“Late Merges” in CPAchecker

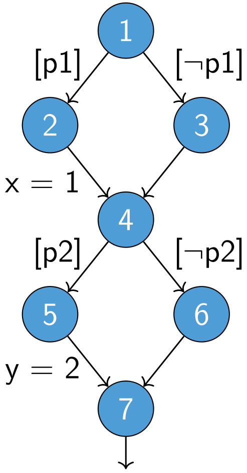
Philipp Wendler

LMU Munich, Germany

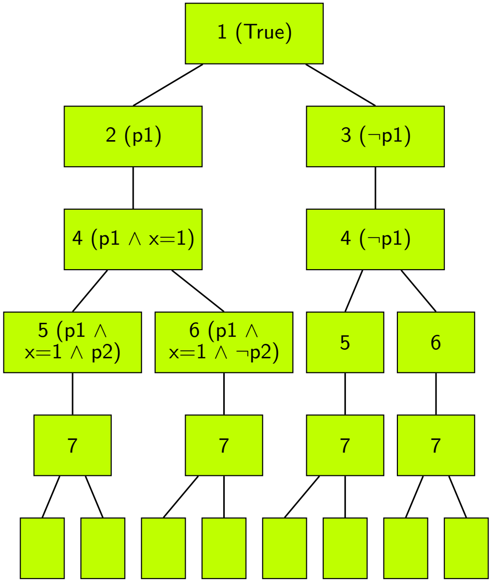
2022-07-11

Software Model Checking

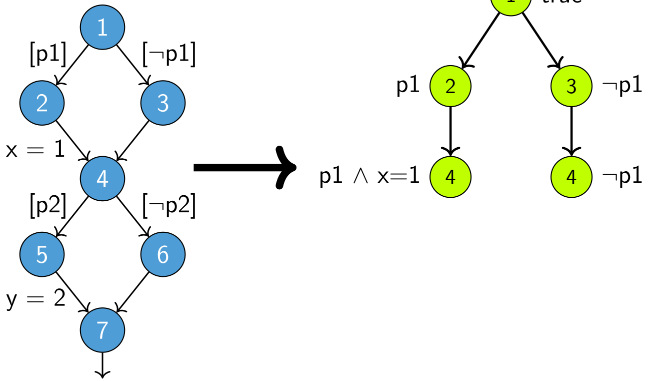
Control-Flow Automaton (CFA)



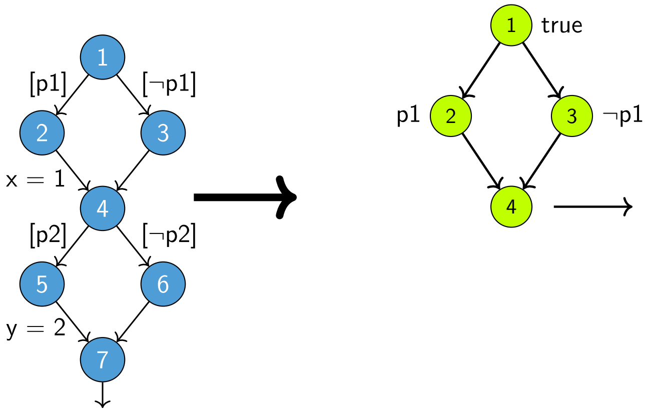
Abstract Reachability Graph (ARG)



Adjustable-Block Encoding [2]

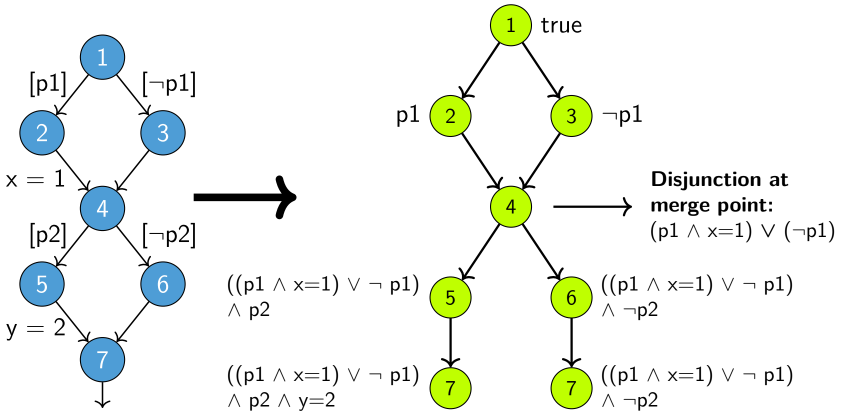


Adjustable-Block Encoding [2]



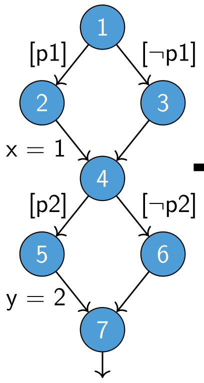
Disjunction at merge point:
(p1 ∧ x=1) ∨ (¬p1)

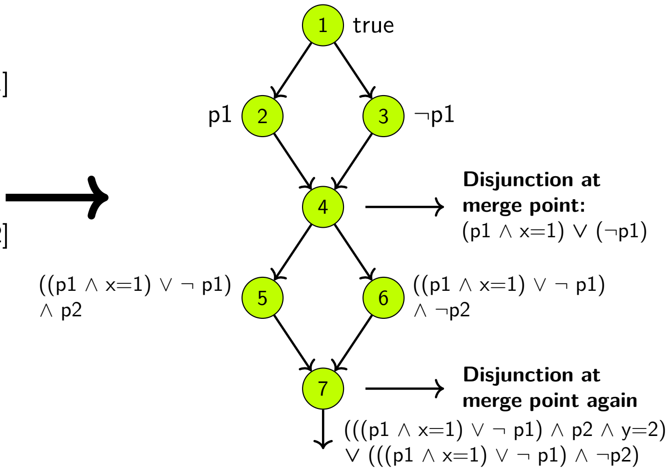
Adjustable-Block Encoding [2]



1
[p1]
[¬p1]
2
3
x = 1
4
[p2]
[¬p2]
5
6
y = 2
7
1
true
p1
2
3
¬p1
4
Disjunction at merge point:
(p1 ∧ x=1) ∨ (¬p1)
((p1 ∧ x=1) ∨ ¬ p1) ∧ p2
5
6
((p1 ∧ x=1) ∨ ¬ p1) ∧ ¬p2
((p1 ∧ x=1) ∨ ¬ p1) ∧ p2 ∧ y=2
7
7
((p1 ∧ x=1) ∨ ¬ p1) ∧ ¬p2

Adjustable-Block Encoding [2]





Disjunction at merge point:
(p1 ∧ x=1) ∨ (¬p1)

((p1 ∧ x=1) ∨ ¬ p1) ∧ p2

((p1 ∧ x=1) ∨ ¬ p1) ∧ ¬p2

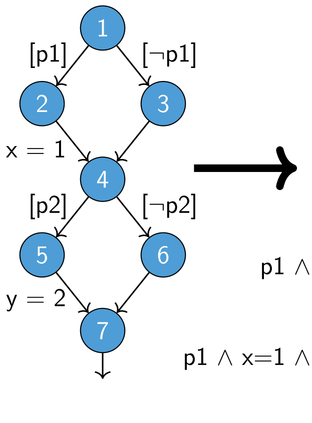
Disjunction at merge point again

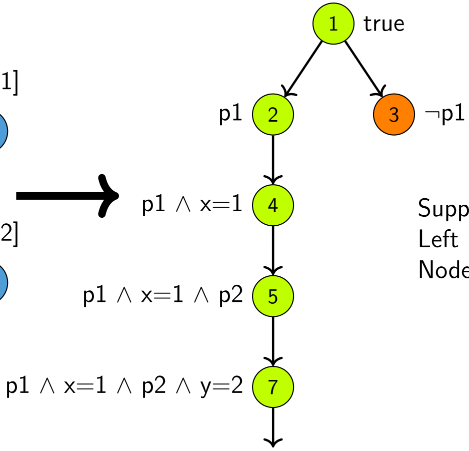
(((p1 ∧ x=1) ∨ ¬ p1) ∧ p2 ∧ y=2) ∨ (((p1 ∧ x=1) ∨ ¬ p1) ∧ ¬p2)

Adjustable-Block Encoding [2]

- Configurable and flexible
- Used for unrolling (parts of) CFA and creating formulas [1]
- For BMC, *k*-induction, PDR, IMC, ISMC, ...

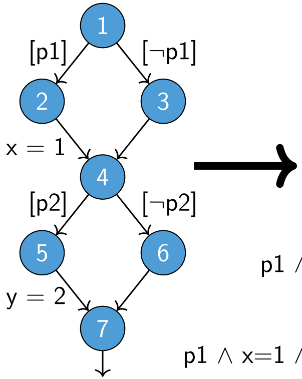
Sensitivity to Traversal Order

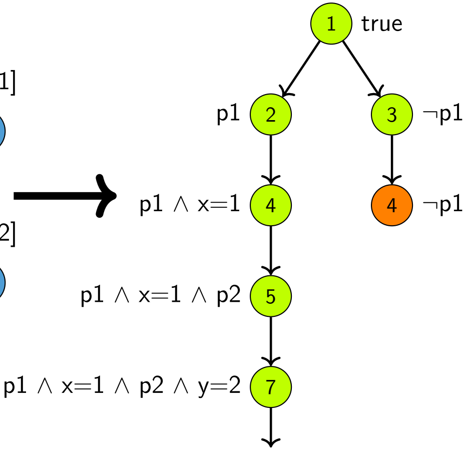




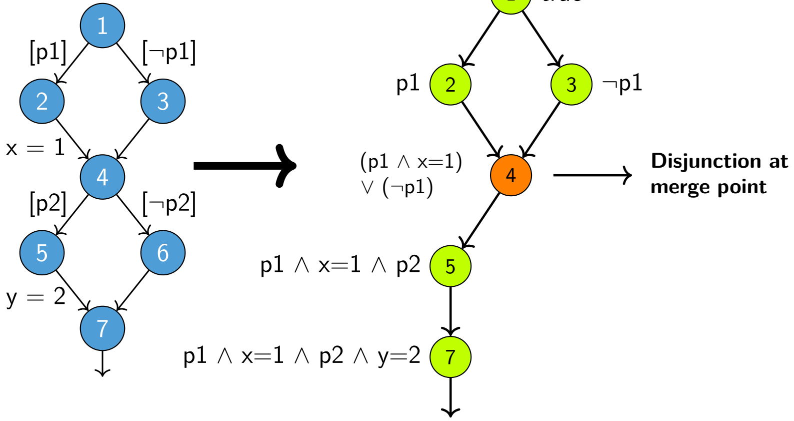
Suppose DFS
Left path finished
Node 3 in waitlist

Sensitivity to Traversal Order

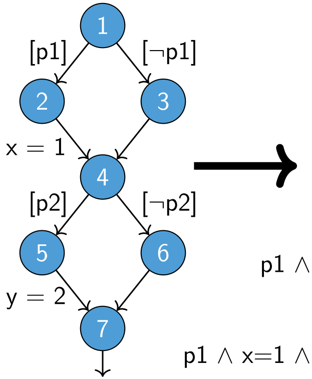


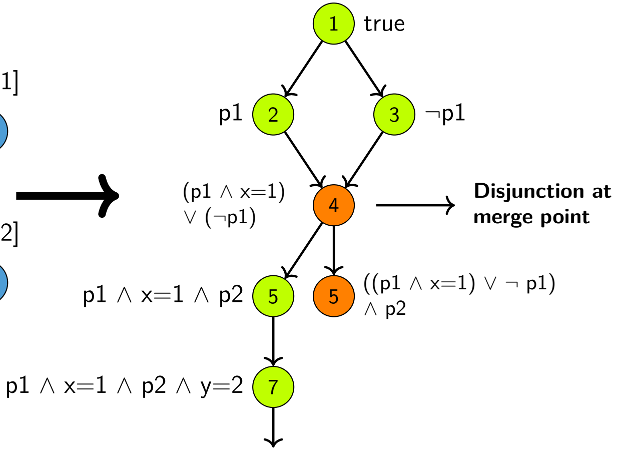


Sensitivity to Traversal Order

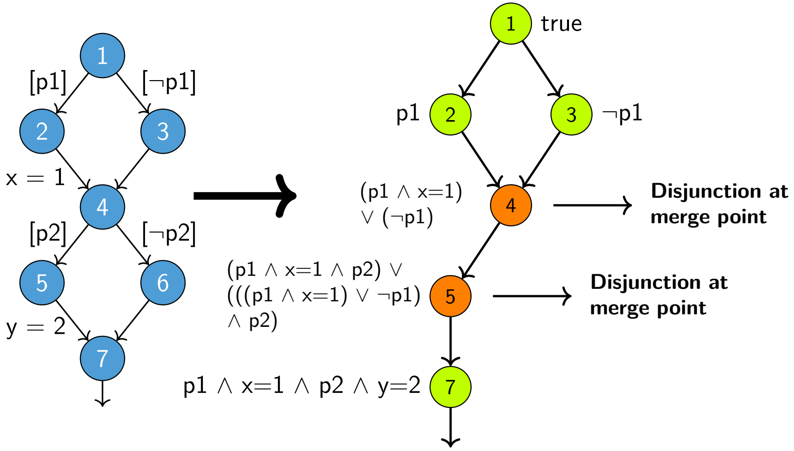


Sensitivity to Traversal Order



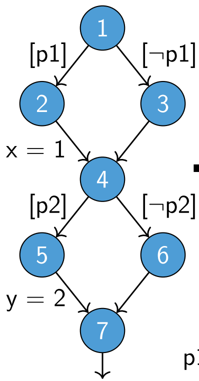


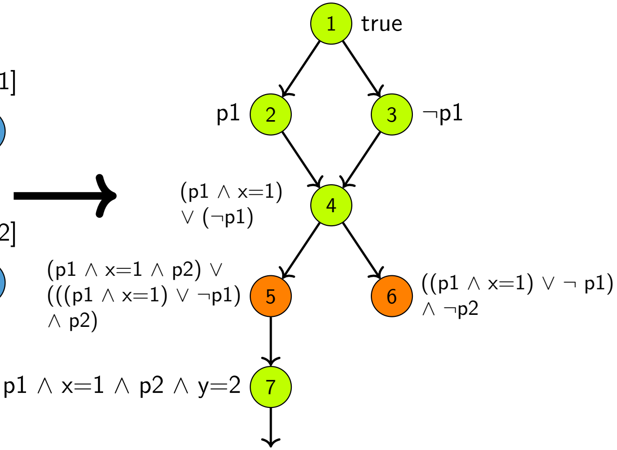
Sensitivity to Traversal Order



1
[p1]
[¬p1]
2
3
x = 1
4
[p2]
[¬p2]
5
6
y = 2
7
1 true
p1 2
3 ¬p1
(p1 ∧ x=1) ∨ (¬p1)
4
Disjunction at merge point
(p1 ∧ x=1 ∧ p2) ∨ (((p1 ∧ x=1) ∨ ¬p1) ∧ p2)
5
Disjunction at merge point
p1 ∧ x=1 ∧ p2 ∧ y=2
7

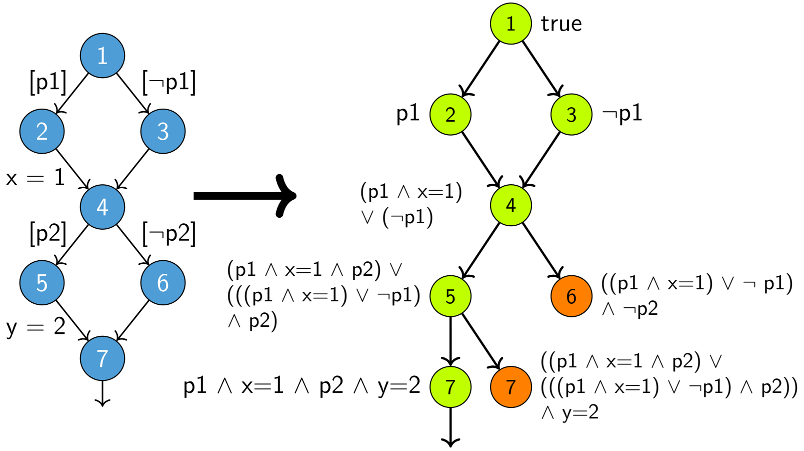
Sensitivity to Traversal Order



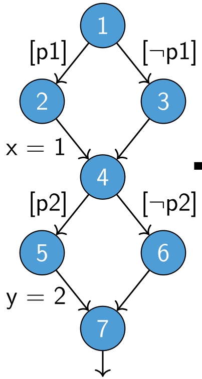


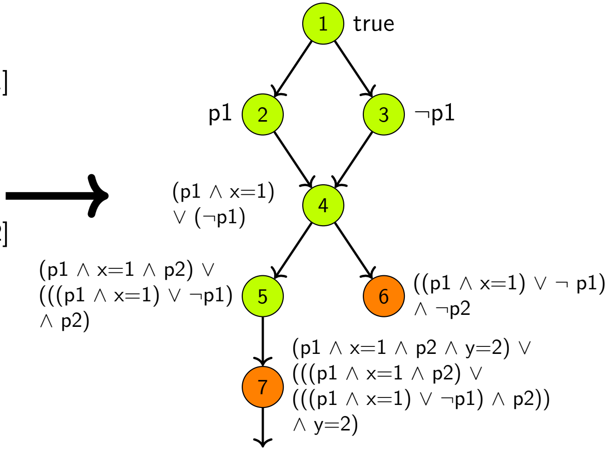
1 true
p1 2
3 ¬p1
(p1 ∧ x=1) ∨ (¬p1)
4
(p1 ∧ x=1 ∧ p2) ∨ (((p1 ∧ x=1) ∨ ¬p1) ∧ p2)
5
6
((p1 ∧ x=1) ∨ ¬ p1) ∧ ¬p2
p1 ∧ x=1 ∧ p2 ∧ y=2
7

Sensitivity to Traversal Order


1
2
3
4
5
6
7
[p1]
[¬p1]
x = 1
[p2]
[¬p2]
y = 2
true
p1
¬p1
(p1 ∧ x=1) ∨ (¬p1)
(p1 ∧ x=1 ∧ p2) ∨ (((p1 ∧ x=1) ∨ ¬p1) ∧ p2)
((p1 ∧ x=1) ∨ ¬ p1) ∧ ¬p2
p1 ∧ x=1 ∧ p2 ∧ y=2
((p1 ∧ x=1 ∧ p2) ∨ (((p1 ∧ x=1) ∨ ¬p1) ∧ p2)) ∧ y=2

Sensitivity to Traversal Order

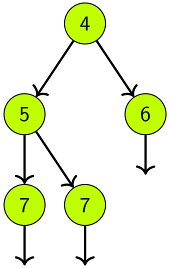



1
true
p1
2
3
¬p1
(p1 ∧ x=1) ∨ (¬p1)
4
(p1 ∧ x=1 ∧ p2) ∨ (((p1 ∧ x=1) ∨ ¬p1) ∧ p2)
5
6
((p1 ∧ x=1) ∨ ¬ p1) ∧ ¬p2
(p1 ∧ x=1 ∧ p2 ∧ y=2) ∨ (((p1 ∧ x=1 ∧ p2) ∨ (((p1 ∧ x=1) ∨ ¬p1) ∧ p2)) ∧ y=2)
7

(to be continued)

Consequences

- Large redundant formulas
- Blow-up of path exploration
- If a SAT check is performed at end of path:
  many and redundant SAT checks
  (first path is checked over and over again)
- If some state on path is non-mergeable, e.g., at node 7:
  - ARG branching at node 5
    (no branching in CFA here)
  - unexpected ARG shape
  - problems for counterexample reconstruction (#883)



Why not just fix traversal order?

Desired traversal order is *reverse post order*, but:

- Sometimes hard to implement
  Example: unrolling with nested loops (#1002)
- Other traversal orders sometimes preferrable,
  e.g. for validating violation witnesses:
  Checking paths eagerly (DFS) more efficient
  than checking all paths together (cf. #907)
- Incorrect traversal order hard to detect
  (only if counterexample reconstruction happens to crash)

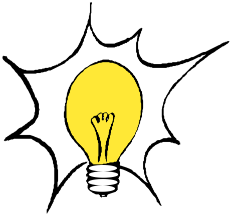
Definition (Late Merge)

merge of abstract states where one state already has children

Solution

- In CPA framework, merging is handled by merge(e, e') (merging e into existing abstract state e')
- merge can decide *whether* to merge!
- E.g., merge$(e, e') = e'$ is valid and avoids all merges

$\Rightarrow$ Just make merge behavior depend on late merge



Implementation

- Configuration option `cpa.arg.lateMerge`
- Possible values:

`allow`	Perform merge as usual (default)
`prevent`	Do not merge if e' has children
`crash`	Throw exception if late merge happens (if configured like an assertion that current analysis should not produce merges)

- Implemented in merge of ARGCPA, no algorithm change

Note

- Does not solve all problems!
- Path exploration may still blow up.
- But better than nothing:
  - Redundant SAT checks avoided if each path is checked
  - No unexpected ARG shape
    (no crash in counterexample creation)
  - Helps debugging when solving the root cause

Current State

- `prevent` used in BMC as workaround for edge cases like #1002
- `prevent` used for internal counterexample checks performed with Predicate CPA
- `prevent` proposed for validating violation witnesses (#907)
- `crash` evaluated for standard predicate analysis, but found another case of suboptimal traversal order (#1004)

Take-Home Messages

- Term “late merge”
- Best traversal order hard to ensure for all edge cases
- …but lots of things depend on it!
- Suboptimal traversal order hard to notice
- Use `cpa.arg.lateMerge = prevent` as workaround.
- Use `cpa.arg.lateMerge = crash` where possible!
- Flexible combination of independent components does have problems.
- Consider safeguards such as assertions.

References

Beyer, D., Dangl, M., Wendler, P.: A unifying view on SMT-based software verification. J. Autom. Reasoning **60**(3), 299–335 (2018). https://doi.org/10.1007/s10817-017-9432-6

Beyer, D., Keremoglu, M.E., Wendler, P.: Predicate abstraction with adjustable-block encoding. In: Proc. FMCAD. pp. 189–197. FMCAD (2010)

CPA Algorithm

```
$Reached, Waitlist := \{e_0\}$
while $Waitlist \neq \emptyset$ do
    remove $e$ from $Waitlist$
    for all $e' \in \underline{\textbf{post}}(e)$ do
        for all $e'' \in Reached$ do
            $e''_{new} := \underline{\textbf{merge}}(e', e'')$
            if $e''_{new} \neq e''$ then
                replace $e''$ in $Reached, Waitlist$ by $e''_{new}$
        if $\neg\underline{\textbf{stop}}(e', Reached)$ then
            add $e'$ to $Reached, Waitlist$
return $Reached$
```