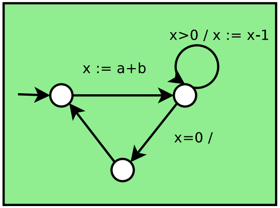
The BINCOA Framework for Binary Code Analysis

Sébastien Bardin, Philippe Herrmann, Jérôme Leroux, Olivier Ly, Renaud Tabary, Aymeric Vincent

CEA LIST (Saclay, Paris)
LABRI (Bordeaux)

Binary code analysis

Model



Source code

```
int foo(int x, int y) {
 int k= x;
 int c=y;
 while (c>0) do {
   k++;
   c--;}
 return k;
}
```

Assembly

```
_start:
  load  A 100
  add B A
  cmp B 0
  jle label

label:
  move @100 B
```

Executable

```
ABFFF780BD70696CA101001BDE45
145634789234ABFFE678ABDCF456
5A2B4C6D009F5F5D1E0835715697
145FEDBCADACBDAD459700346901
3456KAHA305G67H345BFFADECAD3
00113456735FFD451E13AB080DAD
344252FFAADBDA457345FD780001
FFF22546ADDAE989776600000000
```

Binary code analysis at a glimpse

Recent research field

[Codesurfer/x86, SAGE, Jakstab, Osmose, TraceAnalyzer, McVeto, Vine, BAP]

Many promising applications

- off-the-shelf components (including libraries)
- mobile code (including malware)
- third-party certification

Advantages over source-code analysis

- always available
- no “compilation gap”
- allows precise quantitative analysis (ex : wcet)

Very challenging

- conceptual challenges
- practical issues

Practical issues

Engineering issue : many different (large) ISAs

- supporting a new ISA : time-consuming, error-prone, tedious
- consequence : each tool support only a few ISAs (often one !)

Semantic issue : each tool comes with its own formal(?) model

- exact semantics seldom available
- modelling hypothesises often unclear

Consequences

- lots of redundant engineering work between analysers
- difficult to achieve empiric comparisons
- difficult to combine / reuse tools

The BINary COde Analysis project

French research project (CEA, Uni. Bordeaux 1, Uni. Paris 7)

Propose a common formal model for low-level programs

- Dynamic Bitvector Automata (DBA)

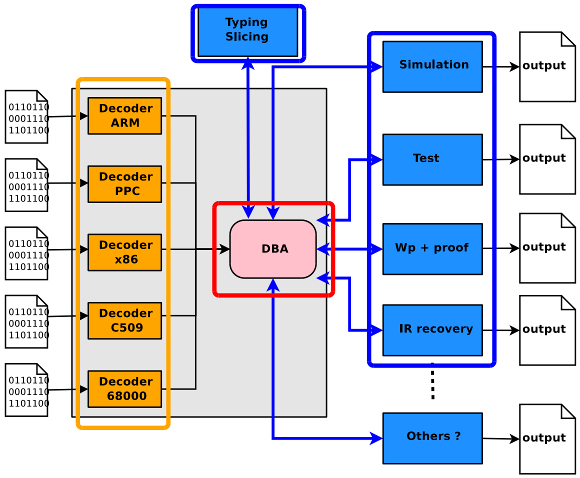
Provide basic open-source tool support

- basic DBA manipulation
- *(future) front-ends from x86, PPC, ARM*

Develop (complementary) binary-level analysers

- OSMOSE (CEA), TraceAnalyzer (CEA), Insight (LABRI)

Long-term objective



- Mutualize engineering work
- Common semantic
- Ease collaboration between analyses

Dynamic Bitvector Automata

Main design ideas

- small set of instructions
- concise and natural modelling of common ISAs
- low-level enough to allow bit-precise modelling

Can model : instruction overlapping, return address smashing, endianness, overlapping memory read/write

Limitations : (strong) no self-modifying code, (weak) no dynamic memory allocation, no FPA

Dynamic Bitvector Automata (2)

Extended automata-like formalism

- bitvector variables and arrays of bytes
- all bv sizes statically known, no side-effects
- standard operations from BVA

Feature 1 : Dynamic transitions

- for dynamic jumps

Feature 2 : Directed multiple-bytes read and write operations

- for endianness and word load/store

Feature 3 : Memory zone properties

- for (simple) environment

Dynamic Bitvector Automata (2)

Feature 1 : Dynamic transitions

- some nodes are labelled by an address
- dynamic transitions have no predefined destination
- destination computed dynamically via a target expression

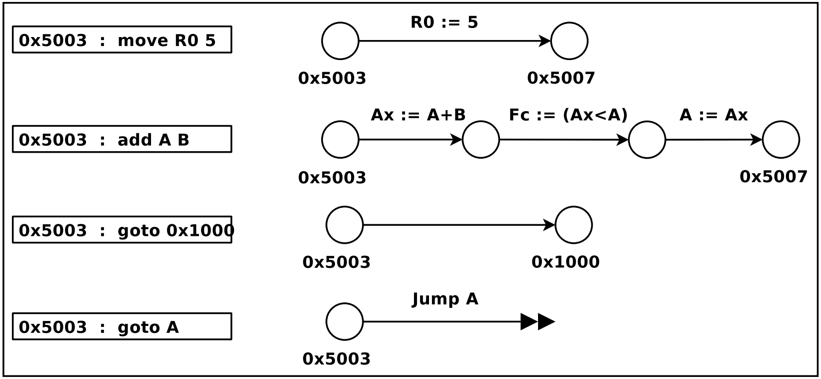
Feature 2 : Directed multiple-bytes read and write operations

- $\mathrm{array}[\mathit{expr}; k^{\#}]$, where $k \in \mathbb{N}$ and $\# \in \{\leftarrow, \rightarrow\}$

Feature 3 : Memory zone properties

- specify special behaviour for some segments of memory
- volatile, write-aborts, write-ignored, read-aborts

Modelling with DBA



Procedure calls / returns : encoded as static / dynamic jumps

Memory zone properties, a few examples : ROM *(write-ignored)*, memory controlled by env *(volatile)*, code section *(write-aborts)*

DBA toolbox

Open-source Ocaml code for basic DBA manipulation

Features

- a datatype for DBAs
- basic “typing” (size checking) over DBAs
- import (export) from (to) a XML format
- DBA simplification (see next)

GPL license, based on xml-light, $\approx$ 3 kloc

DBA toolbox - simplifications

Goal : simplify unduly complex DBAs typically obtained from instruction-wise translation

- useless flag computations / auxiliary variables / etc.

Inspired by standard compilation techniques [peephole, dead code, etc.]

- beware of partial DBAs and dynamic jumps !
- rethink these standard techniques in a partial CFG setting

Results : size reduction of -50% (all instrs), and between -30% and -50% (non-goto instrs)

Binary-level analysers

Osmose (CEA) [ICST-08, STVR-11]

- automatic test data generation (dynamic symbolic execution)
- 75 kloc of OCaml, front-ends : PPC, M6800, Intel c509
- case-studies : programs from aeronautics and energy

> negotiations to become open-source

TraceAnalyzer (CEA, with Franck Védrine) [VMCAI-11]

- safe CFG reconstruction (refinement-based static analysis)
- 29 kloc of C++, front-end : PPC
- case-studies : programs from aeronautics

Insight (LABRI, with Emmanuel Fleury)

- abstract interpretation and weakest precondition
- C++, front-end : x86
- case-studies (on-going) : polymorphic virus analysis

> aims at being open source when the API stabilizes

Conclusion

Current state

- DBAs are a nice formalism to work with
  [improve our former model]
- common semantics allows exchange of information
  [OSMOSE - Traceanalyzer]
- basic DBA support

Ongoing and future work

- open-source front-ends
- extensions of DBAs : support for dynamic memory allocation