Overview of the scientific achievements of DAEDALUS

Seminar for industrial users, Saarbrücken, 27 Sep. 2002

Patrick Cousot
ENS, 45 rue d’Ulm, 75230 Paris cedex 05, France
www.di.ens.fr/cousot
Patrick.Cousot@ens.fr
Formal methods
Static program verification

Program analyzer

Specification

Program
Indecidability of static program verification

Given an input program, a specification, and finite computing resources, any static program verifier will either:

- ask for interactive user help.
- run out of memory, or
- never terminate, or
- answer "I don't know", or
- answer negatively, or
- answer positively, or

which may be a true or a false alarm.

.sources, any static program verifier will either:
Coping with finite computing resources

- Static program verifiers must avoid very long computations:

  - Deductive methods: ask the user to provide a (small) finite model of the program.
  - Abstract interpretation: the analyzer uses an approximate semantics.
  - Model checking: ask the user to provide a (small) finite model of the semantics of the program.

- Semantic interpretation

  - E.g. by providing inductive invariants or assistance in proving theorems.

- Model checking

  - By providing inductive invariants or assistance in proving theorems.

- Deductive methods

  - Ask the user to help the theorem prover...
User interaction versus abstraction (1)

User interaction (deductive methods, model checking) will always succeed positively or negatively for a given program by providing an appropriate proof/model, but:

- Additional maintenance cost (for maintaining both the program and proof/model)
- The human cost for designing the proof/model is prohibitive
- The proof/model is usually not reusable for different programs
- Additional maintenance cost (for designing both the proof/model)

Courses, seminars, theses, publications, and software developments (Second European Workshop on Abstract Interpretation, ECOOP'97, Barcelona, Spain, June 1997).

P. Cousot
Abstraction (abstract interpretation), may partially fail for a program by not providing definite answers to all questions, but:

- Entirely automatic without any user interaction
- Reusable for all programs of a given programming language
- No additional maintenance cost (since the static program analysis is automatic)
- Abstraction (abstract interpretation) may partially fail for a program

User interaction versus abstraction (2)
Abstract Interpretation
An Informal Introduction
An example...

Abstract Interpretation

...thinking tool: the idea of abstraction is central to reasoning

A framework for designing mechanical tools leads to automatic semantics-based formal

Systems/Program manipulation tools...
An infinite set of points:

\[ \ldots \langle 0, 02 \rangle, \langle 20, 02 \rangle, \langle 19, 77 \rangle, \ldots \ldots \]
Abstraction from above:

\{
\cdots, \langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle, \cdots
\}
Effective abstraction from above: Signs

$0 < h$

$0 < x$
Effective abstraction from above: Intervals

\[ x \in [19, 77] \]

\[ y \in [20, 02] \]
Effective abstraction from above: Octagons
Effective abstraction from above: Polyhedra 6

\[
\begin{align*}
19x + 77y & \leq 2002 \\
20x + 02y & \geq 0
\end{align*}
\]

---

Effective abstraction from above: Simple congruences

\[
\begin{align*}
66 \mod 99 &= y \\
77 \mod 19 &= x
\end{align*}
\]
Effective abstraction from above: Linear congruences

\[
\begin{align*}
2x - 1y &\equiv 9 \mod 8 \\
1x + 9y &\equiv 7 \mod 8
\end{align*}
\]}
Effective abstraction from above: Trapezoidal congruences

\[
\begin{align*}
x & \equiv 0 \mod 10 \\
y & \equiv 0 \mod 11
\end{align*}
\]
Ist the operation \( \frac{1}{x+1-y} \) well defined at run-time? 

Concrete semantics: yes
Conservative Approximation

Is the operation $1/(x+1-y)$ well defined at run-time?

Testing: You never know!
Conservative Approximation

Is the operation \( \frac{1}{x+1-y} \) well defined at run-time?

Abstract semantics I: I don’t know

Abstract semantics II: The operation \( \frac{1}{x+1-y} \) well defined at run-time?
Conservative Approximation

Istheoperation $1/(x+y)$ well-defined at run-time?

Abstract semantics 2: Yes

$\bullet$ Is the operation $1/(x+y)$ well defined at run-time?

$\bullet$ Conservative Approximation
Achievements of DAEDALUS
The academic visibility of the DAEDALUS project was worldwide; E.g.: International symposium on static analysis SASS’02, 86 papers submitted worldwide, 32 accepted, 12 papers (co-)authored by members of DAEDALUS teams.
An uncommon number of high-quality software tools have been improved or newly designed; an unforeseen number of these tools will have direct industrial applications in the short or medium term; thanks to the exceptional involvement of the end-user and its high-quality assessment.
A synthetic summary of DEEDELUS achievements

- Tools (to be presented in this seminar)
- Prototypes (prefigurating future tools)
- Basic research (theoretical results paving the way for future practice)

See the Synthetic summary of DEEDELUS achievements on the DEEDELUS website.
An expression of interest on a network of excellence on abstract interpretation (AINoE) has been submitted to FP6.

European lead in Abstract Interpretation
THE END, THANK YOU
Conclusions
Some applications of abstract interpretation are mature for short-term direct industrial applicability.