Merging Techniques for Faster Derivation of WCET Flow Information using Abstract Execution

Jan Gustafsson and Andreas Ermedahl

Mälardalen Real-Time Research Center (MRTC)
Västerås, Sweden

MÄLARDALENS HÖGSKOLA

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Contents

- Abstract Execution finds loop bounds and infeasible paths
- Merging is used to speed up calculations (but may introduce pessimism)
- We show how different merge point placements affects speed and pessimism
- We present a new efficient merging technique based on *sorting of program points*

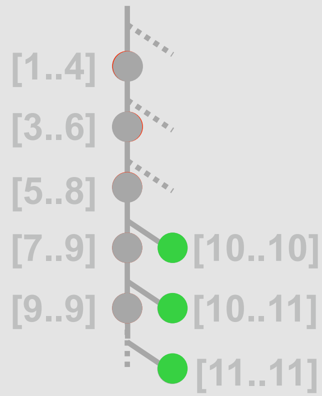
Abstract Execution

* Based on Abstract Interpretation (AI)
    * AI gives safe (over)approximation of possible values of each variable at different program points
* "Executes" the program using abstract state(s)
    * Not using traditional AI fixpoint calculation, instead always running forward
* Result: an (over)approximation of possible execution paths
    * Might potentially include some infeasible paths
    * Infeasible paths found are guaranteed to be infeasible
* Implemented in SWEET (SWEdish Execution Time tool)

Example: Loop bound analysis

```
i = INPUT;
// i = [1..4]
while(i < 10) {
    // point p
    ...
    i = i + 2;
}
// point q
```

Loop iteration	Abstract state at p	Abstract state at q
1	i = [1..4]	⊥
2	i = [3..6]	⊥
3	i = [5..8]	⊥
4	i = [7..9]	i = [10..10]
5	i = [9..9]	i = [10..11]
6	⊥	i = [11..11]



Result
Min. #iterations: 3
Max. # iterations: 5

- Result includes all possible loop executions
- Three new abstract states generated at q
  - Could be merged to one single abstract state:

    i = [10..11]

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Problems with merging

* May yield abstract values that represent concrete values in a less precise way
  - Merging [6..6] and [10..11] yields [6..11], which also contains the concrete values 7, 8, 9
* May lead to overestimated loop bounds
* May lead to that infeasible paths are missed

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Example: Overestimated loop bound

```
int i, x;
// i = [-5..5]
if (i > 0) x=2;
else x=4;
// p
while (x < 10)
 { // q
  if (isOdd(x))
    x++;
  else x=x+2;
 }
```

- 11 possible runs
- Concrete worst cases when

  `i = [1..5]`
- Max. iterations = 4

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Example: Overestimated loop bound

```
int i, x;
// i = [-5..5]
if (i > 0) x=2;
else x=4;
// p
while (x < 10)
 { // q
  if (isOdd(x))
    x++;
  else x=x+2;
 }
```

Loop iteration	State at q last test true	State at q last test false
5	[6..8] etc.	etc.
6	[7..9] etc.	etc.
7	[8..9] etc.	etc.
8	[9..9] etc.	etc.

#parallel states: 128

Result
Max. #iterations: 8

Worst case

* Merging at p yields an overestimation
* ...and a lot of states

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Example: Overestimated loop bound

```
int i, x;
// i = [-5..5]
if (i > 0) x=2;
else x=4;
// p
while (x < 10)
 { // q
  if (isOdd(x))
    x++;
  else x=x+2;
 }
```

...

#parallel states:
6

Result
Max. #iterations: 7

- Merging at q yields *fewer* states and a *smaller* overestimation

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Placement of Merge Points

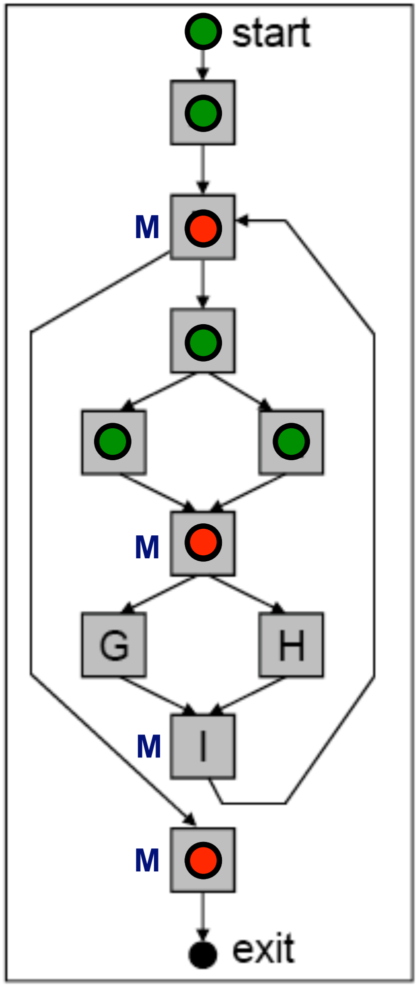
* The placement of merge points is important
* There is a trade-off between precision and speed
* Optimal selection depends on code structure and size
* SWEET allows the following placements:
  * at function entries
  * at function exits
  * at loop body termination, i.e., at the loop header
  * at loop exits
  * at joins after if-statements

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Problems to solve

* AE may give many parallel states
* Merge reduces the number, but for some large programs the problem remains
* For programs with many variables, each abstract state can be large
* For programs with large input spaces, the number of parallel states can be large
* We need to minimize the number of parallel states

Unordered and Ordered Merge

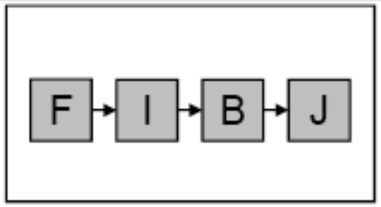


(a) CFG with merge-nodes B, F, I and J

When all states are merged states, how to continue?

Unordered merge = all merged states are released

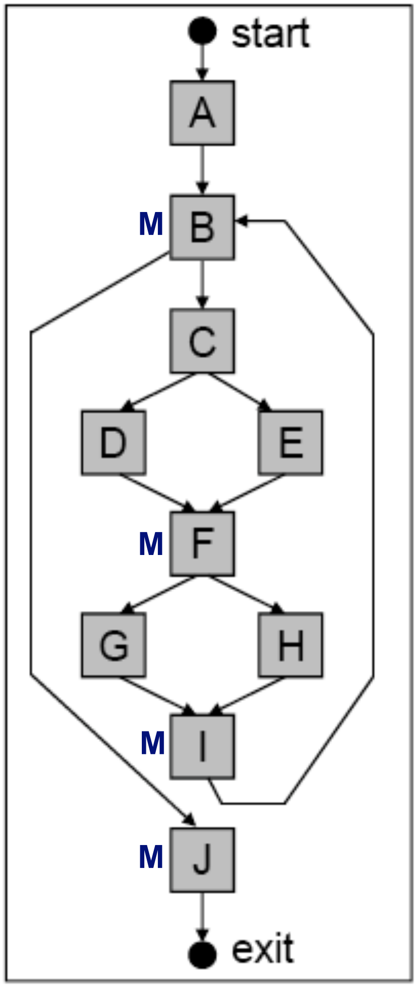
Ordered merge = only the state lagging behind (F) is released => fewer states



(c) Ordered list of merge nodes

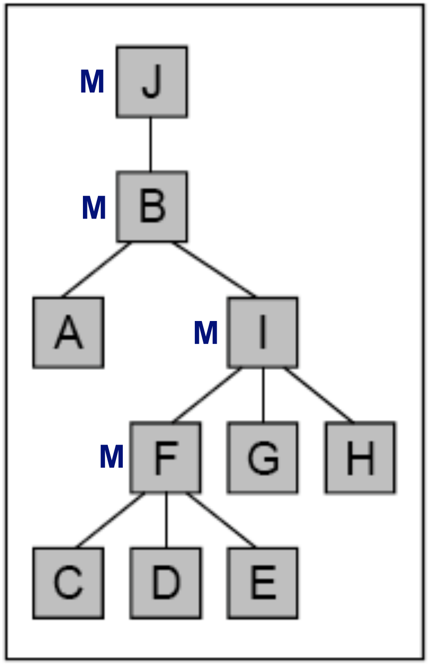
MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Ordered list of merge nodes



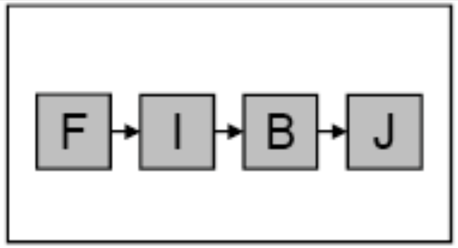
(a) CFG with merge-nodes B, F, I and J

post dominance



(b) Immediate post-dominance tree

bottom up



(c) Ordered list of merge nodes

Results - jcomplex

Merge type		Merge node type					
		FE	FT	LBT	LT	LBI	ALL
Unordered merge	ATime	8.7	15.0	0.72	0.18	0.84	0.77
	WCET	918	918	1053	2087	1053	1053
Ordered merge	ATime	8.7	8.3	0.13	0.18	0.18	0.16
	WCET	918	918	3711	2087	5395	5395

Table 1. Analysis results for `jcomplex`

- No merge: 7.2 seconds
- Unordered merge: fast and some overestimation
- Ordered merge: very fast but large overestimation

Results - insertsort

Merge type		Merge node type					
		FE	FT	LBT	LT	LBI	ALL
Unordered merge	ATime	-	-	1.6	0.08	1.9	1.7
	WCET	-	-	332	332	332	332
Ordered merge	ATime	-	-	0.09	0.08	0.09	0.09
	WCET	-	-	332	332	332	332

Table 2. Analysis results for `insertsort`

- No merge: impossible to analyse (10^{93} inputs)
- Unordered merge: fast and no overestimation
- Ordered merge: very fast and no overestimation

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Results – esab_mod

Merge type		Merge node type					
		FE	FT	LBT	LT	LBI	ALL
Unordered merge	ATime	-	-	-	-	-	-
	WCET	-	-	-	-	-	-
Ordered merge	ATime	-	-	-	-	163	161
	WCET	-	-	-	-	165795	165795

Table 3. Analysis results for esab_mod

- No merge: impossible to analyse (10^{12} inputs)
- Unordered merge: impossible to analyse
- Ordered merge: gives a result in reasonable time

Conclusions

- Merging of states is useful for large programs and programs with large input spaces
- Ordered merge is very efficient
- It is cruical to select optimal type of merging and placement of merge points
- Ordered merge technique has a general use

MÄLARDALENS HÖGSKOLA

The End!

For more information:

www.mrtc.mdh.se/projects/wcet

Unordered and Ordered Merge

* Run analysis to merge points
* Merge all states at merge points
* Let the analysis wait at the merge points
* Unordered merge:
  * When all states are at merge points, release all
* Ordered merge:
  * When all states are at merge points, release only the “state lagging behind”
  * Gives fewer states

MRTC
MÄLARDALEN REAL-TIME RESEARCH CENTRE

Flow Analysis

* Most programs can execute in several ways, due to code structure and inputs
* A flow analysis should provide bounds on all possible execution paths
* Required output to bound WCET estimate:
  - Loop iteration bounds
  - Recursion depth bounds
* Additional output to tighten WCET estimate:
  - Input dependencies
  - Infeasible paths