Games on Graphs
Breaking the O(n m) Barrier for Buechi Games and Maximal End-Component Decomposition

Monika Henzinger

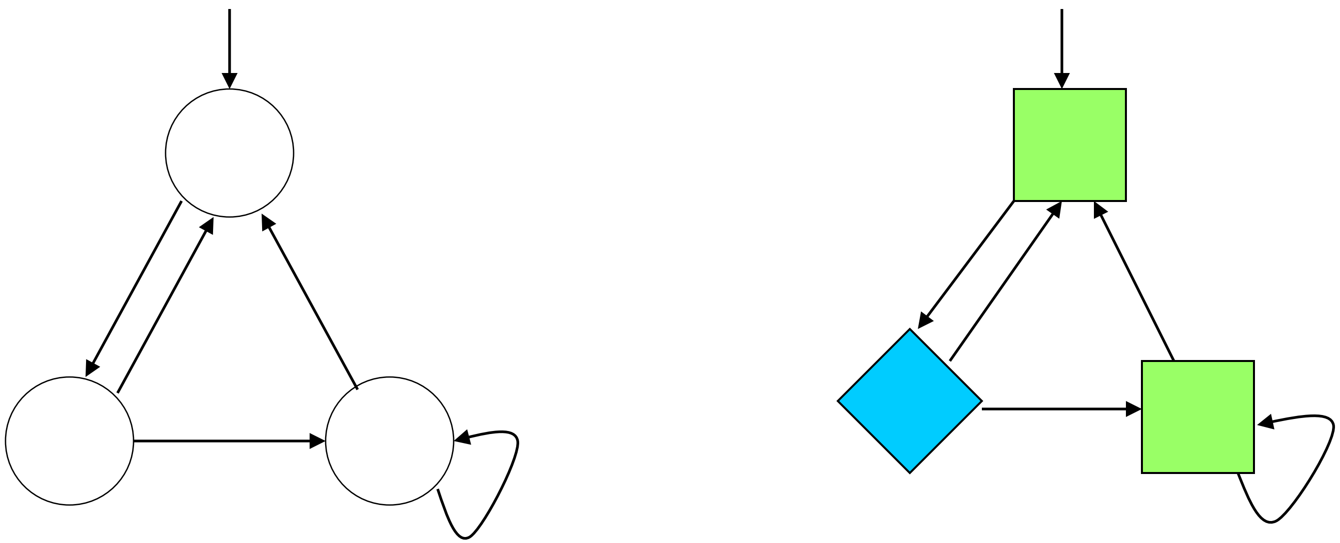
joint work with Krishnendu Chatterjee (IST Austria)

universität wien

This Talk

- Two classical algorithmic problems related to graph games and verification of probabilistic systems:
  - Buechi games
  - Maximal end-component (MEC) decomposition
- Long-standing best known time bounds: O(n m).
- Here: $O(n^2)$ and better ...

Graphs vs. Game Graphs



2 types of nodes representing 2 interacting players in games:

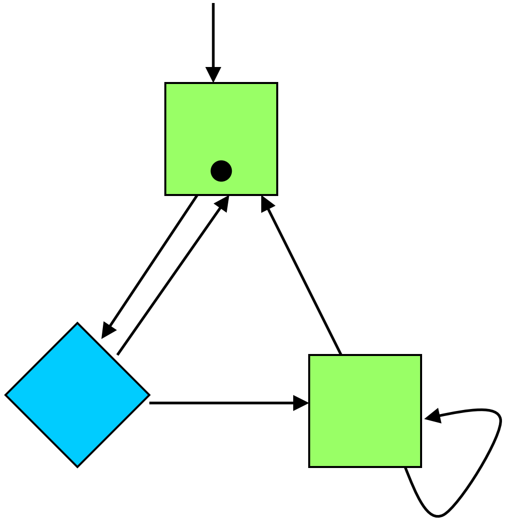
- Player 1 (Box)
- Player 2 (Diamond)

Game Graphs

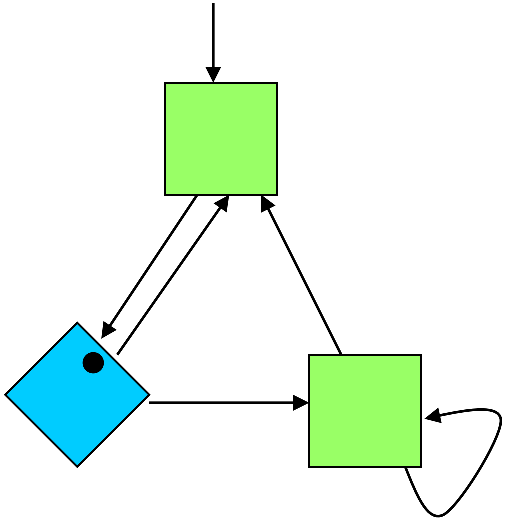
- A game graph $G = ((V,E), (V_1, V_2))$
  - Player 1 states (or vertices) V_1 and player 2 states V_2, and (V_1, V_2) partitions V.
  - E is the set of edges.
    - We assume every vertex has at least one out-edge.
    - Player-*i* edge: out-edge of player *i*
  - Notation: $n = |V|$, $m = |E|$.

- Game played by moving one token forever:
  - For *i* = 1, 2: When player *i* vertex, then player *i* chooses which out-going edge the token traverses next.

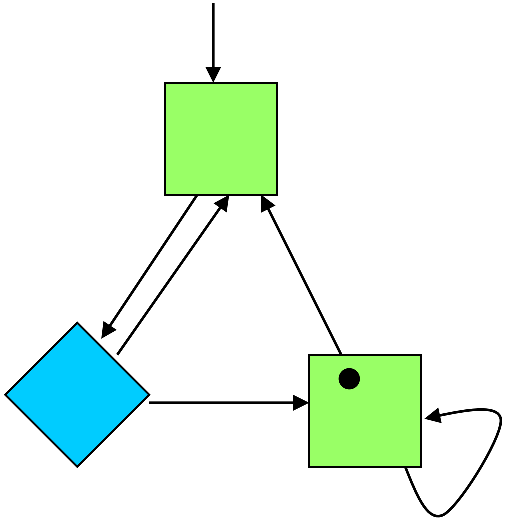
Game Example



Game Example



Game Example



Strategies

- Strategies are rules how to move tokens or how to extend plays.
- Formally, given a history of play (= finite sequence of states) and the current vertex is player *i* vertex, the strategy of player *i* chooses an out-going edge.
  - Player 1 strategy s_1: $V^* \times V_1 \rightarrow V$.
  - Player 2 strategy s_2: $V^* \times V_2 \rightarrow V$.

Goal of graph game?

- Reachability objective: Given a set T of vertices, the objective is the set of infinite paths that visit the target T *at least once*.
- Buechi objective: Given a Buechi set B of vertices, the objective is the set of infinite paths that visit some vertex in B *infinitely often*.
- Winning set: Set of vertices v such that player 1 has *a* strategy to ensure the objective starting at v against *all* strategies of player 2.
- **Remark**: Memoryless strategies are sufficient.
  - Strategy only depends on *current vertex*, not history
- **This talk**: Compute the winning set A for player 1 for Buechi objectives, i.e. *a set of vertices v s. t. there exists a strategy for player 1 that starting from v a vertex of B is visited infinitely often, no matter how player 2 plays*

Motivation for Buechi games

- Formal analysis of reactive system:
  - Vertices represent states, edges represent transitions, infinite paths represent behaviors, and players agents (system vs environment).

- Many other applications in verification
  - Synthesis of specifications given as Buechi automata.
  - Model checking one-alternation[1]-calculus.
  - Numerous other applications in verification.

Previous Result

- Reachability games:
  - O(m) (linear time algorithm) and PTIME-complete [Immerman 81, Beeri 80].
  - Winning set aka alternating reachability set

- Buechi games:
  - Classical algorithm: O(nm) [EJ91].
  - In the special case when m = O(n), an $O(n^2/\log n)$ algorithm [CJH03]

Buechi Games Algorithm

Classical Algorithm

- A simple iterative algorithm using alternating reachability.

- Steps are as follows:
  1. Compute player-1 alt-reach set A to the current Buechi set.

  2. If A is the set of all vertices of current game graph, then stop and output A as the (Buechi-)winning set.

  3. Else U= V \ A. Remove player-2 alt-reach set C to the set U from game graph and continue at Step 1.

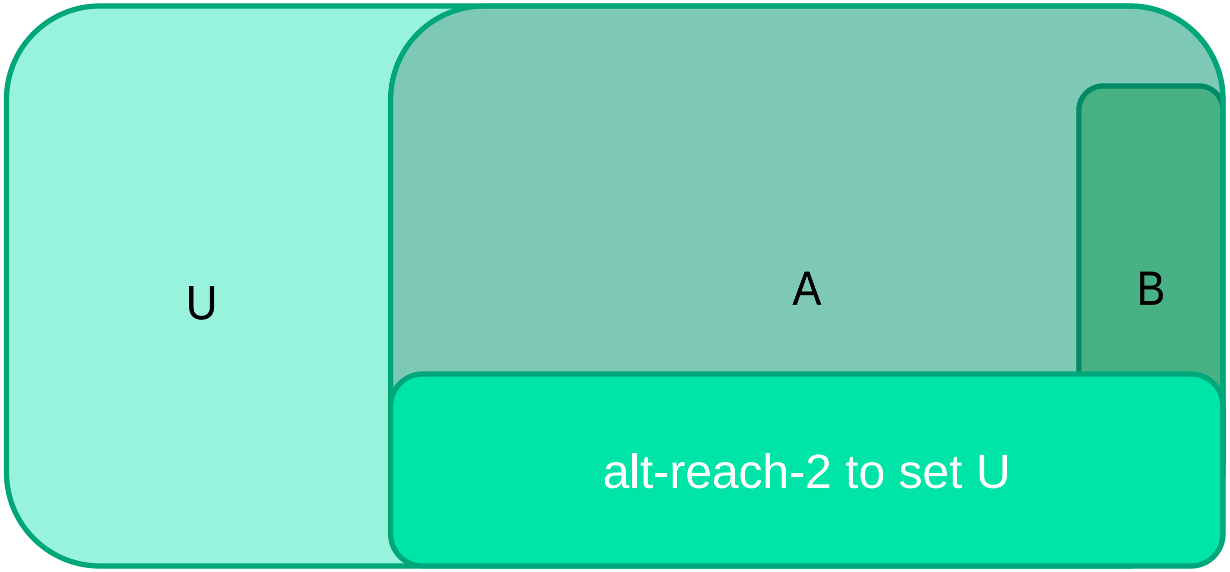
Classical Algorithm

- Compute player 1 alt-reach set A to the set B.

Classical Algorithm

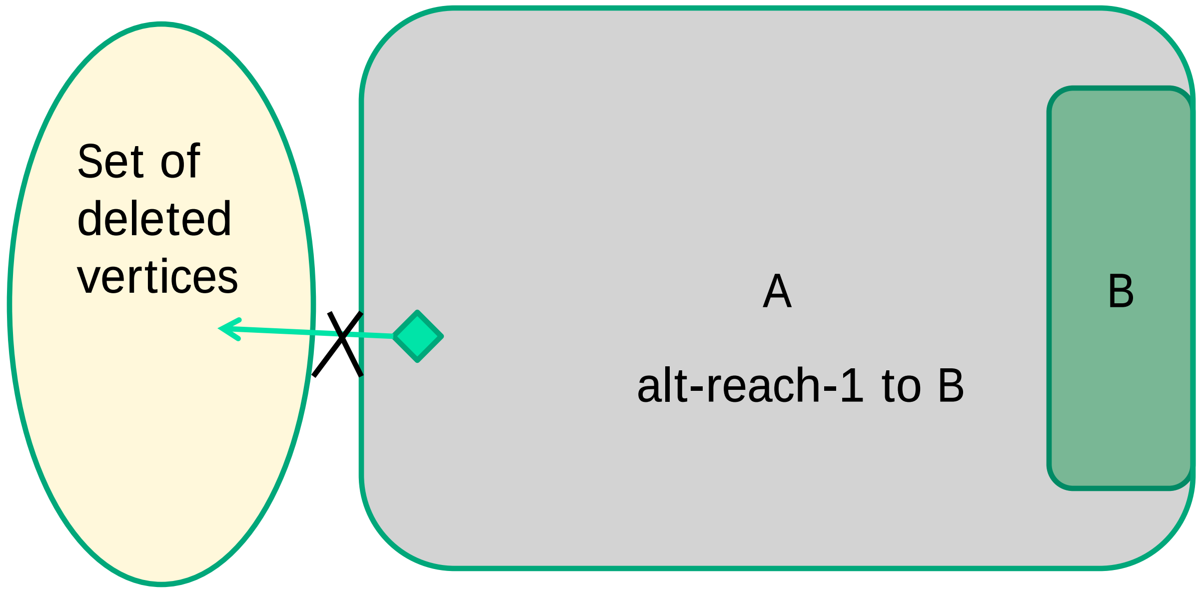
- Let U= V \ A. Then U is a trap. Clearly, no vertex of U is winning for player 1.
  - Trap U:
    - every player-1 edge stays in U
    - every player 2 has at least one edge in U
    - no vertex of U belongs to B
- Hence alt-reach for player 2 to U is also not winning for player 1.

Classical Algorithm



- Iterate on the remaining sub-graph.
- Every iteration what is removed is not part of winning set.
- When the iteration stops, all remaining vertices are winning for player 1.
  - Why?

Correctness Proof Idea



- By construction of A
  - Player 2 cannot have an edge from A to a deleted vertex.
  - Every player 1 vertex has at least one edge to a vertex in A

$\Rightarrow$ Player 1 can ensure from A to reach B, and then to get back to A again, and so on and on.

$\Rightarrow$ A is winning set for player 1.

Classical Algorithm

- Classical algorithm identifies in each iteration the *largest trap* and removes it until no trap exists anymore

$\Rightarrow$ Remaining set is winning set

- **Analysis**: O(nm) total time
  - At most n iterations each performs two alt-reachability computations
    - Take time O(m) each
  - O(nm) is tight for classical algorithm

- **Remark**: Total time for player-2 alt-reachability computation over all iterations is O(m)
  - Edges worked on are removed from the graph

  $\Rightarrow$ need only to speed up time for player-1 alt-reachability

Our New Algorithm

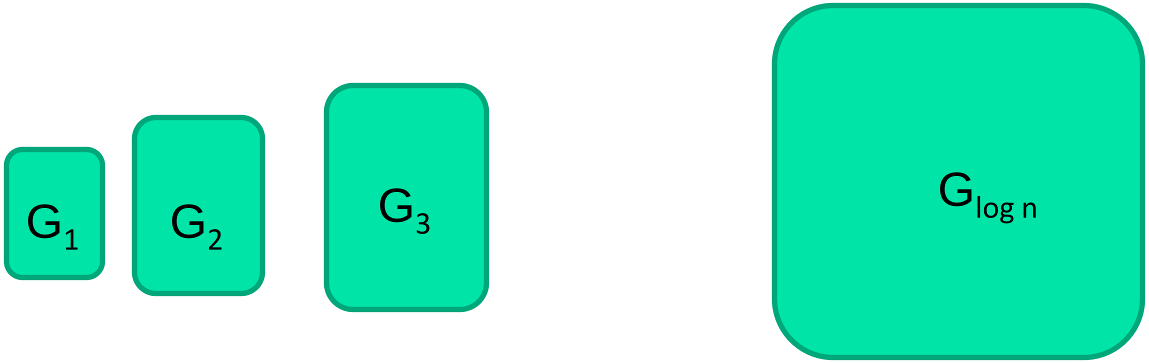
- **Idea 1:** As long as we find traps, we can remove them, need not find the largest trap.

- **Idea 2:** Hierarchical graph decomposition technique
  - Try first to find traps in sparse graphs

- **Running time:** $O(n^2)$
  - Better worst case for dense graphs.
  - Along with previous [CJH03] algorithm breaks O(nm) for all cases.

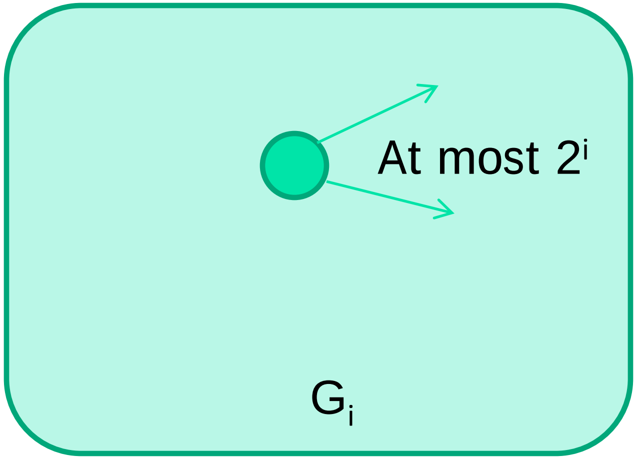
New Algorithm

- For $i = 1, \ldots, \log n$: Build game graph $G_i=(V,E_i)$ s.t.
  - $|E_i| = O(2^i n)$ and
  - Graph G_{i-1} is a sub-graph of G_i.
    - Use fixed ordering of in-edges and out-edges
    - For in-edges order edges from player-2 non-Buechi vertices before all other edges.



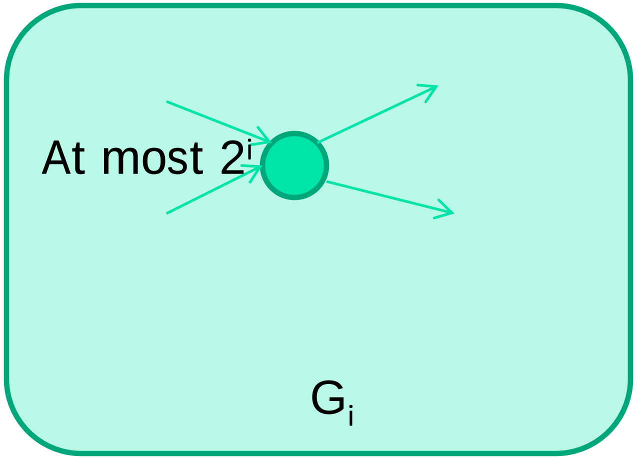
Construction of G_i

- For every vertex add the first 2^i out-edges (or all if its out-degree < 2^i).



Construction of G_i

- For every vertex add the first 2^i out-edges (or all if its out-degree < 2^i)
- Additionally for every vertex add the first 2^i in-edges (or all if its in-degree < 2^i)



Construction of G_i

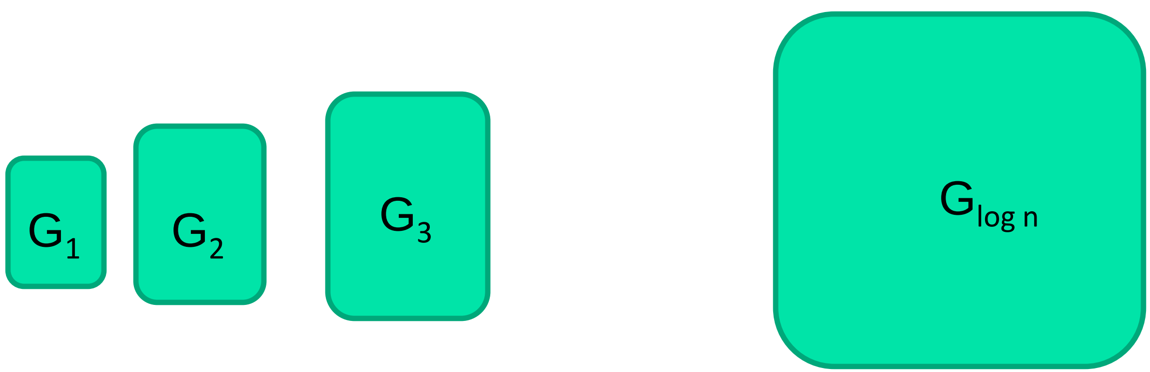
- For every vertex add the first 2^i out-edges (or all if its out-degree < 2^i)
- Additionally for every vertex add the first 2^i in-edges (or all if its in-degree < 2^i)

⇒ Graph G_{i-1} is a sub-graph of G_i

⇒ $G = G_{\log n}$

New Algorithm

1. i = 1
2. While i < log n +1
   - Search for traps in G_i that are also traps in G
   - If such trap U is found then
     - Compute the player-2 alt-reach set C to U, remove it from **all** graphs G_j, and goto Step 1
   - i = i + 1
3. Output remaining vertices as winning set



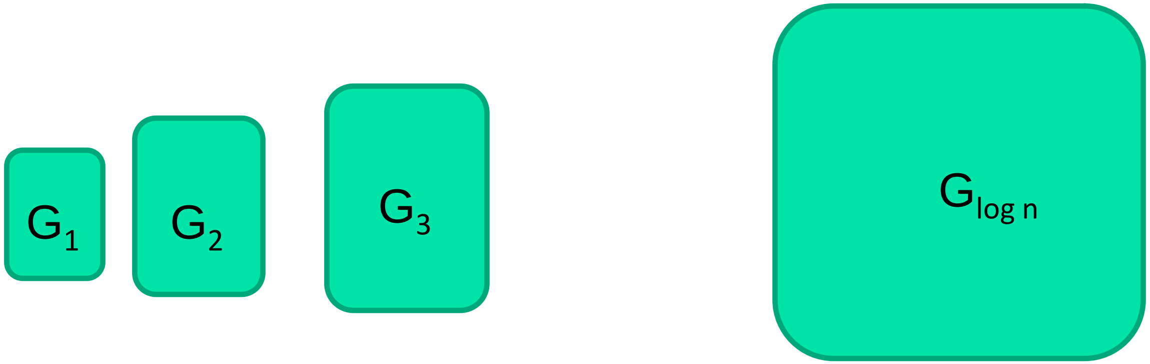
New Algorithm (cont.)

- Call a player-1 vertex with out-degree > 2^i blue in G_i.
- **Problem**: Traps in G_i that contain blue vertices might not be traps in G
- **Idea**: Only search for traps without blue vertices
- **Implementation**: Treat blue vertices like Buechi vertices in player-1 alt reachability computation

- Search for traps in G_i:
  - Compute player-1 alt reachability set A to the set of Buechi **or blue** vertices.
- **Claim:** If V \ A is non-empty, then it is a trap in G.

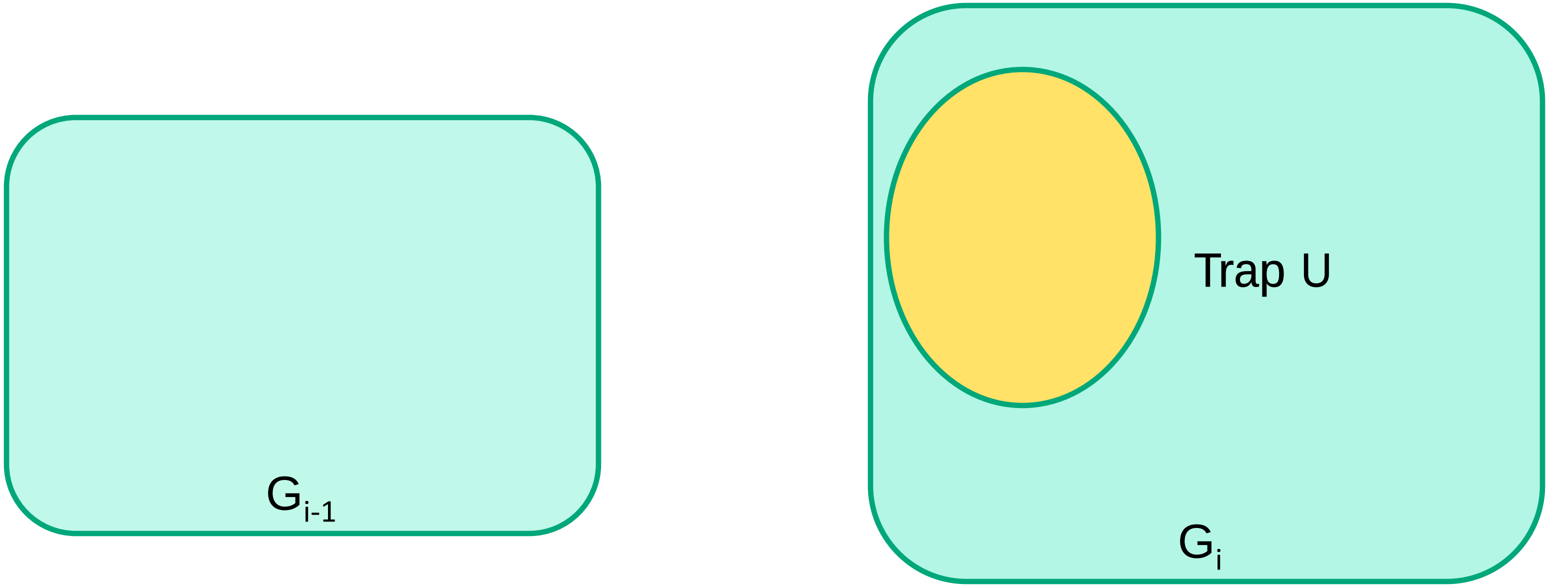
Correctness of New Algorithm

- **Claim:** If V \ A is non-empty, then it is a trap in G.
- **Proof sketch:** When identify a trap U, then all player-1 vertices in U are not-blue
  - All their out-edges of G are in G_i and thus in U
  - No player-1 vertex has an out-edge leaving U in G
  - Every player-2 vertex in U has an out-edge in G_i and thus in G
- When algorithm stops no more traps exist in G as G = $G_{\log n}$
- No traps implies that remaining vertices form winning set (as for classical algorithm)



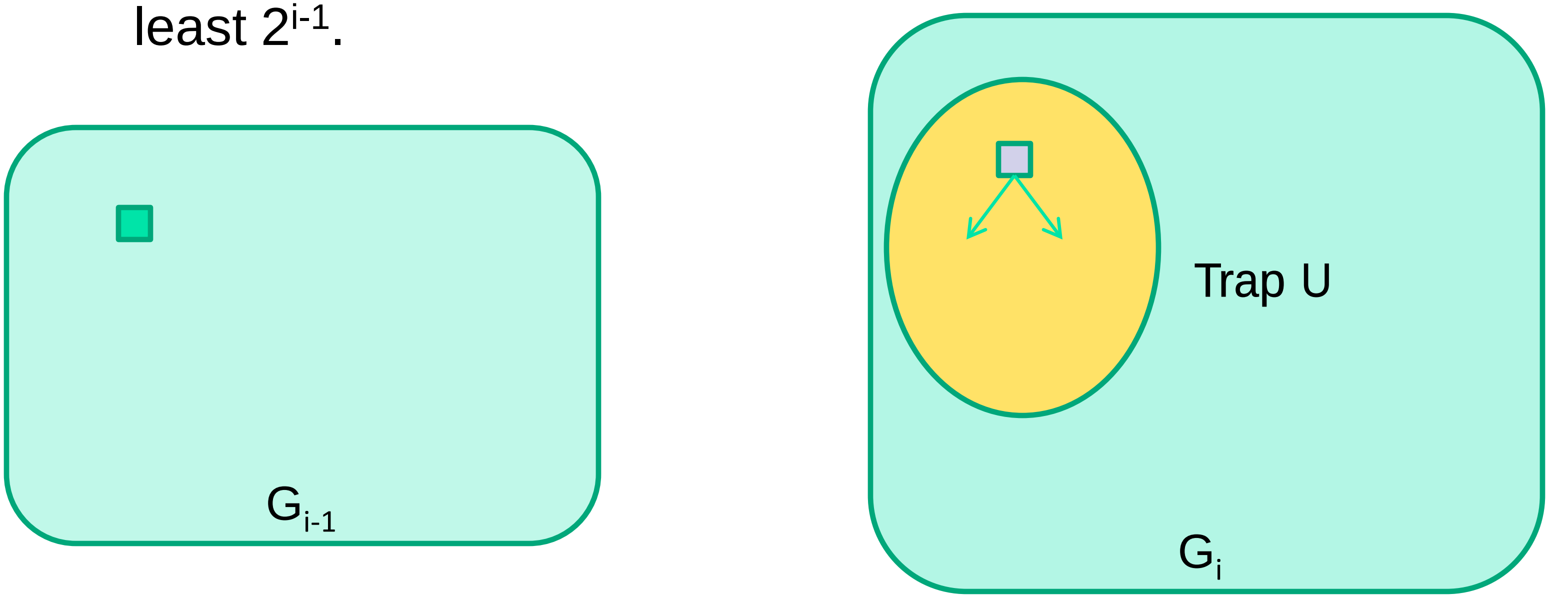
Running Time Analysis

- Analysis of the size of the trap.
  - Trap U identified in G_i but not in G_{i-1}.
  - We analyze the size of the trap we identify.



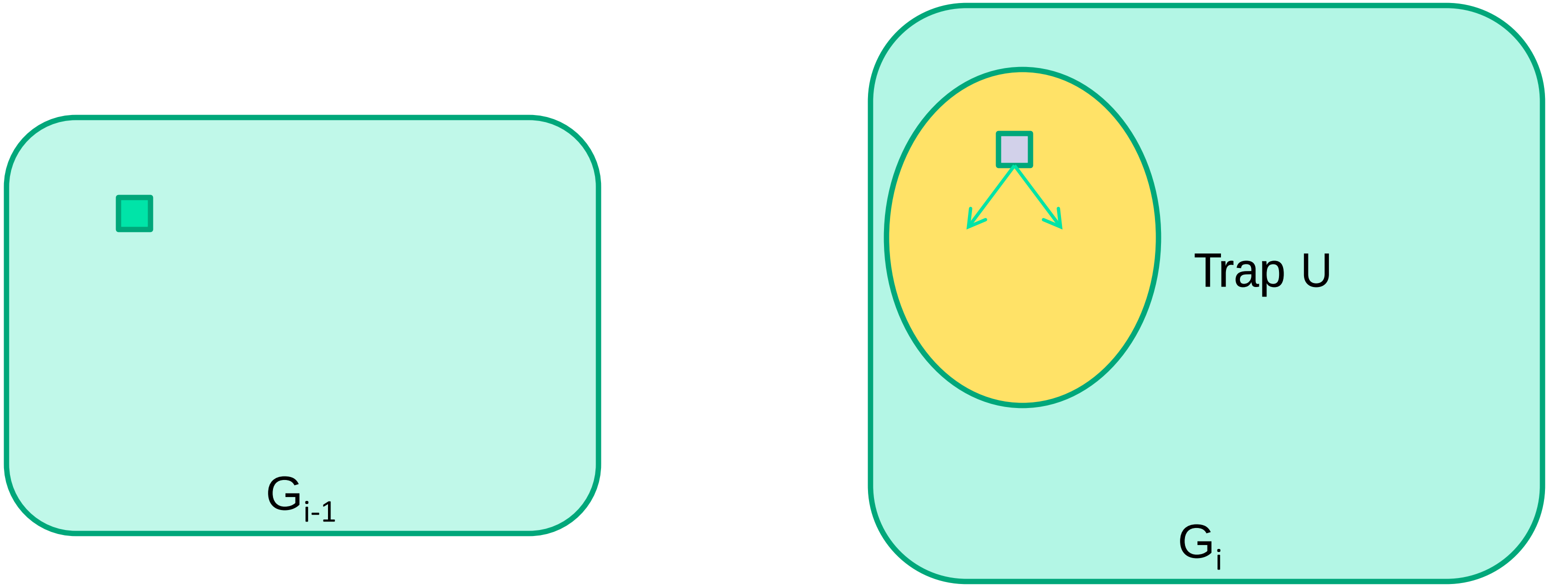
Running Time Analysis

- Analysis of the size of the trap.
  - Trap U identified in G_i but not in G_{i-1}.
  - Case 1: U contains a player-1 vertex v that was blue in G_{i-1}.
  - Then v has at least 2^{i-1} out-edges, otherwise would not have been blue.
  - Since a trap contains all out-going edges from v, size of trap at least 2^{i-1}.



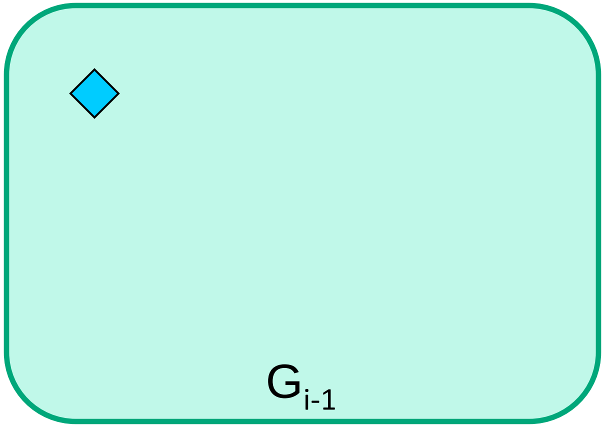
Running Time Analysis (cont.)

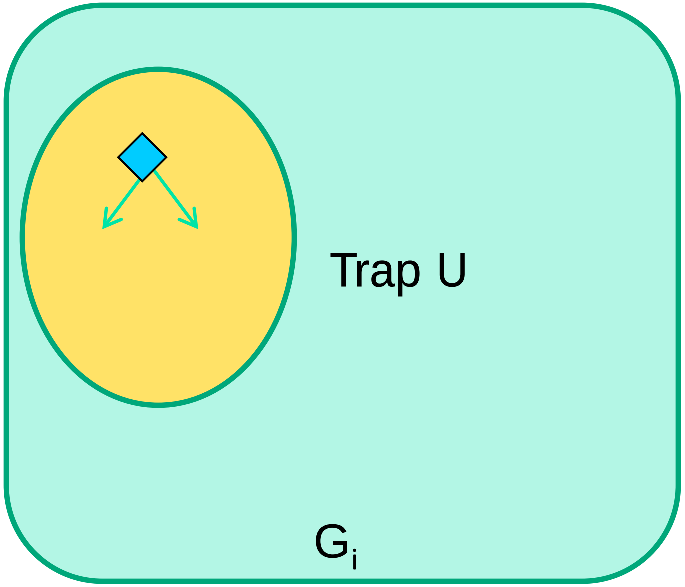
- Analysis of the size of the trap.
  - Trap U identified in G_i but not in G_{i-1}.
  - Case 2: U does not contain a player-1 vertex v that was blue in G_{i-1}. All player-1 edges in G_i and G_{i-1} identical.
    - Two sub-cases to analyze.



Running Time Analysis (cont.)

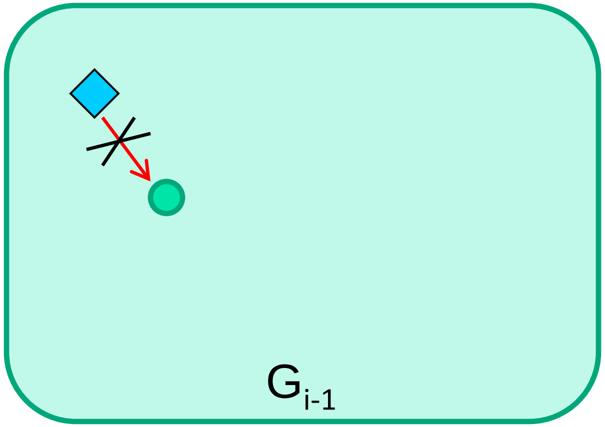
- Analysis of the size of the trap.
  - Case 2: All player-1 edges in G_i and G_{i-1} identical.
  - Case 2 (a): All player-2 edges in G_i and G_{i-1} are identical. Then U is a trap in G_{i-1} and this a contradiction.
  - Case 2(b): One new player-2 edge in the trap.

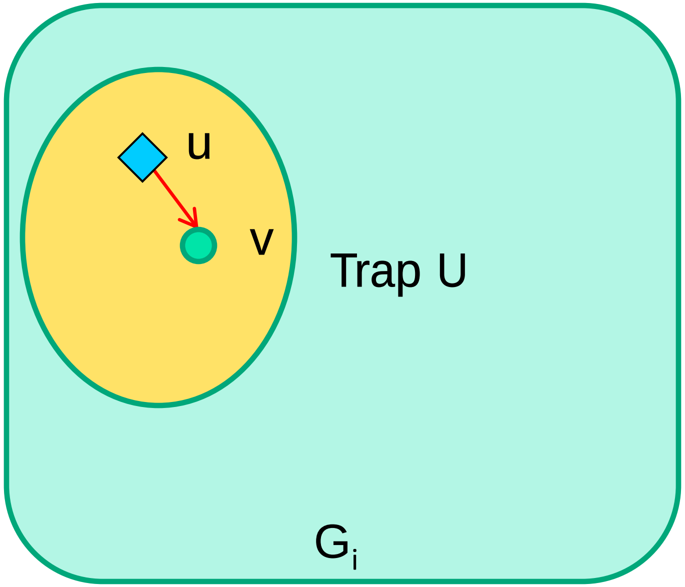




Running Time Analysis (cont.)

- Analysis of the size of the trap.
  - Case 2: All player-1 edges in G_i and G_{i-1} identical.
  - Case 2(b): One new player-2 edge (u,v) in the trap.
    - Vertex u is player-2 non-Buechi vertex, i.e., (u,v) has priority in order of in-edge
    - Vertex v has at least 2^{i-1} in-edges before (u,v) in order of in-edges $\Rightarrow$ v has at least 2^{i-1} in-edges from player-2 non-Buechi vertices



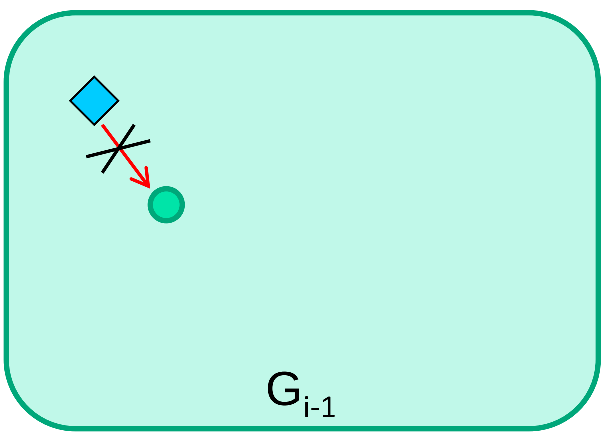


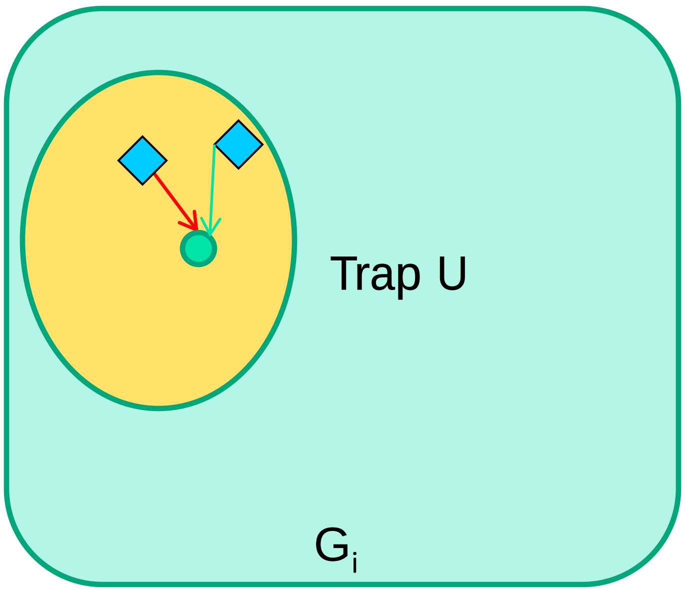
Running Time Analysis (cont.)

- Analysis of the size of the trap.
  - Case 2(b): One new player-2 edge (u,v) in the trap.
    - Vertex v has at least 2^{i-1} in-edges from player-2 non-Buechi vertices $\Rightarrow$ By construction of U no player-2 non-Buechi vertex of V \ U has an edge to U

    $\Rightarrow$ All in-edge of v belong to U

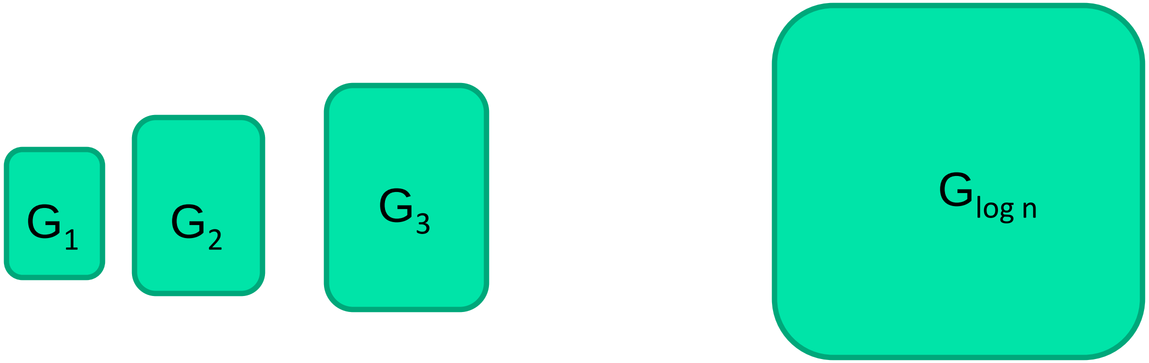
    $\Rightarrow$ U has at least 2^{i-1} vertices





New Algorithm

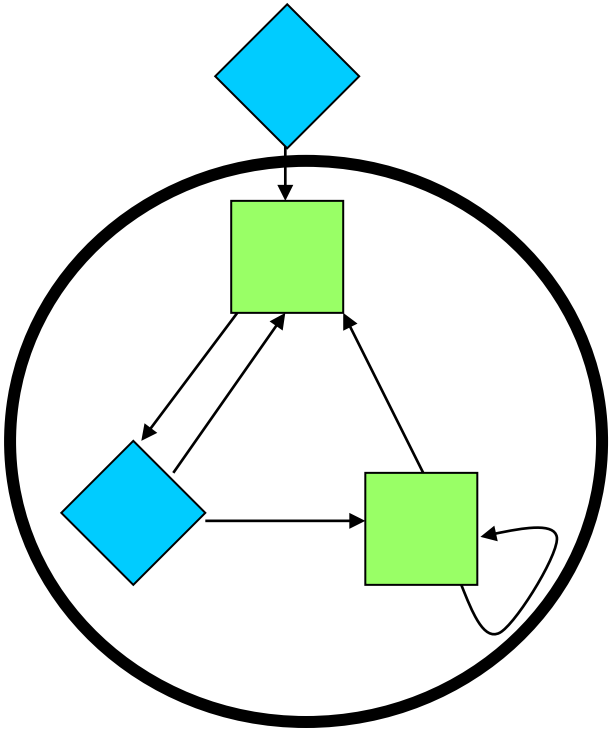
- Time for finding a trap in G_i: $O(2^{i+1}n)$
- **Key argument**: If we find a trap in G_i, then trap of size at least 2^{i-1} is removed from G.
- **Amortized analysis**: Charge O(n) to removed vertices.
- Total time spent until last trap is removed: $O(n^2)$
- Time spent afterwards: $\sum_{i=1}^{\log n} 2^{i+1} n = O(n^2)$



Maximal End-component Decomposition

Maximal End-component Decomposition

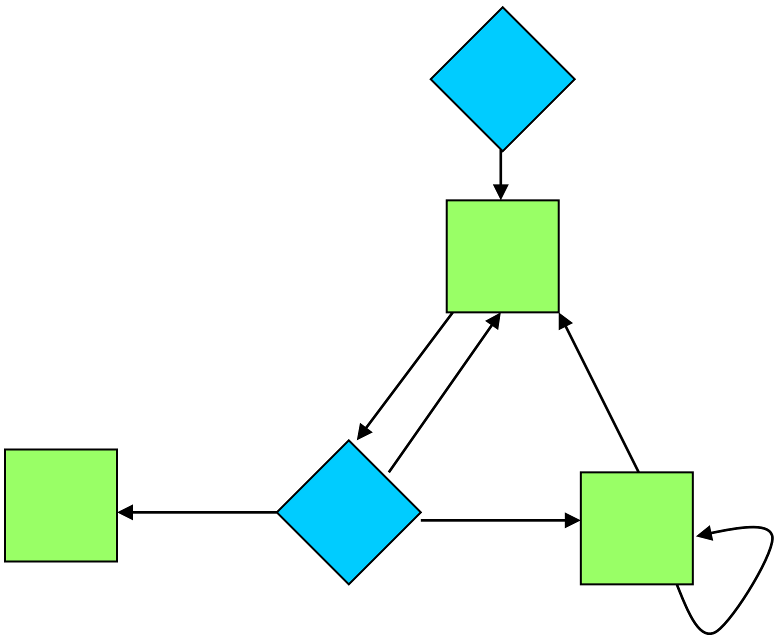
- An end-component U is a set of vertices such that
  - Graph induced by U is strongly connected.
  - If |U| > 1, then for all player-2 vertices in U all out-edges end in U.
  - “strongly connected component with no player-2 out-edges”



Graph decomposed into 1 end-component and 1 individual vertex

Maximal End-component Decomposition

- An end-component U is a set of vertices such that
  - Graph induced by U is strongly connected.
  - If |U| > 1, then for all player-2 vertices in U all out-edges end in U.
  - “strongly connected component with no player-2 out-edges”



Graph decomposed
into 5 individual vertices
(no end-component)

Maximal End-component Decomposition

- Application: Typically used to analyze Markov Decision Processes, where player 2 is the probabilistic player.

- Maximal end-component (MEC) decomposition:
  - Classical algorithm: O(nm) [CY95, deAlfaro97]
- Same algorithm as above but instead of traps search for strongly connected components with no out-edge containing no vertex of B.
  - $O(n^2)$
- Second algorithm: $O(m^{1.5})$

Conclusion

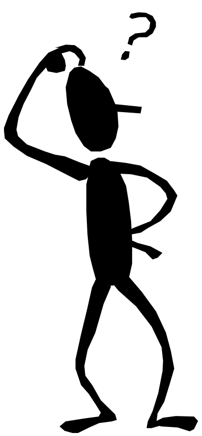
- Buechi games and MEC decomposition:
  - A core algorithmic problem in verification with long-standing O(nm) barrier.
  - We present a simple $O(n^2)$ time algorithm for the problem, also for mec decomposition.
  - For mec decomposition also $O(m^{1.5})$ algorithm that combined gives a worst case $O(mn^{2/3})$ algorithm.

- **Open questions**:
  - $O(mn^{1-\delta})$ or $O(nm^{1-\delta})$ for Buechi games, for some $\delta > 0$
  - $O(mn^{1/2})$ algorithm for mec decomposition.

Generalization of Buechi Games

- **Parity games:**
  - Sub-exponential time deterministic [Jurdzindski, Patterson, Zwick '06], pseudo-polynomial time [Zwick, Paterson '96]
  - No polynomial-time algorithm known
  - NP $\cap$ Co-NP
- **Mean-payoff games:**
  - Sub-exponential time randomized algorithm [Björklund, Vorobyov '07], pseudo-polynomial time [Pisaruk '99]
  - No polynomial-time algorithm known
    - Polynomial-time algorithm for restricted weight-structures [Chatterjee, H, Krinninger, Nanongkai '12]
  - NP $\cap$ Co-NP
- **Open question:** Are they solvable in polynomial time?

Thank you !



Questions ?