Expressing and Exploiting Conflicts over Paths in WCET Analysis

Vincent Mussot, Jordy Ruiz, Pascal Sotin,
Marianne de Michiel, Hugues Cassé

funded by W-SEPT ANR Project

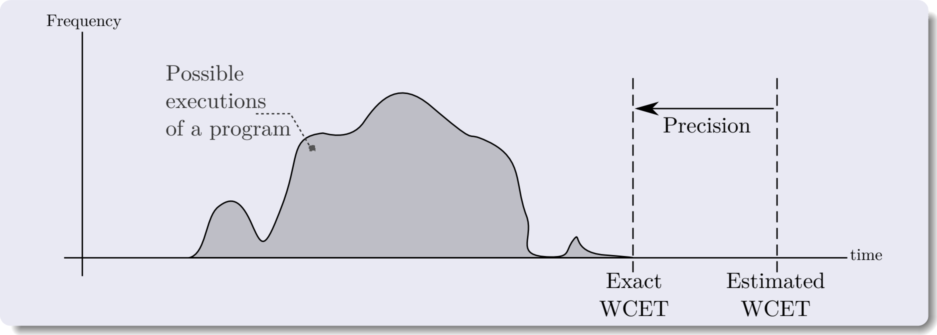
IRIT, Université Paul Sabatier

July 5, 2016

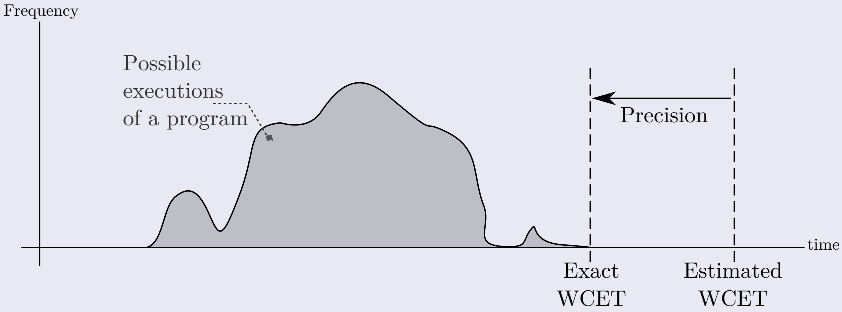
Outline

Outline

Introduction



Introduction



A popular method in WCET static analysis

Implicit Path Enumeration Technique (IPET)

1. Work on the Control Flow Graph (CFG) representation of a program
2. Compute low-level timings for every basic block
3. Turn dependencies and timings into an Integer Linear Program (ILP)
4. Solve this ILP to obtain the WCET of the program.

Flow facts to improve the WCET

- Constraints on execution paths (loop bounds, infeasible paths...)
- Contexts (fonction/call, iteration, condition...)

Flow facts to improve the WCET

- Constraints on execution paths (loop bounds, infeasible paths...)
- Contexts (fonction/call, iteration, condition...)

Our contributions

- The identification of a specific class of infeasible paths.
- The expression in the FFX Format.
- Two distinct methods of integration in the WCET analysis
  - Through CFG transformation (*V.Mussot, RTCSA 2015 [8]*).
  - With additional ILP constraints (*P.Raymond, EMSOFT 2014 [9]*).
- Experimental studies with comparison between both methods.

What does Flow Fact in XML (FFX) support:

Nested contexts and annotations

```
<call name="C1" address="0x8004">
  <function name="f">
    <loop address="0x824a" MAX="10">
      <iteration number="7">
        ...
        <condition address="0x90" ...>
          <case ... executed="false">
          </case>
          ...
        </conditional>
      </iteration>
    </loop>
  </function>
</call>
```

What does Flow Fact in XML (FFX) support:

Nested contexts and annotations

```
<call name="C1" address="0x8004">
  <function name="f">
    <loop address="0x824a" MAX="10">
      <iteration number="7">
        ...
        <condition address="0x90" ...>
          <case ... executed="false">
          </case>
          ...
        </conditional>
      </iteration>
    </loop>
  </function>
</call>
```

Numeric constraints (e.g.: BB1+BB2≤80)

```
<block id="BB1"
  address="0x8848"/>
<block id="BB2"
  address="0x90a4"/>
  ...
<control-constraint>
  <le>
    <add>
      <count id="BB1"/>
      <count id="BB2"/>
    </add>
    <const int="80"/>
  </le>
</control-constraint>
```

Outline

Notion of conflict

Example of conflict

```
if (...)
  x = 0; // A
if (x != 0)
  y = x; // B
```

Conflict:
{A,B}.

ILP constraint: $n_A + n_B \leq 1$

Notion of conflict

Example of conflict

```
if (...)
  x = 0; // A
if (x != 0)
  y = x; // B
```

Conflict:
{A,B}.

ILP constraint: $n_A + n_B \leq 1$

Definition

We define a conflict over a list of edges or blocks. Thus, any program path that contains at least one occurrence of every element of the list is an infeasible path.

Notion of conflict

Example of conflict

```
if (...)
  x = 0; // A
if (x != 0)
  y = x; // B
```

Conflict:
{A,B}.

ILP constraint: $n_A + n_B \leq 1$

Definition

We define a conflict over a list of edges or blocks. Thus, any program path that contains at least one occurrence of every element of the list is an infeasible path.

FFX syntax

```
<conflict>
  <!-- Edge or block identifier 1 -->
  <!-- ... -->
  <!-- Edge or block identifier N -->
</conflict>
```

Contextual conflicts

Context of validity

```
while(...){ //bound=N
   if (...)
     x = 0; // A
   if (x != 0)
     y = x; // B
 }
```

Conflicts:
{A,B} in 1^{st} iteration.
...
{A,B} in n^{th} iteration.

ILP constraint: $n_A + n_B \leq N$

The conflict occurs in each iteration of the loop.

Contextual conflicts

Context of validity

```
while(...){ //bound=N
   if (...)
     x = 0; // A
   if (x != 0)
     y = x; // B
 }
```

Conflicts:

{A,B} in 1^{st} iteration.

...

{A,B} in n^{th} iteration.

ILP constraint: $n_A + n_B \leq N$

The conflict occurs in each iteration of the loop.

FFX Syntax

```
<loop ...>
  <iteration number="*">
    <conflict>
      <edge "A" />
      <edge "B" />
    </conflict>
  </iteration>
</loop>
```

Contextual conflicts

Example with specific instances of edges

```
main(){
  k = 0;
  if (...)
    k = 1; // A
  foo(k); // C_foo
  foo(2);
}
```

```
foo(int k){
  if (...)
    x = 0; // B
  if (x != 0
  || k != 1)
    y = x; // C
}
```

The conflict only holds for instances of edges B and C in the call C_foo.

Contextual conflicts

Example with specific instances of edges

```
main(){
  k = 0;
  if (...)
    k = 1; // A
  foo(k); // C_foo
  foo(2);
}
```

```
foo(int k){
  if (...)
    x = 0; // B
  if (x != 0
  || k != 1)
    y = x; // C
}
```

Conflict:
$\{A, B_{C\_foo}, C_{C\_foo}\}$

ILP constraint:
$n_A + n_{B_{C\_foo}} + n_{C_{C\_foo}} \leq 2$

The conflict only holds for instances of edges B and C in the call C_foo.

Contextual conflicts

Example with specific instances of edges

```
main(){
  k = 0;
  if (...)
    k = 1; // A
  foo(k); // C_foo
  foo(2);
}
```

```
foo(int k){
  if (...)
    x = 0; // B
  if (x != 0
  || k != 1)
    y = x; // C
}
```

Conflict:

$\{A, B_{C\_foo}, C_{C\_foo}\}$

ILP constraint:

$n_A + n_{B_{C\_foo}} + n_{C_{C\_foo}} \leq 2$

The conflict only holds for instances of edges B and C in the call C_foo.

FFX Syntax

```
<conflict>
  <edge "A" />
  <call name="C_foo" ...>
    <edge "B" />
    <edge "C" />
  </call>
</conflict>
```

The ordered attribute

Example of ordered conflict

```
int k = 0;
while(...){
  if (k==0)
    ... // B
  if (...)
    k=1; // A
}
```

Conflict:
{A,B} in that order.

ILP constraint:
???

The conflict only holds for edges A and B in that order. B may appear in the program path before the first A, not after.

The ordered attribute

Example of ordered conflict

```
int k = 0;
while(...){
  if (k==0)
    ... // B
  if (...)
    k=1; // A
}
```

Conflict:
{A,B} in that order.

ILP constraint:
???

The conflict only holds for edges A and B in that order. B may appear in the program path before the first A, not after.

FFX syntax

```
<conflict ordered="yes">
  <edge "A" />
  <edge "B" />
</conflict>
```

Note that the ordered conflict is weaker than the unordered one:

Non-ordered conflict $\Rightarrow$ ordered conflict

Outline

Integrate conflicts through CFG transformation

General idea

- Turn a Control Flow Graph (CFG) into an automaton.
- Express a conflict as an automaton.
- Perform a product between both automata.
- Rebuild a CFG from the result of the product.

The conflict restriction is now carried by the new CFG.

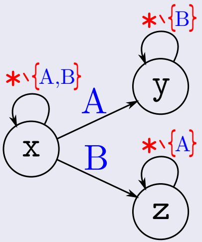
Integrate conflicts through CFG transformation

General idea

- Turn a Control Flow Graph (CFG) into an automaton.
- Express a conflict as an automaton.
- Perform a product between both automata.
- Rebuild a CFG from the result of the product.

The conflict restriction is now carried by the new CFG.

Example



Conflict

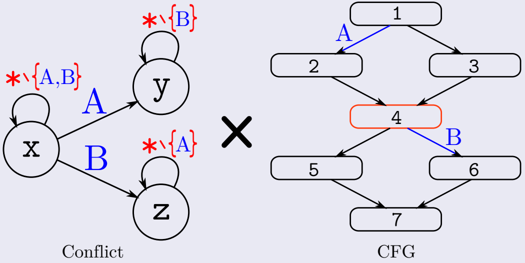
Integrate conflicts through CFG transformation

General idea

- Turn a Control Flow Graph (CFG) into an automaton.
- Express a conflict as an automaton.
- Perform a product between both automata.
- Rebuild a CFG from the result of the product.

The conflict restriction is now carried by the new CFG.

Example



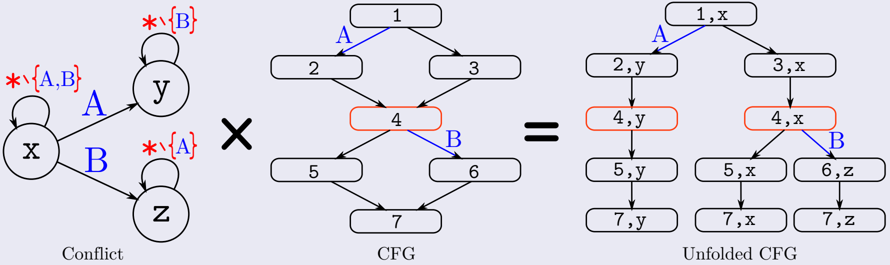
Integrate conflicts through CFG transformation

General idea

- Turn a Control Flow Graph (CFG) into an automaton.
- Express a conflict as an automaton.
- Perform a product between both automata.
- Rebuild a CFG from the result of the product.

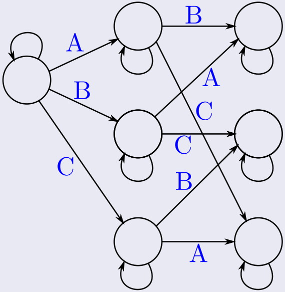
The conflict restriction is now carried by the new CFG.

Example

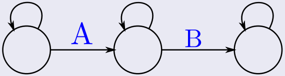


Effects of the ordered attribute

Comparison of ordered and unordered automaton



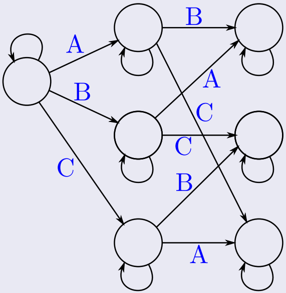
Conflict



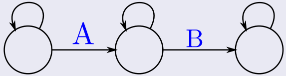
Conflict (ordered)

Effects of the ordered attribute

Comparison of ordered and unordered automaton



Conflict



Conflict (ordered)

Observations

- The size of the *result automaton* depends on the unfolding of the CFG caused by the product.
- The size of the *conflict automaton* explodes in number of states (N_S) with the number of elements (N_E) in conflict: $N_S = 2^{N_E} - 1$.
- The size of a *conflict automaton* for ordered elements is linear: $N_S = N_E - 1$.

Effects of the ordered attribute

Observations

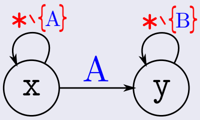
- The size of the *result automaton* depends on the unfolding of the CFG caused by the product.
- The size of the *conflict automaton* explodes in number of states (N_S) with the number of elements (N_E) in conflict: $N_S = 2^{N_E} - 1$.
- The size of a *conflict automaton* for ordered elements is linear: $N_S = N_E - 1$.

Effects of the ordered attribute

Observations

- The size of the *result automaton* depends on the unfolding of the CFG caused by the product.
- The size of the *conflict automaton* explodes in number of states (N_S) with the number of elements (N_E) in conflict: $N_S = 2^{N_E} - 1$.
- The size of a *conflict automaton* for ordered elements is linear: $N_S = N_E - 1$.

Product with an ordered conflict



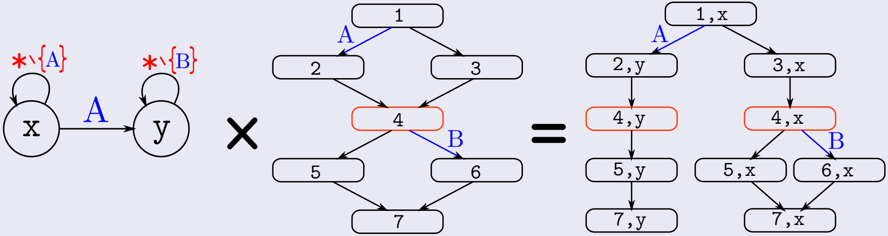
Conflict (ordered)

Effects of the ordered attribute

Observations

- The size of the *result automaton* depends on the unfolding of the CFG caused by the product.
- The size of the *conflict automaton* explodes in number of states (N_S) with the number of elements (N_E) in conflict: $N_S = 2^{N_E} - 1$.
- The size of a *conflict automaton* for ordered elements is linear: $N_S = N_E - 1$.

Product with an ordered conflict



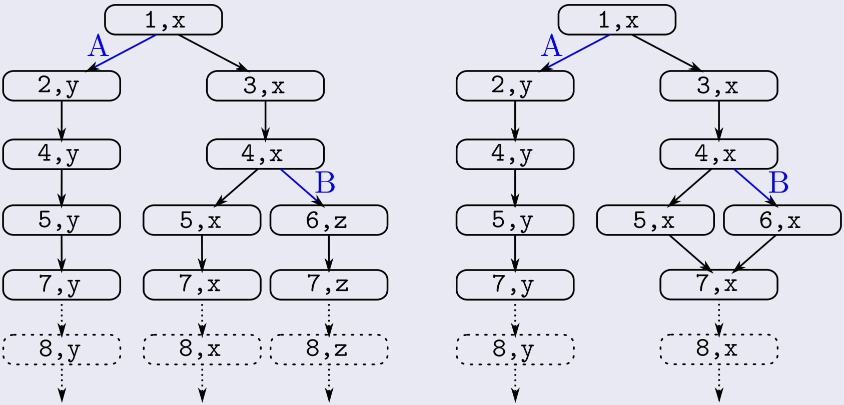
Conflict (ordered) CFG Unfolded CFG

Effects of the ordered attribute

Result comparison

From non-ordered conflict

From ordered conflict



Outline

Integrate conflicts with additional ILP constraints

The formula

P.Raymond presented in *EMSOFT 2014* [9] a general formula that allows to generate an ILP constraint from a set S of conflicting edges:

$$\sum_{x \in X} p_x x \leq (|X| - 1)|S| + \sum_{x \in X} (p_x m_x - |S|)$$

where X is a set of edges and S is a set conflicting *avatars* built upon X

Integrate conflicts with additional ILP constraints

The formula

P.Raymond presented in *EMSOFT 2014* [9] a general formula that allows to generate an ILP constraint from a set S of conflicting edges:

$$\sum_{x \in X} p_x x \leq (|X| - 1)|S| + \sum_{x \in X} (p_x m_x - |S|)$$

where X is a set of edges and S is a set conflicting *avatars* built upon X

The generation of the set S

We developed an OTAWA plug-in that automatically generates sets S from conflict elements. Then we derive new ILP constraints from the formula and integrate these constraints in the WCET analysis.

Integrate conflicts with additional ILP constraints

Example with specific instances of edges

```
while(...){
  i=j=0;
  if (...)
    i=1; // A
  if (...)
    j=1; // B
}
if (i == 1
&& j ==1)
  ... // C
```

Integrate conflicts with additional ILP constraints

Example with specific instances of edges

```
while(...){
  i=j=0;
  if (...)
    i=1; // A
  if (...)
    j=1; // B
}
if (i == 1
&& j ==1)
  ... // C
```

```
<conflict>
  <loop ...>
    <iteration "n">
      <edge "A" />
      <edge "B" />
    </iteration>
  </loop>
  <edge "C" />
</conflict>
```

Conflict:

$\{A_n, B_n, C\}$

Integrate conflicts with additional ILP constraints

Example with specific instances of edges

```
while(...){
  i=j=0;
  if (...)
    i=1; // A
  if (...)
    j=1; // B
}
if (i == 1
&& j ==1)
  ... // C
```

```
<conflict>
  <loop ...>
    <iteration "n">
      <edge "A" />
      <edge "B" />
    </iteration>
  </loop>
  <edge "C" />
</conflict>
```

Conflict:
$\{A_n, B_n, C\}$

Application of the formula

The set of conflicting edges derived from the *conflict* is $S = \{A_n, B_n, C\}$, and if we apply the formula, we obtain:

$$1 \times A + 1 \times B + 1 \times C \leq (3 - 1) \times 1 + 1 \times n - 1 + 1 \times n - 1$$

The newly generated ILP constraint is then $A + B + C \leq 2n$.

Outline

Experiments

Overview

- Detection and expression of conflicts on the *Mälardalen* benchmark suite and other benchmarks using the PathFinder tool.
- Integration of conflicts in a WCET analysis carried by our academic tool OTAWA, using both methods presented here.
- The same binary files, annotations files (with conflicts) and architecture models were used in both cases.

Experiments

Overview

- Detection and expression of conflicts on the *Mälardalen* benchmark suite and other benchmarks using the PathFinder tool.
- Integration of conflicts in a WCET analysis carried by our academic tool OTAWA, using both methods presented here.
- The same binary files, annotations files (with conflicts) and architecture models were used in both cases.

The infeasible path detection tool PathFinder

- Analyzes binary programs, looking for semantic conflicts.
- Models program paths as conjunctions of predicates on registers and memory cells.
- Deduces infeasible paths from the discovery of unsatisfiable conjunctions.
- Attempts to minimize the sets of edges involved in a conflict.
- Supports FFX as an output format.

Experiments

	Nb. of conflicts found		WCET gain simple arch.		WCET gain arm9 + cache	
Program	Total	After minim.	Constraints	Unfolded	Constraints	Unfolded
SMALL MÄLARDALEN BENCHMARKS						
adpcm	174	28	0.00 %	0.00 %	CE	CE
cnt	118	5	0.00 %	0.00 %	0.00 %	0.00 %
cover	3	3	6.95 %	6.95 %	**0.01 %**	**0.25 %**
crc	8	8	0.50 %	0.50 %	**4.10 %**	**9.70 %**
edn	7	6	0.03 %	0.03 %	CE	CE
expint	8	5	0.00 %	0.00 %	**0.00 %**	**0.09 %**
fibcall	1	1	0.72 %	0.72 %	0.32 %	0.32 %
fir	1	1	0.00 %	0.00 %	**3.37 %**	**7.45 %**
select	18	11	0.16 %	0.16 %	0.09 %	0.09 %
sqrt	407	10	0.40 %	0.40 %	0.04 %	0.04 %
LARGE MÄLARDALEN BENCHMARKS						
statemate	1118	71	2.77 %	CE*	1.00 %	CE*
ud	13	1	1.17 %	1.17 %	1.08 %	1.08 %
nsichneu	13648	7684	0.00 %	CE*	0.00 %	CE*
minver	10	9	1.40 %	1.40 %	CE	CE
ludcmp	29	3	0.00 %	0.00 %	0.00 %	0.00 %
lms	2097	141	CE	CE	CE	CE
fft1	830	149	CE	CE	CE	CE
qurt	797	41	CE	CE	CE	CE
ESTEREL BENCHMARKS						
runner	5618	185	9.84 %	CE*	9.12 %	CE*
abcd	4949	274	3.01 %	CE*	5.17 %	CE*

Experiments

	Nb. of conflicts found		WCET gain simple arch.		WCET gain arm9 + cache	
Program	Total	After minim.	Constraints	Unfolded	Constraints	Unfolded
SMALL MÄLARDALEN BENCHMARKS						
cover	3	3	6.95 %	6.95 %	**0.01 %**	**0.25 %**
crc	8	8	0.50 %	0.50 %	**4.10 %**	**9.70 %**
expint	8	5	0.00 %	0.00 %	**0.00 %**	**0.09 %**
fir	1	1	0.00 %	0.00 %	**3.37 %**	**7.45 %**

On the precision improvement of unfolding the CFG

At some points in the WCET analysis, abstract cache states are merged.

→ It injects pessimism.

The CFG is unchanged with the *additional constraint* method.

→ The merge points remains.

The *unfolding* method may cause the separation of some paths.

→ Some merge points may disappear.

Conflicts for WCET analysis

- We identified *conflicts* as a specific class of infeasible paths.
- They have specific properties:
  - → They can replace some numeric constraints.
  - → They are often more simple than equivalent numeric constraints.
  - → They support external and internal contexts.
  - → Their generation from specific infeasible path detection method can be straightforward.
- They can be expressed in an annotation language.
- We presented two method for the integration in the WCET analysis.
  - → One method through CFG transformation that benefits from the *ordered* property to improve its scalability.
  - → One method that generates new ILP constraints.
- The comparison of the two methods has shown:
  - → Significant gain for both methods.
  - → Unfolding can be more precise than adding new constraints.
  - → Unfolding suffers from scalability issues.