k-FAIR = *k*-LIVENESS + FAIR
Revisiting SAT-based Liveness Algorithms

FMCAD’18

Alexander Ivrii
Ziv Nevo
Jason Baumgartner
IBM

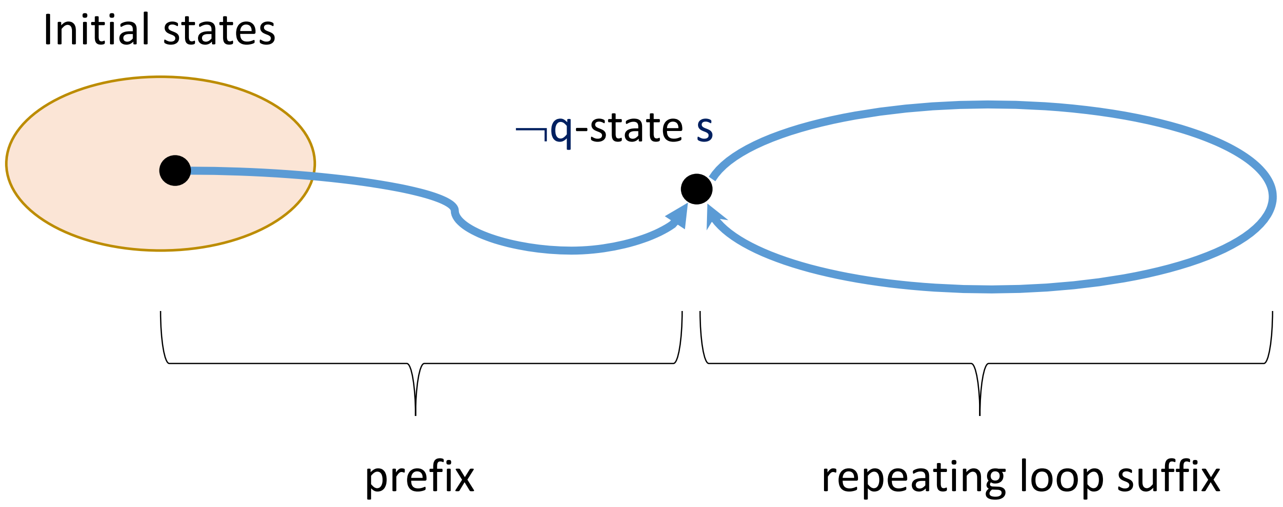
Personal perspective

- Support for ***liveness properties*** in IBM's formal hardware verification tool:
  - ➕ BDD-based algorithms (Clarke-Grumberg-Peled, '99)
    - Compute all reachable states
    - Less scalable but occasionally useful
  - ➕ Liveness-to-Safety translation (Biere-Artho-Schuppan, FMICS'02)
    - Replaces verification of liveness properties by verification of safety properties
    - Extremely useful
  - ⛔ FAIR (Bradley-Somenzi-Hassan-Zhang, FMCAD'11)
    - Will describe in a minute
    - Seems extremely interesting but also hard to implement
  - ➕ *k*-LIVENESS (Claessen-Sörensson, FMCAD'12)
    - Will describe in a minute
    - Quite useful
  - ➕ *k*-FAIR = *k*-LIVENESS + FAIR
    - Will describe in a minute
    - Quite useful

Liveness properties

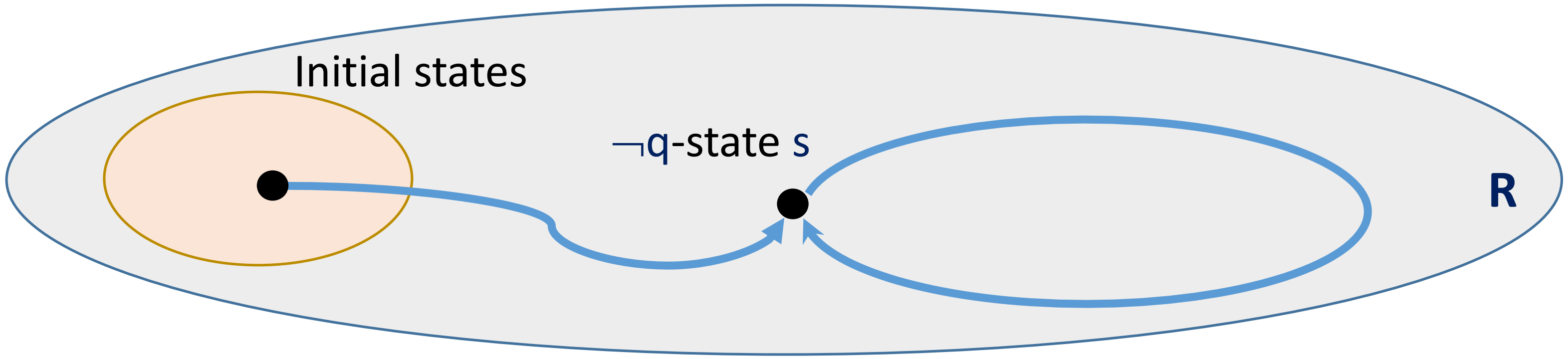
- Consider liveness properties of the form FGq
  - "on every trace q eventually becomes true forever"

- A counterexample to FGq is an infinite trace on which q becomes false infinitely often
  - Commonly represented as a lasso-shaped trace:

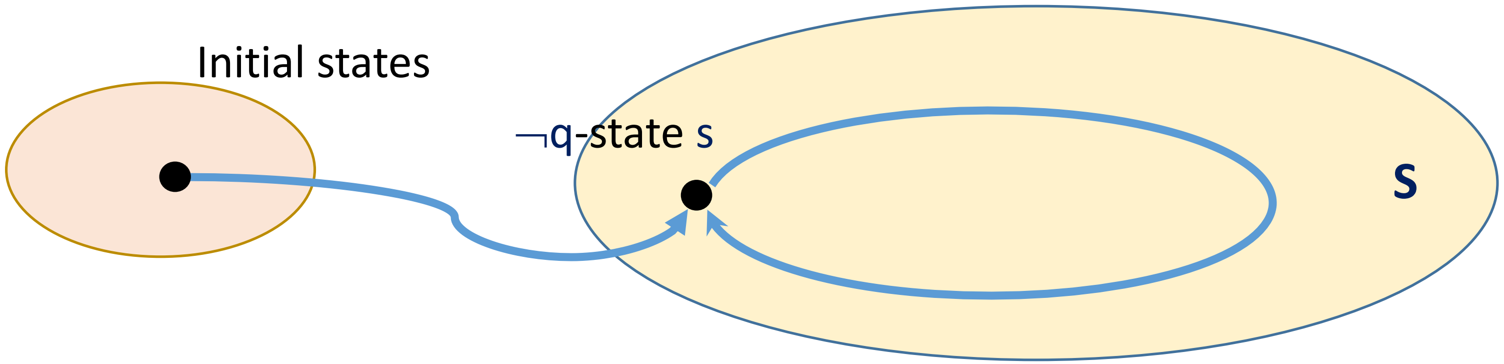


Reachability and stabilizing constraints

- A reachability constraint **R** indicates that all states on a potential lasso-shaped counterexample belong to **R**.



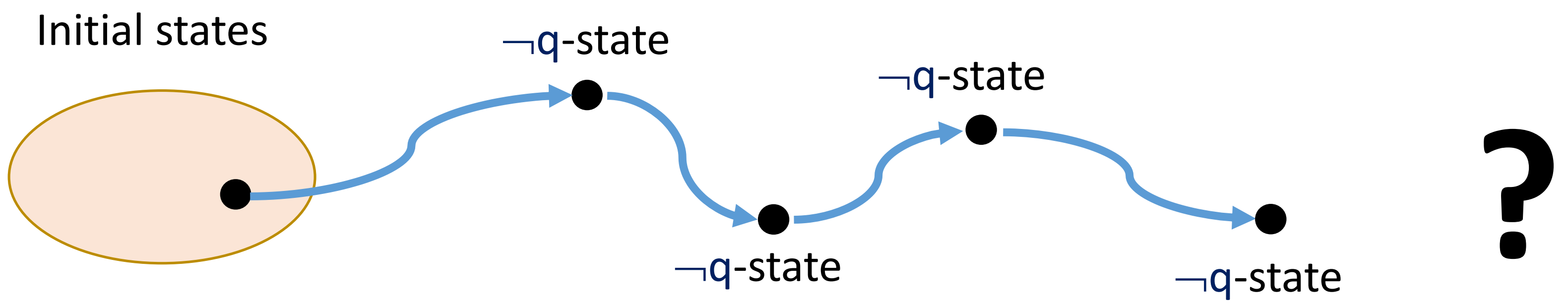
- A stabilizing constraint **S** indicates that all states <u>on the loop</u> of a potential lasso-shaped counterexample belong to **S**.



k-LIVENESS (Claessen-Sörensson, FMCAD'12)

- *k*-LIVENESS tries to bound the number of times that q can become false on any trace.

- Start with $k = 1$.
- ***Make safety query***: is there a trace with at least *k* occurrences of ¬q?



  - If such a trace does not exist: FGq passes.
  - If such a trace exists ***and*** has a repeated ¬q-state: FGq fails.
  - Otherwise: increase *k*.

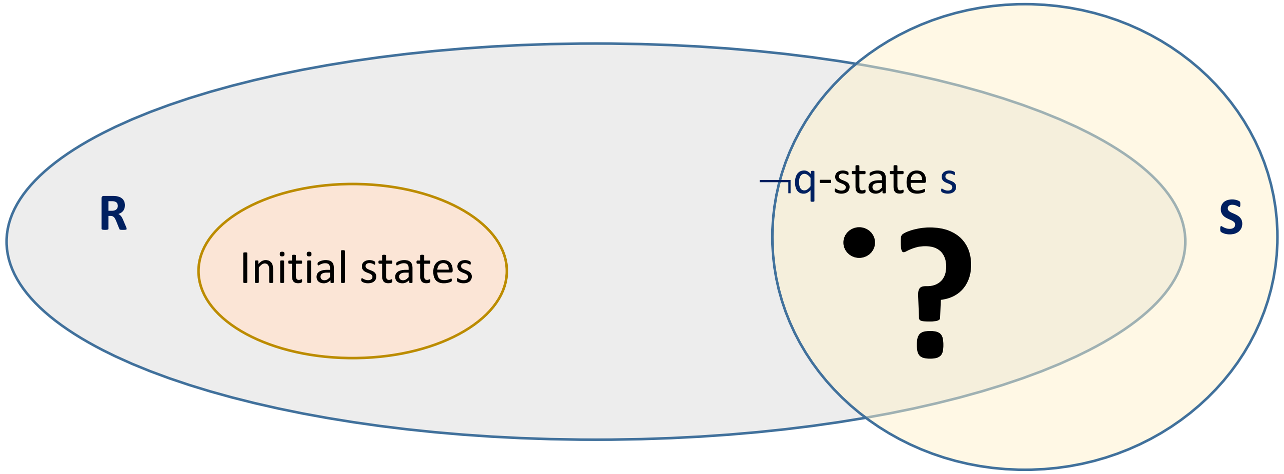
- Any safety model checker can be used (capable of proving properties).
- Using IC3 offers advantage due to close relation between queries for different *k*.

FAIR (Bradley-Somenzi-Hassan-Zhang, FMCAD'11)

- The presented variant is specialized to properties of the form FGq, and is slightly less general (but simpler).

- FAIR incrementally learns reachability and stabilizing constraints on the state space.

- ***Make SAT query***: "is there a ¬q-state s, subject to previously discovered reachability and stabilizing constraints?"



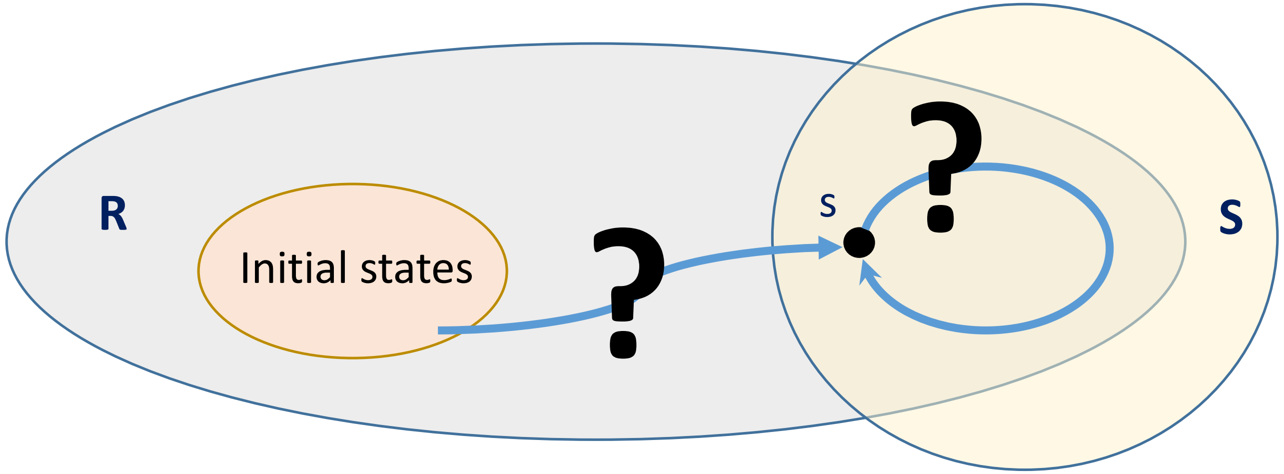
- If such a state does not exist, FGq passes.

continued...

FAIR (Bradley-Somenzi-Hassan-Zhang, FMCAD'11)

- Let's suppose such a state s exists. Let's check if s can be completed to a lasso-shaped counterexample.

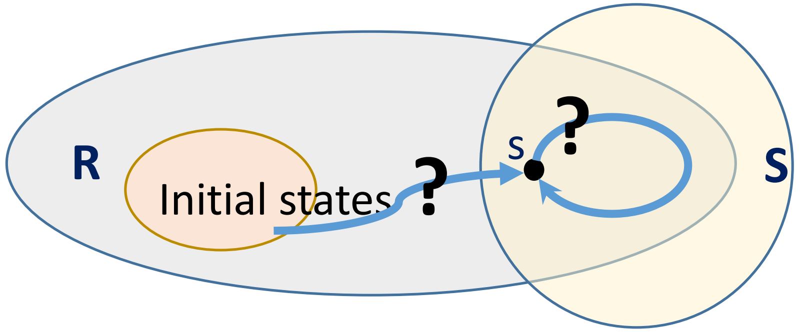
- ***Make safety query***: "is s reachable from initial states, subject to previously discovered reachability constraints?"
- ***Make safety query***: "does s have a loop to itself, subject to previously discovered reachability and stabilizing constraints?"



continued...

FAIR (Bradley-Somenzi-Hassan-Zhang, FMCAD'11)

- If both safety queries are satisfiable: FGq fails.
- For unsatisfiable safety queries, the model checker needs to be capable of producing inductive proofs of unsatisfiability (commonly IC3 is used).
- If s is not reachable from initial states:
  - The inductive proof represents a new reachability constraint (excluding s).
- If s does not have a loop to itself:
  - We can generalize s to a larger set of states without a self-loop.
  - The complement of this set represents a new stabilizing constraint (excluding s).
- In either case, the algorithm makes progress.

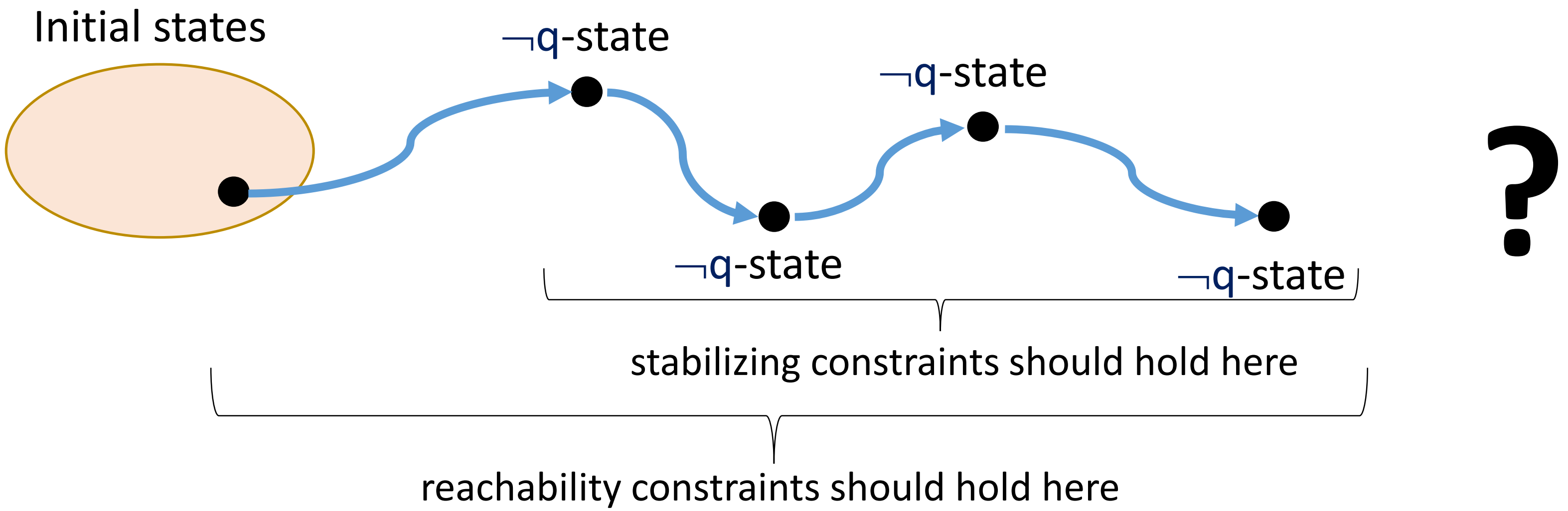


k-Liveness vs. FAIR

- The two algorithms have different strengths:
  - *k*-LIVENESS works well when a small value of *k* is sufficient (otherwise, underlying safety queries usually become complicated).
  - FAIR works well when inductive proofs restrict large portions of state space (otherwise, many iterations are required).
- The presented algorithm *k*-FAIR combines ideas from both approaches.

k-FAIR (the feature presentation)

- Start with $k = 1$.
- ***Make safety query***: "is there a trace with at least k occurrences of ¬q, subject to previously discovered reachability and stabilizing constraints?"



- If such a trace does not exist: FGq passes.
- If such a trace exists ***and*** has a repeated ¬q-state: FGq fails.

continued…

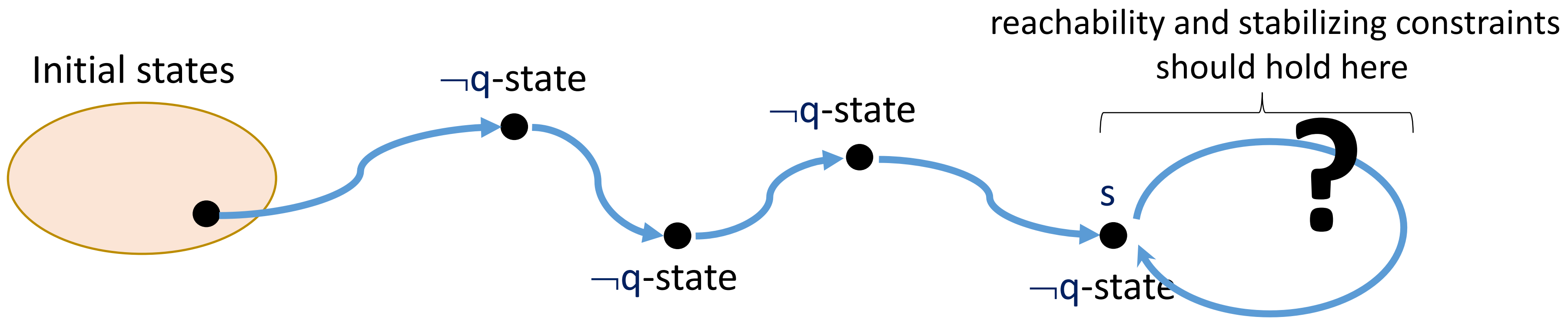
k-FAIR (the feature presentation)

<u>One option</u>:

- Increase *k*.

<u>Another option</u>:

- Select any ¬q-state s from the trace.
- ***Make safety query***: "does s have a loop to itself, subject to previously discovered reachability and stabilizing constraints?"



- If s has a loop to itself: FGq fails.
- If s does not have a loop to itself: extract a new stabilizing constraint (excluding s).
- We have a choice to increase k, leave k the same, or even to decrease k.

Finding additional stabilizing constraints

- Both *k*-LIVENESS and FAIR have an important optimization of looking for single-literal stabilizing constraints.

- There are certain differences:
  - The technique in *k*-LIVENESS is **stronger**, but is used only as **preprocessing**.
    - Considers all nets in the circuit as candidates; uses liveness signal in a stronger way.
  - FAIR looks for new constraints **periodically**.
    - Leverages new reachability and stabilizing constraints.

- *k*-FAIR uses the best of both worlds:
  - Uses the stronger technique; can apply this technique as preprocessing and periodically.
  - *Minor technicality*: how to pass stabilizing constraints to the safety model checker?
    - See the paper for details

Experiments: setup & configurations

- All single-property liveness benchmarks from HWMCC'11 – HWMCC'17, and various in-house
- Preprocessed using standard logic synthesis techniques (à la ABC)
- 3 hours time-limit

- *k*-FAIR Configurations:

	Stabilizing Constraints	Increase k	Look for self-loop
FAIR	Periodically	Never	Every iteration
Combined-50	Periodically	Every 50th iteration	Every iteration
Combined-5	Periodically	Every 5th iteration	Every Iteration
Improved *k*-LIVENESS	Periodically	Every iteration	Never
Standard *k*-LIVENESS	Only initially	Every iteration	Never

- LTS-BMC: Liveness-to-Safety, followed by Bounded Model Checking
- LTS-IC3: Liveness-to-Safety, followed by IC3

Experiments: results

	PASS solved	PASS time	FAIL solved	FAIL time
FAIR	108	338,475	89	351,634
Combined-50	111	301,592	94	321,260
Combined-5	122	166,655	104	240,458
Improved *k*-LIVENESS	123	173,077	97	245,543
Standard *k*-LIVENESS	117	250,431	100	225,971
LTS-BMC	–	–	114	94,103
LTS-IC3	116	225,059	99	226,321
Virtual Best	131	37,270	117	29,315

(green) – best configuration

(yellow) – best additional value

Experiments: conclusions

- Falsification:
  - Liveness-to-Safety followed by BMC is a very strong strategy
  - Combined-5 brings most additional value
  - Running all 7 configurations significantly improves performance further

- Proof:
  - Improved *k*-LIVENESS performs best overall
  - Liveness-to-safety followed by IC3 brings most additional value
  - Running other configurations did not further improve performance at all

- *What we want to be true, but experiments don't show:*
  - Intuitively, *k*-FAIR should especially help when FGq holds:
    - "When *k*-LIVENESS gets stuck, we can stop increasing *k* and apply FAIR".
  - In retrospect, the considered *k*-FAIR configurations are not very intelligent.

Comparison of FAIR in *k*-FAIR and in IIMC

	PASS solved	PASS unique solved	PASS time	FAIL solved	FAIL unique solved	FAIL time
FAIR in *k*-FAIR	108	18	208,875	89	21	70,834
FAIR in IIMC	101	11	277,657	70	2	242,762

- Both variants have unique value
- Overall, FAIR in *k*-FAIR performs substantially better than FAIR in IIMC
  - However, the improvements may be due to a large number of different factors
  - Detailed comparison is difficult
- *The (simplified) variant of FAIR presented here is a viable alternative to the implementation in IIMC*

Summary:

- ***SAT-based liveness is a pure FMCAD-driven research.***
- ***Quick takeaway***: *periodically* detecting single-literals stabilizing constraints improves *k*-LIVENESS, and is simple to implement.
- ***A slightly less quick takeaway***: combining the strengths of *k*-LIVENESS and FAIR brings unique value, and is not too difficult to implement.
- ***There is a lot of room for further improvement***: devising better *k*-FAIR strategies, carefully balancing the efforts spent in different *k*-FAIR components, improving underlying IC3 implementation towards safety queries posed by *k*-FAIR, etc.

Thank You !