

Linear Schemas for Program Dependence

Sebastian Danicic

November 21, 2006



First Schema Meeting 21 Nov 2006

Vague Agenda:

- 10-10.30: Coffee etc.
- 10.30-11.15: Sebastian Danicic: Intro to Schemas Project
- 11.15-12.00: Mike Laurence: A History of Schemas: Results and Open Problems
- 12.00-1.15: Lunch (probably in College)
- 1.15-1.30: "Is that the Post Office Tower?"
(Traditional Circuit of Goldsmiths)

- 1.30-4: Future Work - Discussion
- 4-6: Drinking in the Hobgoblin Pub
- 6- Eating in The Thailand, 15 Lewisham Way



EPSRC

Engineering and Physical Sciences
Research Council



EPSRC

Engineering and Physical Sciences
Research Council



EPSRC

Engineering and Physical Sciences
Research Council



KING'S
College
LONDON
Founded 1829



EPSRC

Engineering and Physical Sciences
Research Council



KING'S
College
LONDON
Founded 1829

Brunel
UNIVERSITY
WEST LONDON



EPSRC

Engineering and Physical Sciences
Research Council



KING'S
College
LONDON
Founded 1829

Brunel
UNIVERSITY
WEST LONDON



University of Essex



EPSRC

Engineering and Physical Sciences
Research Council



University of Essex



EPSRC

Engineering and Physical Sciences
Research Council



University of Essex



History of the Schemas Project



History of the Schemas Project
Aims of the Schemas Project



This Talk

History of the Schemas Project
Aims of the Schemas Project
People in the Schemas Project



This Talk

History of the Schemas Project

Aims of the Schemas Project

People in the Schemas Project

Motivation - Examples from Program Slicing



History of the Schemas Project

Aims of the Schemas Project

People in the Schemas Project

Motivation - Examples from Program Slicing

Semantics of Schemas and Dataflow Minimality



History of the Schemas Project

Aims of the Schemas Project

People in the Schemas Project

Motivation - Examples from Program Slicing

Semantics of Schemas and Dataflow Minimality

Decidability of Equivalence of Schemas



History of the Schemas Project

Aims of the Schemas Project

People in the Schemas Project

Motivation - Examples from Program Slicing

Semantics of Schemas and Dataflow Minimality

Decidability of Equivalence of Schemas

Future Work



History of Schema Theory

- Schema theory was introduced in the 1950s by a Russian Mathematician, Ianov. It was seen as a way of proving the correctness of compiler optimisations.



History of Schema Theory

- Schema theory was introduced in the 1950s by a Russian Mathematician, Ianov. It was seen as a way of proving the correctness of compiler optimisations.
- Schemas are an abstract way of representing classes of programs with identical structure.



Seminal Work on Program Schemas

Some well-known computer scientists have worked on Schemas:



Seminal Work on Program Schemas

Some well-known computer scientists have worked on Schemas:

- Ianov (1960) “The Logical Schemes of Algorithms”
- M.S. Paterson (1968) “Program Schemata”
- D.C. Cooper(1969) “Program Scheme Equivalences and Logic”
- R.Milner(1970) “Equivalences on Program Schemes”
- Ershov (1971) “Theory of Program Schemata”
- Constable and Gries(1972) “On Classes of Program Schemata”
- Garland and Luckham(1973) “Program Schemes, Recursion Schemes and Formal Language”
- A.K.Chandra (1973) “On the Properties and Applications of Program Schemas”



Death of Schemas



The subject more or less died out by the 1980s due to a lack of positive results.



- In the 1990s, Mark and I were working on the computability of Dataflow Minimal Slices.



- In the 1990s, Mark and I were working on the computability of Dataflow Minimal Slices.
- Mark was at a conference and mentioned our problem to Tom Reps of Wisconsin and Tim Teitelbaum of Cornell who suggested we looked at





- In the 1990s, Mark and I were working on the computability of Dataflow Minimal Slices.
- Mark was at a conference and mentioned our problem to Tom Reps of Wisconsin and Tim Teitelbaum of Cornell who suggested we looked at **Schema Theory**.



They were right!



They were right!

- For our problem it turned out we needed a class of schema in which no predicate or function symbol occurs more than once.



They were right!

- For our problem it turned out we needed a class of schema in which no predicate or function symbol occurs more than once.
- We called these **Linear** Schemas.



They were right!

- For our problem it turned out we needed a class of schema in which no predicate or function symbol occurs more than once.
- We called these **Linear** Schemas.
- We were surprised that no work had been done on Linear Schemas.



They were right!

- For our problem it turned out we needed a class of schema in which no predicate or function symbol occurs more than once.
- We called these **Linear** Schemas.
- We were surprised that no work had been done on Linear Schemas.
- Serendipitously, it turned out that the linearity condition helped in proving decidability of equivalence of schemas.



- Mike proved that equivalence of conservative, free, linear schemas is decidable.



- Mike proved that equivalence of conservative, free, linear schemas is decidable.
- and in his thesis he strengthened this by proving that equivalence of liberal, free linear schemas is decidable in polynomial time.



- Mike proved that equivalence of conservative, free, linear schemas is decidable.
- and in his thesis he strengthened this by proving that equivalence of liberal, free linear schemas is decidable in polynomial time.
- This represented significant progress in the field of schema theory after a hiatus of about twenty years.



But we still don't know whether dataflow minimal slices are computable!



Aims of the Schemas Project

- There is strong evidence that the imposition of this extra but natural condition of linearity (or partial forms of linearity) will lead to further decidability results in the theory of schemas.



Aims of the Schemas Project

- There is strong evidence that the imposition of this extra but natural condition of linearity (or partial forms of linearity) will lead to further decidability results in the theory of schemas.
- We hope that our new results will lead to a re-appraisal of the substantial body of work in program schema theory and to further research on its applications in a modern framework.



Aims of the Schemas Project

- There is strong evidence that the imposition of this extra but natural condition of linearity (or partial forms of linearity) will lead to further decidability results in the theory of schemas.
- We hope that our new results will lead to a re-appraisal of the substantial body of work in program schema theory and to further research on its applications in a modern framework.
e.g. Dataflow Minimal Slicing!



People





Dr. Sebastian Danicic (Principal Investigator)





Dr. Sebastian Danicic (Principal Investigator)

- Sebastian is a Lecturer in Computing at Goldsmiths.





Dr. Sebastian Danicic (Principal Investigator)

- Sebastian is a Lecturer in Computing at Goldsmiths.
- His work on slicing began in the mid 1990s with work on its semantic foundations and algorithms.





Dr. Sebastian Danicic (Principal Investigator)

- Sebastian is a Lecturer in Computing at Goldsmiths.
- His work on slicing began in the mid 1990s with work on its semantic foundations and algorithms.
- The Schemas project described here has grown out of the research Sebastian conducted during his PhD. Since then, he has supervised two PhDs continuing this work.





Prof. Mark Harman (Co-Investigator)





Prof. Mark Harman (Co-Investigator)

- Mark is Professor of Software Engineering at King's College London.





Prof. Mark Harman (Co-Investigator)

- Mark is Professor of Software Engineering at King's College London.
- He is the principal investigator on a number of projects including one to design and build an industrial program slicing and variable dependence analysis system.





Prof. Mark Harman (Co-Investigator)

- Mark is Professor of Software Engineering at King's College London.
- He is the principal investigator on a number of projects including one to design and build an industrial program slicing and variable dependence analysis system.
- He is the inventor of Amorphous Slicing





Prof. Mark Harman (Co-Investigator)

- Mark is Professor of Software Engineering at King's College London.
- He is the principal investigator on a number of projects including one to design and build an industrial program slicing and variable dependence analysis system.
- He is the inventor of Amorphous Slicing
- He has also worked on problems in software testing and upon the combination of slicing, transformation and testing.





Prof. Mark Harman (Co-Investigator)

- Mark is Professor of Software Engineering at King's College London.
- He is the principal investigator on a number of projects including one to design and build an industrial program slicing and variable dependence analysis system.
- He is the inventor of Amorphous Slicing
- He has also worked on problems in software testing and upon the combination of slicing, transformation and testing.
- Mark is also well-known for his work on Search-Based Software Engineering for which he has recently obtained a £1,145,357 EPSRC grant.





Prof. Mark Harman (Co-Investigator)

- Mark is Professor of Software Engineering at King's College London.
- He is the principal investigator on a number of projects including one to design and build an industrial program slicing and variable dependence analysis system.
- He is the inventor of Amorphous Slicing
- He has also worked on problems in software testing and upon the combination of slicing, transformation and testing.
- Mark is also well-known for his work on Search-Based Software Engineering for which he has recently obtained a £1,145,357 EPSRC grant.
- Mark used to work at Goldsmiths





Prof. Rob Hierons (Co-Investigator)





Prof. Rob Hierons (Co-Investigator)

- Rob is Professor of Computer Science at Brunel University.





Prof. Rob Hierons (Co-Investigator)

- Rob is Professor of Computer Science at Brunel University.
- Rob has a BA in Mathematics (Trinity College, Cambridge), and a Ph.D. in Computer Science (Brunel University)





Prof. Rob Hierons (Co-Investigator)

- Rob is Professor of Computer Science at Brunel University.
- Rob has a BA in Mathematics (Trinity College, Cambridge), and a Ph.D. in Computer Science (Brunel University)
- Rob has published widely in the areas of testing and software engineering.





Prof. Rob Hierons (Co-Investigator)

- Rob is Professor of Computer Science at Brunel University.
- Rob has a BA in Mathematics (Trinity College, Cambridge), and a Ph.D. in Computer Science (Brunel University)
- Rob has published widely in the areas of testing and software engineering.
- Rob used to work at Goldsmiths





Dr. Chris Fox





Dr. Chris Fox

- Chris is Reader in Computer Science at Essex University.





Dr. Chris Fox

- Chris is Reader in Computer Science at Essex University.
- He has published widely in the areas of Formal Semantics, Philosophy of Language, Business Process Modelling and Program Analysis.





Dr. Chris Fox

- Chris is Reader in Computer Science at Essex University.
- He has published widely in the areas of Formal Semantics, Philosophy of Language, Business Process Modelling and Program Analysis.
- Chris implemented the first conditioned slicer, ConSiT: a system which combines symbolic program execution with theorem proving.





Dr. Chris Fox

- Chris is Reader in Computer Science at Essex University.
- He has published widely in the areas of Formal Semantics, Philosophy of Language, Business Process Modelling and Program Analysis.
- Chris implemented the first conditioned slicer, ConSiT: a system which combines symbolic program execution with theorem proving.
- Chris used to work at Goldsmiths





Dr. Lahcen Ouarbya

- Lahcen is a lecturer at Goldsmiths College.





Dr. Lahcen Ouarbya

- Lahcen is a lecturer at Goldsmiths College.
- He has a BSc in Physics from Cadi Ayad University, Marrakech, Morocco





Dr. Lahcen Ouarbya

- Lahcen is a lecturer at Goldsmiths College.
- He has a BSc in Physics from Cadi Ayad University, Marrakech, Morocco
- ...an MSc in Solid State Physics from St. Petersburg State University





Dr. Lahcen Ouarbya

- Lahcen is a lecturer at Goldsmiths College.
- He has a BSc in Physics from Cadi Ayad University, Marrakech, Morocco
- ...an MSc in Solid State Physics from St. Petersburg State University
- and a PhD in Formal Semantics of Slicing from Goldsmiths, (supervised by Sebastian, Mark, Rob, Chris and John).





Dr. Mohammed (Dave) Daoudi



- Dave is now studying Financial Mathematics at Imperial College.





Dr. Mohammed (Dave) Daoudi



- Dave is now studying Financial Mathematics at Imperial College.
- Dave has a first class honours degree in Mathematics from Goldsmiths





Dr. Mohammed (Dave) Daoudi



- Dave is now studying Financial Mathematics at Imperial College.
- Dave has a first class honours degree in Mathematics from Goldsmiths
- and a PhD in Conditioned Slicing from Goldsmiths, (supervised by Sebastian, Mark, Rob, Chris and John).





Dr. Dr. Mike Laurence (Research Assistant)

- Mike has BSc in Mathematics from Queen Mary College, University of London (1st class honours) and has an MSc in Mathematics from University of Warwick





Dr. Dr. Mike Laurence (Research Assistant)

- Mike has BSc in Mathematics from Queen Mary College, University of London (1st class honours) and has an MSc in Mathematics from University of Warwick
- A PhD in Group Theory from Queen Mary College





Dr. Dr. Mike Laurence (Research Assistant)

- Mike has BSc in Mathematics from Queen Mary College, University of London (1st class honours) and has an MSc in Mathematics from University of Warwick
- A PhD in Group Theory from Queen Mary College
- A PhD in Schema Theory from Goldsmiths, (supervised by Sebastian, Rob, Mark and John).





Dr. Dr. Mike Laurence (Research Assistant)

- Mike has BSc in Mathematics from Queen Mary College, University of London (1st class honours) and has an MSc in Mathematics from University of Warwick
- A PhD in Group Theory from Queen Mary College
- A PhD in Schema Theory from Goldsmiths, (supervised by Sebastian, Rob, Mark and John).
- He has had a number of previous jobs notably as a clothes sorter at an Oxfam shop near Goudge Street.





Dr. Dr. Mike Laurence (Research Assistant)

His favourite food is:





Dr. Dr. Mike Laurence (Research Assistant)

His favourite food is:



Pistachio Nuts!





Dr. John Howroyd (Industrial Collaborator)

- John works as Director of Research and Development at @UK PLC.





Dr. John Howroyd (Industrial Collaborator)

- John works as Director of Research and Development at @UK PLC.
- He has a BSc in Mathematics from Oxford University and a PhD in Mathematics from St. Andrews.





Dr. John Howroyd (Industrial Collaborator)

- John works as Director of Research and Development at @UK PLC.
- He has a BSc in Mathematics from Oxford University and a PhD in Mathematics from St. Andrews.
- His interests include Geometric Measure Theory, Harmonic Analysis, Isometries, Applications of Domain Theory in Computable Mathematics, and Static Program Analysis.





Dr. John Howroyd (Industrial Collaborator)

- John works as Director of Research and Development at @UK PLC.
- He has a BSc in Mathematics from Oxford University and a PhD in Mathematics from St. Andrews.
- His interests include Geometric Measure Theory, Harmonic Analysis, Isometries, Applications of Domain Theory in Computable Mathematics, and Static Program Analysis.
- John used to work at Goldsmiths





Prof. Elaine Weyuker (Industrial Collaborator)

- Elaine works at AT&T Labs-Research.





Prof. Elaine Weyuker (Industrial Collaborator)

- Elaine works at AT&T Labs-Research.
- She has published widely in the area of schema theory and dataflow analysis.





Prof. Elaine Weyuker (Industrial Collaborator)

- Elaine works at AT&T Labs-Research.
- She has published widely in the area of schema theory and dataflow analysis.
- She is the originator of dataflow testing.





Prof. Elaine Weyuker (Industrial Collaborator)

- Elaine works at AT&T Labs-Research.
- She has published widely in the area of schema theory and dataflow analysis.
- She is the originator of dataflow testing.
- She is world renowned for her research into providing formal bases for many areas of computer science.





Prof. Elaine Weyuker (Industrial Collaborator)

- Elaine works at AT&T Labs-Research.
- She has published widely in the area of schema theory and dataflow analysis.
- She is the originator of dataflow testing.
- She is world renowned for her research into providing formal bases for many areas of computer science.
- Elaine isn't here :(





Prof. Elaine Weyuker (Industrial Collaborator)

- Elaine works at AT&T Labs-Research.
- She has published widely in the area of schema theory and dataflow analysis.
- She is the originator of dataflow testing.
- She is world renowned for her research into providing formal bases for many areas of computer science.
- Elaine isn't here :(
- and ...



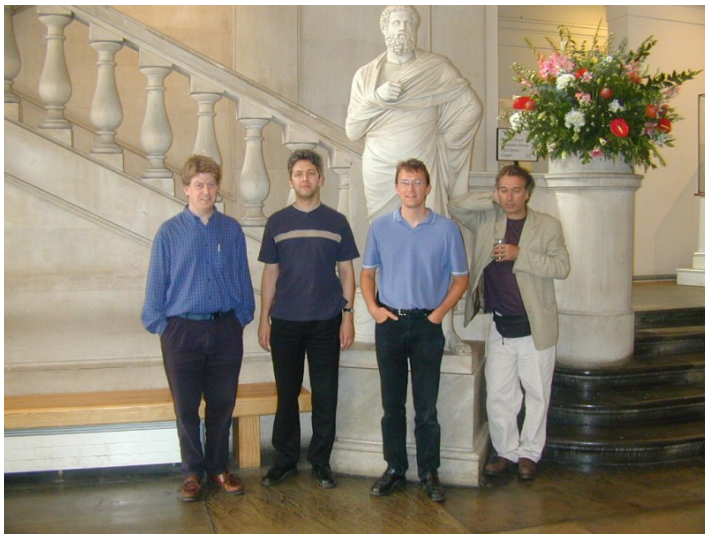


Prof. Elaine Weyuker (Industrial Collaborator)

- Elaine works at AT&T Labs-Research.
- She has published widely in the area of schema theory and dataflow analysis.
- She is the originator of dataflow testing.
- She is world renowned for her research into providing formal bases for many areas of computer science.
- Elaine isn't here :(
- and ...
- Elaine has no connection with Goldsmiths



A few of years ago...



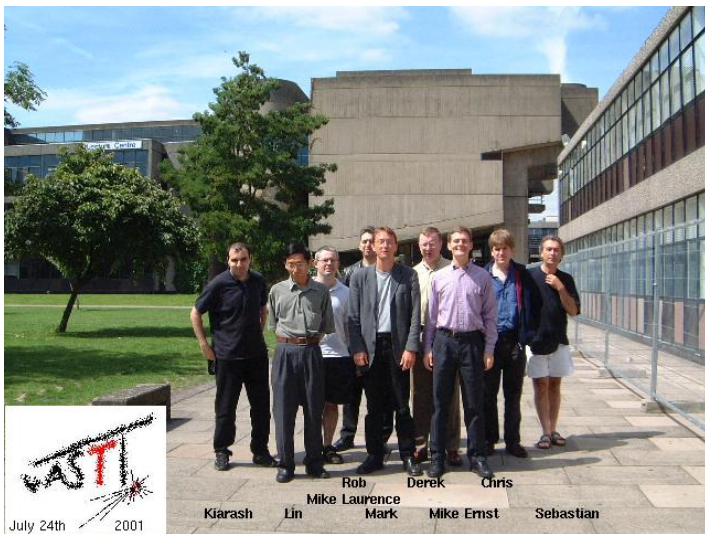
A few of years ago...



A few of years ago...



A few of years ago...



A few of years ago...



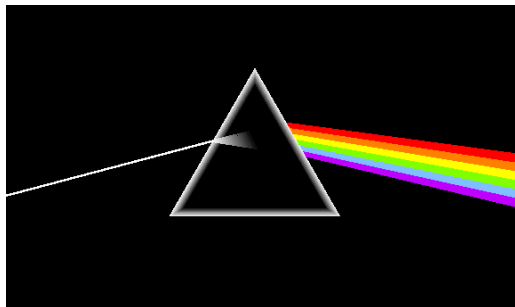
A few of years ago...



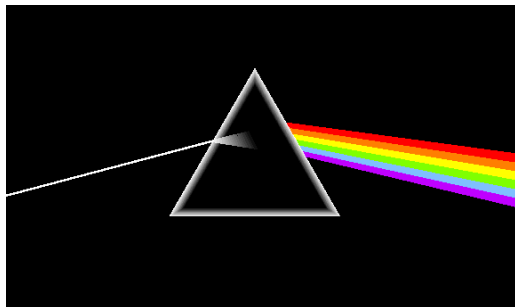
A few of years ago...



Background – Program Slicing



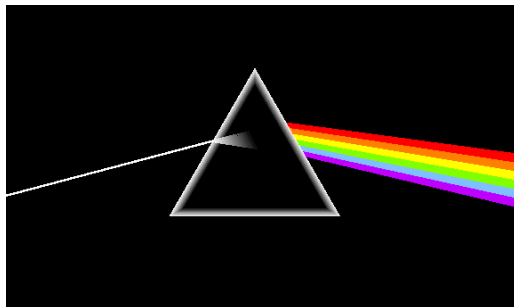
Background – Program Slicing



Program Slicing gives us different views of the same program



Background – Program Slicing



Program Slicing gives us different views of the same program
...depending what we are interested in.



- Which bits of big program P affect the final value of variable x ?



- Which bits of big program P affect the final value of variable x ?
- Which bits of big program P affect the updating of this file?



Questions answered by Slicing

- Which bits of big program P affect the final value of variable x ?
- Which bits of big program P affect the updating of this file?
- Which bits of big program P affect the firing of this missile?



Commercial Slicing Tools: Kaveri/Indus

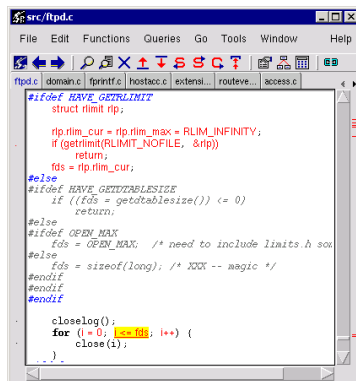


```
63 }
64 }
65 private synchronized boolean checkWrite()
66 {
67     if(nr==0 && nr==0)
68     {
69         nr++; ← Complete Slice Element
70         return true;
71     }
72     else
73         return false;
74 }
75 void end_write()
76 {
77     synchronized(this)
78     {
79         hw--; ← Partial Slice Element
80     }
81     synchronized(objectR) { objectR.notifyAll(); }
82     synchronized(objectW) { objectW.notify(); }
83 }
84 };
```

Kaveri is an eclipse plug-in front-end for the Indus Java slicer. It utilizes the Indus program slicer to calculate slices of Java programs and then displays the results visually in the editor. The purpose of this project is to create an effective tool for simplifying program understanding, program analysis, program debugging and testing.



Commercial Slicing Tools: Codesurfer



```
src/rtpd.c
File Edit Functions Queries Go Tools Window Help
f/rtpd.c domain.c f/print.c hostacc.c extensi... routeve... access.c
#ifdef HAVE_GETRLIMIT
struct rlimit rlp;

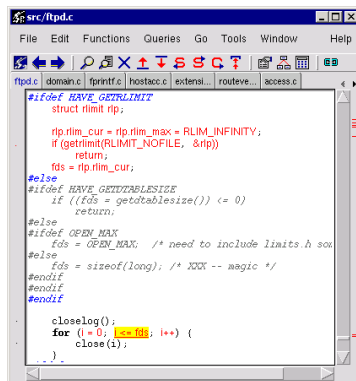
rlp.rlim_cur = rlp.rlim_max = RLIM_INFINITY;
if (getrlimit(RLIMIT_NOFILE, &rlp))
    return;
fds = rlp.rlim_cur;
#else
#ifdef HAVE_GETDTABLESIZE
if ((fds = getdtablesize()) <= 0)
    return;
#else
#ifdef OPEN_MAX
fds = OPEN_MAX; /* need to include limits.h so
#else
fds = sizeof(long); /* XXX -- magic */
#endif
#endif
#endif

closelog();
for (i = 0; i <= fds; i++) {
    close(i);
}
.._.
```

“The backward slice from a program point p includes all points that may influence whether control reaches p , and all points that may influence the values of the variables used at p when control gets there.”



Commercial Slicing Tools: Codesurfer



```
src/rtpd.c
File Edit Functions Queries Go Tools Window Help
rtpd.c domain.c fprint.c hostacc.c extensi... routeve... access.c
#ifdef HAVE_GETRLIMIT
struct rlim rlp;

rlp.rlim_cur = rlp.rlim_max = RLIM_INFINITY;
if (getrlimit(RLIMIT_NOFILE, &rlp))
return;
fds = rlp.rlim_cur;
#else
#ifdef HAVE_GETDTABLESIZE
if ((fds = getdtablesize()) <= 0)
return;
#else
#ifdef OPEN_MAX
fds = OPEN_MAX; /* need to include limits.h so
#else
fds = sizeof(long); /* XXX -- magic */
#endif
#endif
#endif

closelog();
for (i = 0; i <= fds; i++) {
close(i);
}
}
```

*“The backward slice from a program point p includes all points that **may** influence whether control reaches p , and all points that **may** influence the values of the variables used at p when control gets there.”*

What the hell does **may** mean?



The Crux of the Problem

- Slicing algorithms are conservative: They often keep in bits of the program that could be left out.



The Crux of the Problem

- Slicing algorithms are conservative: They often keep in bits of the program that could be left out.
- A statement that a slicing algorithm thinks may affect a variable often does not!



The Crux of the Problem

- Slicing algorithms are conservative: They often keep in bits of the program that could be left out.
- A statement that a slicing algorithm thinks may affect a variable often does not!
- This leads to slices that are too big.



The Crux of the Problem

- Slicing algorithms are conservative: They often keep in bits of the program that could be left out.
- A statement that a slicing algorithm thinks may affect a variable often does not!
- This leads to slices that are too big.
- Small is beautiful. – Big slices aren't very useful.



The Crux of the Problem

- Slicing algorithms are conservative: They often keep in bits of the program that could be left out.
- A statement that a slicing algorithm thinks may affect a variable often does not!
- This leads to slices that are too big.
- Small is beautiful. – Big slices aren't very useful.
- We want to find ways of producing smaller slices.



Examples

We now give two examples showing wrong computation of dependence by slicing algorithms:



We now give two examples showing wrong computation of dependence by slicing algorithms:

- The “c becomes one” example



We now give two examples showing wrong computation of dependence by slicing algorithms:

- The “c becomes one” example
- The “Montreal Boat Trip” example (John Howroyd)



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5)
    {
        z=7;
        c=y+c;
    }
    i=i+1;
}
```



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5)
    {
        z=7;
        c=y+c;
    }
    i=i+1;
}
```

Conventional Program Slicers like Codesurfer will say “all of them!”



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5)
    {
        z=7;
        c=y+c;
    }
    i=i+1;
}
```

but why?



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5) <-----
    {
        z=7; <-----
        c=y+c;
    }
    i=i+1;
}
```

z=7 is *control-dependent* on *(c<5)*



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5) <-----
    {
        z=7;
        c=y+c; <-----
    }
    i=i+1;
}
```

Because its in a loop $c < 5$ is data-dependent upon $c = y + c$;



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k) <-----  
{  
    if (c<5) <-----  
    {  
        z=7;  
        c=y+c;  
    }  
    i=i+1;  
}
```

The **if** is control-dependent the guard of the **while**



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k) <-----  
{  
    if (c<5)  
    {  
        z=7;  
        c=y+c;  
    }  
    i=i+1; <-----  
}
```

The guard of the **while** is data-dependent on **i=i+1**



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5)
    {
        z=7;
        c=y+c;
    }
    i=i+1;
}
```

So slicing on z gives the whole program



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5)
    {
        z=7;
        c=y+c;
    }
    i=i+1;
}
```

Slicing Algorithms compute the transitive closure of the dependence relation



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5)
    {
        z=7;
        c=y+c; <-----
    }
    i=i+1;
}
```

But is z really dependent on this?



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5)
    {
        z=7;
        c=y+c; <-----
    }
    i=i+1;
}
```

Clearly not because if we do execute $c=y+c$ the value of z can't change any further, so it is irrelevant if we go through the true part of the if after that.



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5)
    {
        z=7;
        c=y+c; <-----
    }
    i=i+1;
}
```

So Transitive closure of dependence doesn't seem to be the most accurate way of computing dependencies.



The “c becomes one” example

Which lines of this program affect the final value of z?

```
while (i<k)
{
    if (c<5)
    {
        z=7;
        c=y+c; <-----
    }
    i=i+1;
}
```

As we have seen, dependence is not transitive.



Conventional Slicing Removes Non-termination

```
while (true)
{

}
z=2;
```

What do we get if we slice on the final value of z?



Conventional Slicing Removes Non-termination

```
while (true)
{

}
z=2;
```

The loop is removed since $z=2$ is not data or control dependent on it.



Conventional Slicing Removes Non-termination

```
while (true)
{

}
z=2;
```

So transitive closure of dependence can introduce termination.



Conventional Slicing Removes Non-termination

```
while (true)
{

}
z=2;
```

So, formally a program p and its slice s need only agree in initial states where p terminates.



Conventional Slicing Removes Non-termination

```
while (true)
{

}
z=2;
```

So, formally a program p and its slice s need only agree in initial states where p terminates.

So, there's an even smaller slice of this program.



Conventional Slicing Removes Non-termination

```
while (true)
{

}
z=2;
```

So, formally a program p and its slice s need only agree in initial states where p terminates.

So, there's an even smaller slice of this program.

The empty program – all statements can be removed.



The Semantics of End-Slicing

Definition:

q is an **end slice** of p with respect to variable v
if and only if

for all initial states σ

$$\mathcal{M}[|p|]\sigma \neq \perp \implies \mathcal{M}[|p|]\sigma v = \mathcal{M}[|q|]\sigma v$$



The Semantics of End-Slicing

Definition:

q is an **end slice** of p with respect to variable v
if and only if

for all initial states σ

$$\mathcal{M}[|p|]\sigma \neq \perp \implies \mathcal{M}[|p|]\sigma v = \mathcal{M}[|q|]\sigma v$$

- Slicing is reflexive.



The Semantics of End-Slicing

Definition:

q is an **end slice** of p with respect to variable v
if and only if

for all initial states σ

$$\mathcal{M}[|p|]\sigma \neq \perp \implies \mathcal{M}[|p|]\sigma v = \mathcal{M}[|q|]\sigma v$$

- Slicing is reflexive.
- Slicing is transitive.



The Semantics of End-Slicing

Definition:

q is an **end slice** of p with respect to variable v
if and only if

for all initial states σ

$$\mathcal{M}[|p|]\sigma \neq \perp \implies \mathcal{M}[|p|]\sigma v = \mathcal{M}[|q|]\sigma v$$

- Slicing is reflexive.
- Slicing is transitive.
- Slicing is not symmetric.



The Semantics of End-Slicing

Definition:

q is an **end slice** of p with respect to variable v
if and only if

for all initial states σ

$$\mathcal{M}[|p|]\sigma \neq \perp \implies \mathcal{M}[|p|]\sigma v = \mathcal{M}[|q|]\sigma v$$

- Slicing is reflexive.
- Slicing is transitive.
- Slicing is not symmetric.
- Slicing is not anti-symmetric.



The "Montreal Boat Trip" example

THE MONTREAL
BOAT TRIP SLICING
EXAMPLE:

```
while p(j)
{
  if q(k) k = f1(k);
  else {k = f2(k); j = f3(j)}
}
```

Slice on j at ^{end} of program?

$x = 1$
 $x = 1$
 $y = x$
slice on y

Written on the white board by me at SCAM 2002 in Montreal!



The “Montreal Boat Trip” example

THE MONTREAL
BOAT TRIP SLICING
EXAMPLE:

```
while p(j)
{
  if q(k) k = f1(k);
  else {k = f2(k); j = f3(j)}
}
```

Slice on j at ^{end} of program?

$x = 1$
 $x = 1$
 $y = x$
slice on y

I asked, “What is the slice on variable j ”



The "Montreal Boat Trip" example

THE MONTREAL
BOAT TRIP SLICING
EXAMPLE:

```
while p(j)
{
  if q(k) k = f1(k);
  else {k = f2(k); j = f3(j)}
}
```

Slice on j at ^{end}₂ of program?

$x = 1$
 $x = 1$
 $y = x$
slice on y

To which Ira Baxter wittily replied:



The “Montreal Boat Trip” example



Who cares!



The “Montreal Boat Trip” example

What is the slice on j at the end of the program?

```
while p(j)
{
    if q(k) k=f(k);
    else
    {
        k=g(k);
        j=h(j);
    }
}
```



The “Montreal Boat Trip” example

What is the slice on j at the end of the program?

```
while p(j)
{
    if q(k) k=f(k);
    else
    {
        k=g(k);
        j=h(j);
    }
}
```

Again, transitive closure of dependence gives the whole program.



The “Montreal Boat Trip” example

What is the slice on j at the end of the program?

```
while p(j)
{
    if q(k) k=f(k);
    else
    {
        k=g(k);    ←-----
        j=h(j);
    }
}
```

But what about this line?



The “Montreal Boat Trip” example

What is the slice on j at the end of the program?

```
while p(j)
{
    if q(k) k=f(k);
    else
    {
        k=g(k);    <-----
        j=h(j);
    }
}
```

It either causes the program to non-terminate or increases the number of iterations of the loop before termination.



The “Montreal Boat Trip” example

What is the slice on j at the end of the program?

```
while p(j)
{
    if q(k) k=f(k);
    else
    {
        k=g(k);    <-----
        j=h(j);
    }
}
```

In initial states where the program terminates this line doesn't affect the final value of j .



The “Montreal Boat Trip” example

What is the slice on j at the end of the program?

```
while p(j)
{
    if q(k) k=f(k);
    else
    {
        k=g(k);    <-----
        j=h(j);
    }
}
```

So $k=g(k)$ can be removed from the slice.



The “Montreal Boat Trip” example

Linear Schemas

```
while p(j)
{
    if q(k) k=f(k);
    else
    {
        k=g(k);
        j=h(j);
    }
}
```

The program above is in fact a **Schema**



The “Montreal Boat Trip” example

Linear Schemas

```
while p(j)
{
    if q(k) k=f(k);
    else
    {
        k=g(k);
        j=h(j);
    }
}
```

it is in fact a *linear* Schema -because each function and predicate symbol occurs at most once



The “Montreal Boat Trip” example

Linear Schemas

```
while p(j)
{
    if q(k) k=f(k);
    else
    {
        k=f(k);
        j=h(j);
    }
}
```

Now it's not *linear*



The Data Flow Minimality Problem:

- We have shown that transitive closure of dependence gives over-large slices.



The Data Flow Minimality Problem:

- We have shown that transitive closure of dependence gives over-large slices.
- Small slices are good.



The Data Flow Minimality Problem:

- We have shown that transitive closure of dependence gives over-large slices.
- Small slices are good.
- So is it possible to produce **minimal slices** at this level of abstraction?



The Data Flow Minimality Problem:

- We have shown that transitive closure of dependence gives over-large slices.
- Small slices are good.
- So is it possible to produce **minimal slices** at this level of abstraction?
- A **minimal slice** is a slice all of whose proper sub-programs are not slices.



The Semantics of Schemas



The Semantics of Schemas

$$\text{States} = [\text{Variables} \rightarrow \text{Terms}] \cup \{\perp\}$$



The Semantics of Schemas

States = [Variables \rightarrow Terms] \cup $\{\perp\}$

(Herbrand) Interpretations = [Terms \rightarrow {True,False}]



The Semantics of Schemas

States = [Variables \rightarrow Terms] \cup $\{\perp\}$

Interpretations = [Terms \rightarrow {True,False}]



The Semantics of Schemas

States = [Variables \rightarrow Terms] \cup $\{\perp\}$

Interpretations = [Terms \rightarrow {True,False}]

\mathcal{M} : Schemas \rightarrow Interpretations \rightarrow States \rightarrow States.



The Semantics of Schemas

States = [Variables \rightarrow Terms] \cup $\{\perp\}$

Interpretations = [Terms \rightarrow {True,False}]

\mathcal{M} : Schemas \rightarrow Interpretations \rightarrow States \rightarrow States.

```
while p(j)
{
  if q(k) k=f(k);
  else
  {
    k=g(k);
    j=h(j);
  }
}
```

Show John's Haskell Schema Interpreter readSch "boat.sch"



Decidability of Equivalence of Schemas

- Two Schemas are equivalent if they are semantically equivalent under all Herbrand interpretations.



Decidability of Equivalence of Schemas

- Two Schemas are equivalent if they are semantically equivalent under all Herbrand interpretations.
- The Decidability of Equivalence of Schemas implies the computability of minimal slices.



Decidability of Equivalence of Schemas

- Two Schemas are equivalent if they are semantically equivalent under all Herbrand interpretations.
- The Decidability of Equivalence of Schemas implies the computability of minimal slices.
- Why?



Decidability of Equivalence of Schemas

- Two Schemas are equivalent if they are semantically equivalent under all Herbrand interpretations.
- The Decidability of Equivalence of Schemas implies the computability of minimal slices.
- Why?
- Crudely, we can simply try all combinations of deleting statements and check for equivalence of the resulting schema with the original.



The Bad News

Paterson (1967):



Equivalence of Schemas is Undecidable.



Not So Bad News

For Linear Schemas decidability of equivalence is an open problem.



The Good News

Mike Laurence (2006)



For certain classes of Linear Schemas equivalence is decidable.



- Is equivalence of free conservative linear schemas decidable?



Open Problems in Decidability to Investigate

- Is equivalence of free conservative linear schemas decidable? **Yes**



Open Problems in Decidability to Investigate

- Is equivalence of free conservative linear schemas decidable? **Yes**
- Is equivalence of free liberal linear schemas decidable?



Open Problems in Decidability to Investigate

- Is equivalence of free conservative linear schemas decidable? **Yes**
- Is equivalence of free liberal linear schemas decidable? **Yes**



Open Problems in Decidability to Investigate

- Is equivalence of free conservative linear schemas decidable? **Yes**
- Is equivalence of free liberal linear schemas decidable? **Yes**
- Is equivalence of free linear schemas decidable?



Open Problems in Decidability to Investigate

- Is equivalence of free conservative linear schemas decidable? **Yes**
- Is equivalence of free liberal linear schemas decidable? **Yes**
- Is equivalence of free linear schemas decidable? **Don't know**



Open Problems in Decidability to Investigate

- Is equivalence of free conservative linear schemas decidable? **Yes**
- Is equivalence of free liberal linear schemas decidable? **Yes**
- Is equivalence of free Σ linear schemas decidable? **Don't know**
- Is equivalence of Σ linear schemas decidable?



Open Problems in Decidability to Investigate

- Is equivalence of free conservative linear schemas decidable? **Yes**
- Is equivalence of free liberal linear schemas decidable? **Yes**
- Is equivalence of free Σ linear schemas decidable? **Don't know**
- Is equivalence of Σ linear schemas decidable? **Don't know**



Open Problems in Decidability to Investigate

- Is equivalence of free conservative linear schemas decidable? **Yes**
- Is equivalence of free liberal linear schemas decidable? **Yes**
- Is equivalence of free Σ linear schemas decidable? **Don't know**
- Is equivalence of Σ linear schemas decidable? **Don't know**
- Is freeness of linear schemas decidable?



Open Problems in Decidability to Investigate

- Is equivalence of free conservative linear schemas decidable? **Yes**
- Is equivalence of free liberal linear schemas decidable? **Yes**
- Is equivalence of free Σ linear schemas decidable? **Don't know**
- Is equivalence of Σ linear schemas decidable? **Don't know**
- Is freeness of linear schemas decidable? **Don't know**



- A new lazy trace semantics for Schemas for general Slicing



More Accurate Notions of Dependency

- A new lazy trace semantics for Schemas for general Slicing
- Characterisation of Slicing in terms of this new Semantics



More Accurate Notions of Dependency

- A new lazy trace semantics for Schemas for general Slicing
- Characterisation of Slicing in terms of this new Semantics
- Investigation of Decidability of Equivalence in terms of the new Semantics



More Accurate Notions of Dependency

- A new lazy trace semantics for Schemas for general Slicing
- Characterisation of Slicing in terms of this new Semantics
- Investigation of Decidability of Equivalence in terms of the new Semantics
- Extending the Syntax and Semantics of Linear Schemas to handle more features e.g. functions and procedures.



More Accurate Notions of Dependency

- A new lazy trace semantics for Schemas for general Slicing
- Characterisation of Slicing in terms of this new Semantics
- Investigation of Decidability of Equivalence in terms of the new Semantics
- Extending the Syntax and Semantics of Linear Schemas to handle more features e.g. functions and procedures.
- New algorithms for computing dependency



Plan for the Schemas Project



Publications so Far

- Michael R. Laurence, Sebastian Danicic, Mark Harman, Rob Hierons, and John Howroyd.
Equivalence of conservative, free, linear program schemas is decidable.
Theoretical Computer Science, 290:831–862, January 2003.



Publications so Far

- Michael R. Laurence, Sebastian Danicic, Mark Harman, Rob Hierons, and John Howroyd.
Equivalence of conservative, free, linear program schemas is decidable.
Theoretical Computer Science, 290:831–862, January 2003.
- Sebastian Danicic, Chris Fox, Mark Harman, Robert Mark Hierons, John Howroyd, and Mike Laurence.
Slicing algorithms are minimal for programs which can be expressed as linear, free, liberal schemas.
The Computer Journal, 48(6):737–748, 2005.



Publications so Far

- Michael R. Laurence, Sebastian Danicic, Mark Harman, Rob Hierons, and John Howroyd.
Equivalence of conservative, free, linear program schemas is decidable.
Theoretical Computer Science, 290:831–862, January 2003.
- Sebastian Danicic, Chris Fox, Mark Harman, Robert Mark Hierons, John Howroyd, and Mike Laurence.
Slicing algorithms are minimal for programs which can be expressed as linear, free, liberal schemas.
The Computer Journal, 48(6):737–748, 2005.
- Sebastian Danicic, Mark Harman, Robert Mark Hierons, John Howroyd, and Mike Laurence.
Equivalence of linear, free, liberal, structured program schemas is decidable in polynomial time.
Theoretical Computer Science, 2006.



Mike – Over to you!

