Refinement-Based CFG Reconstruction from Unstructured Programs

Sébastien Bardin, Philippe Herrmann, Franck Védrine

CEA LIST
(Paris, France)

Overview

Automatic analysis of executable files

- recent research field [Codesurfer/x86, SAGE, Jakstab, Osmose, etc.]
- many promising applications (COTS, mobile code, malware, etc.)

A key issue : Control-Flow Graph (CFG) reconstruction

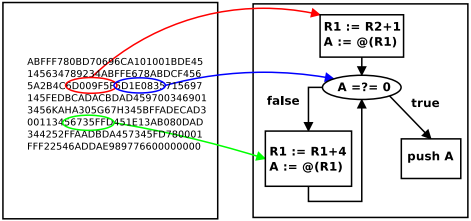
- prior to any other static analysis (SA)
- must be safe : otherwise, other SA unsafe
- must be precise : otherwise, other SA imprecise

This talk is about CFG reconstruction (from executable files)

- safe and precise technique
- based on abstraction-refinement

Input

- an executable file, i.e. an array of bytes
- the address of the initial instruction
- a basic decoder : exec f. $\times$ address $\mapsto$ instruction $\times$ size



Output : CFG of the program

CFG reconstruction (2)

Successor addresses are often syntactically known

- `addr : move a b` → successor at `addr+size`
- `addr : goto 100` → successor at 100
- `addr : ble 100` → successors at 100 and `addr+size`

But not always : successors of `goto a` ?

Dynamic jump is the enemy !

Dynamic jumps are pervasive : introduced by compilers

- `switch`, function pointers, virtual methods, etc.

Safe CFG reconstruction

Need to mix value analysis and standard CFG analysis

- [Balakrishnan-Reps 04, Kinder-Zuleger-Veith 09]

Very difficult to get precise

---

1. A very sensitive analysis : imprecision on jump expressions → extra propagation on false targets → more imprecision on value analysis → possibly more imprecision on jump expressions → ...

- need to be very precise on jump targets

2. Sets of jump targets lack regularity (arbitrary values from compiler)

- standard domains imprecise on jump targets

Related work

CodeSurfer/x86 [Balakrishnan-Reps 04]

- abstract domain : strided intervals (+ affine relationships)
- lots of features : local variable recovery, type recovery, etc.
- abstract domain not suited to sets of jump targets

Jakstab [Kinder-Veith 08]

- abstract domain : sets of bounded cardinality (k-sets)
- precise when the bound K is well-tuned
- not robust to the parameter K : possibly inefficient if K too large, but very imprecise if K not large enough

Contribution

Key observations

- k-sets are the only domain well-suited to precise CFG reconstruction
- for most programs, only a few facts need to be tracked precisely to resolve dynamic jumps
- good candidate for abstraction-refinement

Contribution

- A refinement-based approach to safe CFG reconstruction
- An implementation and a few experiments
- The technique is safe, precise, robust and reasonably efficient

Rest of the talk

Formalisation : unstructured programs and the VAPR problem

The Propagate-and-Refine procedure for VAPR

Experiments

Unstructured programs

Unstructured Programs : $P = (L, V, A, T, l_0)$ where

- $L \subseteq \mathbb{N}$ finite set of code addresses
- V finite set of program variables, A finite set of arrays
- T maps code addresses to instructions
- l_0 initial code address
- instructions : assignments v `:=`e and $a[e_1]$ `:=`e_2, static jumps `goto` l, branching instructions `ite`$(cond, l_1, l_2)$, dynamic jumps `cgoto`(v)

Value Analysis with Precision Requirements

Value Analysis with Precision Requirements (VAPR)

- input : a program P and a set of precision requirements $\mathcal{C}$
- problem : compute an over-approximation M of the collecting semantics of P such that $M \models \mathcal{C}$

Precision Requirement : a (memory) location (l, v), written $\varphi\langle l, v\rangle$

- $M \models \varphi\langle l, v\rangle$ if $M(l, v) \neq \top$

CFG reconstruction can be achieved through VAPR

- add a requirement $\varphi\langle l, v\rangle$ for each $(l, cgoto\ v)$ in P
- rather weak constraint, but sufficient in practise (see after)

The Propagate-and-Refine procedure (PaR) for VAPR

Input : $(P, \mathcal{C})$
Parameter : $Kmax$
Output : an over-approximation M of the collecting semantics of P such that $M \models \mathcal{C}$, or FAIL

Two interleaved-steps : propagation and refinement

Propagation based on k-sets

Each location has its own cardinality bound ($\leq Kmax$)

Refinement : done by increasing some cardinality bounds

Propagation : original features

Cardinality bounds : abstract values downcast to destination bound

- role : lose information, increase efficiency

$\top$-labels to track initial precision losses (ipl)

- $\top_{init}$: input $\top$-values, $\top_{\langle c_1,\ldots,c_q\rangle}$: $\top$-abstraction of $\{c_1,\ldots,c_q\}$
- dedicated propagation rules : $\top_{init}$ and $\top_{\langle\ldots\rangle}$ "stay in place"
- role : pinpoint ipl, give clue for correction

Transitions involving faulty locations are not fired

- role : avoid noise propagation

Update a journal of the computation

- records alias values, jump values and branches that have been fired during propagation
- role : prune irrelevant backward data dependencies

Refinement

For each faulty location, find a set of possible ipl

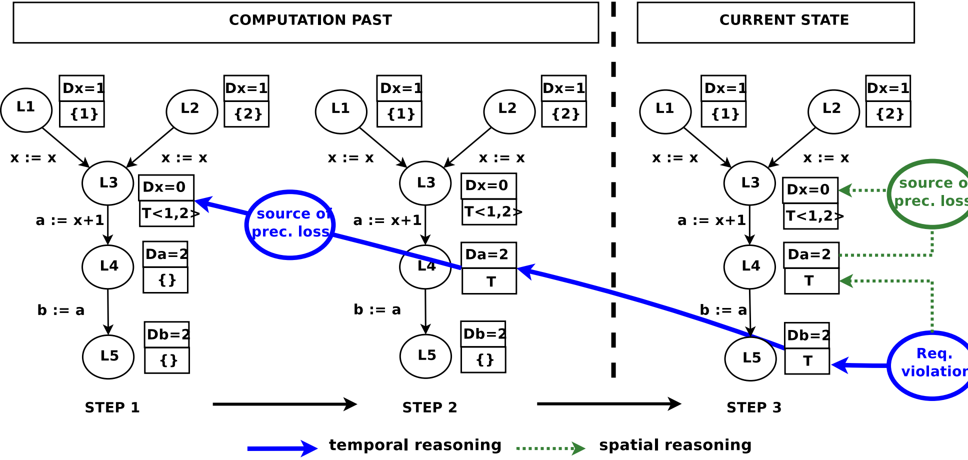
- follows backward data dependencies, guided by $\top$-labels
- stop on ipl : $\top_{init}$ and $\top_{\langle c_1,\ldots,c_q \rangle}$
- data dependencies pruned wrt the journal (cgoto, alias)

Try to "correct" every ipl

- $\top_{init}$ cannot be avoided
- $\top_{\langle c_1,\ldots,c_q \rangle}$ may be avoided if $q \leq Kmax$ (set local bound to q)

If no domain update then fail, else restart propagation with new domains

Intuition



COMPUTATION PAST
CURRENT STATE
L1
Dx=1
{1}
L2
Dx=1
{2}
x := x
x := x
L3
Dx=0
T<1,2>
a := x+1
L4
Da=2
{}
b := a
L5
Db=2
{}
STEP 1
source of prec. loss
L1
Dx=1
{1}
L2
Dx=1
{2}
x := x
x := x
L3
Dx=0
T<1,2>
a := x+1
L4
Da=2
T
b := a
L5
Db=2
{}
STEP 2
L1
Dx=1
{1}
L2
Dx=1
{2}
x := x
x := x
L3
Dx=0
T<1,2>
a := x+1
L4
Da=2
T
b := a
L5
Db=2
T
source of prec. loss
Req. violation
STEP 3
temporal reasoning
spatial reasoning

Properties of PaR

Soundness and termination : PaR(P, $\mathcal{C}$) terminates and is sound, i.e. it returns either FAIL or a safe approximation M of the collecting semantics of P such that $M \models \mathcal{C}$

Complexity : PaR(P, $\mathcal{C}$) runs in polynomial-time

Relative completeness : PaR is relatively complete if PaR($P, \mathcal{C}$) with parameter *Kmax* returns successfully when the forward k-set propagation with parameter *Kmax* does.

- no relative completeness in the general case
  [mainly because of control dependencies]
- relative completeness for a non trivial subclass [see the paper]

Experiments

Implementation : CFG reconstruction from 32-bit PowerPC (PPC)
Only a preliminary implementation

Test bench

- T1 : 12 small hand-written C programs compiled with gcc. From 60 to 1000 PPC instructions
- T2 : real-life embedded program (aeronautic) : 32,000 instructions, 51 dynamic jumps, up to 16 targets for one jump

Some results (1)

Precision

- no target evaluates to $\top$
- on T1 only 7% of false targets
  (k-set 7%, perfect I : 4300%, perfect I+C : 400%)
- on T2, only 7% of false targets
  (k-set : 1.5%)

Robustness : results independent of *Kmax* (if large enough)

Efficiency : between 1x and 3x faster than adequate k-set propag

- lots of redundant work from one refinement step to the other
- can probably be improved

Some results (2)

Locality

- max-k always very close to max # targets
- average-k always low : between 1.08 and 1.18

Scalability : PaR needs 18 minutes for T2 (32 kl)

- ok for a preliminary implementation
- already sufficient for some industrial application
- however (as expected) procedure inlining is an issue

We investigate safe CFG reconstruction from executable files

Results

- a refinement-based procedure to solve VAPR problems
- leads to a safe, precise, robust and reasonably efficient CFG reconstruction
- both theoretical and empirical evidence

Future work

- better implementation and more experiments [dynamic alloc]
- extensions to other abstract domains, optimisations
- investigate other applications of VAPR