_0, x \mapsto \delta_0]$)

Control Flow Analysis with Context (2)

$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} c^{\ell}$ always

$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} x^{\ell}$ iff $\hat{\rho}(x, \boxed{ce(x)}) \subseteq \hat{\mathsf{C}}(\ell, \delta)$

$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} (\texttt{if}\ t_0^{\ell_0}\ \texttt{then}\ t_1^{\ell_1}\ \texttt{else}\ t_2^{\ell_2})^{\ell}$ iff
$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} t_0^{\ell_0} \wedge (\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} t_1^{\ell_1} \wedge (\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} t_2^{\ell_2} \wedge$
$\hat{\mathsf{C}}(\ell_1, \delta) \subseteq \hat{\mathsf{C}}(\ell, \delta) \wedge \hat{\mathsf{C}}(\ell_2, \delta) \subseteq \hat{\mathsf{C}}(\ell, \delta)$

$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} (\texttt{let}\ x\ \texttt{=}\ t_1^{\ell_1}\ \texttt{in}\ t_2^{\ell_2})^{\ell}$ iff
$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} t_1^{\ell_1} \wedge (\hat{\mathsf{C}}, \hat{\rho}) \models^{ce'}_{\delta} t_2^{\ell_2} \wedge$
$\hat{\mathsf{C}}(\ell_1, \delta) \subseteq \hat{\rho}(x, \delta) \wedge \hat{\mathsf{C}}(\ell_2, \delta) \subseteq \hat{\mathsf{C}}(\ell, \delta)$
where $ce' = ce[x \mapsto \delta]$

$(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} (t_1^{\ell_1}\ op\ t_2^{\ell_2})^{\ell}$ iff $(\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} t_1^{\ell_1} \wedge (\hat{\mathsf{C}}, \hat{\rho}) \models^{ce}_{\delta} t_2^{\ell_2}$

Example:

$$\texttt{(let f = (fn x => x}^{1}\texttt{)}^{2}\texttt{ in ((f}^{3}\texttt{ f}^{4}\texttt{)}^{5}\texttt{ (fn y => y}^{6}\texttt{)}^{7}\texttt{)}^{8}\texttt{)}^{9}$$

Contexts of interest for uniform 1-CFA:

- Λ: the initial context
- 5: the context when the application point labelled 5 has been passed
- 8: the context when the application point labelled 8 has been passed

Context environments of interest for uniform 1-CFA:

- $ce_0 = [\,]$ the initial (empty) context environment
- $ce_1 = ce_0[\texttt{f} \mapsto \Lambda]$ the context environment for the analysis of the body of the `let`-construct
- $ce_2 = ce_0[\texttt{x} \mapsto 5]$ the context environment used for the analysis of the body of `f` initiated at the application point 5
- $ce_3 = ce_0[\texttt{x} \mapsto 8]$ the context environment used for the analysis of the body of `f` initiated at the application point 8.

Example: Let us take $\widehat{C}_{id}'$ and $\widehat{\rho}_{id}'$ to be:

$$\widehat{C}_{id}'(1,5) = \{(\texttt{fn x => x}^{1},ce_0)\} \qquad \widehat{C}_{id}'(1,8) = \{(\texttt{fn y => y}^{6},ce_0)\}$$
$$\widehat{C}_{id}'(2,\Lambda) = \{(\texttt{fn x => x}^{1},ce_0)\} \qquad \widehat{C}_{id}'(3,\Lambda) = \{(\texttt{fn x => x}^{1},ce_0)\}$$
$$\widehat{C}_{id}'(4,\Lambda) = \{(\texttt{fn x => x}^{1},ce_0)\} \qquad \widehat{C}_{id}'(5,\Lambda) = \{(\texttt{fn x => x}^{1},ce_0)\}$$
$$\widehat{C}_{id}'(7,\Lambda) = \{(\texttt{fn y => y}^{6},ce_0)\} \qquad \widehat{C}_{id}'(8,\Lambda) = \{(\texttt{fn y => y}^{6},ce_0)\}$$
$$\widehat{C}_{id}'(9,\Lambda) = \{(\texttt{fn y => y}^{6},ce_0)\}$$

$$\widehat{\rho}_{id}'(\texttt{f},\Lambda) = \{(\texttt{fn x => x}^{1},ce_0)\}$$
$$\widehat{\rho}_{id}'(\texttt{x},5) = \{(\texttt{fn x => x}^{1},ce_0)\} \qquad \widehat{\rho}_{id}'(\texttt{x},8) = \{(\texttt{fn y => y}^{6},ce_0)\}$$

This is an acceptable analysis result:

$$(\widehat{C}_{id}', \widehat{\rho}_{id}') \models^{ce_0}_{\Lambda} \texttt{(let f = (fn x => x}^{1}\texttt{)}^{2}\texttt{ in ((f}^{3}\texttt{ f}^{4}\texttt{)}^{5}\texttt{ (fn y => y}^{6}\texttt{)}^{7}\texttt{)}^{8}\texttt{)}^{9}$$

Complexity

Uniform k-CFA has exponential worst case complexity even when $k = 1$

Assume that the expression has size n and that it has p different variables. Then Δ has $O(n)$ elements and hence there will be $O(p \cdot n)$ different pairs (x, δ) and $O(n^2)$ different pairs (ℓ, δ). This means that $(\widehat{C}, \widehat{\rho})$ can be seen as an $O(n^2)$ tuple of values from $\widehat{\mathbf{Val}}$. Since $\widehat{\mathbf{Val}}$ itself is a powerset of pairs of the form (t, ce) and there are $O(n \cdot n^p)$ such pairs it follows that $\widehat{\mathbf{Val}}$ has height $O(n \cdot n^p)$. Since $O(p) = O(n)$ we have the exponential worst case complexity.

0-CFA analysis has polynomial worst case complexity

It corresponds to letting Δ be a singleton. Repeating the above calculations we can see $(\widehat{C}, \widehat{\rho})$ as an $O(p+n)$ tuple of values from $\widehat{\mathbf{Val}}$, and $\widehat{\mathbf{Val}}$ will be a lattice of height $O(n)$.

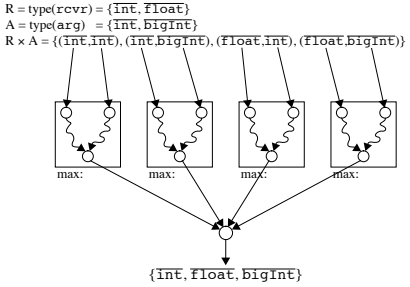
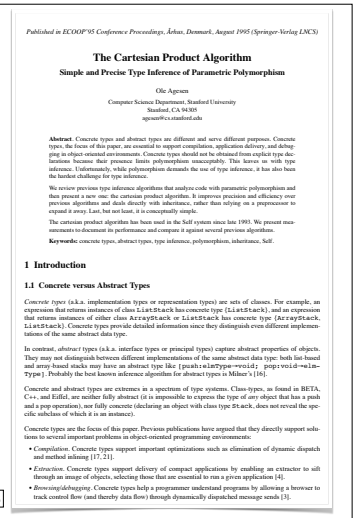


Fig. 6. CPA forms the cartesian product of the receiver type and argument types of the send it is analyzing. Each member of the product is propagated to a template reserved exclusively for that receiver and argument combination. The result type of the send is obtained by collecting the contributions from each template.

Ole Agesen. *The Cartesian Product Algorithm: Simple and Precise Type Inference of Parametric Polymorphism* ECOOP'95



Published in ECOOP'95 Conference Proceedings, Århus, Denmark, August 1995 (Springer-Verlag LNCS)

The Cartesian Product Algorithm

Simple and Precise Type Inference of Parametric Polymorphism

Ole Agesen
Computer Science Department, Stanford University
Stanford, CA 94305
agesen@cs.stanford.edu

Abstract. Concrete types and abstract types are different and serve different purposes. Concrete types, the focus of this paper, are essential to support compilation, application delivery, and debugging in object-oriented environments. Concrete types should not be obtained from explicit type declarations because their presence limits polymorphism unacceptably. This leaves us with type inference. Unfortunately, while polymorphism demands the use of type inference, it has also been the hardest challenge for type inference.

We review previous type inference algorithms that analyze code with parametric polymorphism and then present a new one: the cartesian product algorithm. It improves precision and efficiency over previous algorithms and deals directly with inheritance, rather than relying on a preprocessor to expand it away. Last, but not least, it is conceptually simple.

The cartesian product algorithm has been used in the Self system since late 1993. We present measurements to document its performance and compare it against several previous algorithms.

Keywords: concrete types, abstract types, type inference, polymorphism, inheritance, Self.

1 Introduction

1.1 Concrete versus Abstract Types

Concrete types (a.k.a. implementation types or representation types) are sets of classes. For example, an expression that returns instances of class `ListStack` has concrete type {`ListStack`}, and an expression that returns instances of either class `ArrayStack` or `ListStack` has concrete type {`ArrayStack`, `ListStack`}. Concrete types provide detailed information since they distinguish even different implementations of the same abstract data type.

In contrast, *abstract types* (a.k.a. interface types or principal types) capture abstract properties of objects. They may not distinguish between different implementations of the same abstract data type: both list-based and array-based stacks may have an abstract type like [`push:elmType→void; pop:→void→elmType`]. Probably the best known inference algorithm for abstract types is Milner's [16].

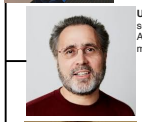
Concrete and abstract types are extremes in a spectrum of type systems. Class-types, as found in BETA, C++, and Eiffel, are neither fully abstract (it is impossible to express the type of *any* object that has a push and a pop operation), nor fully concrete (declaring an object with class type `Stack`, does not reveal the specific subclass of which it is an instance).

Concrete types are the focus of this paper. Previous publications have argued that they directly support solutions to several important problems in object-oriented programming environments:

- *Compilation.* Concrete types support important optimizations such as elimination of dynamic dispatch and method inlining [17, 21].
- *Extraction.* Concrete types support delivery of compact applications by enabling an extractor to sift through an image of objects, selecting those that are essential to run a given application [4].
- *Browsing/debugging.* Concrete types help a programmer understand programs by allowing a browser to track control flow (and thereby data flow) through dynamically dispatched message sends [3].

Dave Ungar obtained a PhD from UC Berkeley in 1985 with David Patterson on *The Design and Evaluation of a High-Performance Smalltalk System*; it won the 1986 ACM Dissertation Award. He was an assistant professor at Stanford. In 1991, he joined Sun and became a distinguished engineer. In 2006 he was recognized as a Distinguished Engineer by the ACM and in 2010 a Fellow. Ungar's 1984 paper, *Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm* won a Retrospective SIGSOFT Impact Paper Award. He was awarded the Dahl-Nygaard Senior Prize in 2009.

Self



Urs Hölzle is a Swiss software engineer and technology executive. He is the senior vice president of technical infrastructure and Google Fellow at Google. As Google's eighth employee and its first VP of Engineering, he has shaped much of Google's development processes and infrastructure.

Craig Chambers has been at Google since 2007.[1] Prior to this, he was a Professor at the University of Washington. He received his B.S. from MIT in 1986 and a Ph.D. from Stanford in 1992. He is known for Self, which introduced prototypes as an alternative to classes, and code-splitting, a compilation technique that generates separate code paths for fast and general cases to speed execution of dynamically-typed programs.

Ole Agesen joined VMware in 1999 was appointed as a VMware Fellow in 2012. During his 12-year tenure as a principal engineer, He has earned 52 issued patents, 25 of which are assigned to VMware and made major contributions to VMware vSphere. Agesen worked for Sun Microsystems as a technical lead for the company's first server-class Java Virtual Machine, ExactVM. He holds a master's from Aarhus in Denmark and a Ph.D. in from Stanford.

Jeff Dean joined Google in '99, he headsArtificial Intelligence. The projects he worked:

- Spanner, a scalable, multi-version, globally distributed, and replicated database
- BigTable, a large-scale semi-structured storage system[2]
- MapReduce, a system for large-scale data processing applications[2]
- LevelDB, an open-source on-disk key-value store
- DistBelief, a machine-learning system for deep neural nets refactored into TensorFlow

Scaling Program Analysis

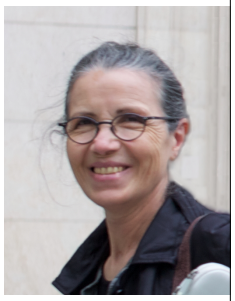
CS7575

- Lecture 5 — Abstract Interpretation - PoPA Chapter 5

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

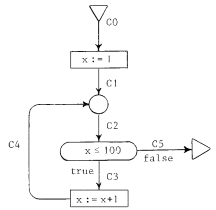
Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France



Acknowledgements

We wish to thank M. Sintzoff for stimulating discussions. We were very lucky to have F. Blanc do the typing for us.



Let us note [a, b] where a ≤ b the predicate a ≤ x ≤ b. The system of equations corresponding to the example is :

(0) C0 = [,]
(1) C1 = [1, 1]
(2) C2 = C1 ∪ C4
(3) C3 = C2 ∩ [-∞, 100]
(4) C4 = C3 + [1, 1]
(5) C5 = C2 ∩ [101, +∞]

Assignment statements are treated using an interval arithmetic (e.g. [i, j] + [k, ℓ] = [i+k, j+ℓ] naturally extended to include the case of the empty interval). Similarly tests are treated using an "interval logic". Since there exist infinite Kleene's sequences (e.g. [,] < [0, 0] < [0, 1] < ... < [0, +∞] for the program x := 0 ; while true do x := x + 1), we must use an approximation sequence. Hence the results will be somewhat inaccurate but runtime subscript tests may be inserted in the absence of certainty.

Let us define the widening ∇ of intervals by :

- [,] is the null element of ∇
- [i, j] ∇ [k, ℓ] = [if k < i then -∞ else i fi, if ℓ > j then +∞ else j fi]

∇ satisfies the requirements of 9.1.3. According to 9.1.3.4 the system of equations is modified by :

(2) C2 = C2 ∇ (C1 ∪ C4)

The corresponding approximation sequence is :

Ci = [,] for i ∈ [0, 5]
* C1 = [1, 1]
C2 = C2 ∇ (C1 ∪ C4)
= [,] ∇ ([1, 1] ∪ [,])
= [,] ∇ [1, 1]
= [1, 1]
C3 = C2 ∩ [-∞, 100]
= [1, 1] ∩ [-∞, 100]
= [1, 1]
C4 = C3 + [1, 1]
= [1, 1] + [1, 1]
= [2, 2]
C2 = C2 ∇ (C1 ∪ C4)
= [1, 1] ∇ ([1, 1] ∪ [2, 2])
= [1, 1] ∇ [1, 2]
* C2 = [1, +∞]
C3 = C2 ∩ [-∞, 100]
= [1, +∞] ∩ [-∞, 100]
* C3 = [1, 100]
C4 = C3 + [1, 1]
= [1, 100] + [1, 1]
* C4 = [2, 101]
Note : C1 ∪ C4 = [1, 101] ≤ C2 = [1, +∞]
stop on that path.
C5 = C2 ∩ [101, +∞]
= [1, +∞] ∩ [101, +∞]
* C5 = [101, +∞]
exit, stop.

The final context on each arc is marked by a star *. Note that the results are approximate ones, (e.g. C5).

In this example the widening is a very rough operation which introduces a great loss of information. However it can be seen in the trace that tests behave like filters. Furthermore, for PASCAL like languages, one can first use the bounds given in the declaration of x before widening to infinite limits.

Early years (1972-73): formal semantics

Patrick Cousot: works on the operational semantics of programming languages and the derivation of implementations from the formal definition.

Patrick Cousot. *Définition interprétative et implantation de languages de programmation*. Thèse de Docteur Ingénieur en Informatique, Université Joseph Fourier, Grenoble, France, 14 Décembre 1974 (submitted in 1973 but defended after finishing military service with J.D. Ichbiah at CII).

Static analysis of the formal definition and transformation to get the implementation by “pre-evaluation” (similar to the more recent “partial evaluation”)

Before starting (1972-73): formal syntax

- Radhia Rezig: works on precedence parsing (R.W. Floyd, N. Wirth and H. Weber, etc.) for Algol 68
  - ➡ Pre-processing (by static analysis and transformation) of the grammar before building the bottom-up parser
- Patrick Cousot: works on context-free grammar parsing (J. Earley and F. De Remer)
  - ➡ Pre-processing (by static analysis and transformation) of the grammar before building the top-down parser

- Radhia Rezig. Application de la méthode de précédence totale à l’analyse d’Algol 68, Master thesis, Université Joseph Fourier, Grenoble, France, September 1972.
- Patrick Cousot. Un analyseur syntaxique pour grammaires hors contexte ascendant sélectif et général. In Congrès AFCET 72, Brochure 1, pages 106-130, Grenoble, France, 6-9 November 1972.

1972

Intervals →

Assertions →

Static analysis →

pas le niveau de "compréhension" des programmes. Les langages actuels ne sont pas faits pour l'optimisation. Entre autres, il y a certains faits sur un programme qui sont connus du programmeur et qui ne sont pas explicites dans le programme. On pourrait y remédier en incluant des assertions, tout comme on insère des déclarations de type pour les variables.

Exemple :

(1) - pour i de 0 à 10 faire a[i] := i ; fin ;

(2) - pour i de 11 à 10000 faire a[i] := 0 ; fin ;

(3) - a[(a[j] + 1) x a [j + 1]] := j ;

(4) - si a[j x j + 2 x j + 1] ≠ a[j] aller à étiquette ;

Pour un tel programme, il est important de savoir que $1 \leq j < 99$ (à charge éventuellement au système de le déduire à partir d'autres assertions), parce qu'on peut alors remplacer (4) par (4') :

(4') si j < 10 aller à étiquette ;

Cette insertion d'assertions peut donc servir de guide à une analyse automatique des programmes essentielle pour l'optimisation (mais également pour la mise au point, la documentation automatique, la décompilation, l'adaptation à un changement d'environnement d'exécution...). Dans tous les exemples que nous avons pris, (équivalence de définitions de données, équivalence de définition d'opérateurs) nous avons conduit cette analyse sémantique à la main.

La possibilité de son automation, nous semble conditionner les progrès dans le domaine de l'optimisation de l'implantation automatisée d'un langage étant donnée sa définition, aussi bien que dans celui de l'optimisation des programmes [41].

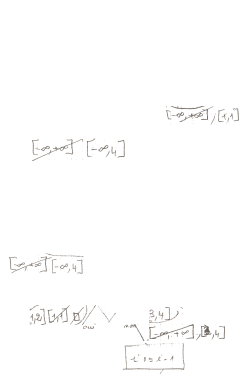
1974: The origins

Radhia Rezig shows her interval analysis ideas to Patrick Cousot

- ➡ Patrick very critical on going backwards from [-∞, +∞] and claims that going forward would be much better
- ➡ Patrick also very skeptical on forward termination for loops

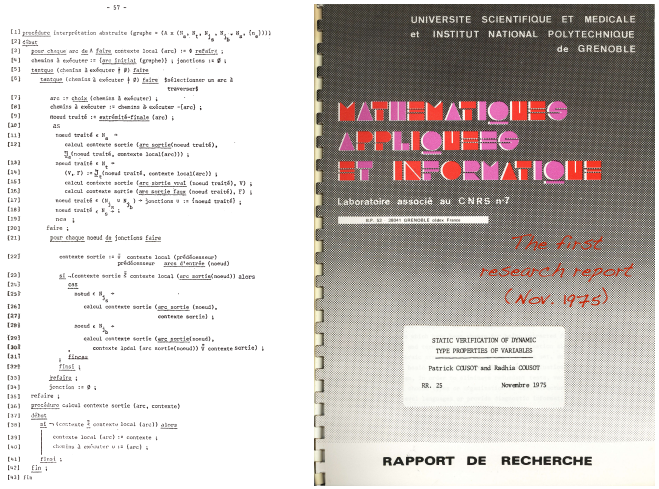

Radhia comes back with the idea of extrapolating bounds to ±∞ for the forward analysis

We discover widening = induction in the abstract and that the idea is very general!!



Notes of Radhia Rezig on forward iteration from $\square = \bot$ (forward least fix-point) versus backward iteration from $[-\infty, +\infty]$ (backward greatest fix-point)

The first abstract interpreter with widening (as of 23 Sep. 1975)



UNIVERSITE SCIENTIFIQUE ET MEDICALE
et INSTITUT NATIONAL POLYTECHNIQUE
de GRENOBLE

MATHEMATIQUES APPLIQUEES ET INFORMATIQUE

Laboratoire associé au CNRS n°7

The first reSearch report (Nov. 1975)

STATIC VERIFICATION OF DYNAMIC TYPE PROPERTIES OF VARIABLES

Patrick COUSOT and Radhia COUSOT

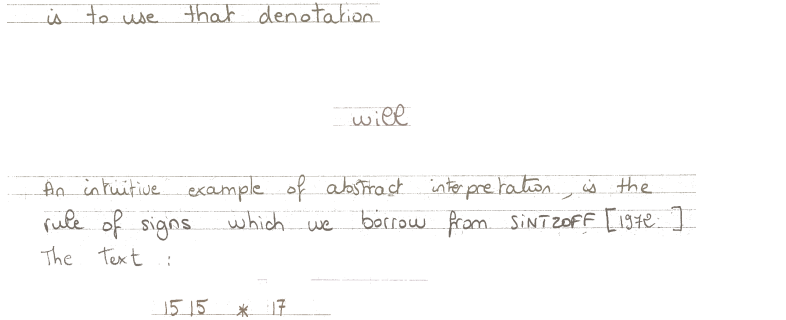
RR. 25 Novembre 1975

RAPPORT DE RECHERCHE

On submitting to POPL1977

For 4th POPL'77, Patrick and Radhia submitted on August 12, 1976 hard copies of a two-hands written manuscript of 100 pages.

Never published in a "scientific journal".



_0_ INTRODUCTION

A program denotes (complicated)

is to use that denotation

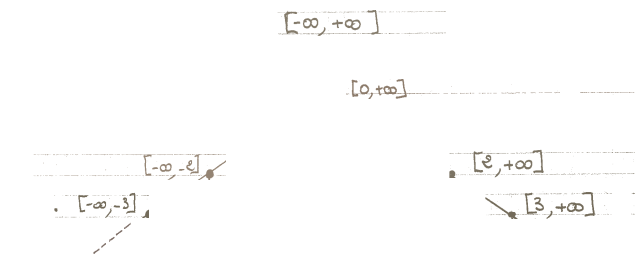
will

An intuitive example of abstract interpretation, is the rule of signs which we borrow from SINTZOFF [1972]. The Text :

15 15 * 17

KEY WORDS AND PHRASES

of programming law



5 CONCLUDING REMARKS

days

program speci

dels

A Mundane Approach to Semantic Correctness

Semantics:

$$p \vdash v_1 \rightsquigarrow v_2$$

where $v_1, v_2 \in V$.

Note: $\rightsquigarrow$ might be deterministic.

Program analysis:

$$p \vdash l_1 \triangleright l_2$$

where $l_1, l_2 \in L$.

Note: $\triangleright$ should be deterministic:

$$f_p(l_1) = l_2.$$

What is the relationship between the semantics and the analysis?

Restrict attention to analyses where properties directly describe sets of values i.e. *"first-order" analyses* (rather than *"second-order" analyses*).

Example: Data Flow Analysis

Structural Operational Semantics:

Values: $V = \mathbf{State}$

Transitions:

$$S_\star \vdash \sigma_1 \rightsquigarrow \sigma_2$$

iff

$$\langle S_\star, \sigma_1 \rangle \rightarrow^* \sigma_2$$

Constant Propagation Analysis:

Properties: $L = \widehat{\mathbf{State}}_{\mathsf{CP}} = (\mathbf{Var}_\star \rightarrow \mathbf{Z}^\top)_\perp$

Transitions:

$$S_\star \vdash \widehat{\sigma}_1 \rhd \widehat{\sigma}_2$$

iff

$$\widehat{\sigma}_1 = \iota$$
$$\widehat{\sigma}_2 = \bigsqcup\{\mathsf{CP}_\bullet(\ell) \mid \ell \in \mathit{final}(S_\star)\}$$
$$(\mathsf{CP}_\circ, \mathsf{CP}_\bullet) \models \mathsf{CP}^=(S_\star)$$

Example: Control Flow Analysis

Structural Operational Semantics:

Values: $V = \mathbf{Val}$

Transitions:

$$e_\star \vdash v_1 \rightsquigarrow v_2$$

iff

$$[\,] \vdash (e_\star\ v_1^{\ell_1})^{\ell_2} \rightarrow^* v_2^{\ell_2}$$

Pure 0-CFA Analysis:

Properties: $L = \widehat{\mathbf{Env}} \times \widehat{\mathbf{Val}}$

Transitions:

$$e_\star \vdash (\widehat{\rho}_1, \widehat{v}_1) \rhd (\widehat{\rho}_2, \widehat{v}_2)$$

iff

$$\widehat{\mathsf{C}}(\ell_1) = \widehat{v}_1$$
$$\widehat{\mathsf{C}}(\ell_2) = \widehat{v}_2$$
$$\widehat{\rho}_1 = \widehat{\rho}_2 = \widehat{\rho}$$
$$(\widehat{\mathsf{C}}, \widehat{\rho}) \models (e_\star\ \mathsf{c}^{\ell_1})^{\ell_2}$$

for some place holder constant c

Correctness Relations

$$R : V \times L \rightarrow \{\mathit{true}, \mathit{false}\}$$

Idea: $v\ R\ l$ means that the value v is described by the property l.

Correctness criterion: R is preserved under computation:

$$\begin{array}{ccccc} p & \vdash & v_1 & \rightsquigarrow & v_2 \\ & & \vdots & & \vdots \\ & & R & \Rightarrow & R \\ & & \vdots & & \vdots \\ p & \vdash & l_1 & \rhd & l_2 \end{array}$$

logical relation:

$(p \vdash \cdot \rightsquigarrow \cdot)\ (R \twoheadrightarrow R)\ (p \vdash \cdot \rhd \cdot)$

Admissible Correctness Relations

$$v\ R\ l_1 \ \wedge\ l_1 \sqsubseteq l_2 \ \Rightarrow\ v\ R\ l_2$$
$$(\forall l \in L' \subseteq L : v\ R\ l) \ \Rightarrow\ v\ R\ (\bigsqcap L')$$

Two consequences:

$$v\ R\ \top$$
$$v\ R\ l_1 \ \wedge\ v\ R\ l_2 \ \Rightarrow\ v\ R\ (l_1 \sqcap l_2)$$

Assumption: $(L, \sqsubseteq)$ is a complete lattice.

Example: Data Flow Analysis

Correctness relation

$$R_{\mathsf{CP}} : \mathbf{State} \times \widehat{\mathbf{State}}_{\mathsf{CP}} \to \{\mathit{true}, \mathit{false}\}$$

is defined by

$$\sigma \; R_{\mathsf{CP}} \; \widehat{\sigma} \quad \text{iff} \quad \forall x \in \mathit{FV}(S_\star) : (\widehat{\sigma}(x) = \top \;\vee\; \sigma(x) = \widehat{\sigma}(x))$$

Example: Control Flow Analysis

Correctness relation

$$R_{\mathsf{CFA}} : \mathbf{Val} \times (\widehat{\mathbf{Env}} \times \widehat{\mathbf{Val}}) \to \{\mathit{true}, \mathit{false}\}$$

is defined by

$$v \; R_{\mathsf{CFA}} \; (\widehat{\rho}, \widehat{v}) \quad \text{iff} \quad v \; \mathcal{V} \; (\widehat{\rho}, \widehat{v})$$

where $\mathcal{V}$ is given by:

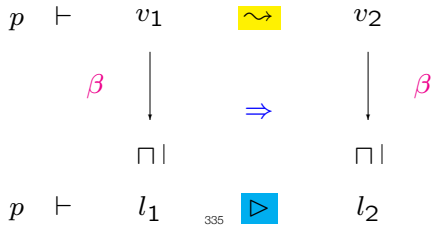
$$v \; \mathcal{V} \; (\widehat{\rho}, \widehat{v}) \text{ iff } \begin{cases} \mathit{true} & \text{if } v = c \\ t \in \widehat{v} \wedge \forall x \in \mathit{dom}(\rho) : \rho(x) \; \mathcal{V} \; (\widehat{\rho}, \widehat{\rho}(x)) & \text{if } v = \texttt{close } t \texttt{ in } \rho \end{cases}$$

Representation Functions

$$\beta : V \to L$$

Idea: β maps a value to the *best* property describing it.

Correctness criterion:



Equivalence of Correctness Criteria

Given a representation function β we define a correctness relation R_β by

$$v \; R_\beta \; l \quad \text{iff} \quad \beta(v) \sqsubseteq l$$

Given a correctness relation R we define a representation function β_R by

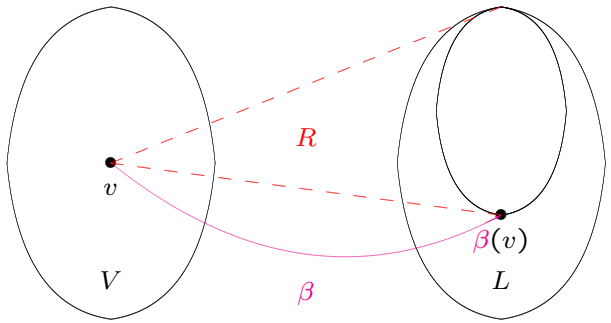
$$\beta_R(v) \;=\; \bigsqcap \{l \mid v \; R \; l\}$$

Lemma:

(i) Given $\beta : V \to L$, then the relation $R_\beta : V \times L \to \{\mathit{true}, \mathit{false}\}$ is an admissible correctness relation such that $\beta_{R_\beta} = \beta$.

(ii) Given an admissible correctness relation $R : V \times L \to \{\mathit{true}, \mathit{false}\}$, then β_R is well-defined and $R_{\beta_R} = R$.

Equivalence of Criteria: R is *generated* by β



Example: Data Flow Analysis

Representation function

$$\beta_{\mathsf{CP}} : \mathbf{State} \to \widehat{\mathbf{State}}_{\mathsf{CP}}$$

is defined by

$$\beta_{\mathsf{CP}}(\sigma) = \lambda x.\sigma(x)$$

R_{CP} is *generated by* β_{CP}:

$$\sigma \; R_{\mathsf{CP}} \; \widehat{\sigma} \quad \underline{\text{iff}} \quad \beta_{\mathsf{CP}}(\sigma) \sqsubseteq_{\mathsf{CP}} \widehat{\sigma}$$

Example: Control Flow Analysis

Representation function

$$\beta_{\mathsf{CFA}} : \mathbf{Val} \to \widehat{\mathbf{Env}} \times \widehat{\mathbf{Val}}$$

is defined by

$$\beta_{\mathsf{CFA}}(v) = \begin{cases} (\lambda x.\emptyset, \emptyset) & \text{if } v = c \\ (\beta^E_{\mathsf{CFA}}(\rho), \{t\}) & \text{if } v = \texttt{close } t \texttt{ in } \rho \end{cases}$$

$$\beta^E_{\mathsf{CFA}}(\rho)(x) = \bigcup\{\widehat{\rho}_y(x) \mid \beta_{\mathsf{CFA}}(\rho(y)) = (\widehat{\rho}_y, \widehat{v}_y) \text{ and } y \in dom(\rho)\} \cup \begin{cases} \{\widehat{v}_x\} & \text{if } x \in dom(\rho) \text{ and } \beta_{\mathsf{CFA}}(\rho(x)) = (\widehat{\rho}_x, \widehat{v}_x) \\ \emptyset & \text{otherwise} \end{cases}$$

R_{CFA} is *generated by* β_{CFA}:

$$v \; R_{\mathsf{CFA}} \; (\widehat{\rho}, \widehat{v}) \quad \underline{\text{iff}} \quad \beta_{\mathsf{CFA}}(v) \sqsubseteq_{\mathsf{CFA}} (\widehat{\rho}, \widehat{v})$$

A Modest Generalisation

Semantics:

$$p \vdash v_1 \rightsquigarrow v_2$$

where $v_1 \in V_1, v_2 \in V_2$

Program analysis:

$$p \vdash l_1 \triangleright l_2$$

where $l_1 \in L_1, l_2 \in L_2$

$$\begin{array}{ccccc} p & \vdash & v_1 & \rightsquigarrow & v_2 \\ & & \vdots & & \vdots \\ & & R_1 & \Rightarrow & R_2 \\ & & \vdots & & \vdots \\ p & \vdash & l_1 & \triangleright & l_2 \end{array}$$

logical relation:

$$(p \vdash \cdot \rightsquigarrow \cdot) \; (R_1 \twoheadrightarrow R_2) \; (p \vdash \cdot \triangleright \cdot)$$

Higher-Order Formulation

Assume that

- R_1 is an admissible correctness relation for V_1 and L_1
  that is *generated by* the representation function $\beta_1 : V_1 \to L_1$
- R_2 is an admissible correctness relation for V_2 and L_2
  that is *generated by* the representation function $\beta_2 : V_2 \to L_2$

Then the relation $R_1 \twoheadrightarrow R_2$ is an admissible correctness relation for $V_1 \to V_2$ and $L_1 \to L_2$
that is *generated by* the representation function $\beta_1 \twoheadrightarrow \beta_2$ defined by

$$(\beta_1 \twoheadrightarrow \beta_2)(\rightsquigarrow) = \lambda l_1. \bigsqcup\{\beta_2(v_2) \mid \beta_1(v_1) \sqsubseteq l_1 \ \wedge \ v_1 \rightsquigarrow v_2\}$$

Example:

Semantics:

$\texttt{plus} \vdash (z_1, z_2) \rightsquigarrow z_1 + z_2$

where $z_1, z_2 \in \mathbf{Z}$

Program analysis:

$\texttt{plus} \vdash ZZ \rhd \{z_1 + z_2 \mid (z_1, z_2) \in ZZ\}$

where $ZZ \subseteq \mathbf{Z} \times \mathbf{Z}$

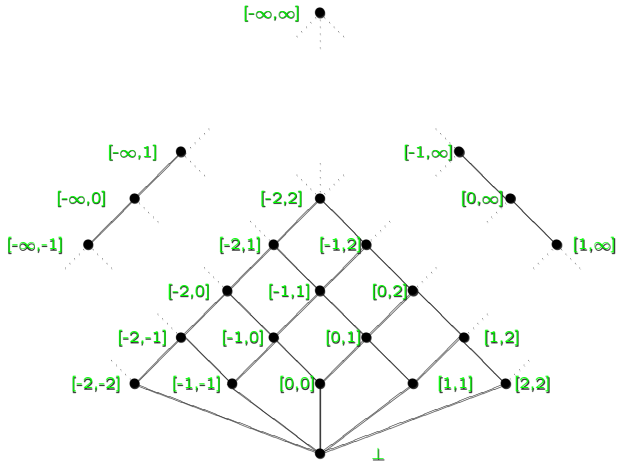
	Correctness relations	Representation functions
result	R_Z	$\beta_Z(z) = \{z\}$
argument	$R_{Z \times Z}$	$\beta_{Z \times Z}(z_1, z_2) = \{(z_1, z_2)\}$
plus	$(\texttt{plus} \vdash \cdot \rightsquigarrow \cdot) \ (R_{Z \times Z} \twoheadrightarrow R_Z) \ (\texttt{plus} \vdash \cdot \rhd \cdot)$	$(\beta_{Z \times Z} \twoheadrightarrow \beta_Z)(\texttt{plus} \vdash \cdot \rightsquigarrow \cdot) \sqsubseteq (\texttt{plus} \vdash \cdot \rhd \cdot)$

Approximation of Fixed Points

- Fixed points
- Widening
- Narrowing

Example: lattice of intervals for *Array Bound Analysis*

The complete lattice **Interval** = (**Interval**, $\sqsubseteq$)



Fixed points

Let $f : L \to L$ be a *monotone function* on a complete lattice $L = (L, \sqsubseteq, \sqcup, \sqcap, \bot, \top)$.

l is a *fixed point*	iff	$f(l) = l$	$Fix(f) = \{l \mid f(l) = l\}$
f is *reductive* at l	iff	$f(l) \sqsubseteq l$	$Red(f) = \{l \mid f(l) \sqsubseteq l\}$
f is *extensive* at l	iff	$f(l) \sqsupseteq l$	$Ext(f) = \{l \mid f(l) \sqsupseteq l\}$

Tarski's Theorem ensures that

$$lfp(f) = \bigsqcap Fix(f) = \bigsqcap Red(f) \in Fix(f) \subseteq Red(f)$$

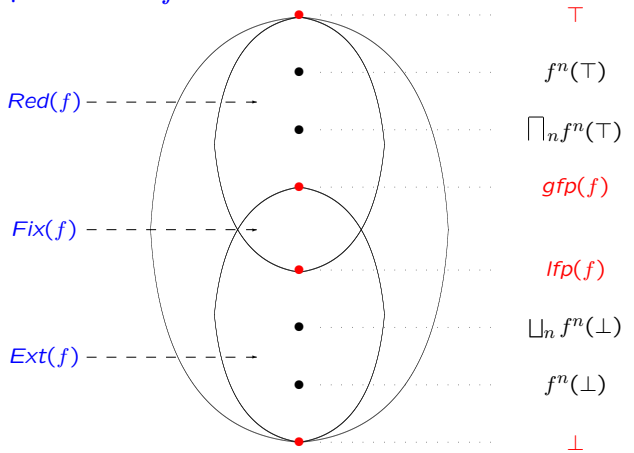
$$gfp(f) = \bigsqcup Fix(f) = \bigsqcup Ext(f) \in Fix(f) \subseteq Ext(f)$$

Fixed points of f

$Fix(f) = \{l \mid f(l) = l\}$

$Red(f) = \{l \mid f(l) \sqsubseteq l\}$

$Ext(f) = \{l \mid f(l) \sqsupseteq l\}$



Widening Operators

Problem: We cannot guarantee that $(f^n(\bot))_n$ eventually stabilises nor that its least upper bound necessarily equals $lfp(f)$.

Idea: We replace $(f^n(\bot))_n$ by a new sequence $(f^n_\nabla)_n$ that is known to eventually stabilise and to do so with a value that is a safe (upper) approximation of the least fixed point.

The new sequence is parameterised on the widening operator ∇: an upper bound operator satisfying a finiteness condition.

Upper bound operators

$\triangledown : L \times L \to L$ is an *upper bound operator* iff

$$l_1 \sqsubseteq l_1 \triangledown l_2 \sqsupseteq l_2$$

for all $l_1, l_2 \in L$.

Let $(l_n)_n$ be a sequence of elements of L. Define the sequence $(l_n^\triangledown)_n$ by:

$$l_n^\triangledown = \begin{cases} l_n & \text{if } n = 0 \\ l_{n-1}^\triangledown \triangledown l_n & \text{if } n > 0 \end{cases}$$

Fact: If $(l_n)_n$ is a sequence and $\triangledown$ is an upper bound operator then $(l_n^\triangledown)_n$ is an ascending chain; furthermore $l_n^\triangledown \sqsupseteq \bigsqcup\{l_0, l_1, \cdots, l_n\}$ for all n.

Example:

Let *int* be an arbitrary but fixed element of **Interval**.

An upper bound operator:

$$int_1 \,\breve{\sqcup}^{int}\, int_2 = \begin{cases} int_1 \sqcup int_2 & \text{if } int_1 \sqsubseteq int \;\vee\; int_2 \sqsubseteq int_1 \\ [-\infty, \infty] & \text{otherwise} \end{cases}$$

Example: $[1,2]\breve{\sqcup}^{[0,2]}[2,3] = [1,3]$ and $[2,3]\breve{\sqcup}^{[0,2]}[1,2] = [-\infty,\infty]$.

Transformation of:	$[0,0], [1,1], [2,2], [3,3], [4,4], [5,5], \cdots$
If $int = [0,\infty]$:	$[0,0], [0,1], [0,2], [0,3], [0,4], [0,5], \cdots$
If $int = [0,2]$:	$[0,0], [0,1], [0,2], [0,3], [-\infty,\infty], [-\infty,\infty], \cdots$

Widening operators

An operator $\nabla : L \times L \to L$ is a *widening operator* iff

- it is an upper bound operator, and
- for all ascending chains $(l_n)_n$ the ascending chain $(l_n^{\nabla})_n$ eventually stabilises.

Widening operators

Given a monotone function $f : L \to L$ and a widening operator ∇ define the sequence $(f_{\nabla}^n)_n$ by

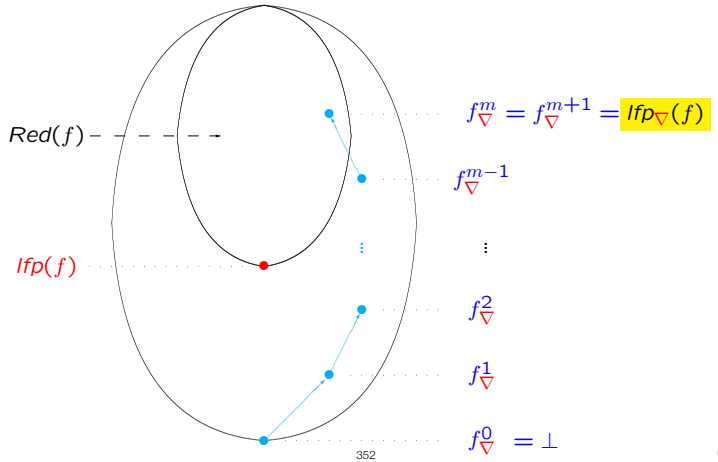
$$f_{\nabla}^n = \begin{cases} \bot & \text{if } n = 0 \\ f_{\nabla}^{n-1} & \text{if } n > 0 \;\wedge\; f(f_{\nabla}^{n-1}) \sqsubseteq f_{\nabla}^{n-1} \\ f_{\nabla}^{n-1} \;\nabla\; f(f_{\nabla}^{n-1}) & \text{otherwise} \end{cases}$$

One can show that:

- $(f_{\nabla}^n)_n$ is an ascending chain that eventually stabilises
- it happens when $f(f_{\nabla}^m) \sqsubseteq f_{\nabla}^m$ for some value of m
- Tarski's Theorem then gives $f_{\nabla}^m \sqsupseteq lfp(f)$

$lfp_{\nabla}(f) = f_{\nabla}^m$

The widening operator ∇ applied to f



Example:

Let K be a *finite* set of integers, e.g. the set of integers explicitly mentioned in a given program.

We shall define a widening operator ∇ based on K.

Idea: $[z_1, z_2] \nabla [z_3, z_4]$ is

$$[\ \mathrm{LB}(z_1, z_3)\ ,\ \mathrm{UB}(z_2, z_4)\ ]$$

where

- $\mathrm{LB}(z_1, z_3) \in \{z_1\} \cup K \cup \{-\infty\}$ is the best possible lower bound, and
- $\mathrm{UB}(z_2, z_4) \in \{z_2\} \cup K \cup \{\infty\}$ is the best possible upper bound.

The effect: a change in any of the bounds of the interval $[z_1, z_2]$ can only take place finitely many times – corresponding to the cardinality of K

Example (cont.) — formalisation:

Let $z_i \in \mathbf{Z}' = \mathbf{Z} \cup \{-\infty, \infty\}$ and write:

$$\mathrm{LB}_K(z_1, z_3) = \begin{cases} z_1 & \text{if } z_1 \leq z_3 \\ k & \text{if } z_3 < z_1 \ \wedge\ k = \max\{k \in K \mid k \leq z_3\} \\ -\infty & \text{if } z_3 < z_1 \ \wedge\ \forall k \in K : z_3 < k \end{cases}$$

$$\mathrm{UB}_K(z_2, z_4) = \begin{cases} z_2 & \text{if } z_4 \leq z_2 \\ k & \text{if } z_2 < z_4 \ \wedge\ k = \min\{k \in K \mid z_4 \leq k\} \\ \infty & \text{if } z_2 < z_4 \ \wedge\ \forall k \in K : k < z_4 \end{cases}$$

$$int_1 \nabla int_2 = \begin{cases} \bot & \text{if } int_1 = int_2 = \bot \\ [\ \mathrm{LB}_K(\inf(int_1), \inf(int_2))\ ,\ \mathrm{UB}_K(\sup(int_1), \sup(int_2))\ ] & \text{otherwise} \end{cases}$$

Example (cont.):

Consider the ascending chain $(int_n)_n$

$$[0,1], [0,2], [0,3], [0,4], [0,5], [0,6], [0,7], \cdots$$

and assume that $K = \{3, 5\}$.

Then $(int_n^{\nabla})_n$ is the chain

$$[0,1], [0,3], [0,3], [0,5], [0,5], [0,\infty], [0,\infty], \cdots$$

which eventually stabilises.

Narrowing Operators

Status: Widening gives us an upper approximation $lfp_{\nabla}(f)$ of the least fixed point of f.

Observation: $f(lfp_{\nabla}(f)) \sqsubseteq lfp_{\nabla}(f)$ so the approximation can be improved by considering the iterative sequence $(f^n(lfp_{\nabla}(f)))_n$.

It will satisfy $f^n(lfp_{\nabla}(f)) \sqsupseteq lfp(f)$ for all n so we can stop at an arbitrary point.

The notion of narrowing is *one way* of encapsulating a termination criterion for the sequence.

Narrowing

An operator $\triangle : L \times L \to L$ is a *narrowing operator* iff

- $l_2 \sqsubseteq l_1 \;\Rightarrow\; l_2 \sqsubseteq (l_1 \triangle l_2) \sqsubseteq l_1$ for all $l_1, l_2 \in L$, and
- for all descending chains $(l_n)_n$ the sequence $(l_n^{\triangle})_n$ eventually stabilises.

Recall: The sequence $(l_n^{\triangle})_n$ is defined by:

$$l_n^{\triangle} = \begin{cases} l_n & \text{if } n = 0 \\ l_{n-1}^{\triangle} \triangle l_n & \text{if } n > 0 \end{cases}$$

Narrowing

We construct the sequence $([f]^n_{\triangle})_n$

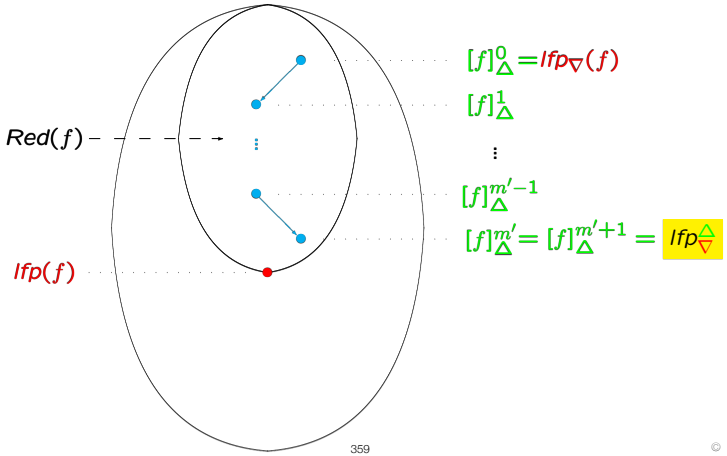
$$[f]^n_{\triangle} = \begin{cases} lfp_{\nabla}(f) & \text{if } n = 0 \\ [f]^{n-1}_{\triangle} \triangle f([f]^{n-1}_{\triangle}) & \text{if } n > 0 \end{cases}$$

One can show that:

- $([f]^n_{\triangle})_n$ is a descending chain where all elements satisfy $lfp(f) \sqsubseteq [f]^n_{\triangle}$
- the chain eventually stabilises so $[f]^{m'}_{\triangle} = [f]^{m'+1}_{\triangle}$ for some value m'

$lfp^{\triangle}_{\nabla}(f) = [f]^{m'}_{\triangle}$

The narrowing operator $\triangle$ applied to f



Example:

The complete lattice (**Interval**, $\sqsubseteq$) has two kinds of infinite descending chains:

- those with elements of the form $[-\infty, z]$, $z \in \mathbf{Z}$
- those with elements of the form $[z, \infty]$, $z \in \mathbf{Z}$

Idea: Given some fixed non-negative number N
the narrowing operator $\triangle_N$ will force an infinite descending chain

$$[z_1, \infty], [z_2, \infty], [z_3, \infty], \cdots$$

(where $z_1 < z_2 < z_3 < \cdots$) to stabilise when $z_i > N$

Similarly, for a descending chain with elements of the form $[-\infty, z_i]$ the narrowing operator will force it to stabilise when $z_i < -N$

Example (cont.) — formalisation:

Define $\triangle = \triangle_N$ by

$$int_1 \triangle int_2 = \begin{cases} \bot & \text{if } int_1 = \bot \vee int_2 = \bot \\ [z_1, z_2] & \text{otherwise} \end{cases}$$

where

$$z_1 = \begin{cases} \inf(int_1) & \text{if } N < \inf(int_2) \wedge \sup(int_2) = \infty \\ \inf(int_2) & \text{otherwise} \end{cases}$$

$$z_2 = \begin{cases} \sup(int_1) & \text{if } \inf(int_2) = -\infty \wedge \sup(int_2) < -N \\ \sup(int_2) & \text{otherwise} \end{cases}$$

Example (cont.):

Consider the infinite descending chain $([n, \infty])_n$

$$[0, \infty], [1, \infty], [2, \infty], [3, \infty], [4, \infty], [5, \infty], \cdots$$

and assume that $N = 3$.

Then the narrowing operator $\triangle_N$ will give the sequence $([n, \infty]^{\triangle})_n$

$$[0, \infty], [1, \infty], [2, \infty], [3, \infty], [3, \infty], [3, \infty], \cdots$$

Galois Connections

- Galois connections and adjunctions
- Extraction functions
- Galois insertions
- Reduction operators

Galois connections

$$L \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} M$$

α: *abstraction function*

γ: *concretisation function*

is a Galois connection if and only if

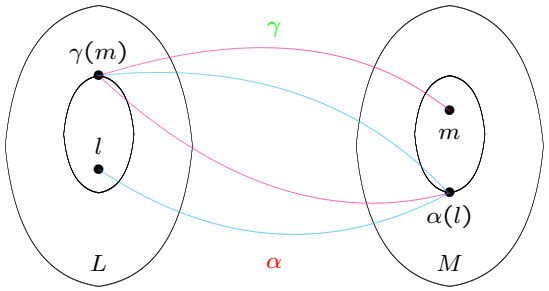
α and γ are monotone functions

that satisfy

$$\gamma \circ \alpha \sqsupseteq \lambda l.l$$

$$\alpha \circ \gamma \sqsubseteq \lambda m.m$$

Galois connections



$\gamma \circ \alpha \sqsupseteq \lambda l.l$ $\quad$ $\alpha \circ \gamma \sqsubseteq \lambda m.m$

Example:

Galois connection

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\mathrm{ZI}}, \gamma_{\mathrm{ZI}}, \textbf{Interval})$$

with concretisation function

$$\gamma_{\mathrm{ZI}}(int) = \{z \in \mathbf{Z} \mid \mathrm{inf}(int) \leq z \leq \mathrm{sup}(int)\}$$

and abstraction function

$$\alpha_{\mathrm{ZI}}(Z) = \begin{cases} \bot & \text{if } Z = \emptyset \\ [\mathrm{inf}'(Z), \mathrm{sup}'(Z)] & \text{otherwise} \end{cases}$$

Examples:

$$\begin{array}{rcl} \gamma_{\mathrm{ZI}}([0,3]) & = & \{0,1,2,3\} \\ \gamma_{\mathrm{ZI}}([0,\infty]) & = & \{z \in \mathbf{Z} \mid z \geq 0\} \\ \alpha_{\mathrm{ZI}}(\{0,1,3\}) & = & [0,3] \\ \alpha_{\mathrm{ZI}}(\{2*z \mid z > 0\}) & = & [2,\infty] \end{array}$$

Galois connections from representation functions

A representation function $\beta : V \to L$ gives rise to a Galois connection

$$(\mathcal{P}(V), \alpha, \gamma, L)$$

where

$$\begin{array}{rcl} \alpha(V') & = & \bigsqcup\{\beta(v) \mid v \in V'\} \\ \gamma(l) & = & \{v \in V \mid \beta(v) \sqsubseteq l\} \end{array}$$

for $V' \subseteq V$ and $l \in L$.

Galois connections from extraction functions

An *extraction function*

$$\eta : V \to D$$

maps the values of V to their best descriptions in D.

It gives rise to a representation function $\beta_\eta : V \to \mathcal{P}(D)$ (corresponding to $L = (\mathcal{P}(D), \subseteq)$) defined by

$$\beta_\eta(v) = \{\eta(v)\}$$

The associated Galois connection is

$$(\mathcal{P}(V), \alpha_\eta, \gamma_\eta, \mathcal{P}(D))$$

where

$$\begin{array}{rcl} \alpha_\eta(V') & = & \bigcup\{\beta_\eta(v) \mid v \in V'\} \quad = \quad \{\eta(v) \mid v \in V'\} \\ \gamma_\eta(D') & = & \{v \in V \mid \beta_\eta(v) \subseteq D'\} \quad = \quad \{v \mid \eta(v) \in D'\} \end{array}$$

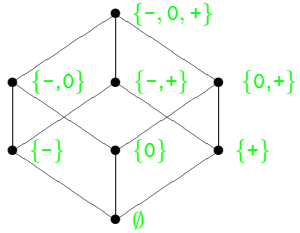
Example:

Extraction function

$$\text{sign} : \mathbf{Z} \to \mathbf{Sign}$$

specified by

$$\text{sign}(z) = \begin{cases} - & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ + & \text{if } z > 0 \end{cases}$$



Galois connection

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\text{sign}}, \gamma_{\text{sign}}, \mathcal{P}(\mathbf{Sign}))$$

with

$$\begin{aligned} \alpha_{\text{sign}}(Z) &= \{\text{sign}(z) \mid z \in Z\} \\ \gamma_{\text{sign}}(S) &= \{z \in \mathbf{Z} \mid \text{sign}(z) \in S\} \end{aligned}$$

Properties of Galois Connections

Lemma: If (L, α, γ, M) is a Galois connection then:

- α uniquely determines γ by $\gamma(m) = \bigsqcup\{l \mid \alpha(l) \sqsubseteq m\}$
- γ uniquely determines α by $\alpha(l) = \bigsqcap\{m \mid l \sqsubseteq \gamma(m)\}$
- α is completely additive and γ is completely multiplicative

In particular $\alpha(\bot) = \bot$ and $\gamma(\top) = \top$.

Lemma:

- If $\alpha : L \to M$ is completely additive then there exists (an upper adjoint) $\gamma : M \to L$ such that (L, α, γ, M) is a Galois connection.
- If $\gamma : M \to L$ is completely multiplicative then there exists (a lower adjoint) $\alpha : L \to M$ such that (L, α, γ, M) is a Galois connection.

Fact: If (L, α, γ, M) is a Galois connection then

- $\alpha \circ \gamma \circ \alpha = \alpha$ and $\gamma \circ \alpha \circ \gamma = \gamma$

Example:

Define $\gamma_{\text{IS}} : \mathcal{P}(\mathbf{Sign}) \to \mathbf{Interval}$ by:

$$\begin{aligned} \gamma_{\text{IS}}(\{-,0,+\}) &= [-\infty, \infty] & \qquad \gamma_{\text{IS}}(\{-,0\}) &= [-\infty, 0] \\ \gamma_{\text{IS}}(\{-,+\}) &= [-\infty, \infty] & \qquad \gamma_{\text{IS}}(\{0,+\}) &= [0, \infty] \\ \gamma_{\text{IS}}(\{-\}) &= [-\infty, -1] & \qquad \gamma_{\text{IS}}(\{0\}) &= [0, 0] \\ \gamma_{\text{IS}}(\{+\}) &= [1, \infty] & \qquad \gamma_{\text{IS}}(\emptyset) &= \bot \end{aligned}$$

Does there exist an abstraction function

$$\alpha_{\text{IS}} : \mathbf{Interval} \to \mathcal{P}(\mathbf{Sign})$$

such that $(\mathbf{Interval}, \alpha_{\text{IS}}, \gamma_{\text{IS}}, \mathcal{P}(\mathbf{Sign}))$ is a Galois connection?

Example (cont.):

Is γ_{IS} completely multiplicative?

- if yes: then there exists a Galois connection
- if no: then there cannot exist a Galois connection

Lemma: If L and M are complete lattices and M is finite then $\gamma : M \to L$ is completely multiplicative if and only if the following hold:

- $\gamma : M \to L$ is monotone,
- $\gamma(\top) = \top$, and
- $\gamma(m_1 \sqcap m_2) = \gamma(m_1) \sqcap \gamma(m_2)$ whenever $m_1 \not\sqsubseteq m_2 \wedge m_2 \not\sqsubseteq m_1$

We calculate

$$\begin{aligned} \gamma_{\text{IS}}(\{-,0\} \cap \{-,+\}) &= \gamma_{\text{IS}}(\{-\}) &&= [-\infty, -1] \\ \gamma_{\text{IS}}(\{-,0\}) \sqcap \gamma_{\text{IS}}(\{-,+\}) &= [-\infty, 0] \sqcap [-\infty, \infty] &&= [-\infty, 0] \end{aligned}$$

showing that there is no Galois connection involving γ_{IS}.

Galois Connections are the Right Concept

We use the mundane approach to correctness to demonstrate this for:

- Admissible correctness relations
- Representation functions

The mundane approach: correctness relations

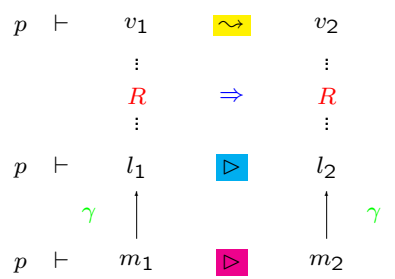
Assume

- $R : V \times L \to \{true, false\}$ is an admissible correctness relation
- (L, α, γ, M) is a Galois connection

Then $S : V \times M \to \{true, false\}$ defined by

$$v \; S \; m \quad \underline{\text{iff}} \quad v \; R \; (\gamma(m))$$

is an admissible correctness relation between V and M



The mundane approach: representation functions

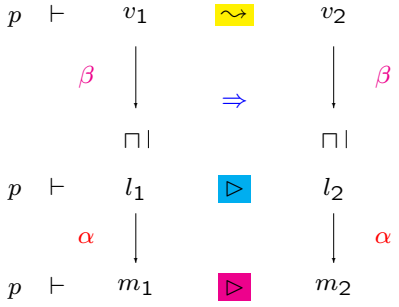
Assume

- $R : V \times L \to \{true, false\}$ is *generated by* $\beta : V \to L$
- (L, α, γ, M) is a Galois connection

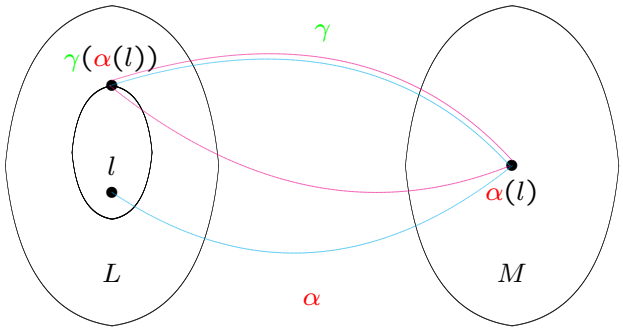
Then $S : V \times M \to \{true, false\}$ defined by

$$v \; S \; m \quad \underline{\text{iff}} \quad v \; R \; (\gamma(m))$$

is *generated by* $\alpha \circ \beta : V \to M$



Galois Insertions



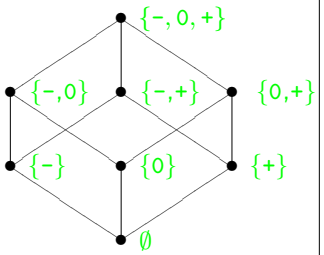
Monotone functions satisfying: $\gamma \circ \alpha \sqsupseteq \lambda l.l$ $\alpha \circ \gamma = \lambda m.m$

Example (1):

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\text{sign}}, \gamma_{\text{sign}}, \mathcal{P}(\mathbf{Sign}))$$

where sign : $\mathbf{Z} \rightarrow \mathbf{Sign}$ is specified by:

$$\text{sign}(z) = \begin{cases} - & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ + & \text{if } z > 0 \end{cases}$$



Is it a Galois insertion?

Example (2):

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\text{signparity}}, \gamma_{\text{signparity}}, \mathcal{P}(\mathbf{Sign} \times \mathbf{Parity}))$$

where $\mathbf{Sign} = \{-, 0, +\}$ and $\mathbf{Parity} = \{\text{odd}, \text{even}\}$

and signparity : $\mathbf{Z} \rightarrow \mathbf{Sign} \times \mathbf{Parity}$:

$$\text{signparity}(z) = \begin{cases} (\text{sign}(z), \text{odd}) & \text{if } z \text{ is odd} \\ (\text{sign}(z), \text{even}) & \text{if } z \text{ is even} \end{cases}$$

Is it a Galois insertion?

Example (1) reconsidered:

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\text{sign}}, \gamma_{\text{sign}}, \mathcal{P}(\mathbf{Sign}))$$

$$\text{sign}(z) = \begin{cases} - & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ + & \text{if } z > 0 \end{cases}$$

is a Galois insertion because sign is surjective.

Example (2) reconsidered:

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\text{signparity}}, \gamma_{\text{signparity}}, \mathcal{P}(\mathbf{Sign} \times \mathbf{Parity}))$$

$$\text{signparity}(z) = \begin{cases} (\text{sign}(z), \text{odd}) & \text{if } z \text{ is odd} \\ (\text{sign}(z), \text{even}) & \text{if } z \text{ is even} \end{cases}$$

is not a Galois insertion because signparity is not surjective.

Properties of Galois Insertions

Lemma: For a Galois connection (L, α, γ, M) the following claims are equivalent:

(i) (L, α, γ, M) is a Galois insertion;

(ii) α is surjective: $\forall m \in M : \exists l \in L : \alpha(l) = m$;

(iii) γ is injective: $\forall m_1, m_2 \in M : \gamma(m_1) = \gamma(m_2) \Rightarrow m_1 = m_2$; and

(iv) γ is an order-similarity: $\forall m_1, m_2 \in M : \gamma(m_1) \sqsubseteq \gamma(m_2) \Leftrightarrow m_1 \sqsubseteq m_2$.

Corollary: A Galois connection specified by an *extraction* function $\eta : V \rightarrow D$ is a Galois insertion if and only if η is surjective.

Reduction Operators

Given a Galois connection (L, α, γ, M) it is always possible to obtain a Galois insertion by enforcing that the concretisation function γ is injective.

Idea: remove the superfluous elements from M using a *reduction operator*

$$\varsigma : M \to M$$

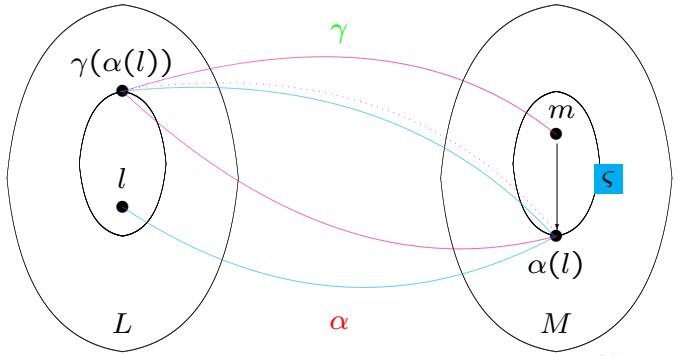
defined from the Galois connection.

Proposition: Let (L, α, γ, M) be a Galois connection and define the reduction operator $\varsigma : M \to M$ by

$$\varsigma(m) = \bigsqcap \{m' \mid \gamma(m) = \gamma(m')\}$$

Then $\varsigma[M] = (\{\varsigma(m) \mid m \in M\}, \sqsubseteq_M)$ is a complete lattice and $(L, \alpha, \gamma, \varsigma[M])$ is a Galois insertion.

The reduction operator $\varsigma : M \to M$



Reduction operators from extraction functions

Assume that the Galois connection $(\mathcal{P}(V), \alpha_\eta, \gamma_\eta, \mathcal{P}(D))$ is given by an extraction function $\eta : V \to D$.

Then the reduction operator ς_η is given by

$$\varsigma_\eta(D') = D' \cap \eta[V]$$

where $\eta[V] = \{d \in D \mid \exists v \in V : \eta(v) = d\}$.

Since $\varsigma_\eta[\mathcal{P}(D)]$ is isomorphic to $\mathcal{P}(\eta[V])$ the resulting Galois insertion is isomorphic to

$$(\mathcal{P}(V), \alpha_\eta, \gamma_\eta, \mathcal{P}(\eta[V]))$$

Systematic Design of Galois Connections

The "functional composition" (or "sequential composition") of two Galois connections is also a Galois connection:

$$L_0 \underset{\alpha_1}{\overset{\gamma_1}{\leftrightarrows}} L_1 \underset{\alpha_2}{\overset{\gamma_2}{\leftrightarrows}} L_2 \underset{\alpha_3}{\overset{\gamma_3}{\leftrightarrows}} \cdots \underset{\alpha_k}{\overset{\gamma_k}{\leftrightarrows}} L_k$$

A catalogue of techniques for combining Galois connections:

- independent attribute method
- direct product
- reduced product
- total function space
- relational method
- direct tensor product
- reduced tensor product
- monotone function space

Running Example: Array Bound Analysis

Approximation of the difference in magnitude between two numbers (typically the index and the bound):

- a Galois connection for approximating pairs (z_1, z_2) of integers by their difference $|z_1| - |z_2|$
- a Galois connection for approximating integers using a finite lattice $\{\texttt{<-1}, \texttt{-1}, \texttt{0}, \texttt{+1}, \texttt{>+1}\}$
- a Galois connection for their functional composition

Example: Difference in Magnitude

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{\text{diff}}, \gamma_{\text{diff}}, \mathcal{P}(\mathbf{Z}))$$

where the extraction function $\text{diff} : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$ calculates the difference in magnitude:

$$\text{diff}(z_1, z_2) = |z_1| - |z_2|$$

The abstraction and concretisation functions are

$$\begin{aligned} \alpha_{\text{diff}}(ZZ) &= \{|z_1| - |z_2| \mid (z_1, z_2) \in ZZ\} \\ \gamma_{\text{diff}}(Z) &= \{(z_1, z_2) \mid |z_1| - |z_2| \in Z\} \end{aligned}$$

for $ZZ \subseteq \mathbf{Z} \times \mathbf{Z}$ and $Z \subseteq \mathbf{Z}$.

Example: Finite Approximation

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\text{range}}, \gamma_{\text{range}}, \mathcal{P}(\textbf{Range}))$$

where $\textbf{Range} = \{\texttt{<-1}, \texttt{-1}, \texttt{0}, \texttt{+1}, \texttt{>+1}\}$
and the extraction function $\text{range} : \mathbf{Z} \to \textbf{Range}$ is

$$\text{range}(z) = \begin{cases} \texttt{<-1} & \text{if } z < -1 \\ \texttt{-1} & \text{if } z = -1 \\ \texttt{0} & \text{if } z = 0 \\ \texttt{+1} & \text{if } z = 1 \\ \texttt{>+1} & \text{if } z > 1 \end{cases}$$

The abstraction and concretisation functions are

$$\begin{aligned} \alpha_{\text{range}}(Z) &= \{\text{range}(z) \mid z \in Z\} \\ \gamma_{\text{range}}(R) &= \{z \mid \text{range}(z) \in R\} \end{aligned}$$

for $Z \subseteq \mathbf{Z}$ and $R \subseteq \textbf{Range}$.

Example: Functional Composition

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{\text{R}}, \gamma_{\text{R}}, \mathcal{P}(\textbf{Range}))$$

where

$$\begin{aligned} \alpha_{\text{R}} &= \alpha_{\text{range}} \circ \alpha_{\text{diff}} \\ \gamma_{\text{R}} &= \gamma_{\text{diff}} \circ \gamma_{\text{range}} \end{aligned}$$

The explicit formulae for the abstraction and concretisation functions

$$\begin{aligned} \alpha_{\text{R}}(ZZ) &= \{\text{range}(|z_1| - |z_2|) \mid (z_1, z_2) \in ZZ\} \\ \gamma_{\text{R}}(R) &= \{(z_1, z_2) \mid \text{range}(|z_1| - |z_2|) \in R\} \end{aligned}$$

correspond to the extraction function $\text{range} \circ \text{diff}$.

Approximation of Pairs

Independent Attribute Method

Let $(L_1, \alpha_1, \gamma_1, M_1)$ and $(L_2, \alpha_2, \gamma_2, M_2)$ be Galois connections.

The *independent attribute method* gives a Galois connection

$$(L_1 \times L_2, \alpha, \gamma, M_1 \times M_2)$$

where

$$\begin{aligned} \alpha(l_1, l_2) &= (\alpha_1(l_1), \alpha_2(l_2)) \\ \gamma(m_1, m_2) &= (\gamma_1(m_1), \gamma_2(m_2)) \end{aligned}$$

Example: Detection of Signs Analysis

Given

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\mathsf{sign}}, \gamma_{\mathsf{sign}}, \mathcal{P}(\mathbf{Sign}))$$

using the extraction function sign.

The independent attribute method gives

$$(\mathcal{P}(\mathbf{Z}) \times \mathcal{P}(\mathbf{Z}), \alpha_{\mathsf{SS}}, \gamma_{\mathsf{SS}}, \mathcal{P}(\mathbf{Sign}) \times \mathcal{P}(\mathbf{Sign}))$$

where

$$\begin{aligned} \alpha_{\mathsf{SS}}(Z_1, Z_2) &= (\{\mathsf{sign}(z) \mid z \in Z_1\}, \{\mathsf{sign}(z) \mid z \in Z_2\}) \\ \gamma_{\mathsf{SS}}(S_1, S_2) &= (\{z \mid \mathsf{sign}(z) \in S_1\}, \{z \mid \mathsf{sign}(z) \in S_2\}) \end{aligned}$$

Motivating the Relational Method

The independent attribute method often leads to imprecision!

Semantics: The expression `(x,-x)` may have a value in

$$\{(z, -z) \mid z \in \mathbf{Z}\}$$

Analysis: When we use $\mathcal{P}(\mathbf{Z}) \times \mathcal{P}(\mathbf{Z})$ to represent sets of pairs of integers we cannot do better than representing $\{(z, -z) \mid z \in \mathbf{Z}\}$ by

$$(\mathbf{Z}, \mathbf{Z})$$

Hence the best property describing it will be

$$\alpha_{\mathsf{SS}}(\mathbf{Z}, \mathbf{Z}) = (\{\mathtt{-}, \mathtt{0}, \mathtt{+}\}, \{\mathtt{-}, \mathtt{0}, \mathtt{+}\})$$

Relational Method

Let $(\mathcal{P}(V_1), \alpha_1, \gamma_1, \mathcal{P}(D_1))$ and $(\mathcal{P}(V_2), \alpha_2, \gamma_2, \mathcal{P}(D_2))$ be Galois connections.

The *relational method* will give rise to the Galois connection

$$(\mathcal{P}(V_1 \times V_2), \alpha, \gamma, \mathcal{P}(D_1 \times D_2))$$

where

$$\begin{aligned} \alpha(VV) &= \bigcup\{\alpha_1(\{v_1\}) \times \alpha_2(\{v_2\}) \mid (v_1, v_2) \in VV\} \\ \gamma(DD) &= \{(v_1, v_2) \mid \alpha_1(\{v_1\}) \times \alpha_2(\{v_2\}) \subseteq DD\} \end{aligned}$$

Generalisation to arbitrary complete lattices: use *tensor products*.

Relational Method from Extraction Functions

Assume that the Galois connections $(\mathcal{P}(V_i), \alpha_i, \gamma_i, \mathcal{P}(D_i))$ are given by *extraction functions* $\eta_i : V_i \to D_i$ as in

$$\alpha_i(V_i') = \{\eta_i(v_i) \mid v_i \in V_i'\}$$
$$\gamma_i(D_i') = \{v_i \mid \eta_i(v_i) \in D_i'\}$$

Then the Galois connection $(\mathcal{P}(V_1 \times V_2), \alpha, \gamma, \mathcal{P}(D_1 \times D_2))$ has

$$\alpha(VV) = \{(\eta_1(v_1), \eta_2(v_2)) \mid (v_1, v_2) \in VV\}$$
$$\gamma(DD) = \{(v_1, v_2) \mid (\eta_1(v_1), \eta_2(v_2)) \in DD\}$$

which also can be obtained directly from the extraction function $\eta : V_1 \times V_2 \to D_1 \times D_2$ defined by

$$\eta(v_1, v_2) = (\eta_1(v_1), \eta_2(v_2))$$

Example: Detection of Signs Analysis

Using the relational method we get a Galois connection

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{SS'}, \gamma_{SS'}, \mathcal{P}(\mathbf{Sign} \times \mathbf{Sign}))$$

where

$$\alpha_{SS'}(ZZ) = \{(\mathsf{sign}(z_1), \mathsf{sign}(z_2)) \mid (z_1, z_2) \in ZZ\}$$
$$\gamma_{SS'}(SS) = \{(z_1, z_2) \mid (\mathsf{sign}(z_1), \mathsf{sign}(z_2)) \in SS\}$$

corresponding to an extraction function twosigns : $\mathbf{Z} \times \mathbf{Z} \to \mathbf{Sign} \times \mathbf{Sign}$ defined by

$$\mathsf{twosigns}(z_1, z_2) = (\mathsf{sign}(z_1), \mathsf{sign}(z_2))$$

Advantages of the Relational Method

Semantics: The expression `(x,-x)` may have a value in

$$\{(z, -z) \mid z \in \mathbf{Z}\}$$

In the present setting $\{(z, -z) \mid z \in \mathbf{Z}\}$ is an element of $\mathcal{P}(\mathbf{Z} \times \mathbf{Z})$.

Analysis: The best "relational" property describing it is

$$\alpha_{SS'}(\{(z, -z) \mid z \in \mathbf{Z}\}) = \{(-,+), (0,0), (+,-)\}$$

whereas the best "independent attribute" property was

$$\alpha_{SS}(\mathbf{Z}, \mathbf{Z}) = (\{-,0,+\}, \{-,0,+\})$$

Function Spaces

Total Function Space

Let (L, α, γ, M) be a Galois connection and let S be a set.

The Galois connection for the *total function space*

$$(S \to L, \alpha', \gamma', S \to M)$$

is defined by

$$\alpha'(f) = \alpha \circ f \qquad \gamma'(g) = \gamma \circ g$$

Monotone Function Space

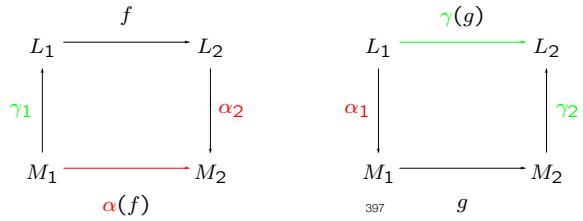
Let $(L_1, \alpha_1, \gamma_1, M_1)$ and $(L_2, \alpha_2, \gamma_2, M_2)$ be Galois connections.

The Galois connection for the *monotone function space*

$$(L_1 \to L_2, \alpha, \gamma, M_1 \to M_2)$$

is defined by

$$\alpha(f) = \alpha_2 \circ f \circ \gamma_1 \qquad \gamma(g) = \gamma_2 \circ g \circ \alpha_1$$



Performing Analyses Simultaneously

Direct Product

Let $(L, \alpha_1, \gamma_1, M_1)$ and $(L, \alpha_2, \gamma_2, M_2)$ be Galois connections.

The *direct product* is the Galois connection

$$(L, \alpha, \gamma, M_1 \times M_2)$$

defined by

$$\begin{aligned} \alpha(l) &= (\alpha_1(l), \alpha_2(l)) \\ \gamma(m_1, m_2) &= \gamma_1(m_1) \sqcap \gamma_2(m_2) \end{aligned}$$

Example:

Combining the detection of signs analysis for pairs of integers with the analysis of difference in magnitude.

We get the Galois connection

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{\mathsf{SSR}}, \gamma_{\mathsf{SSR}}, \mathcal{P}(\mathbf{Sign} \times \mathbf{Sign}) \times \mathcal{P}(\mathbf{Range}))$$

where

$$\begin{aligned} \alpha_{\mathsf{SSR}}(ZZ) &= (\{(\mathsf{sign}(z_1), \mathsf{sign}(z_2)) \mid (z_1, z_2) \in ZZ\}, \\ &\quad \{\mathsf{range}(|z_1| - |z_2|) \mid (z_1, z_2) \in ZZ\}) \\ \gamma_{\mathsf{SSR}}(SS, R) &= \{(z_1, z_2) \mid (\mathsf{sign}(z_1), \mathsf{sign}(z_2)) \in SS\} \\ &\quad \cap \ \{(z_1, z_2) \mid \mathsf{range}(|z_1| - |z_2|) \in R\} \end{aligned}$$

Motivating the Direct Tensor Product

The expression `(x, 3*x)` may have a value in

$$\{(z, 3 * z) \mid z \in \mathbf{Z}\}$$

which is described by

$$\alpha_{\mathsf{SSR}}(\{(z, 3 * z) \mid z \in \mathbf{Z}\}) = (\{(-,-), (0,0), (+,+)\}, \{0, <-1\})$$

But

- any pair described by (0,0) will have a difference in magnitude described by 0
- any pair described by (-,-) or (+,+) will have a difference in magnitude described by <-1

and the analysis cannot express this.

Direct Tensor Product

Let $(\mathcal{P}(V), \alpha_1, \gamma_1, \mathcal{P}(D_1))$ and $(\mathcal{P}(V), \alpha_2, \gamma_2, \mathcal{P}(D_2))$ be Galois connections.

The *direct tensor product* is the Galois connection

$$(\mathcal{P}(V), \alpha, \gamma, \mathcal{P}(D_1 \times D_2))$$

defined by

$$\begin{aligned} \alpha(V') &= \bigcup\{\alpha_1(\{v\}) \times \alpha_2(\{v\}) \mid v \in V'\} \\ \gamma(DD) &= \{v \mid \alpha_1(\{v\}) \times \alpha_2(\{v\}) \subseteq DD\} \end{aligned}$$

Direct Tensor Product from Extraction Functions

Assume that the Galois connections $(\mathcal{P}(V), \alpha_i, \gamma_i, \mathcal{P}(D_i))$ are given by *extraction functions* $\eta_i : V \to D_i$ as in

$$\begin{aligned} \alpha_i(V') &= \{\eta_i(v) \mid v \in V'\} \\ \gamma_i(D_i') &= \{v \mid \eta_i(v) \in D_i'\} \end{aligned}$$

The Galois connection $(\mathcal{P}(V), \alpha, \gamma, \mathcal{P}(D_1 \times D_2))$ has

$$\begin{aligned} \alpha(V') &= \{(\eta_1(v), \eta_2(v)) \mid v \in V'\} \\ \gamma(DD) &= \{v \mid (\eta_1(v), \eta_2(v)) \in DD\} \end{aligned}$$

corresponding to the extraction function $\eta : V \to D_1 \times D_2$ defined by

$$\eta(v) = (\eta_1(v), \eta_2(v))$$

Example:

Using the direct tensor product to combine the detection of signs analysis for pairs of integers with the analysis of difference in magnitude.

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{SSR'}, \gamma_{SSR'}, \mathcal{P}(\mathbf{Sign} \times \mathbf{Sign} \times \mathbf{Range}))$$

is given by

$$\begin{aligned} \alpha_{SSR'}(ZZ) &= \{(\mathsf{sign}(z_1), \mathsf{sign}(z_2), \mathsf{range}(|z_1| - |z_2|)) \mid (z_1, z_2) \in ZZ\} \\ \gamma_{SSR'}(SSR) &= \{(z_1, z_2) \mid (\mathsf{sign}(z_1), \mathsf{sign}(z_2), \mathsf{range}(|z_1| - |z_2|)) \in SSR\} \end{aligned}$$

corresponding to $\mathsf{twosignsrange} : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Sign} \times \mathbf{Sign} \times \mathbf{Range}$ given by

$$\mathsf{twosignsrange}(z_1, z_2) = (\mathsf{sign}(z_1), \mathsf{sign}(z_2), \mathsf{range}(|z_1| - |z_2|))$$

Advantages of the Direct Tensor Product

The expression `(x,3*x)` may have a value in $\{(z, 3*z) \mid z \in \mathbf{Z}\}$ which in the direct tensor product can be described by

$$\alpha_{SSR'}(\{(z, 3*z) \mid z \in \mathbf{Z}\}) = \{(-,-,\texttt{<-1}), (0,0,0), (+,+,\texttt{<-1})\}$$

compared to the direct product that gave

$$\alpha_{SSR}(\{(z, 3*z) \mid z \in \mathbf{Z}\}) = (\{(-,-), (0,0), (+,+)\}, \{0, \texttt{<-1}\})$$

Note that the Galois connection is *not* a Galois insertion because

$$\gamma_{SSR'}(\emptyset) = \emptyset = \gamma_{SSR'}(\{(0,0,\texttt{<-1})\})$$

so $\gamma_{SSR'}$ is not injective and hence we do not have a Galois insertion.

From Direct to Reduced

Reduced Product

Let $(L, \alpha_1, \gamma_1, M_1)$ and $(L, \alpha_2, \gamma_2, M_2)$ be Galois connections.

The *reduced product* is the Galois *insertion*

$$(L, \alpha, \gamma, \varsigma[M_1 \times M_2])$$

defined by

$$\begin{aligned} \alpha(l) &= (\alpha_1(l), \alpha_2(l)) \\ \gamma(m_1, m_2) &= \gamma_1(m_1) \sqcap \gamma_2(m_2) \\ \varsigma(m_1, m_2) &= \bigsqcap\{(m_1', m_2') \mid \gamma_1(m_1) \sqcap \gamma_2(m_2) = \gamma_1(m_1') \sqcap \gamma_2(m_2')\} \end{aligned}$$

Reduced Tensor Product

Let $(\mathcal{P}(V), \alpha_1, \gamma_1, \mathcal{P}(D_1))$ and $(\mathcal{P}(V), \alpha_2, \gamma_2, \mathcal{P}(D_2))$ be Galois connection.

The *reduced tensor product* is the Galois *insertion*

$$(\mathcal{P}(V), \alpha, \gamma, \varsigma[\mathcal{P}(D_1 \times D_2)])$$

defined by

$$\begin{aligned} \alpha(V') &= \bigcup\{\alpha_1(\{v\}) \times \alpha_2(\{v\}) \mid v \in V'\} \\ \gamma(DD) &= \{v \mid \alpha_1(\{v\}) \times \alpha_2(\{v\}) \subseteq DD\} \\ \varsigma(DD) &= \bigcap\{DD' \mid \gamma(DD) = \gamma(DD')\} \end{aligned}$$

Example: Array Bounds Analysis

The superfluous elements of $\mathcal{P}(\mathbf{Sign} \times \mathbf{Sign} \times \mathbf{Range})$ will be removed when we use a reduced tensor product:

The reduction operator $\varsigma_{SSR'}$ amounts to

$$\varsigma_{SSR'}(SSR) = \bigcap\{SSR' \mid \gamma_{SSR'}(SSR) = \gamma_{SSR'}(SSR')\}$$

where $SSR, SSR' \subseteq \mathbf{Sign} \times \mathbf{Sign} \times \mathbf{Range}$.

The singleton sets constructed from the following 16 elements

$$\begin{array}{llll} (-,0,<\!-1), & (-,0,-1), & (-,0,0), & \\ (0,-,0), & (0,-,+1), & (0,-,>\!+1), & \\ (0,0,<\!-1), & (0,0,-1), & (0,0,+1), & (0,0,>\!+1), \\ (0,+,0), & (0,+,+1), & (0,+,>\!+1), & \\ (+,0,<\!-1), & (+,0,-1), & (+,0,0) & \end{array}$$

will be mapped to the empty set (as they are useless).

Example (cont.): Array Bounds Analysis

The remaining 29 elements of $\mathbf{Sign} \times \mathbf{Sign} \times \mathbf{Range}$ are

$$\begin{array}{lllll} (-,-,<\!-1), & (-,-,-1), & (-,-,0), & (-,-,+1), & (-,-,>\!+1), \\ (-,0,+1), & (-,0,>\!+1), & & & \\ (-,+,<\!-1), & (-,+,-1), & (-,+,0), & (-,+,+1), & (-,+,>\!+1), \\ (0,-,<\!-1), & (0,-,-1), & (0,0,0), & (0,+,<\!-1), & (0,+,-1), \\ (+,-,<\!-1), & (+,-,-1), & (+,-,0), & (+,-,+1), & (+,-,>\!+1), \\ (+,0,+1), & (+,0,>\!+1), & & & \\ (+,+,<\!-1), & (+,+,-1), & (+,+,0), & (+,+,+1), & (+,+,>\!+1) \end{array}$$

and they describe disjoint subsets of $\mathbf{Z} \times \mathbf{Z}$.

Any collection of properties can be descibed in 4 bytes.

Summary

The Array Bound Analysis has been designed from three simple Galois connections specified by extraction functions:

(i) an analysis approximating integers by their sign,

(ii) an analysis approximating pairs of integers by their difference in magnitude, and

(iii) an analysis approximating integers by their closeness to 0, 1 and -1.

These analyses have been combined using:

(iv) the relational product of analysis (i) with itself,

(v) the functional composition of analyses (ii) and (iii), and

(vi) the reduced tensor product of analyses (iv) and (v).

Induced Operations

Given: Galois connections $(L_i, \alpha_i, \gamma_i, M_i)$ so that M_i is more approximate than (i.e. is coarser than) L_i.

Aim: Replace an existing analysis over L_i with an analysis making use of the coarser structure of M_i.

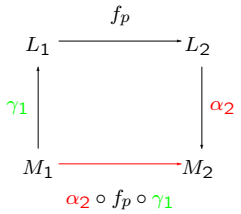
Methods:

- Inducing along the abstraction function: move the computations from L_i to M_i.
- Application to Data Flow Analysis.
- Inducing along the concretisation function: move a widening from M_i to L_i.

Inducing along the Abstraction Function

Given Galois connections $(L_i, \alpha_i, \gamma_i, M_i)$ so that M_i is more approximate than L_i.

Replace an existing analysis $f_p : L_1 \to L_2$ with a new and more approximate analysis $g_p : M_1 \to M_2$: take $g_p = \alpha_2 \circ f_p \circ \gamma_1$.



The analysis $\alpha_2 \circ f_p \circ \gamma_1$ is *induced* from f_p and the Galois connections.

Example:

A very precise analysis for `plus` based on $\mathcal{P}(\mathbf{Z})$ and $\mathcal{P}(\mathbf{Z} \times \mathbf{Z})$:

$$f_{\mathsf{plus}}(ZZ) = \{z_1 + z_2 \mid (z_1, z_2) \in ZZ\}$$

Two Galois connections

$$(\mathcal{P}(\mathbf{Z}), \alpha_{\mathsf{sign}}, \gamma_{\mathsf{sign}}, \mathcal{P}(\mathbf{Sign}))$$

$$(\mathcal{P}(\mathbf{Z} \times \mathbf{Z}), \alpha_{\mathsf{SS'}}, \gamma_{\mathsf{SS'}}, \mathcal{P}(\mathbf{Sign} \times \mathbf{Sign}))$$

An approximate analysis for `plus` based on $\mathcal{P}(\mathbf{Sign})$ and $\mathcal{P}(\mathbf{Sign} \times \mathbf{Sign})$:

$$g_{\mathsf{plus}} = \alpha_{\mathsf{sign}} \circ f_{\mathsf{plus}} \circ \gamma_{\mathsf{SS'}}$$

Example (cont.):

We calculate

$$\begin{aligned} g_{\mathsf{plus}}(SS) &= \alpha_{\mathsf{sign}}(f_{\mathsf{plus}}(\gamma_{SS'}(SS))) \\ &= \alpha_{\mathsf{sign}}(f_{\mathsf{plus}}(\{(z_1,z_2) \in \mathbf{Z}\times\mathbf{Z} \mid (\mathsf{sign}(z_1),\mathsf{sign}(z_2)) \in SS\})) \\ &= \alpha_{\mathsf{sign}}(\{z_1+z_2 \mid z_1,z_2 \in \mathbf{Z}, (\mathsf{sign}(z_1),\mathsf{sign}(z_2)) \in SS\}) \\ &= \{\mathsf{sign}(z_1+z_2) \mid z_1,z_2 \in \mathbf{Z}, (\mathsf{sign}(z_1),\mathsf{sign}(z_2)) \in SS\} \\ &= \bigcup\{s_1 \oplus s_2 \mid (s_1,s_2) \in SS\} \end{aligned}$$

where $\oplus : \mathbf{Sign}\times\mathbf{Sign} \to \mathcal{P}(\mathbf{Sign})$ is the "addition" operator on signs (so e.g. $+\oplus+ = \{+\}$ and $+\oplus- = \{-,0,+\}$).

The Mundane Correctness of f_p carries over to g_p

The correctness relation R_i for V_i and L_i:

$$R_i : V_i \times L_i \to \{\mathit{true}, \mathit{false}\} \quad \text{is } \textit{generated by } \beta_i : V_i \to L_i$$

Correctness of f_p means

$$(p \vdash \cdot \rightsquigarrow \cdot)\,(R_1 \twoheadrightarrow R_2)\,f_p$$

(with $R_1 \twoheadrightarrow R_2$ being generated by $\beta_1 \twoheadrightarrow \beta_2$).

The correctness relation S_i for V_i and M_i:

$$S_i : V_i \times M_i \to \{\mathit{true}, \mathit{false}\} \text{ is } \textit{generated by } \alpha_i \circ \beta_i : V_i \to M_i$$

One can prove that

$$\begin{aligned} &(p \vdash \cdot \rightsquigarrow \cdot)\,(R_1 \twoheadrightarrow R_2)\,f_p \;\wedge\; \alpha_2 \circ f_p \circ \gamma_1 \sqsubseteq g_p \\ &\quad\Rightarrow\; (p \vdash \cdot \rightsquigarrow \cdot)\,(S_1 \twoheadrightarrow S_2)\,g_p \end{aligned}$$

with $S_1 \twoheadrightarrow S_2$ being generated by $(\alpha_1 \circ \beta_1) \twoheadrightarrow (\alpha_2 \circ \beta_2)$.

Fixed Points in the Induced Analysis

Let $f_p = \mathit{lfp}(F)$ for a monotone function $F : (L_1 \to L_2) \to (L_1 \to L_2)$.

The Galois connections $(L_i, \alpha_i, \gamma_i, M_i)$ give rise to a Galois connection $(L_1 \to L_2, \alpha, \gamma, M_1 \to M_2)$.

Take $g_p = \mathit{lfp}(G)$ where $G : (M_1 \to M_2) \to (M_1 \to M_2)$ is an "upper approximation" to F: we demand that $\alpha \circ F \circ \gamma \sqsubseteq G$.

Then for all $m \in M_1 \to M_2$:

$$G(m) \sqsubseteq m \Rightarrow F(\gamma(m)) \sqsubseteq \gamma(m)$$

and $\mathit{lfp}(F) \sqsubseteq \gamma(\mathit{lfp}(G))$ and $\alpha(\mathit{lfp}(F)) \sqsubseteq \mathit{lfp}(G)$

Application to Data Flow Analysis

A *generalised Monotone Framework* consists of:

- the property space: a complete lattice $L = (L, \sqsubseteq)$;
- the set $\mathcal{F}$ of monotone functions from L to L.

An *instance* A of a generalised Monotone Framework consists of:

- a finite flow, $F \subseteq \mathbf{Lab}\times\mathbf{Lab}$;
- a finite set of extremal labels, $E \subseteq \mathbf{Lab}$;
- an extremal value, $\iota \in L$; and
- a mapping $f_\cdot$ from the labels $\mathbf{Lab}$ of F and E to monotone transfer functions from L to L.

Application to Data Flow Analysis

Let (L, α, γ, M) be a Galois connection.

Consider an instance B of the generalised Monotone Framework M that satisfies

- the mapping $g_\cdot$ from the labels $\mathbf{Lab}$ of F and E to monotone transfer functions of $M \to M$ satisfies $g_\ell \sqsupseteq \alpha \circ f_\ell \circ \gamma$ for all ℓ; and
- the extremal value $\jmath$ satisfies $\gamma(\jmath) = \iota$;

and otherwise B is as A.

One can show that a solution to the B-constraints gives rise to a solution to the A-constraints:

$$(B_\circ, B_\bullet) \models \mathsf{B}^{\sqsupseteq} \quad \text{implies} \quad (\gamma \circ B_\circ, \gamma \circ B_\bullet) \models \mathsf{A}^{\sqsupseteq}$$

The Mundane Approach to Semantic Correctness

Here $F = \mathit{flow}(S_\star)$ and $E = \{\mathit{init}(S_\star)\}$.

Correctness of every solution to $\mathsf{A}^{\sqsupseteq}$ amounts to:

Assume $(A_\circ, A_\bullet) \models \mathsf{A}^{\sqsupseteq}$ and $\langle S_\star, \sigma_1\rangle \to^* \sigma_2$.
Then $\beta(\sigma_1) \sqsubseteq \iota$ implies $\beta(\sigma_2) \sqsubseteq \bigsqcup\{A_\bullet(\ell) \mid \ell \in \mathit{final}(S_\star)\}$.

where $\beta : \mathbf{State} \to L$.

One can then prove the correctness result for B:

Assume $(B_\circ, B_\bullet) \models \mathsf{B}^{\sqsupseteq}$ and $\langle S_\star, \sigma_1\rangle \to^* \sigma_2$.
Then $(\alpha \circ \beta)(\sigma_1) \sqsubseteq \jmath$ implies $(\alpha \circ \beta)(\sigma_2) \sqsubseteq \bigsqcup\{B_\bullet(\ell) \mid \ell \in \mathit{final}(S_\star)\}$.

Sets of States Analysis

Generalised Monotone Framework over $(\mathcal{P}(\mathbf{State}), \subseteq)$.
Instance SS for $S_\star$:

- the flow F is $\mathit{flow}(S_\star)$;
- the set E of extremal labels is $\{\mathit{init}(S_\star)\}$;
- the extremal value ι is $\mathbf{State}$; and
- the transfer functions are given by $f^{\mathsf{SS}}_\cdot$:

$$\begin{array}{lll} [x := a]^\ell & f^{\mathsf{SS}}_\ell(\Sigma) & = \{\sigma[x \mapsto \mathcal{A}[\![a]\!]\sigma] \mid \sigma \in \Sigma\} \\ [\mathtt{skip}]^\ell & f^{\mathsf{SS}}_\ell(\Sigma) & = \Sigma \\ [b]^\ell & f^{\mathsf{SS}}_\ell(\Sigma) & = \Sigma \end{array}$$

where $\Sigma \subseteq \mathbf{State}$.

Correctness: Assume $(SS_\circ, SS_\bullet) \models \mathsf{SS}^{\sqsupseteq}$ and $\langle S_\star, \sigma_1\rangle \to^* \sigma_2$.
Then $\sigma_1 \in \mathbf{State}$ implies $\sigma_2 \in \bigcup\{SS_\bullet(\ell) \mid \ell \in \mathit{final}(S_\star)\}$.

Constant Propagation Analysis

Generalised Monotone Framework over $\widehat{\mathbf{State}}_{\mathsf{CP}} = ((\mathbf{Var} \to \mathbf{Z}^\top)_\perp, \sqsubseteq)$.
Instance CP for $S_\star$:

- the flow F is $\mathit{flow}(S_\star)$;
- the set E of extremal labels is $\{\mathit{init}(S_\star)\}$;
- the extremal value ι is $\lambda x.\top$; and
- the transfer functions are given by the mapping $f^{\mathsf{CP}}_\cdot$:

$$\begin{array}{lll} [x := a]^\ell : & f^{\mathsf{CP}}_\ell(\widehat{\sigma}) & = \begin{cases} \perp & \text{if } \widehat{\sigma} = \perp \\ \widehat{\sigma}[x \mapsto \mathcal{A}_{\mathsf{CP}}[\![a]\!]\widehat{\sigma}] & \text{otherwise} \end{cases} \\ [\mathtt{skip}]^\ell : & f^{\mathsf{CP}}_\ell(\widehat{\sigma}) & = \widehat{\sigma} \\ [b]^\ell : & f^{\mathsf{CP}}_\ell(\widehat{\sigma}) & = \widehat{\sigma} \end{array}$$

Galois Connection

The representation function $\beta_{\mathsf{CP}} : \mathbf{State} \to \widehat{\mathbf{State}}_{\mathsf{CP}}$ is defined by

$$\beta_{\mathsf{CP}}(\sigma) = \sigma$$

This gives rise to a Galois connection

$$(\mathcal{P}(\mathbf{State}), \alpha_{\mathsf{CP}}, \gamma_{\mathsf{CP}}, \widehat{\mathbf{State}}_{\mathsf{CP}})$$

where $\alpha_{\mathsf{CP}}(\Sigma) = \bigsqcup\{\beta_{\mathsf{CP}}(\sigma) \mid \sigma \in \Sigma\}$ and $\gamma_{\mathsf{CP}}(\hat{\sigma}) = \{\sigma \mid \beta_{\mathsf{CP}}(\sigma) \sqsubseteq \hat{\sigma}\}$.

One can show that for all labels ℓ

$f_\ell^{\mathsf{CP}} \sqsupseteq \alpha_{\mathsf{CP}} \circ f_\ell^{\mathsf{SS}} \circ \gamma_{\mathsf{CP}}$ as well as $\gamma_{\mathsf{CP}}(\lambda x.\top) = \mathbf{State}$

It follows that CP is an upper approximation to the analysis induced from SS and the Galois connection; therefore it is correct.

Inducing along the Concretisation Function

Given an upper bound operator

$$\nabla_M : M \times M \to M$$

and a Galois connection (L, α, γ, M).

Define an upper bound operator

$$\nabla_L : L \times L \to L$$

by

$$l_1 \ \nabla_L \ l_2 = \gamma(\ \alpha(l_1) \ \nabla_M \ \alpha(l_2)\ )$$

It defines a widening operator if one of the following conditions holds:

(i) M satisfies the Ascending Chain Condition, or

(ii) (L, α, γ, M) is a Galois insertion and $\nabla_M : M \times M \to M$ is a widening.

Precision of the Induced Widening Operator

Lemma: Let (L, α, γ, M) be a Galois insertion such that $\gamma(\bot_M) = \bot_L$ and let $\nabla_M : M \times M \to M$ be a widening operator.

Then the widening operator $\nabla_L : L \times L \to L$ defined by

$$l_1 \ \nabla_L \ l_2 = \gamma(\alpha(l_1) \ \nabla_M \ \alpha(l_2))$$

satisfies

$$\mathit{lfp}_{\nabla_L}(f) = \gamma(\mathit{lfp}_{\nabla_M}(\alpha \circ f \circ \gamma))$$

for all monotone functions $f : L \to L$.

Precision of the Induced Widening Operator

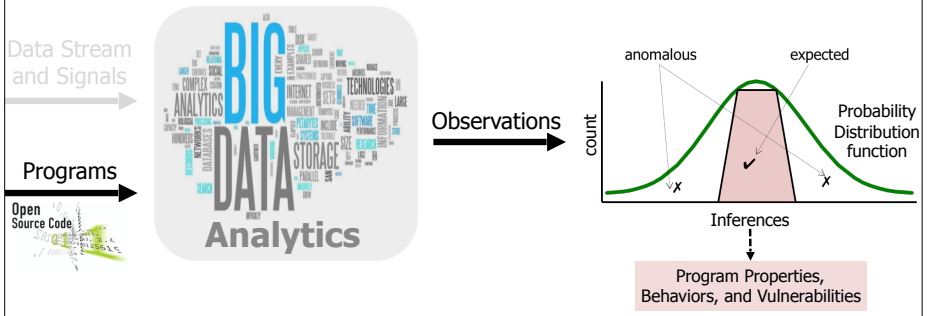
Corollary: Let M be of finite height, let (L, α, γ, M) be a Galois insertion (such that $\gamma(\bot_M) = \bot_L$), and let ∇_M equal the least upper bound operator $\sqcup_M$.

Then the above lemma shows that $\mathit{lfp}_{\nabla_L}(f) = \gamma(\mathit{lfp}(\alpha \circ f \circ \gamma))$.

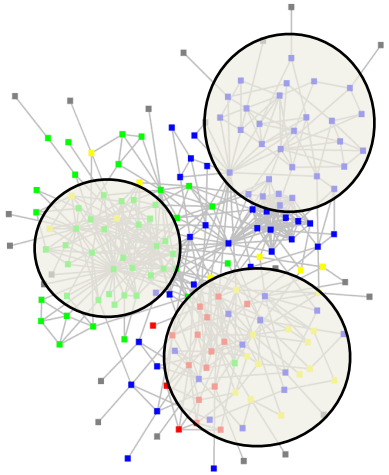
This means that $\mathit{lfp}_{\nabla_L}(f)$ *equals* the result we would have obtained if we decided to work with $\alpha \circ f \circ \gamma : M \to M$ instead of the given $f : L \to L$; furthermore the number of iterations needed turn out to be the same. However, for all other operations the increased precision of L is available.

Analyzing Big Code

Apply principles of Big Data Analytics to a large corpus



Treat programs (and, semantic objects extracted from programs) as data
Observations and inferences applied to program properties



Redundancies in the corpus exposed as dense components (enclaves) in the mined network

- Nodes represent properties facts, claims, and evidence
- Edges connect related properties

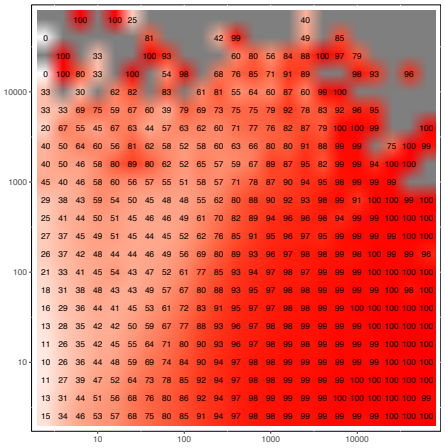
Anomalous properties have small number of connections

Likely invariants have large number of connections

https://muse-portal.net/publications

Duplication

		JavaScript
Counts	# projects (total)	4,479,173
	# projects (non-fork)	2,011,875
	# projects (downloaded)	1,778,679
	# projects (analyzed)	1,755,618
	# files (analyzed)	261,676,091



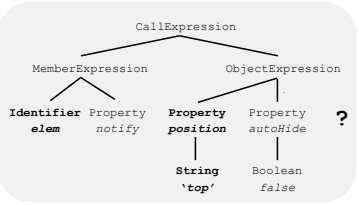
Lopes, Maj, Vitek, et al. *DéjàVu: A Map of Code Duplicates on GitHub.* **OOPSLA'17**

Probabilistic models of code

Program

```
elem.notify({
  position: 'top',
  autoHide: false,
  ?
});
```

AST



Probabilistic Model

P(element | S ())

S is a **learned semantic analyzer**

Can predict entire expressions at once, precisely

Raychev, Bielik, Vechev. *Probabilistic Model for Code with Decision Trees.* ACM OOPSLA 2016

Language Translation

JavaScript

CoffeeScript

Sounds easy, but very hard problem: 0 margin for error, **different than** NLP (where similar intent fine)

Easy to get data, learning **hard to get scale, with 0 mistakes!**

Deep Learning **does not work well**

Statistical Deobfuscation

Major challenge: fast inference with precision. Various works sacrifice one, making tool less usable in practice

Analyzer again learns **right few features** to use. Fast to extract, yet precise.

```
function FZ(e, t) {
var n = [];   var r =
e.length;   var i = 0;
for (; i < r; i += t)
if (i + t < r)
n.push(e.substring(i,
i + t)); else
n.push(e.substring(i,
r)); return n;
}
```

JSNice.org

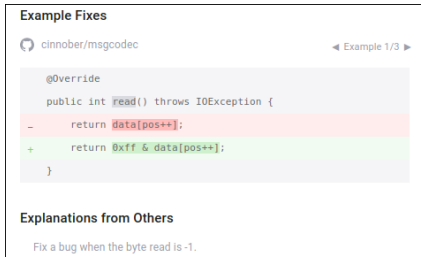
```
function chunkData(str: string,
step: number) {
 var colNames = [];
 var len = str.length;
 var i = 0;
 for (; i < len; i += step)
  if (i + step < len)

colNames.push(str.substring(i,i+
step));
  else

colNames.push(str.substring(i,
len));
  return colNames;
}
```

Raychev, Vechev. *Predicting Program Properties from "Big Code".* POPL 2015

Learn from thousands of commits



Predict bugs elsewhere
Make explainable predictions

Semantic analysis over large corpus of code and learning using **interpretable** model

Key for displaying **actionable** suggestions to users

Enable learning of new static analysis checks [PLDI'18]

Code Vectors

Symbolic trace

```
int example() {
  buf = alloc(12);
  if (buf != 0) {
    bar(buf);
    free(buf);
    return 0;
  } else {
    return -ENOMEM;
  }
}
```

Figure 1: An example procedure

```
call alloc(12);
assume alloc(12) != 0;
call bar(alloc(12));
call free(alloc(12));
return 0;
```

(a) Trace 1

```
call alloc(12);
assume alloc(12) == 0;
return -ENOMEM;
```

(b) Trace 2

Devise a parametric encoding of programs into word sequences that (i) can be tuned to capture different representation choices on the spectrum from (mainly) syntactic to (mainly) semantic, (ii) is amenable to word-vector-learning techniques, and (iii) can be obtained from programs efficiently.

There are too many words in a symbolic trace.

Local variables, user defined functions, literals and path conditions combine to create an extraordinarily large vocabulary.

Abstracted Symbolic Traces

```
lock(&obj->lock);
foo = alloc(12);
if (foo != 0) {
  obj->baz =
    bar(foo);

} else {
  unlock(
    &obj->lock);
  return -ENOMEM;

}
unlock(
  &obj->lock);
return 0;
```

```
Start
Called(lock)
Called(alloc)
RetEq(alloc, 0)
ParamShare(unlock, lock)
Called(unlock)
Error
RetError(ENOMEM)
End
```

Henkel, Liblit, Reps. *Code Vectors: Understanding Programs Through Embedded Abstracted Symbolic Traces.* FSE'18.