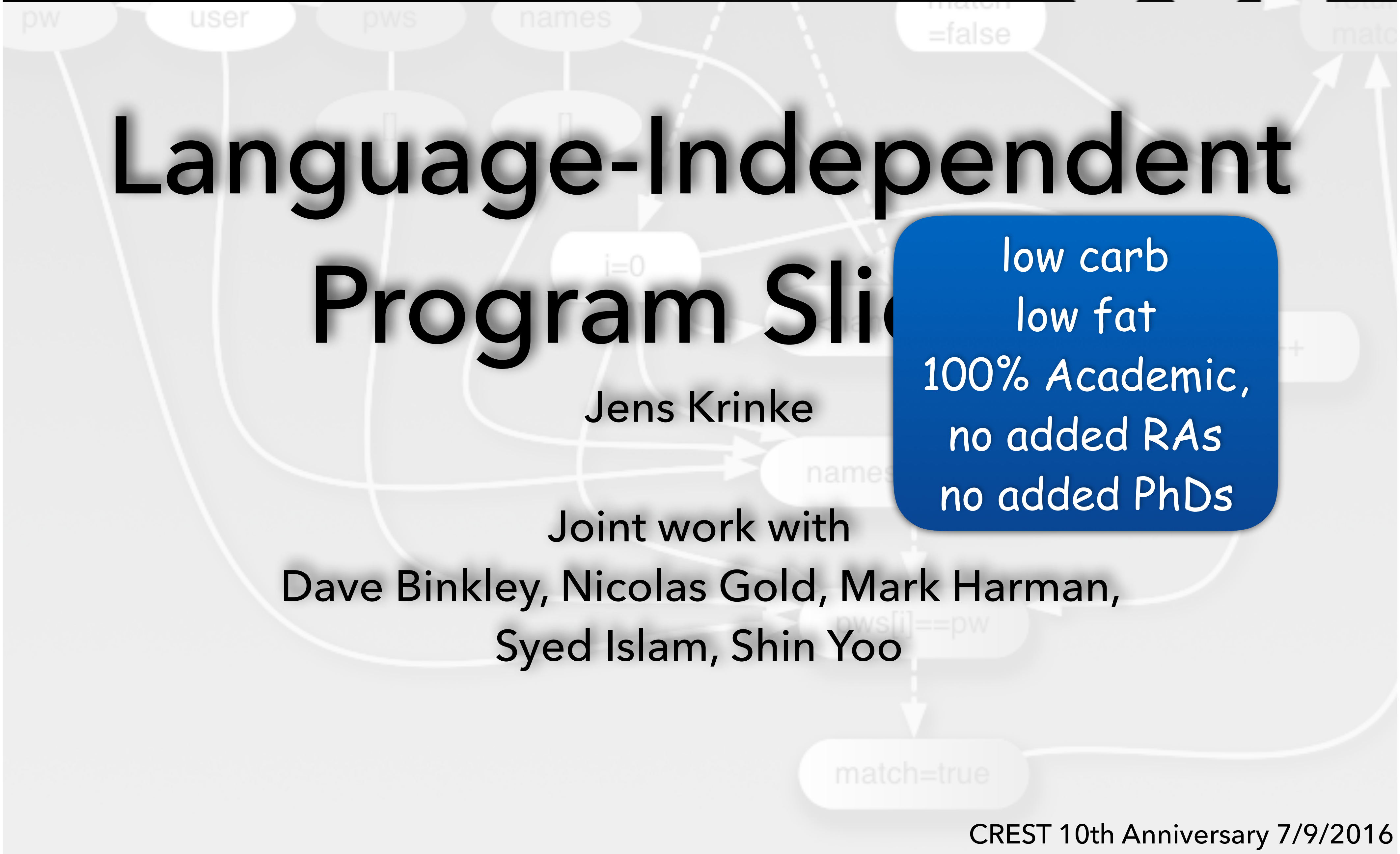


Language-Independent Program Sli

low carb
low fat
100% Academic,
no added RAs
no added PhDs

Jens Krinke

Joint work with
Dave Binkley, Nicolas Gold, Mark Harman,
Syed Islam, Shin Yoo

CREST 10th Anniversary 7/9/2016

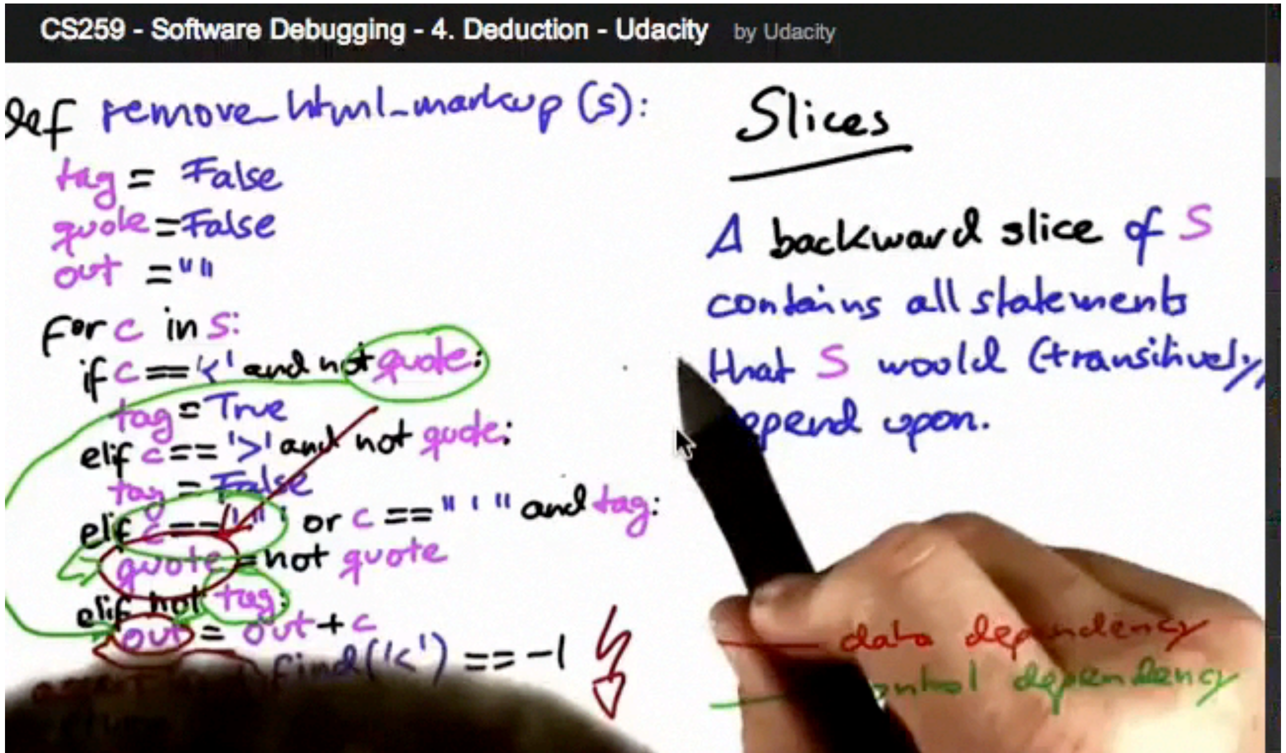
Dependence: Core of Software Engineering

- What are the modules that the system needs?
- What are the requirements this module implements?
- What is impacted by a change?
- On what input does this output depend?
- Where is the secret data flowing to?
- Can untrusted user input reach vulnerable functions?

Traversing Dependence: Program Slicing

What is dependent on a point of interest?

On what is a point of interest dependent?



CS259 - Software Debugging - 4. Deduction - Udacity by Udacity

```
ef remove_html_markup(s):
   tag = False
   quote=False
   out  =""
  for c in s:
    if c=='<' and not quote:
      tag = True
    elif c=='>' and not quote:
      tag = False
    elif c=='"' or c=="'" and tag:
      quote = not quote
    elif not tag:
      out = out + c
          find('<') == -1
```

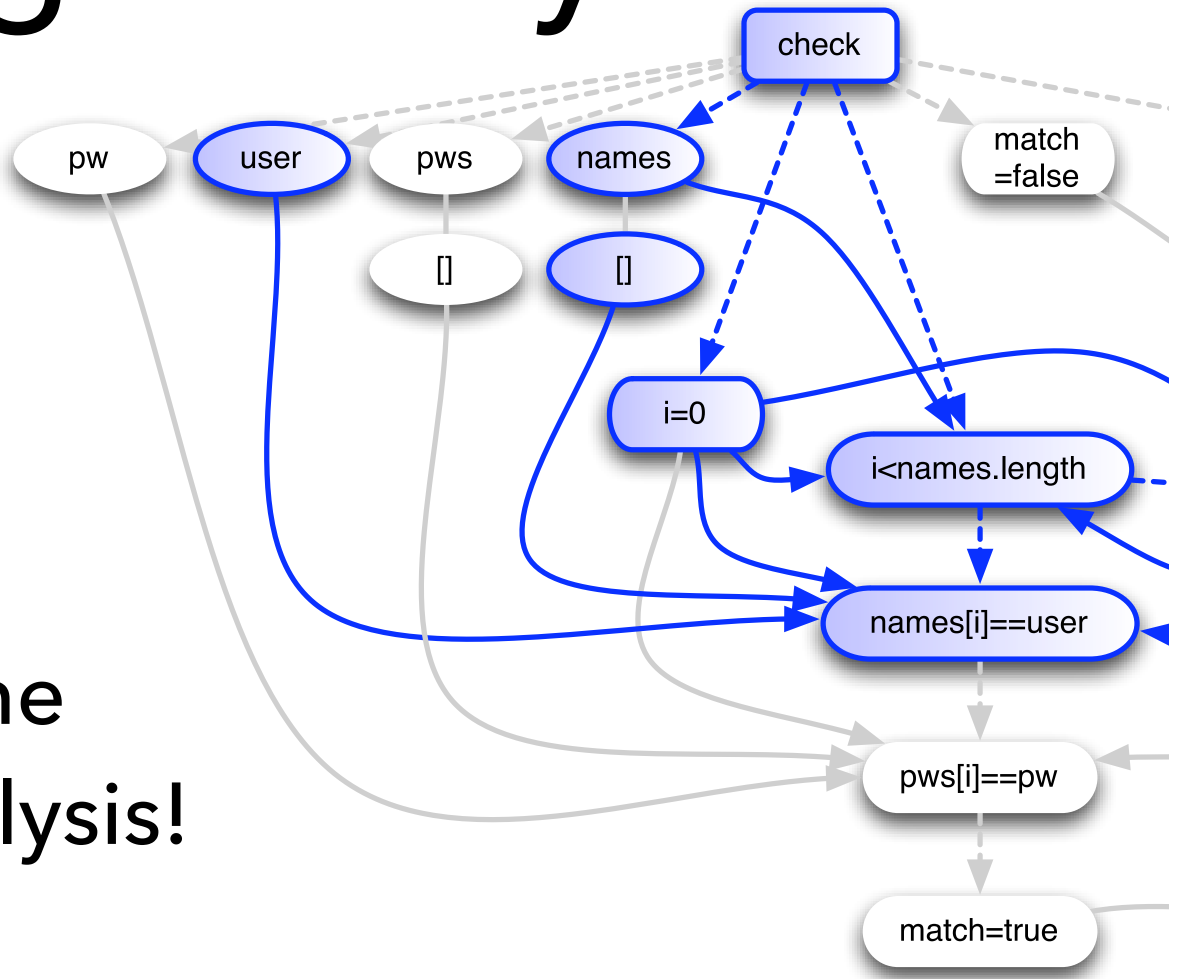
Slices

A backward slice of S contains all statements that S would (transitively epend upon.

data dependency
ontrol dependency

Slicing is easy.

- Slicing is just a traversal of dependences.
- The hard part is the Dependence Analysis!
- Not to mention the Pointer Analysis...



Dependence-Based Slicing

… must implement full semantics.

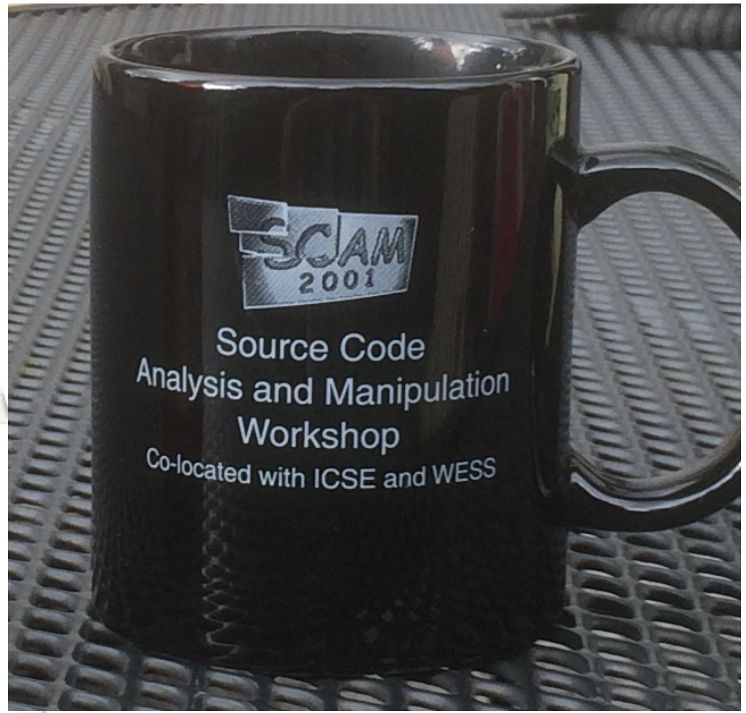
- Semantics are insufficient to capture dependencies arising through interaction.

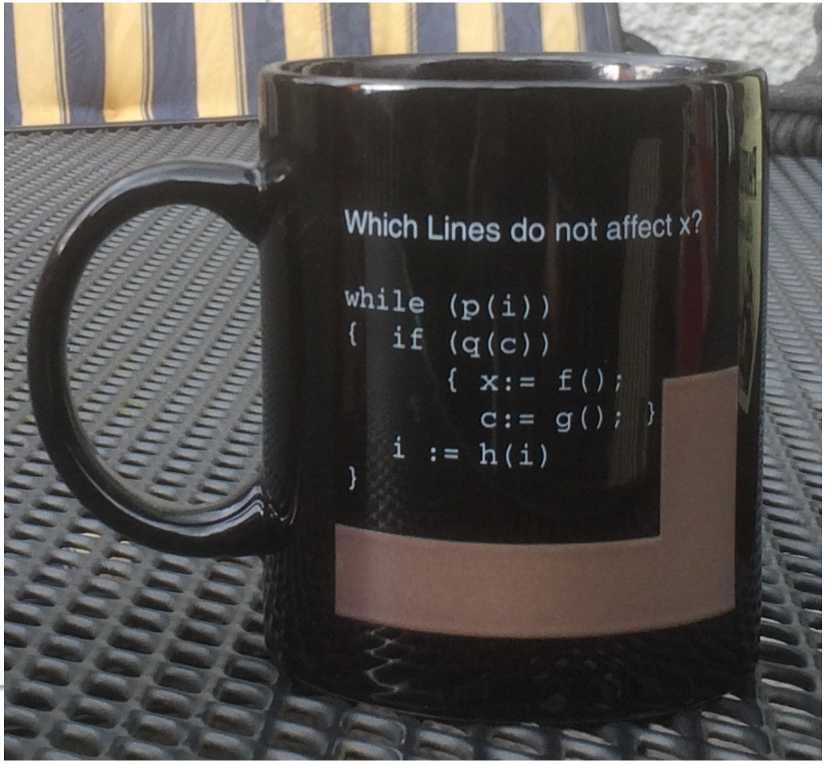
✗ Its is unlikely that a dependence-based slicer will ever be able to capture such dependencies.

SCAM 2001

Which Lines do not affect x?

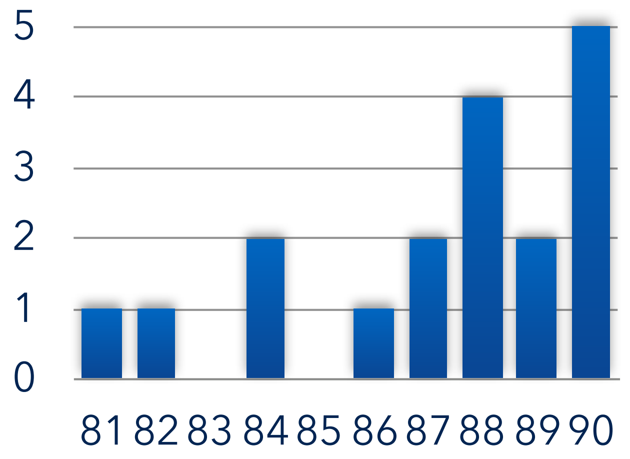
```c
int mug(int i, int c, int x)
{
  while (p(i))
  {
    if (q(c))
    {
      x = f();
      c = g();
    }
    i = h(i);
  }
  printf("@%d\n", x);
}
```


SCAM 2001
Source Code Analysis and Manipulation Workshop
Co-located with ICSE and WESS



```
while (p(i))
{  if (q(c))
      { x:= f();
        c:= g(); }
   i := h(i)
}
```

First 10 years



79, 81, 82, 84 – Mark Weiser's articles

84 – Slicing in Dependence Graphs

86 – Dicing

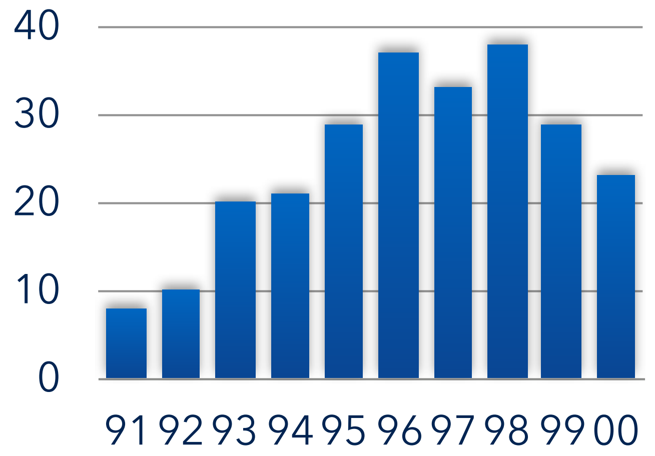
87 – Fault Localisation

88 – Dynamic Slicing

88 – Applications: Maintenance, Differencing

88 – Semantics

Busy 10 years



91 – Quasi-static slicing

92 – Testing

93 – Pointers

93 – Concurrency

93 – Specifications

93 – Functional Languages

93 – Function Extraction

94 – Chopping

94 – OOP

95 – Parametric Slicing

95 – Frank Tip's Survey

96 – Prolog

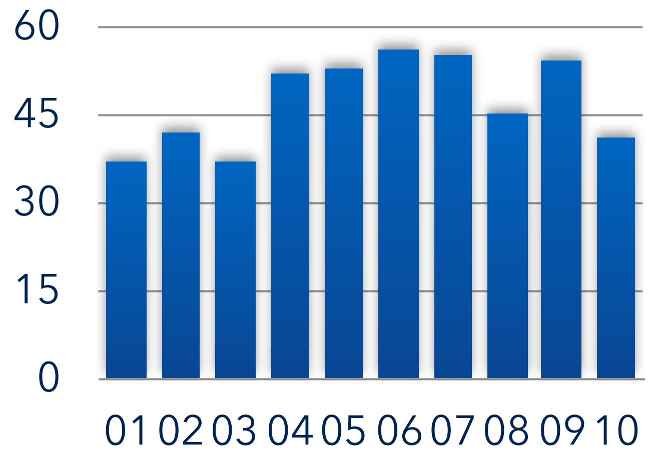
96 – VHDL

97 – Amorphous Slicing

98 – Conditioned Slicing

98 – State Machines

Stable 10 years



- Improvements in precision, efficiency, applications, usability, applicability, ...
- Empirical studies
- Tool(s): CodeSurfer and some prototypes (Kaveri, JSlice, Sprite, Unravel, Frama-c, WET, WALA, LLVM, Joana, JavaSlicer,...)

http://www.dagstuhl.de/05451

November 6 – 11 , 2005, Dagstuhl Seminar 05451

Beyond Program Slicing

Organizers

Dave Binkley (Loyola College – Baltimore, US)
Mark Harman (King's College London, GB)
Jens Krinke (FernUniversität in Hagen, DE)

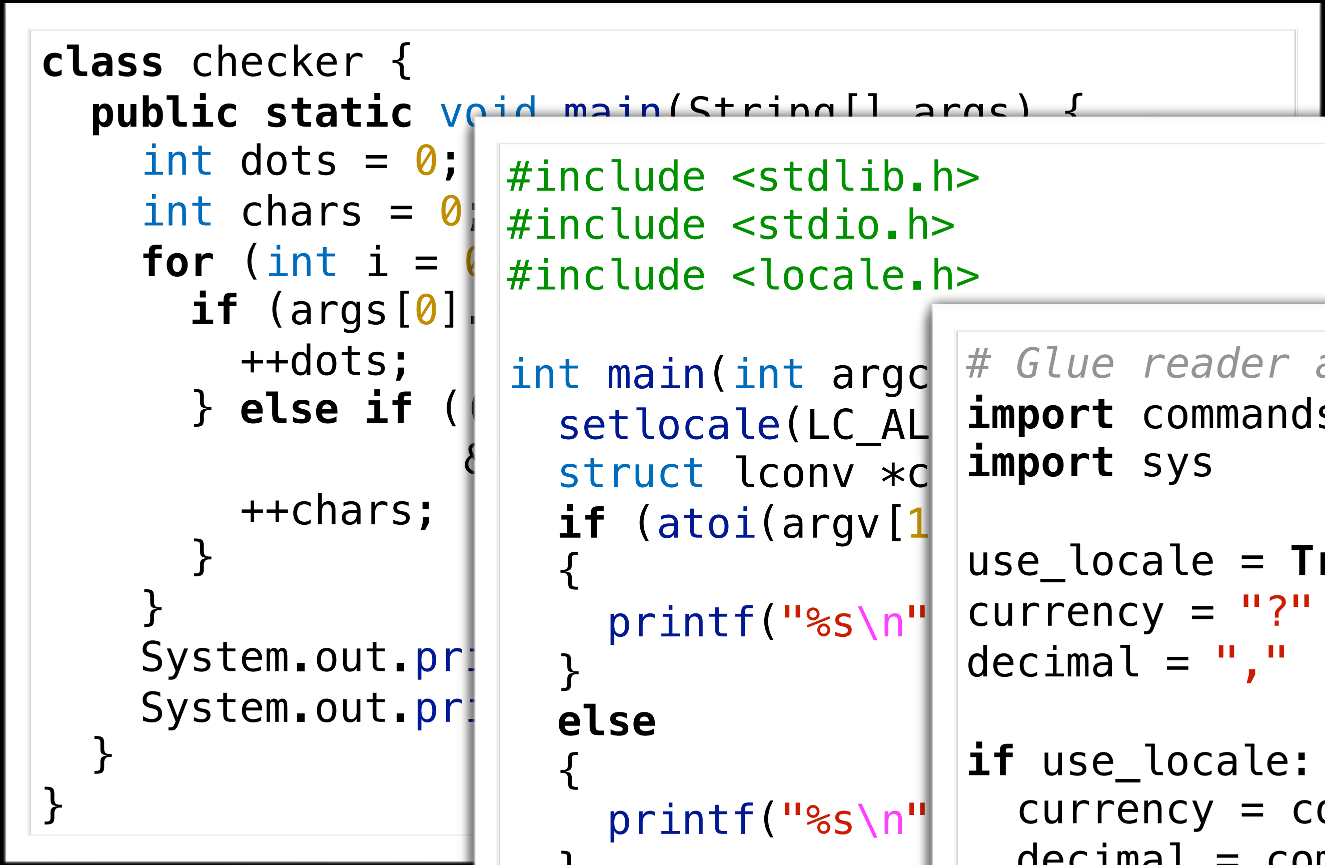
Program Slicing: Challenges

Almost no advances in the past 10 years!

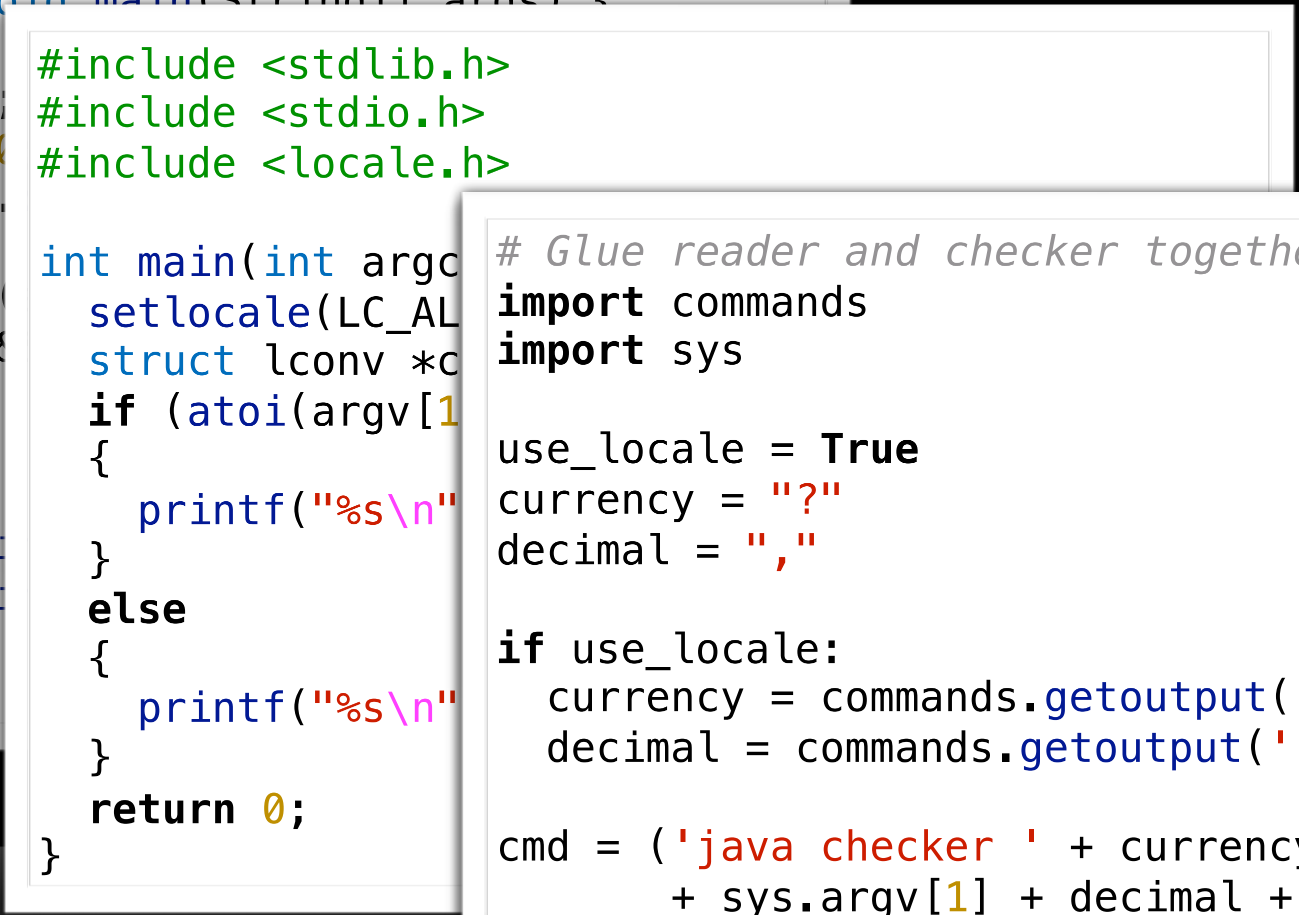
Tools cannot handle real world software:

- Exhaustive analyses are impossible, source code is not available or compilable.
- Systems programmed in various languages, including scripting and configurations.

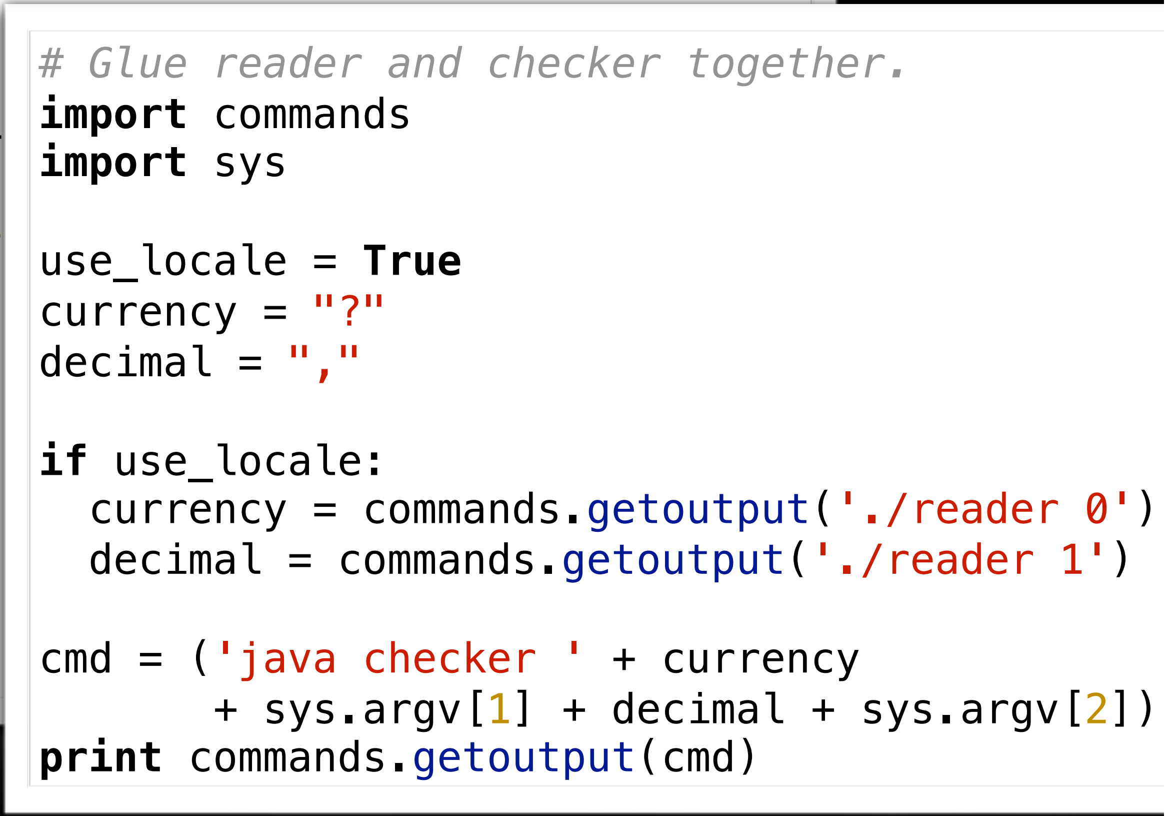
Who can slice this?



```
class checker {
  public static void main(String[] args) {
    int dots = 0;
    int chars = 0;
    for (int i = (
      if (args[0].
        ++dots;
      } else if ((
                 &
        ++chars;
      }
    }
    System.out.pr
    System.out.pr
  }
}
```

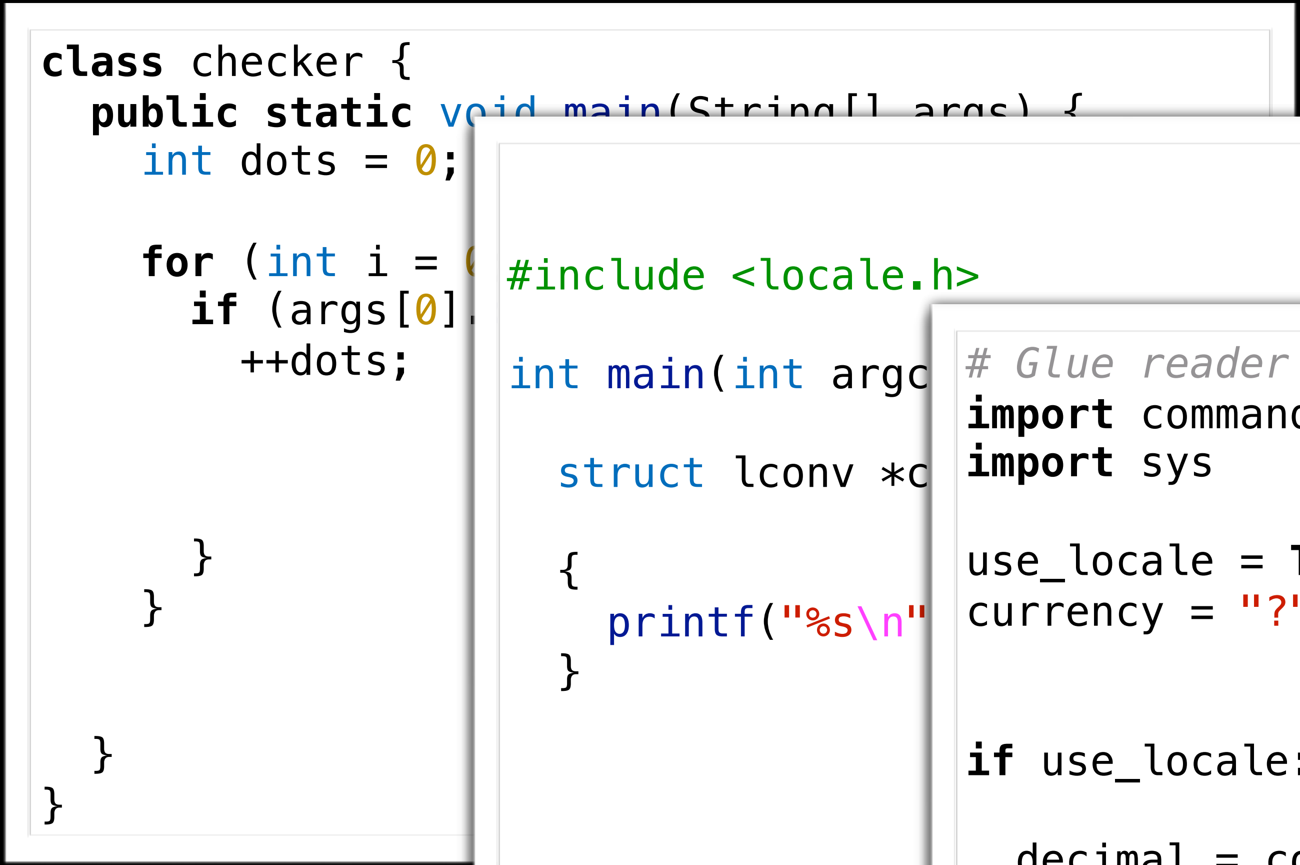


```
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>

int main(int argc
  setlocale(LC_AL
  struct lconv *c
  if (atoi(argv[1
  {
    printf("%s\n"
  }
  else
  {
    printf("%s\n"
  }
  return 0;
}
```

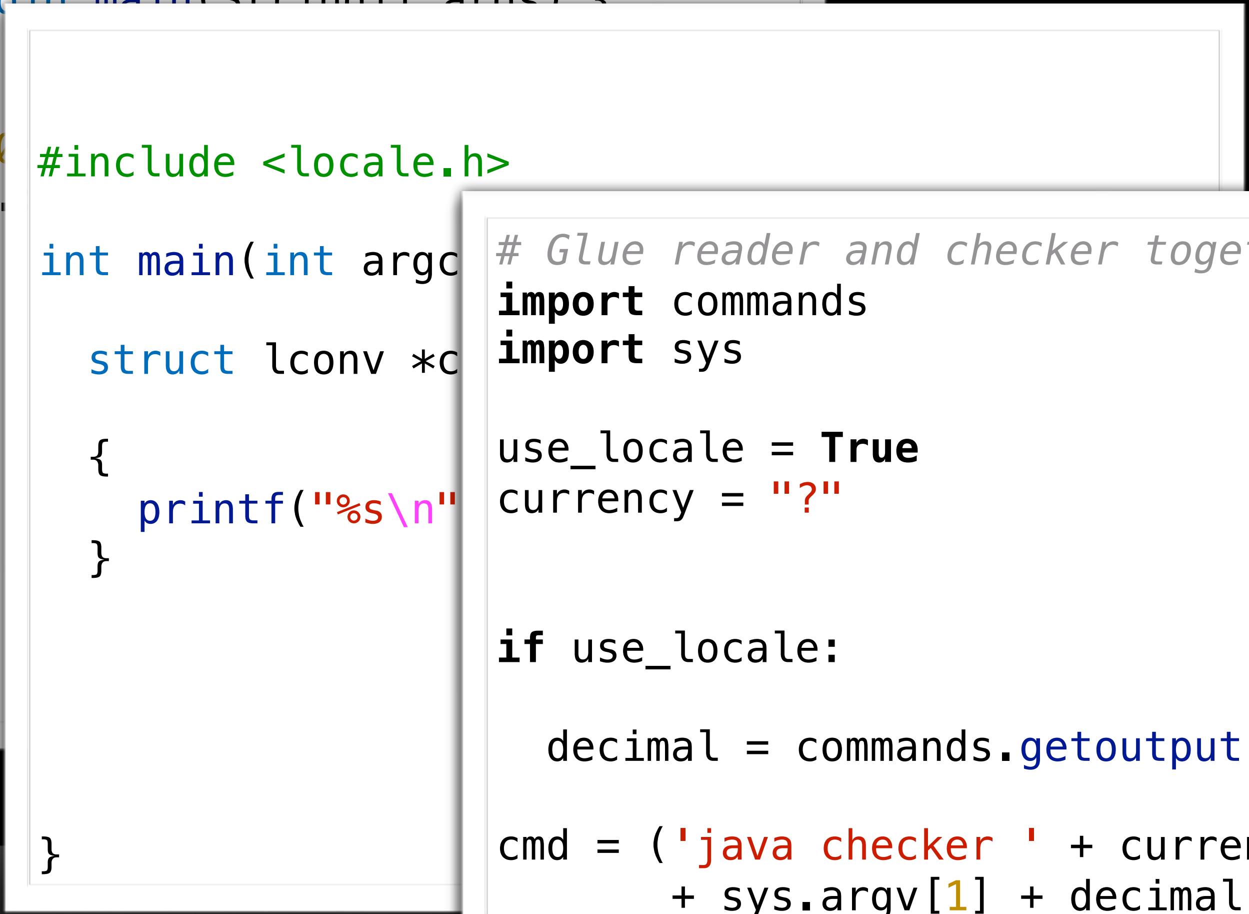


```
# Glue reader and checker together.
import commands
import sys

use_locale = True
currency = "?"
decimal = ","

if use_locale:
  currency = commands.getoutput('./reader 0')
  decimal = commands.getoutput('./reader 1')

cmd = ('java checker ' + currency
       + sys.argv[1] + decimal + sys.argv[2])
print commands.getoutput(cmd)
```

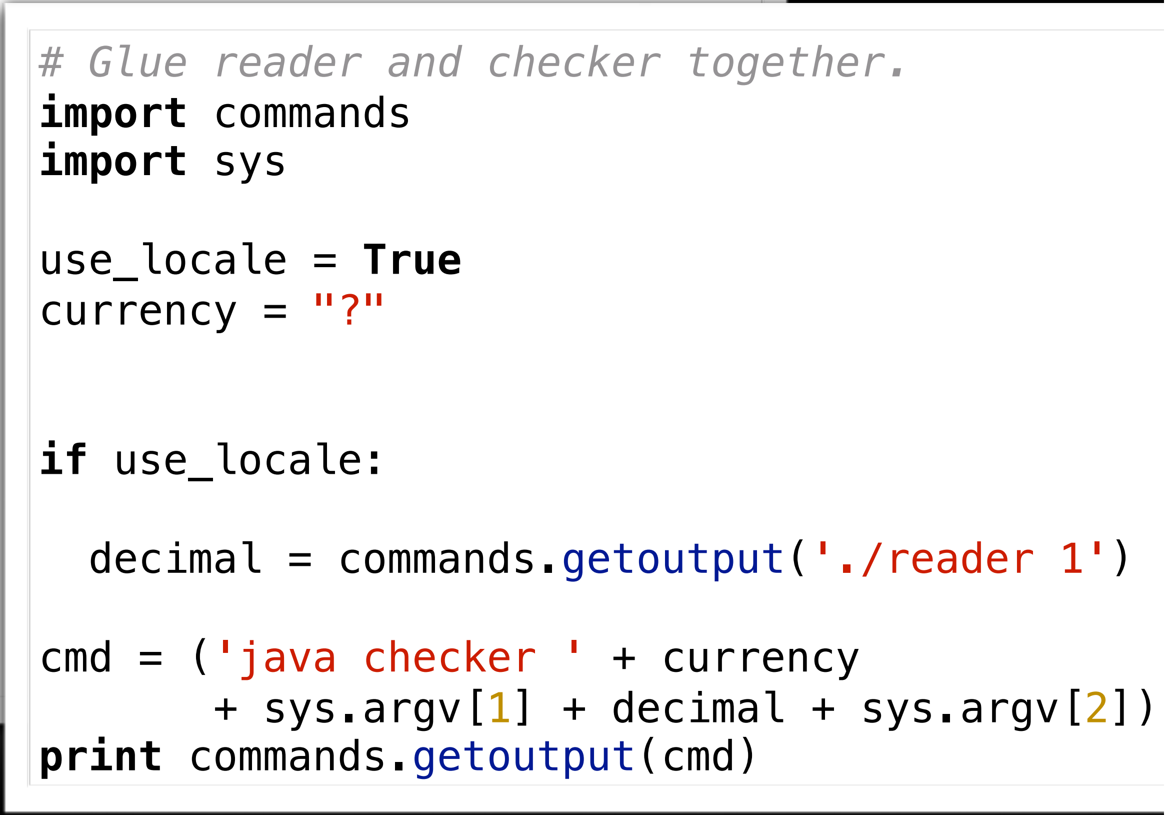
Yes, we can!



```java
class checker {
  public static void main(String[] args) {
    int dots = 0;

    for (int i = (
      if (args[0].
        ++dots;



      }
    }


  }
}
```



```c
#include <locale.h>

int main(int argc

  struct lconv *c

  {
    printf("%s\n"
  }






}
```



```python
# Glue reader and checker together.
import commands
import sys

use_locale = True
currency = "?"


if use_locale:

  decimal = commands.getoutput('./reader 1')

cmd = ('java checker ' + currency
       + sys.argv[1] + decimal + sys.argv[2])
print commands.getoutput(cmd)
```

Slicing (Weiser)

A slice *S* of program *P* on slicing criterion *C*
is any executable program with:

1. *S* can be obtained from *P*
   by deleting zero or more statements from *P*.
2. Whenever *P* halts on input *i*
   with state trajectory *T*,
   then *S* also halts on input *i* with state trajectory *T'*,
   and $\mathrm{PROJ}_C(T) = \mathrm{PROJ}_C(T')$, where PROJ_C is the
   projection function associated with criterion *C*.

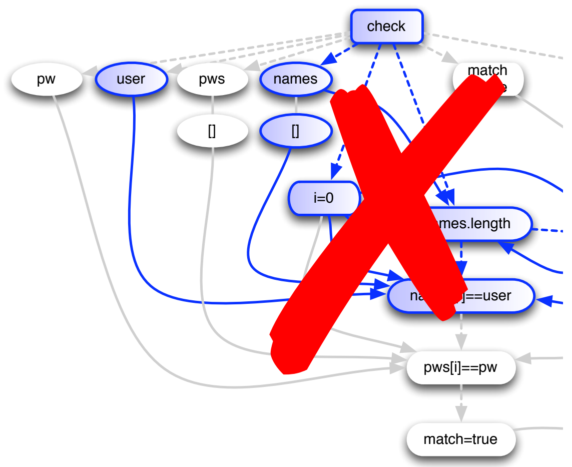
Dynamic Slicing

A dynamic slice *S* of program *P* on slicing criterion *C* **for inputs *I*** is any executable program with:

1. *S* can be obtained from *P*
   by deleting zero or more statements from *P*.

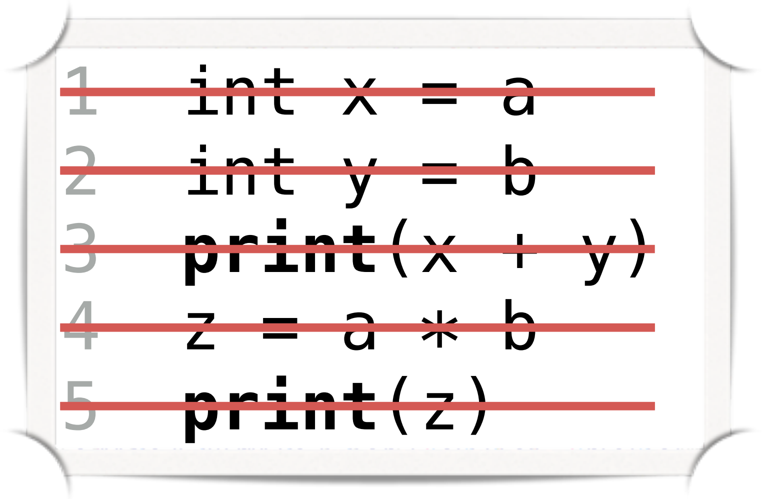
2. Whenever *P* halts on input *i* **from *I***
   with state trajectory *T*,
   then *S* also halts on input *i* with state trajectory *T′*,
   and $\text{PROJ}_C(T) = \text{PROJ}_C(T')$, where PROJ_C is the projection function associated with criterion *C*.

Observation-based Slicing



- **delete** statements (lines)
- **execute** the candidate slice
- **observe** the behaviour for a given criterion
- accept deletion if behaviour is unchanged
- repeat until no statement can be deleted

Example



```
~~1   int x = a~~
~~2   int y = b~~
~~3   print(x + y)~~
~~4   z = a * b~~
~~5   print(z)~~
```

Input:

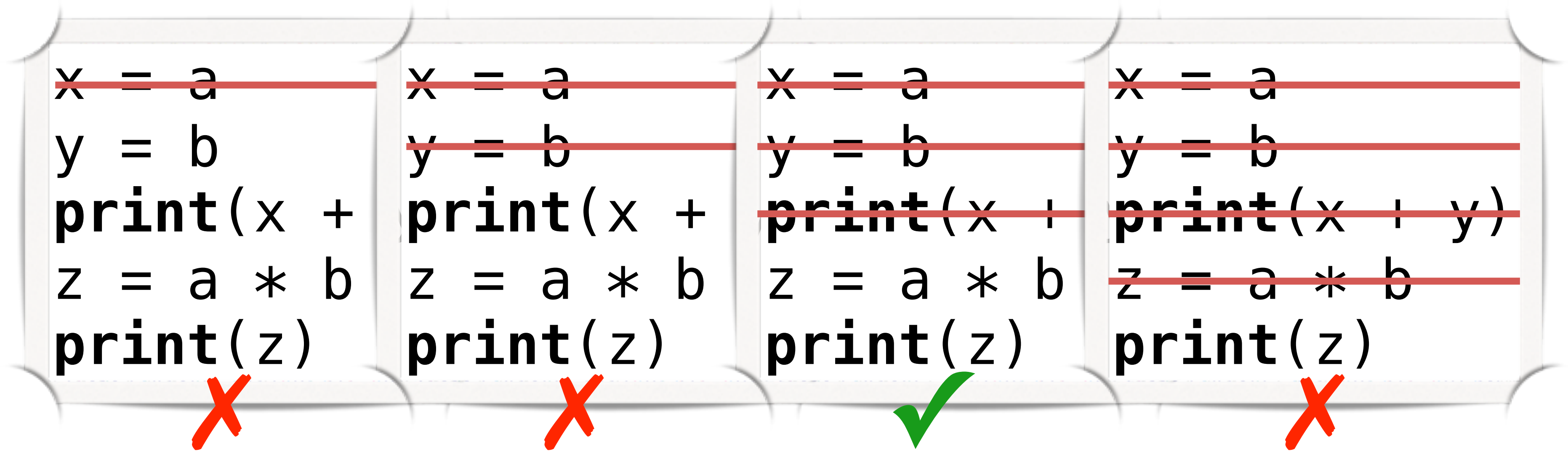
a = 2

b = 3

Criterion:

z in line 5

Parallel Version



~~x = a~~
y = b
print(x +
z = a * b
print(z)
✗

~~x = a~~
~~y = b~~
print(x +
z = a * b
print(z)
✗

~~x = a~~
~~y = b~~
~~**print**(x +~~
z = a * b
print(z)
✓

~~x = a~~
~~y = b~~
~~**print**(x + y)~~
~~z = a * b~~
print(z)
✗

- up to 82% less time for window size four
- larger window sizes lead to less time and often to more deletions

ORBS

- is language independent
  (it is not even aware of the language)

- manipulates files,
  builds and executes the system as usual

- allows binary components or libraries

- creates executable slices (by construction)

Case Study: bash

- 1153 files
- 118,167 SLOC
- 8 different languages
- includes generated source code
- contains libraries

```
#!/bin/bash
```

Criterion

- Variable ‘val’ at line 1393 in ‘expr.c’
  (result of converting a string to an int)

- Test cases ‘arith.tests’ are used as inputs
  (executes the arithmetic functions)

Criterion is executed 80,425 times
(i.e. 80,425 elements in the trajectory)

Scenario

Files to be sliced:

variables.c (variables are used in tests)
parse.y (defines input format, yacc grammar)

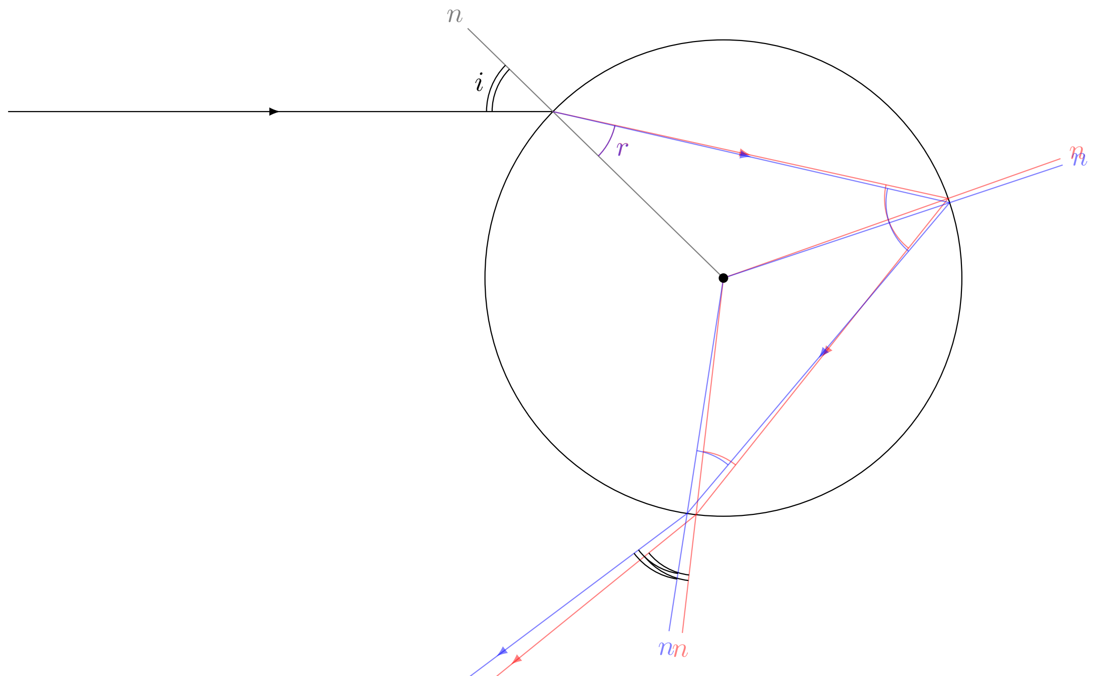
Results:

- 9,417 of 10,804 lines are deleted (13% – 17% SLOC), 42,793 compilations, 5,370 executions
- only 88 lines of 849 grammar lines are left
  8 rules have been removed completely

Non-Standard Semantics

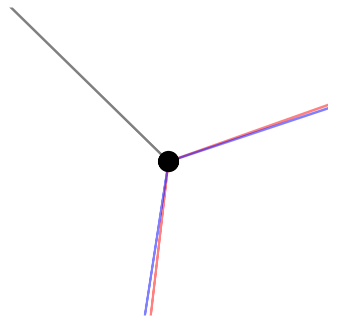
ORBS should work with languages that have *non-traditional semantics!*

For example, *Picture Description Languages*:
Postscript, pic, xfig, TikZ/PGF, Latex, HTML, ….



Slicing Criteria

Slice with respect to this bit



Slice pic

```
program Main
  i := 1
  while i < 11 do
    i := i + 1
  od end(i)
```

Edge Key	
➡	control dependence
→	loop-independent flow dependence
─┼→	loop-carried flow dependence
- - →	def-order dependence

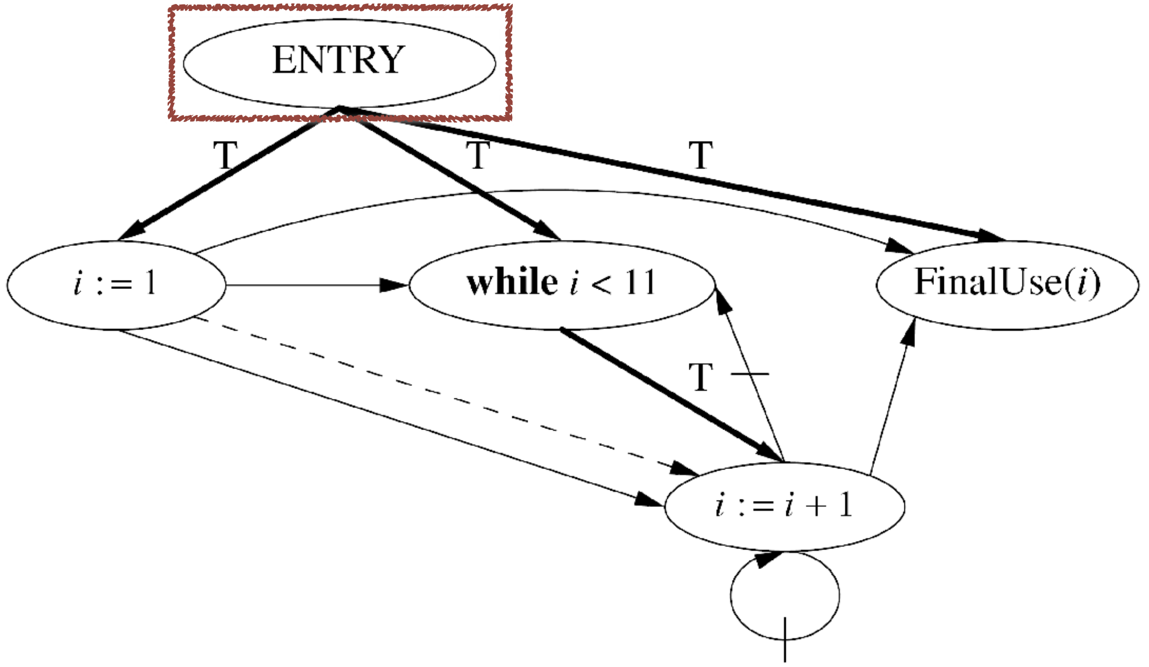
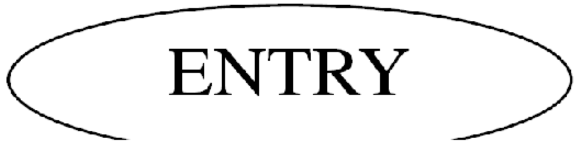


Figure <|pdg_slice_figure|>. The graph and the corresponding program that result from slicing the program dependence graph from Figure 1 with respect to the final-use vertex for *i*.

Slice pic

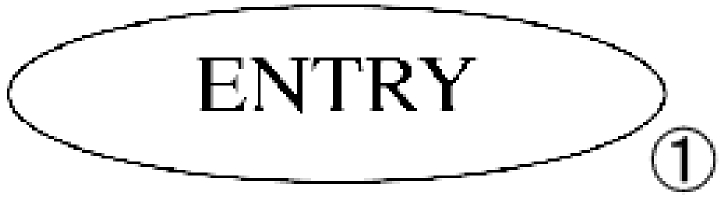


Slicing Criteria

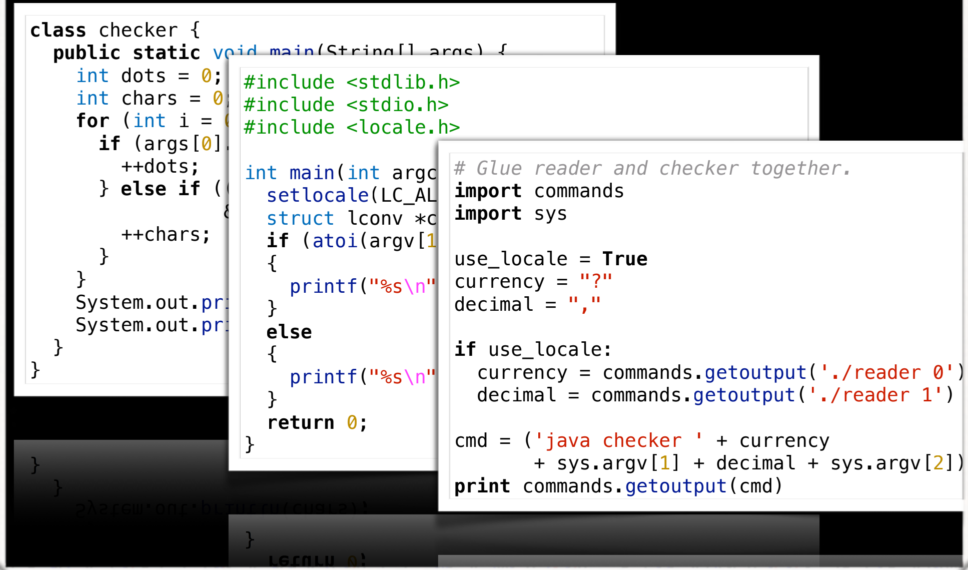
The Slice

```
.PS
ellipseht = ellipseht*.6
ellipsewid = ellipsewid*1.4
Entry: ellipse “ENTRY”   ①
```

Rendered Slice



Who can slice this?



```
class checker {
  public static void main(String[] args) {
    int dots = 0;
    int chars = 0
    for (int i = 
      if (args[0].
        ++dots;
      } else if (
        
        ++chars;
      }
    }
    System.out.pr
    System.out.pr
  }
}
```

```
#include <stdlib.h>
#include <stdio.h>
#include <locale.h>

int main(int argc
  setlocale(LC_AL
  struct lconv *c
  if (atoi(argv[1
  {
    printf("%s\n"
  }
  else
  {
    printf("%s\n"
  }
  return 0;
}
```

```
# Glue reader and checker together.
import commands
import sys

use_locale = True
currency = "?"
decimal = ","

if use_locale:
  currency = commands.getoutput('./reader 0')
  decimal = commands.getoutput('./reader 1')

cmd = ('java checker ' + currency
       + sys.argv[1] + decimal + sys.argv[2])
print commands.getoutput(cmd)
```

Slice pic

```
program Main
  i := 1
  while i < 11 do
    i := i + 1
  od end(i)
```

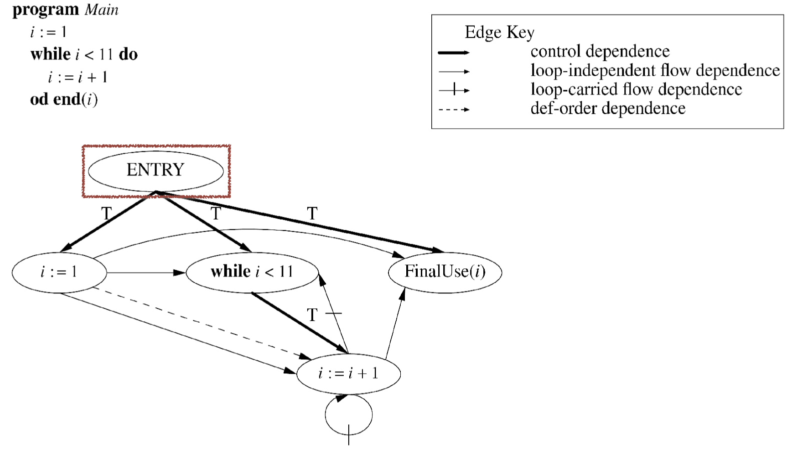


Figure <|pdg_slice_figure|>. The graph and the corresponding program that result from slicing the program dependence graph from Figure 1 with respect to the final-use vertex for *i*.

ORBS

- is the first language-independent slicer
- that slices systems in multiple languages
- and allows binary components or libraries
- to compute executable and minimal slices.