Multi-architecture Value Analysis for Machine Code

H. Cassé, IRIT, université de Toulouse
WCET, June, 9th 2013, Paris

Université de Toulouse
Université Paul Sabatier TOULOUSE III

W-SEPT
WCET: **SE**mantics, **P**recision and **T**raceability

Introduction

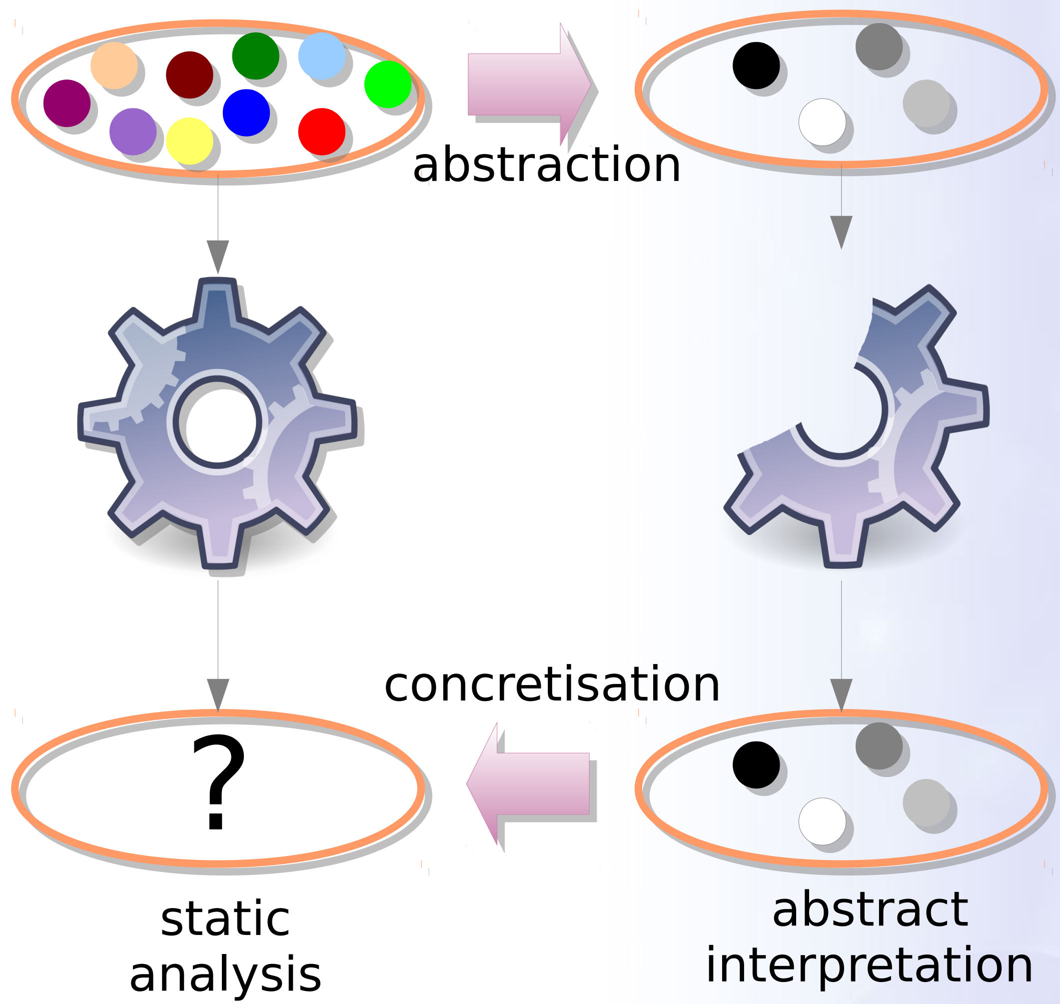
- computing WCET by static analysis
  - survey the behaviour in the hardware
  - handle the program in machine code
- value analysis
  - provide an expression
  - for each register / memory cell of the state
  - at each point of the program
- more and more required
  - resolve complex control flow, loop bound determination
  - data cache analysis
  - infeasible path analysis

Plan

- Introduction
- **Context**
- Language Definition
- Use Cases
- Conclusion

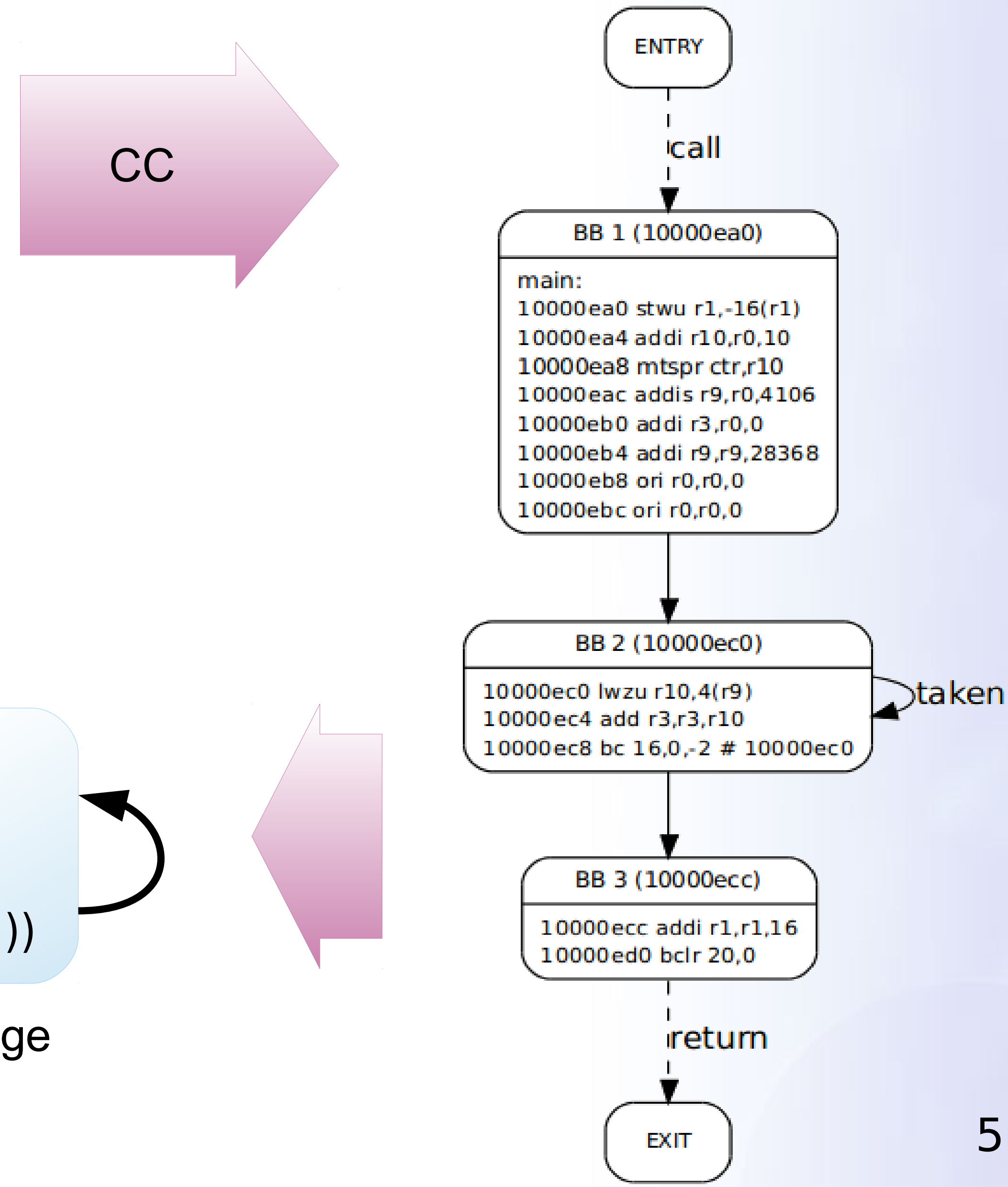
Abstract Interpretation (AI)

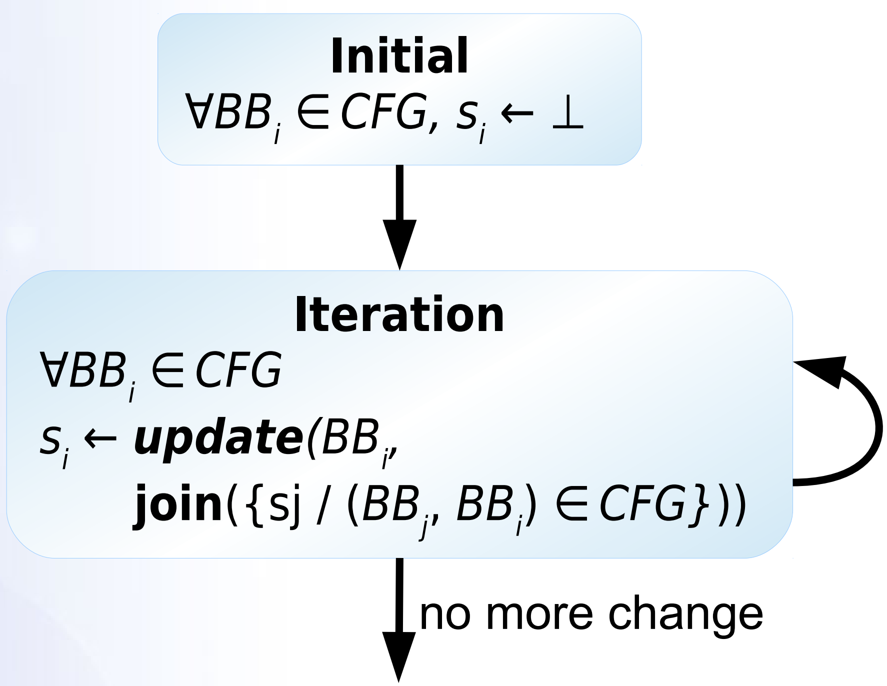
- looking for a program property
  - all possible execution paths
  - for any initial states
  - not tractable in practice
- abstract interpretation
  - domain: concrete → abstract
  - designed for the looked property
  - faster to compute
  - abstraction of the interpreter
- successful for high level languages



AI applied to Control Flow Graph

```
s = 0;
i = 0;
while(i < 10) {
        s = s + t[i];
        i = i + 1;
}
```





AI on Machine Code

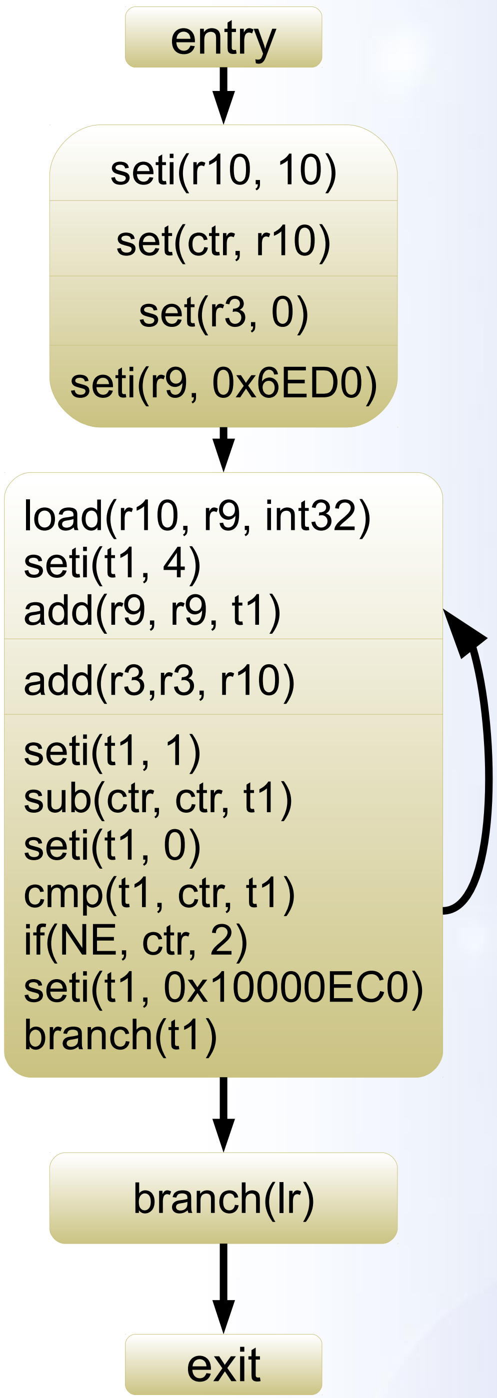
- complexity
  - AI costly: fixpoint reaching of loops at level *n*
    → 2^n interpretations
  - program in machine code : x100,000 instructions
  - need to be fast and concise
- lots of instruction sets
  - PowerPC, ARM, TriCore, Sparc, etc
  - requirement for a language
    - independent of the hardware
    - ability to express any instruction set

Plan

- Introduction
- Context
- **Language Definition**
- Use Cases
- Conclusion

Semantics Language

- inspired by RISC
- one function / instruction
  - add, sub, shl, shr, asr, set
  - minimal number of abstract functions
- simple operands: 3-operands
  - register r_i, ctr
  - temporaries t_i
- specialized instructions
  - memory access: load / store
  - immediate values: seti
- compact representation
  - constants to identify registers
  - only 8 bytes (for 32-bits machine)

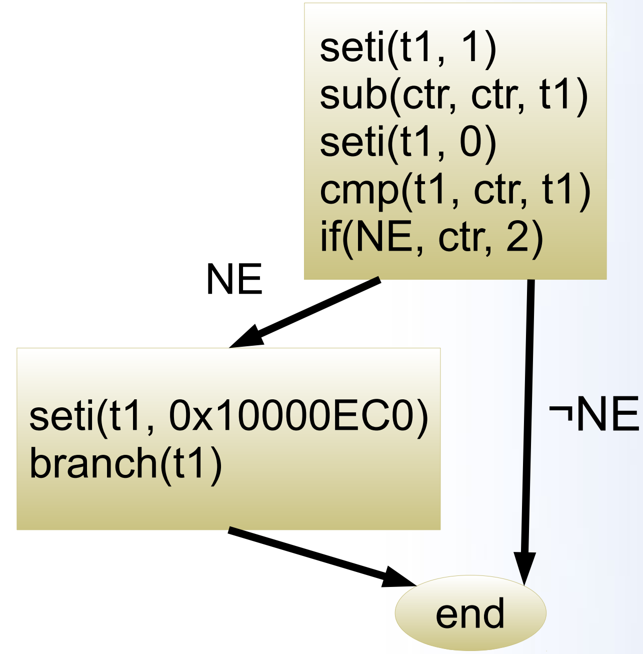


Conditional Instructions

- branch(r) =

  modification au PC
- cont → stop
- cmp(r_d, r_1, r_2)

  generation of condition
- if(r, c, n)
  - if c in r is false, skip n instructions
  - n positive → no loop, no fixpoint




```
update(I, s)
B = semantics instructions(I)
w ← { (0, s) }
s_result ← ⊥
while w do
      (i, s_i)::w ← w
      if i >= |B| then s_result ← join(s_result, s_i)
      else w ← w ∪ update(B[i], s_i)

update(add(r_d, r_1, r_2), s) =
      (i + 1, s[rd → add(s[r1], s[r2]))
...
update(if(r, c, n), s) =
      (i + 1, s)
       ∪ (i + n, s)
```

Support of any Instruction Set

- cannot support any instruction
  - using ⊤, “any value”
  - scratch(r_i)
- instantiation of instructions
  - parametric in instruction set
  - fixed in the code at the analysis time

semantics of lmw in instruction set

op lmw(r: uint(5), a: uint(5), d: int(16))
 ea ← R[a] + d
 <u>for</u> i = r_r <u>to</u> 31 <u>do</u>
  R[i] ← M_{32}[ea]
  ea ← ea + 4

lmw r29, r1, 0 (at analysis time)

seti(t_1, 0)
add(t_1, r_{28}, t_1)
seti(t_2, 4)
load(r_{29}, t_1, uint32)
add(t_1, t_1, t_2)
load(r_{30}, t_1, uint32)
add(t_1, t_1, t_2)
load(r_{31}, t_1, uint32)
add(t_1, t_1, t_2)

Plan

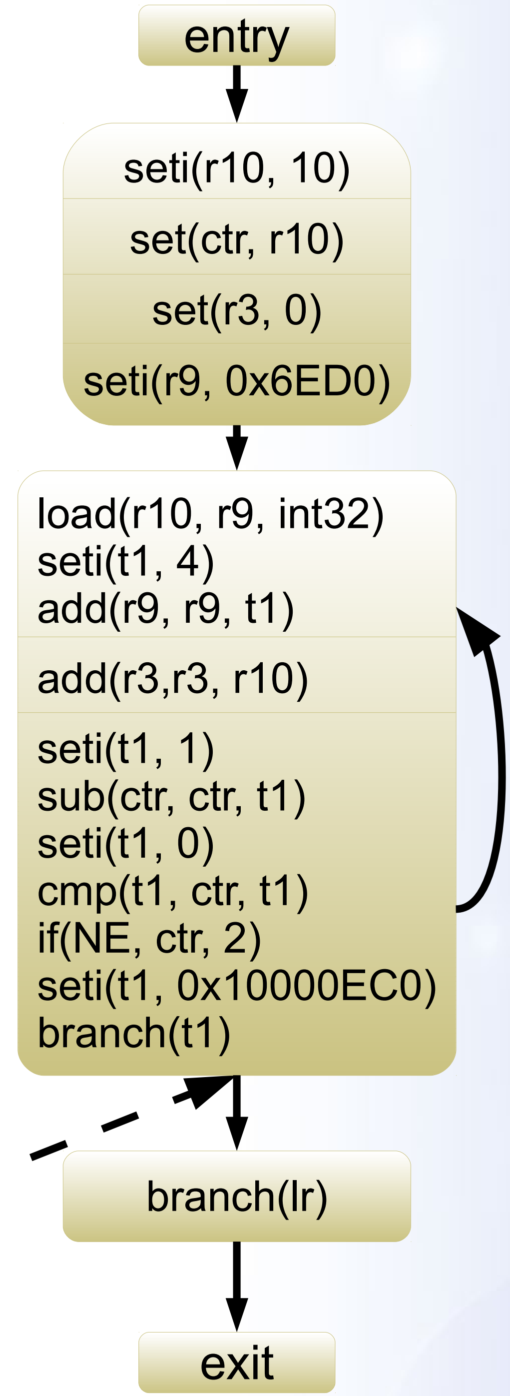
- Introduction
- Context
- Language Definition
- **Use Cases**
- Conclusion

Circular-Linear Progression Analysis

$D = \{(b, \delta, n) \,/\, b, \delta, n \in \mathbb{Z}^3\}$
$\Leftrightarrow d \in D \rightarrow d = \{\, b + k\,\delta \,/\, 0 \le k \le n \}$

$S = Reg \cup Addr \rightarrow D$

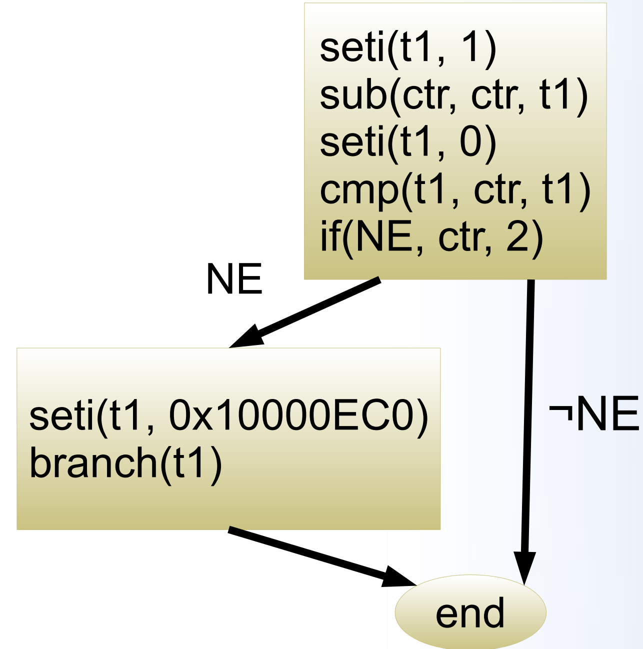
update(add(r_d, r_1, r_2), s) =
    let $(b_1, \delta_1, n_1) = s[r_1]$ in
    let $(b_2, \delta_2, n_2) = s[r_2]$ in
    let $\delta = \gcd(\delta_1, \delta_2)$ in
    let $b = b_1 + b_2$ in
    let $n = m_1\delta_1 / \delta + m_2\delta_2 / \delta)$ in
    $s[r_d \rightarrow (b, \delta, n)]$



$r_{10} \rightarrow (0, 1, 2^{32}\text{-}1)$
$r_3 \rightarrow (0, 1, 2^{32}\text{-}1)$
$ctr \rightarrow (9, -1, 2^{32}\text{-}1)$
$r_9 \rightarrow (0x6ED4, 4, 2^{32}/4)$

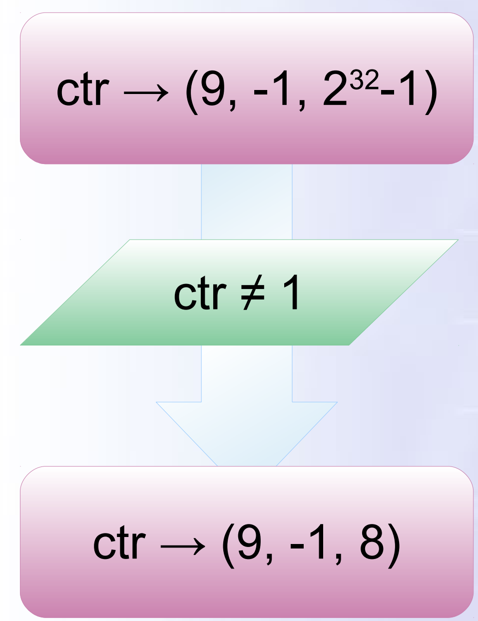
Filtering / Narrowing

- ctr → (9, -1, 2^{32}-1)
  - valid, safe
  - overestimated
- filter by condition
  - predicate associated with paths
  - taken paths, not-taken path
  - backward: condition → data
  - build all paths by simple depth-first traversal of block



left path

{ taken ⋀ ctr - 1 ≠ 0 }
seti(t_1, 1)
{ taken ⋀ ctr - t_1 ≠ 0 }
sub(ctr, ctr, t_1)
{ taken ⋀ ctr ≠ 0 }
seti(t1, 0)
{ taken ⋀ ctr ≠ t_1) }
cmp(t_1, ctr, t1)
{ taken ⋀ sr.1 ≠ sr.2) }
if(NE, sr, 2)
{ taken }
seti(t_1, 0x10000EC0)
{ taken }
branch(t1)



Aliasing problem

- generated code
  - variable: often stored in memory
  - register: temporary container of a variable
  - condition on register
  - how to alias with memory?

```
lwz      r9, 28(r31)
cmpwi    c7, r9, 9
ble      c7,.L3
```

$\{ \text{taken} \wedge r_9 \leq 9 \wedge M(r_{31} + 28) \leq 9 \}$
seti(t1, 28)
$\{ \text{taken} \wedge r_9 \leq 9 \wedge M(r_{31} + t_1) \leq 9 \}$
add(t1, r_{31}, t_1)
$\{ \text{taken} \wedge r_9 \leq 9 \wedge M(t_1) \leq 9 \}$
load(r_9, t_1, int32)
$\{ \text{taken} \wedge r_9 \leq 9 \}$
seti(t_1, 9)
$\{ \text{taken} \wedge r_9 \leq t_1 \}$
cmp(c_7, r_9, t_1)
$\{ \text{taken} \wedge c_7.1 \leq c_7.2 \}$
if(LE, c_7, 1)
$\{ \text{taken} \}$
branch(.L3)

Plan

- Introduction
- Context
- Language Definition
- Use Cases
- **Conclusion**

Conclusion

- speed (PowerPC)
  - 242,594 machine instructions / s
- efficiency
  - address calculation: 74% of non-$\top$
  - value filtering: 68% of non-$\top$
- future works
  - introduce new operations (multiplication, float calculation for conditions)
  - supports new instructions set (at work: ARM, TriCore, Sparc)

Any Question ?