Reinhard Wilhelm

Patrick Cousot

cims.nyu.edu/~pcousot
pcousot@cims.nyu.edu

Farewell Colloquium on the Occasion of Reinhard Wilhelm's 68th birthday
Saarbrüchen, November, 28th 2014
You said Reinhard Wilhelm?
But who is Reinhard Wilhelm?
But who is Reinhard Wilhelm?
But who is Reinhard Wilhelm?

You have understood the limitations of \texttt{``Big data''} and \texttt{``Advanced machine learning''}
But who is Reinhard Wilhelm?

1. You have understood the limitations of "Big data" and "Advanced machine learning"

2. This is THE Reinhard Wilhelm:

Reinhard Wilhelm

Reinhard Wilhelm is a German computer scientist. Wikipedia

Born: June 5, 1946 (age 68), Finnentrop, Germany
Education: University of Münster
But who is Reinhard Wilhelm?

1. You have understood the limitations of "Big data" and "Advanced machine learning"

2. This is THE Reinhard Wilhelm:

   [Image of Reinhard Wilhelm]

   Reinhard Wilhelm is a German computer scientist. Wikipedia
   Born: June 5, 1946 (age 68), Finntentrop, Germany
   Education: University of Münster

   sorry, this was 2 months ago on Wikipedia, thanks to the true Reinhard Wilhelm for updating his picture last month!
But who is Reinhard Wilhelm?

1. You have understood the limitations of ``Big data'' and ``Advanced machine learning''

2. This is THE Prof. em. Dr. Dr. h.c. Reinhard Wilhelm:

   Reinhard Wilhelm

   **Born**
   5 June 1946 (age 68)
   Finnentrop, Germany

   **Fields**
   Computer Scientist

   **Institutions**
   Saarland University

   **Alma mater**
   University of Münster, Stanford University, Technical University Munich

   **Known for**
   compiler technology

   **Notable awards**
   Konrad Zuse Medal (2009)
   Merit Cross on Ribbon (2010)
   ACM Distinguished Service Award (2011)
There is only one, the proof is by Google
There is only one, the proof is by Google
And more …
Great Achievements
Great Achievements of Reinhard (I)
Great Achievements of Reinhard (II)
Great Achievements of Reinhard (II)
Not forgetting... Wilhelm Reinhard
Great Achievements of Reinhard (III)
Great Achievements of Reinhard (IV)

Reinhard Wilhelm

Contributions to Logics and Programming. Programming Logics 2013: 1-18


Reinhard Wilhelm, Jan Reineke: Embedded systems: Many cores - Many problems. SIES 2012: 176-180


Daniel Grund, Jan Reineke, Reinhard Wilhelm: A Template for Predictability Definitions with Supporting Evidence. PPES 2011: 22-31


Weidenbach, the special section on rigorous embedded systems design.


Reinhard Wilhelm, Jan Reineke: Embedded systems: Many cores - Many problems. SIES 2012: 176-180


Benoit Triquet

Benoit Triquet


Farewell Colloquium on the Occasion of Reinhard Wilhelm’s 68th birthday, Saarbrücken, November 28, 2014


Reinhard Wilhelm, Matteo Maffei: Ubiquitous Verification of Ubiquitous Systems. SEUS 2010: 47-58


Thomas Heinz, Reinhard Wilhelm: Towards device emulation code generation. LCTES 2009: 109-118

Reinhard Wilhelm: The PROMPT design principles for predictable multi-core architectures. SCOPES 2009: 31-32

Niklas Holsti, Guillaume Bernat, Christian Ferdinand, Peter P. Puschner, Reinhard Wilhelm: Teaching WCET Analysis in Academia and Industry: A Panel Discussion. WCET 2009


Reinhard Wilhelm, Björn Wachter: Abstract Interpretation with Applications to Timing Validation. CAV 2008: 22-36


Verification: Infinite-State Model Checking and Static Program Analysis 2006


Dierk Johannes, Raimund Seidel, Reinhard Wilhelm: *Algorithm animation using shape analysis: visualising abstract...

**2005**


Dierk Johannes, Raimund Seidel, Reinhard Wilhelm: *Algorithm animation using shape analysis: visualising abstract...
Farewell Colloquium on the Occasion of Reinhard Wilhelm’s 68th birthday, Saarbrücken, November 28, 2014


Analysis of the Worst-Case Execution Time by Abstract Interpretation of Executable Code. ASWSD 2004: 1-14

Component-Wise Instruction-Cache Behavior Prediction. ATVA 2004: 211-229

Static Program Analysis via 3-Valued Logic. CAV 2004: 15-30

Requirements for and Design of a Processor with Predictable Timing. Design of Systems with Predictable Behaviour 2004

Abstracts Collection. Design of Systems with Predictable Behaviour 2004

Design for Time-Predictability. Design of Systems with Predictable Behaviour 2004


Formal Analysis of Processor Timing Models. SPIN 2004: 1-4

Why AI + ILP Is Good for WCET, but MC Is Not, Nor ILP Alone. VMCAI 2004: 309-322


Explanations: Visualizing Abstract States and Invariants.


Verifying Temporal Heap Properties Specified via Evolution Logic. ESOP 2003: 204-222

Run-Time Guarantees for Real-Time Systems. FORMATS 2003: 166-167

Validation of the Zeitverhaltens von kritischer Echtzeit-Software. GI Jahrestagung (1) 2003: 335-339

Convenient User Annotations for a WCET Tool. WCET 2003: 17-20

Industrial Requirements for WCET Tools - Answers to the ARTIST Questionnaire. WCET 2003: 39-43


Software Visualization 2001: 381-394


Reinhard Wilhelm, Shmuel Sagiv, Thomas W. Reps: Shape Analysis. CC 2000: 1-17


Shmuel Sagiv, Thomas W. Reps, Reinhard Wilhelm: Parametric Shape Analysis via 3-Valued Logic. POPL 1999: 105-118


Implementing 2DT on a Multiprocessor. 

Yosi Ben-Asher

Spuroptimierung. 

Optimierung ausnahmebehafteter Programme durch 

Peter G. Bouillon

construction, génération.

Reinhard Wilhelm

CLaX - A Visualized Compiler.

Georg Sander

2DT-FP: A parallel functional programming language on 

Yosi Ben-Asher

Reinhard Wilhelm

ISBN 3-406-40338-7

Amwendungen - Perspektiven [Forum "Perspektiven der 

1996

Martin Alt

POPL 1996

Shmuel Sagiv

1996

1996

Christian Ferdinand

Reinhard Wilhelm

Transformation Development: TrafoLa-H Subsystem. 

PROSPECTRA Book 1993: 539-576

1995

Reinhard Wilhelm, Dieter Maurer: 


Reinhard Wilhelm, Dieter Maurer: 


1994

Christian Ferdinand, Helmut Seidl, Reinhard Wilhelm: 


[c34] Georg Sander, Martin Alt, Christian Ferdinand, Reinhard Wilhelm: 

CLaX - A Visualized Compiler. Graph Drawing 1995: 459-462

[c35] Martin Alt, Christian Ferdinand, Florian Martin, Reinhard Wilhelm: 

Cache Behavior Prediction by Abstract Interpretation. SAS 1996: 52-66

[c36] Shmuel Sagiv, Thomas W. Reps, Reinhard Wilhelm: 

Solving Shape-Analysis Problems in Languages with Destructive Updating. 

POPL 1996: 16-31

[c32] Yosi Ben-Asher, Gudula Rünger, Reinhard Wilhelm, Assaf Schuster: 

Implementing 2DT on a Multiprocessor. CC 1994: 113-127

[c37] Reinhard Wilhelm, Martin Alt, Florian Martin, Martin Raber: 

Parallel Implementation of Functional Languages. LOMAPS 1996: 279-295

1993

Yosi Ben-Asher, Gudula Rünger, Assaf Schuster, Reinhard Wilhelm: 

2DT-FP: An FP Based Programming Language for E

[c39] Volker Claus, Reinhard Wilhelm: 

Einleitung. Perspektiven der Informatik 1993: 9-12


Perspektiven der Informatik 1993: 84-85


Perspektiven der Informatik 1993: 86-90


[c27] Yosi Ben-Asher, Gudula Rünger, Assaf Schuster, Reinhard Wilhelm: 

2DT-FP: An FP Based Programming Language for Efficient Parallel Programming of Multiprocessor Networks. PARLE 1993: 42-55


PROSPECTRA Book 1993: 539-576

1992

Yosi Ben-Asher, Gudula Rünger, Reinhard Wilhelm: 

Solving Shape-Analysis Problems in Languages with Destructive Updating. 

POPL 1996: 16-31

Christian Ferdinand, Helmut Seidl, Reinhard Wilhelm: 

Tree Automata for Code Selection. Code Generation 1991: 30-50

Christian Ferdinand, Helmut Seidl, Reinhard Wilhelm: 

Tree Automata for Code Selection. Code Generation 1991: 30-50

[c24] Christian Ferdinand, Helmut Seidl, Reinhard Wilhelm: 

Tree Automata for Code Selection. Code Generation 1991: 30-50

1991


Attribute Grammars, Applications and Systems 1991: 151-186


[c21] Reinhard Wilhelm: Attributive Reevaluation in OPTRAN. 

Attribute Grammars, Applications and Systems 1991: 507

Shmuel Sagiv, Nissim Francez, Michael Rodeh, Reinhard Wilhelm: 

A Logic-Based Approach to Data Flow Analysis Problem. PLILP

1990

[c20] Shmuel Sagiv, Nissim Francez, Michael Rodeh, Reinhard Wilhelm: 

A Logic-Based Approach to Data Flow Analysis Problem. PLILP

[c20] Shmuel Sagiv, Nissim Francez, Michael Rodeh, Reinhard Wilhelm: A Logic-Based Approach to Data Flow Analysis Problem. PLILP
Innovative Informations-Infrastrukturen 1988

Programmiersprache HOPE mit Hilfe von Kombinatoren.

Wilhelm M. Baston

1988

Complied Graph Reduction on a Processor Network.

Fritz Müller (1988)

Attribute (Re)evaluation in OPTRAN.

Winfried Thome, Reinhard Wilhelm


1988


Martin Raber, Thomas Remmel, Erwin Hoffmann, Dieter Maurer, Fritz Müller, Hans-Georg Oberhauser, Reinhard Wilhelm: Complied Graph Reduction on a Processor Network. ARCS 1988: 198-212


1990: 277-292

Gerhard Becker, Brigitte Kuhn, Dieter Maurer, Reinhard Wilhelm: \S\TeX - eine interaktive Arbeitsumgebung für \TeX. Innovative Informations-Infrastrukturen 1988: 162-169


Reinhard Wilhelm: Inverse Currying Transformation on Attribute Grammars. POPL 1984: 140-147


Invariance of Approximate Semantics with Respect to Program Transformations. GI Jahrestagung 1981: 1-10


Implementierbarkeit attributer Grammatiken. GI Jahrestagung 1977: 17-36


Code-Optimierung Mittels Attributierter Transformationsgrammatiken. GI Jahrestagung 1974: 257-266

data released under the ODC-BY 1.0 license. See also our legal information page last updated on 2014-11-19 23:07 CET by the dblp team
Great sustainable productivity
Too much to read so let’s have numbers

### Citation indices

<table>
<thead>
<tr>
<th></th>
<th>All</th>
<th>Since 2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citations</td>
<td>7053</td>
<td>3497</td>
</tr>
<tr>
<td>h-index</td>
<td>35</td>
<td>27</td>
</tr>
<tr>
<td>i10-index</td>
<td>75</td>
<td>41</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>2006</th>
<th>2007</th>
<th>2008</th>
<th>2009</th>
<th>2010</th>
<th>2011</th>
<th>2012</th>
<th>2013</th>
<th>2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citations</td>
<td><img src="image" alt="Bar Chart" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **The hit parade (even cited before published!):**

<table>
<thead>
<tr>
<th>Title</th>
<th>Cited by</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parametric shape analysis via 3-valued logic</td>
<td>1047</td>
<td>2002</td>
</tr>
<tr>
<td>M. Sagiv, T. Reps, R. Wilhelm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACM Transactions on Programming Languages and Systems (TOPLAS) 24 (3),</td>
<td></td>
<td></td>
</tr>
<tr>
<td>217-298</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The worst-case execution-time problem—overview of methods and survey of tools</td>
<td>1004</td>
<td>2008</td>
</tr>
<tr>
<td>R. Wilhelm, J. Engblom, A. Ermendahl, N. Holsti, S. Thesing, D. Whalley, ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACM Transactions on Embedded Computing Systems (TECS) 7 (3), 36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solving shape-analysis problems in languages with destructive updating</td>
<td>470</td>
<td>1998</td>
</tr>
<tr>
<td>M. Sagiv, T. Reps, R. Wilhelm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACM Transactions on Programming Languages and Systems (TOPLAS) 20 (1),</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1-50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The flop:

<table>
<thead>
<tr>
<th>An abstract machine for an object-oriented language with top-level classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Böschen, C Fecht, AV Hense, R Wilhelm</td>
</tr>
</tbody>
</table>

Total citations | Cited by 2
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>2</td>
</tr>
</tbody>
</table>

© P Cousot
The flop:

An abstract machine for an object-oriented language with top-level classes
C Böschen, C Fecht, AV Hense, R Wilhelm

Total citations: Cited by 2

yes, but cited 11 years before the pretend publication date!
Science
Main contributions

• Coming number one in static analysis, world-wide:

Reinhard Wilhelm
Professor of Computer Science, Saarland University
Verified email at cs.uni-saarland.de
Cited by 7053
embedded systems, compilers, static program analysis
What is static analysis?

Static program analysis

From Wikipedia, the free encyclopedia

Static program analysis is the analysis of computer software that is performed without actually executing programs (analysis performed on executing programs is known as dynamic analysis).

by a computer at least the static analyzer must execute!
A short introduction to static analysis
The very first static analysis

Brahmagupta (Sanskrit: ब्रह्मगुप्त; listen (help·info)) (598–c.670 CE) was an Indian mathematician and astronomer who wrote two important works on Mathematics and Astronomy: the Brāhmaṇaśuṭasiddhānta (Extensive Treatise of Brahma) (628), a theoretical treatise, and the Khaṇḍakādyaka, a more practical text.
The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative;
The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative;

• The abstraction is that you do not (always) need to known the absolute value of the arguments to know the sign of the result;
The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative;

- The abstraction is that you do not (always) need to known the absolute value of the arguments to know the sign of the result;
- Sometimes imprecise (don’t know the sign of the sum of a positive and a negative)
The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative;

- The **abstraction** is that you do not (always) need to known the **absolute value** of the arguments to know the **sign** of the result;
- Sometimes **imprecise** (don’t know the sign of the sum of a positive and a negative)
- **Useful in practice** (if you know what to do when you don’t know the sign)
The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative;

• The abstraction is that you do not (always) need to known the absolute value of the arguments to know the sign of the result;

• Sometimes imprecise (don’t know the sign of the sum of a positive and a negative)

• Useful in practice (if you know what to do when you don’t know the sign)

• e.g. in compilation: do not optimize (a division by 2 into a shift when positive(*)

(*) Unless processor uses 2’s complement and can shift the sign.
The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative; [...]

18.32. A negative minus zero is negative, a positive [minus zero] positive; zero [minus zero] is zero. When a positive is to be subtracted from a negative or a negative from a positive, then it is to be added.
The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative; [...]  

18.32. A negative minus zero is negative, a positive [minus zero] positive; zero [minus zero] is zero. When a positive is to be subtracted from a negative or a negative from a positive, then it is to be added.  

18.33. The product of a negative and a positive is negative, of two negatives positive, and of positives positive; the product of zero and a negative, of zero and a positive, or of two zeros is zero.
The rule of signs by Brahmagupta (628)

18.30. [The sum] of two positives is positives, of two negatives negative; [...]

18.32. A negative minus zero is negative, a positive [minus zero] positive; zero [minus zero] is zero. When a positive is to be subtracted from a negative or a negative from a positive, then it is to be added.

18.33. The product of a negative and a positive is negative, of two negatives positive, and of positives positive; the product of zero and a negative, of zero and a positive, or of two zeros is zero.

18.34. A positive divided by a positive or a negative divided by a negative is positive; **a zero divided by a zero is zero**; a positive divided by a negative is negative; a negative divided by a positive is [also] negative.

wrong
The rule of signs by Michel Sintzoff (1972)

For example, \( a \times a + b \times b \) yields the value 25 when \( a \) is 3 and \( b \) is -4, and when \( + \) and \( \times \) are the arithmetic multiplication and addition. But \( a \times a + b \times b \) yields always the object "pos" when \( a \) and \( b \) are the objects "pos" or "neg", and when the valuation is defined as follows:

\[
\begin{align*}
\text{pos} + \text{pos} &= \text{pos} \\
\text{pos} + \text{neg} &= \text{pos}, \text{neg} \\
\text{neg} + \text{pos} &= \text{pos}, \text{neg} \\
\text{neg} + \text{neg} &= \text{neg} \\
V(p+q) &= V(p) + V(q) \\
V(p \times q) &= V(p) \times V(q) \\
V(0) &= V(1) = \ldots = \text{pos} \\
V(-1) &= V(-2) = \ldots = \text{neg}
\end{align*}
\]

The valuation of \( a \times a + b \times b \) yields "pos" by the following computations:

\[
\begin{align*}
V(a) &= \text{pos}, \text{neg} \\
V(b) &= \text{pos}, \text{neg} \\
V(a \times a) &= \text{pos} \times \text{pos}, \text{neg} \times \text{neg} = \text{pos}, \text{pos} = \text{pos} \\
V(b \times b) &= \text{pos} \times \text{pos}, \text{neg} \times \text{neg} = \text{pos}, \text{pos} = \text{pos} \\
V(a \times a + b \times b) &= V(a \times a) + V(b \times b) = \text{pos} + \text{pos} = \text{pos}
\end{align*}
\]

This valuation proves that the result of \( a \times a + b \times b \) is always positive and hence allows to compute its square root without any preliminary dynamic test on its sign. On the other hand, the
The rule of signs by Michel Sintzoff (1972)

For example, \( a \times a + b \times b \) yields the value 25 when \( a \) is 3 and \( b \) is \(-4\), and when \( + \) and \( \times \) are the arithmetic multiplication and addition. But \( a \times a + b \times b \) yields always the object "pos" when \( a \) and \( b \) are the objects "pos" or "neg", and when the valuation is defined as follows:

\[
\begin{align*}
\text{pos} + \text{pos} &= \text{pos} & \text{pos} \times \text{pos} &= \text{pos} \\
\text{pos} + \text{neg} &= \text{pos}, \text{neg} & \text{pos} \times \text{neg} &= \text{neg} \\
\text{neg} + \text{pos} &= \text{pos}, \text{neg} & \text{neg} \times \text{pos} &= \text{neg} \\
\text{neg} + \text{neg} &= \text{neg} & \text{neg} \times \text{neg} &= \text{pos} \\
V(p+q) &= V(p) + V(q) & V(p \times q) &= V(p) \times V(q)
\end{align*}
\]

\[
\begin{align*}
V(0) &= V(1) = \ldots = \text{pos} \\
V(-1) &= V(-2) = \ldots = \text{neg}
\end{align*}
\]

The valuation of \( a \times a + b \times b \) yields "pos" by the following computations:

\[
\begin{align*}
V(a) &= \text{pos}, \text{neg} & V(b) &= \text{pos}, \text{neg} \\
V(a \times a) &= \text{pos} \times \text{pos}, \text{neg} \times \text{neg} & V(b \times b) &= \text{pos} \times \text{pos}, \text{neg} \times \text{neg} \\
&= \text{pos}, \text{pos} = \text{pos} & = \text{pos}, \text{pos} = \text{pos} \\
V(a \times a + b \times b) &= V(a \times a) + V(b \times b) = \text{pos} + \text{pos} = \text{pos}
\end{align*}
\]

This valuation proves that the result of \( a \times a + b \times b \) is always positive and hence allows to compute its square root without any preliminary dynamic test on its sign. On the other hand, the
The rule of signs by Michel Sintzoff (1972)

For example, \( a \times a + b \times b \) yields the value 25 when \( a \) is 3 and \( b \) is -4, and when \( + \) and \( \times \) are the arithmetic multiplication and addition. But \( a \times a + b \times b \) yields always the object "pos" when \( a \) and \( b \) are the objects "pos" or "neg", and when the valuation is defined as follows:

\[
\begin{align*}
pos + pos &= pos \\
pos + neg &= pos, neg \\
neg + pos &= pos, neg \\
neg + neg &= neg
\end{align*}
\]

\[
V(p+q) = V(p) + V(q)
\]

\[
\begin{align*}
V(0) &= V(1) = \ldots = pos \\
V(-1) &= V(-2) = \ldots = neg
\end{align*}
\]

The valuation of \( a \times a + b \times b \) yields "pos" by the following computations:

\[
\begin{align*}
V(a) &= pos, neg \\
V(b) &= pos, neg \\
V(a \times a) &= pos \times pos, neg \times neg \\
V(b \times b) &= pos \times pos, neg \times neg \\
&= pos, pos = pos \\
&= pos, pos = pos \\
V(a \times a + b \times b) &= V(a \times a) + V(b \times b) = pos + pos = pos
\end{align*}
\]

This valuation proves that the result of \( a \times a + b \times b \) is always positive and hence allows to compute its square root without any preliminary dynamic test on its sign. On the other hand, the
2 Example — Rules-of-Sign Analysis

Problem: Determine at each program point the sign of the values of all variables of numeric type.

Example program:
1: x = 0;
2: y = 1;
3: while (y > 0) do
4: y = y + x;
5: x = x + (-1);

Program representation as control-flow graphs

The analysis should "bind" program variables to elements in Signs. So, the abstract domain is \( D = (\text{Vars} \rightarrow \text{Signs}) \perp \), a Sign-environment. \( \perp \in D \) is the function mapping all arguments to \{\}. The partial order on \( D \) is \( D_1 \subseteq D_2 \) iff \( D_1 = \perp \) or \( D_1 \ x \subseteq D_2 \ x \ (x \in \text{Vars}) \).

Intuition?

How is a solution found?
Iterating until a fixed-point is reached

We construct the abstract domain for single variables starting with the lattice \( \text{Signs} = 2^{\{-,0,\} \times \{0,+,\}} \) with the relation "\( \subseteq \)" = "\( \subseteq \)".

The analysis should "bind" program variables to elements in Signs. So, the abstract domain is \( D = (\text{Vars} \rightarrow \text{Signs}) \perp \), a Sign-environment. \( \perp \in D \) is the function mapping all arguments to \{\}.

The partial order on \( D \) is \( D_1 \subseteq D_2 \) iff \( D_1 = \perp \) or \( D_1 \ x \subseteq D_2 \ x \ (x \in \text{Vars}) \).

Intuition?

\( D_1 \) is at least as precise as \( D_2 \) since \( D_2 \) admits at least as many signs as \( D_1 \).

How did we analyze the program?

In particular, how did we walk the lattice for \( y \) at program point 5?

How is a solution found?
Iterating until a fixed-point is reached

Idea:

- We want to determine the sign of the values of expressions.
- For some sub-expressions, the analysis may yield \{+, -, 0\}, which means, it couldn’t find out.  

Idea:

- We want to determine the signs of the values of expressions.
- For some sub-expressions, the analysis may yield \{+, -, 0\}, which means, it couldn’t find out.
- We replace the concrete operators \( \Box \) working on values by abstract operators \( \Box^2 \) working on signs.
The rule of signs by Reinhard Wilhelm (2012/13)

Idea:

- We want to determine the signs of the values of expressions.
- For some sub-expressions, the analysis may yield \{+, -, 0\}, which means, it couldn’t find out.
- We replace the concrete operators \(\otimes\) working on values by abstract operators \(\otimes\) working on signs:
  - The abstract operators allow to define an abstract evaluation of expressions:
    \[
    [\varepsilon] : (\text{Vars} \rightarrow \text{Signs}) \rightarrow \text{Signs}
    \]

Determining the sign of expressions in a Sign-environment works as follows:

\[
[c] D = \begin{cases} 
\{+\} & \text{if } c > 0 \\
\{-\} & \text{if } c < 0 \\
\{0\} & \text{if } c = 0 
\end{cases}
\]

\[
[v] = D(v)
\]

\[
[e_1 \otimes e_2] D = [e_1] D \otimes [e_2] D
\]

\[
[e] D = \square[e] D
\]

Abstract operators working on signs (Addition)

<table>
<thead>
<tr>
<th>(\oplus)</th>
<th>({0})</th>
<th>({+})</th>
<th>({-})</th>
<th>({+, -})</th>
<th>({0, +})</th>
<th>({-, 0, +})</th>
</tr>
</thead>
<tbody>
<tr>
<td>({0})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({+})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({-})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({+, -})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({0, +})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
</tbody>
</table>

Abstract operators working on signs (Multiplication)

<table>
<thead>
<tr>
<th>(\times)</th>
<th>({0})</th>
<th>({+})</th>
<th>({-})</th>
<th>({+, -})</th>
<th>({0, +})</th>
<th>({-, 0, +})</th>
</tr>
</thead>
<tbody>
<tr>
<td>({0})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({+})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({-})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({+, -})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({0, +})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
</tbody>
</table>

Abstract operators working on signs (unary minus)

<table>
<thead>
<tr>
<th>(-)</th>
<th>({0})</th>
<th>({+})</th>
<th>({-})</th>
<th>({+, -})</th>
<th>({0, +})</th>
<th>({-, 0, +})</th>
</tr>
</thead>
<tbody>
<tr>
<td>({0})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({+})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({-})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({+, -})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
<tr>
<td>({0, +})</td>
<td>({0})</td>
<td>({+})</td>
<td>({-})</td>
<td>({+, -})</td>
<td>({0, +})</td>
<td>({-, 0, +})</td>
</tr>
</tbody>
</table>

Working an example:

\[
\varphi D = \{x \mapsto \{+, y \mapsto \{+\}\}
\]

\[
\]

\[
= \{+\} + \{+\}
\]

\[
= \{+\}
\]

\[
[x + (-y)] D = (\{+\} + \{-[-y]\} D)
\]

\[
= \{+\} + \{-[-y]\}
\]

\[
= \{+\} + \{-\}
\]

\[
= \{+, -, 0\}
\]

Thus, we obtain the following effects of edges \([lab]\) :

\[
[.] D = D
\]

\[
[\text{true}(e)] D = D
\]

\[
[\text{false}(e)] D = D
\]

\[
[x = e_1] D = D \oplus \{x \mapsto \{e_1\} D\}
\]

\[
[x = M[e_1]] D = D \oplus \{x \mapsto \{+, -, 0\}\}
\]

\[
[M[e_1] = e_2] D = D
\]

... whenever \(D \neq \perp\)

---

Attention to details
The rule of signs by Reinhard Wilhelm (2012/13)

Idea:

- We want to determine the signs of the values of expressions.
- For some sub-expressions, the analysis may yield \{+, -, 0\}, which means, it couldn’t find out.
- We replace the concrete operators \(\boxtimes\) working on values by
  abstract operators \(\boxtimes\) working on signs:
- The abstract operators allow to define an abstract evaluation of expressions:

\[ e^\#: (\text{Vars} \rightarrow \text{Signs}) \rightarrow \text{Signs} \]

Determining the sign of expressions in a Sign-environment works as follows:

\[ [c]^D = \begin{cases} 
{+} & \text{if } c > 0 \\
{-} & \text{if } c < 0 \\
{0} & \text{if } c = 0 
\end{cases} \]

\[ [v]^D = D(v) \]
\[ [e_1 \boxtimes e_2]^D = [e_1]^D \boxtimes [e_2]^D \]
\[ [\boxtimes e]^D = \boxtimes [e]^D \]

Abstract operators working on signs (Addition)

\[
+^\#
\begin{array}{ccccccc}
0 & \{0\} & \{+\} & \{-\} & \{-, 0\} & \{-, +\} & \{-, 0, +\} \\
{+} & \{0\} & \{+\} & \{-\} & \{-, 0\} & \{-, +\} & \{-, 0, +\} \\
{-} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, +\} & \{-, 0, +\} \\
{-, 0} & \{-, 0\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
{-, +} & \{-, +\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
{0, +} & \{0\} & \{+\} & \{-\} & \{-, 0\} & \{-, 0\} & \{-, 0, +\} \\
{-, 0, +} & \{-, 0, +\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
\end{array}
\]

Abstract operators working on signs (Multiplication)

\[
\times^\#
\begin{array}{ccccccc}
0 & \{0\} & \{+\} & \{-\} & \{-, 0\} & \{-, +\} & \{-, 0, +\} \\
{+} & \{0\} & \{+\} & \{-\} & \{-, 0\} & \{-, +\} & \{-, 0, +\} \\
{-} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, +\} & \{-, 0, +\} \\
{-, 0} & \{-, 0\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
{-, +} & \{-, +\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
{0, +} & \{0\} & \{+\} & \{-\} & \{-, 0\} & \{-, 0\} & \{-, 0, +\} \\
{-, 0, +} & \{-, 0, +\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
\end{array}
\]

Abstract operators working on signs (unary minus)

\[
-^\#
\begin{array}{ccccccc}
0 & \{0\} & \{+\} & \{-\} & \{-, 0\} & \{-, +\} & \{-, 0, +\} \\
{+} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, +\} & \{-, 0, +\} \\
{-} & \{+\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
{-, 0} & \{-, 0\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
{-, +} & \{-, +\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
{0, -} & \{0\} & \{+\} & \{-\} & \{-, 0\} & \{-, 0\} & \{-, 0, +\} \\
{-, 0, -} & \{-, 0, +\} & \{-\} & \{0\} & \{+\} & \{-, 0\} & \{-, 0, +\} \\
\end{array}
\]

Working an example:

\[ \forall D = \{x \mapsto \{+, y \mapsto \{+\}\} \}
\]
\[ [x + 7]^D = [x]^D +^\# [7]^D = \{+\} +^\# [+] = [+] \]
\[ [x + (-y)]^D = [+] +^\# (-[y]^D) = [+] +^\# (\{-\}) = [+] +^\# [+] \]
\[ [x = e]^D = D \oplus \{x \mapsto [e]^D\} \]
\[ [x = M[e]]^D = D \oplus \{x \mapsto \{+, -, 0\}\} \]
\[ [M[e_1] = e_2]^D = D \]

Thus, we obtain the following effects of edges \([lab]^D\):

\[ D \neq \perp \]

if the program does not terminate isn’t it correct to say that \(x\) is 0 upon its termination?

Attention to details

if the program does not terminate isn’t it correct to say that \(x\) is 0 upon its termination?
That’s where you recognize a great scientist: make simple what is complicated!
Suggestions for an happy retirement
Have ambitious objectives!
Have ambitious objectives!

- Move Dagstuhl close to an airport (or an airport close to Dagstuhl)
Remain active in science!
Remain active in science!

- Start working on cyberimbedded systems
Remain active in science!

- Start working on cyberimbedded systems
- Consider dynamic methods for static analysis
Remain active in science!

- Start working on **cyberimbedded systems**
- Consider **dynamic methods for static analysis**
- Write a book on **decompilation**
Remain active in science!

- Start working on cyberimbedded systems
- Consider dynamic methods for static analysis
- Write a book on decompilation, by duality
Time for a serious conclusion
Thanks a lot for 30 years of friendship
Thanks a lot for 30 years of friendship, with lots of problems!
Thanks a lot for 30 years of friendship, with lots of problems!
The End, thank you
The beginning, thank you
The beginning, thank you of retirement
The beginning, thank you of retirement
of retirement

The beginning, thank you
The beginning, thank you

of retirement