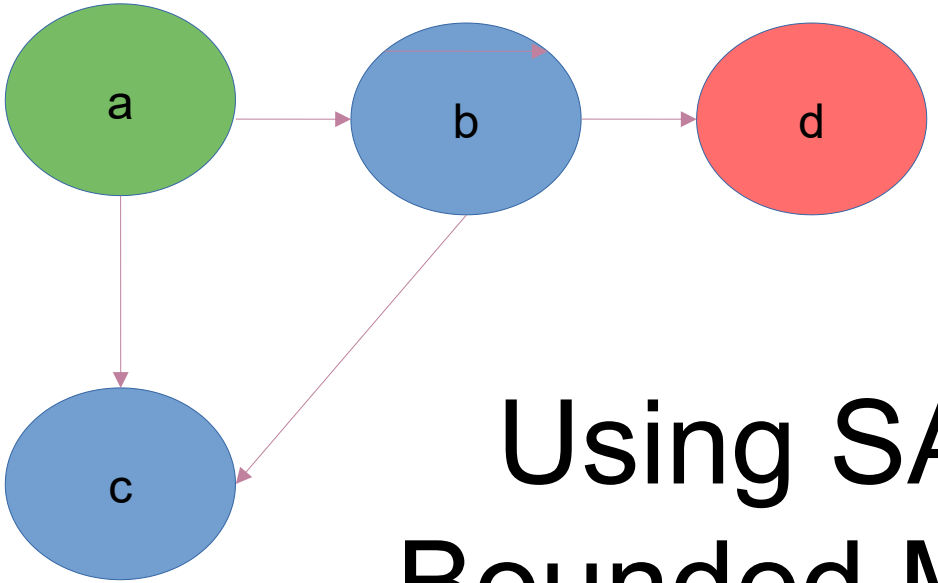


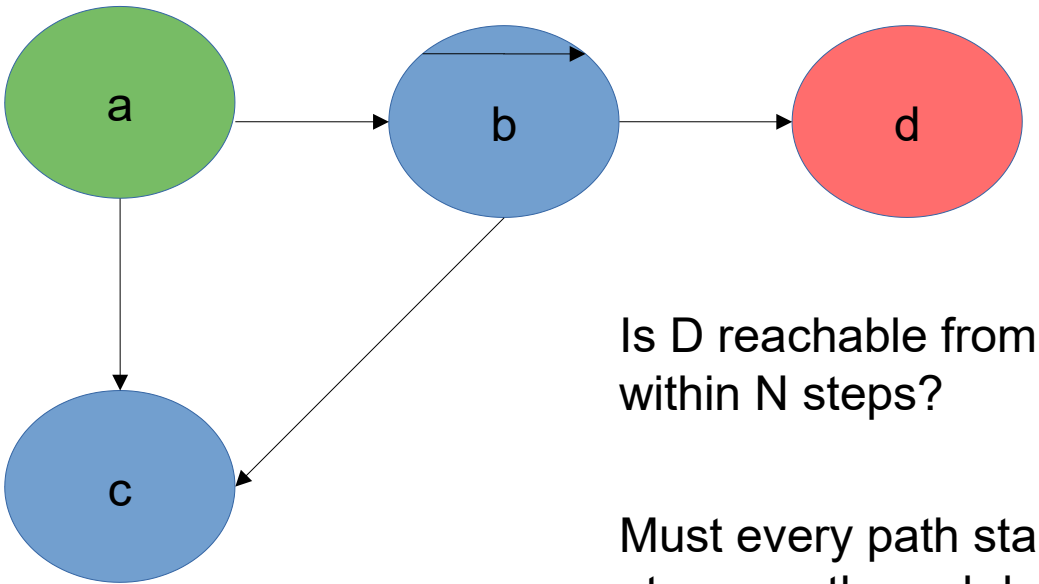
Using SAT Solvers for Bounded Model Checking

- Daniel Scanteianu

Using SAT Solver for BMC

- Safety Property: Bad state is unreachable in N steps
- Liveness Property: A state (or set of states) is always reached in n steps

Problem Visualization



Is D reachable from A within N steps?

Must every path starting at a pass through b within the first N steps?

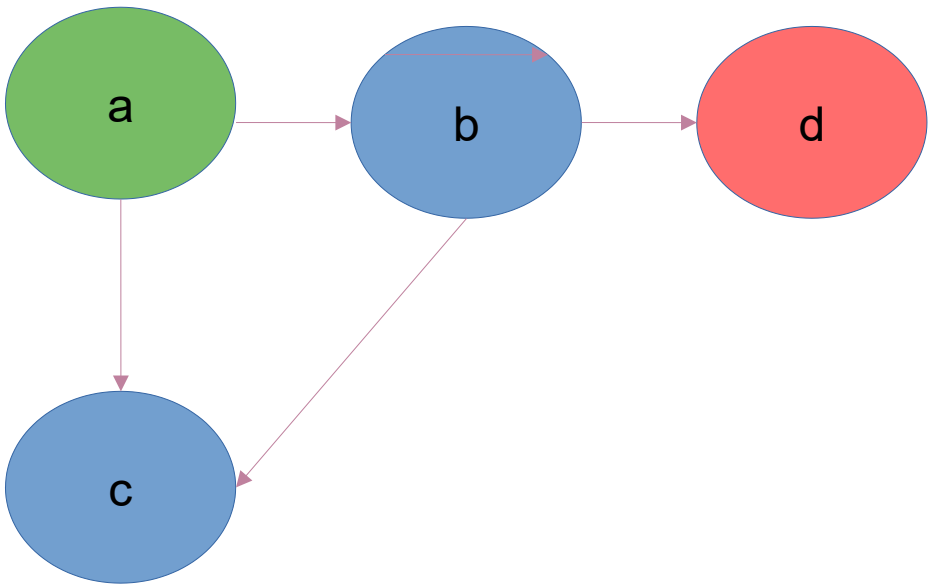
Reachability to Sat

- Unrolling
  - Removes cycles
- Prolog reachability
  - reach(X,Y) :- edge(X,Y).
  - reach(X,Y) :- edge(X,Z), reach(Z,Y).
  - http://fmv.jku.at/biere/talks/Biere-SATSMTAR18-talk.pdf

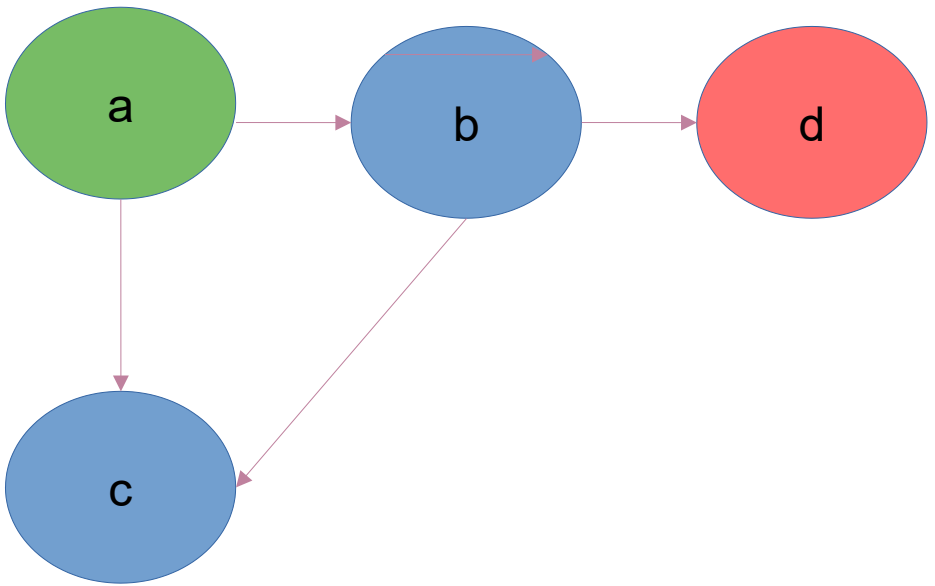
Reachability to Sat

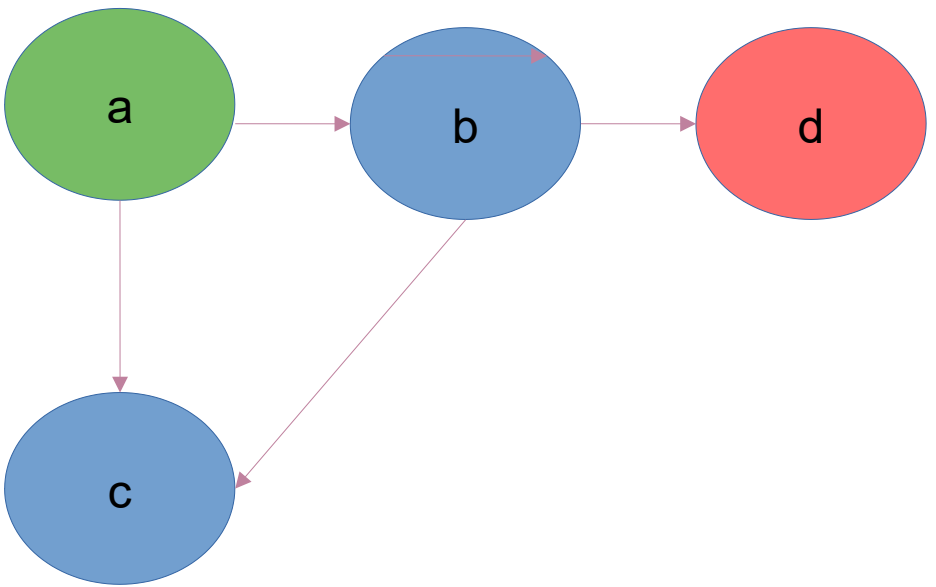
- **Unrolling**
  - **Removes cycles**
- Prolog reachability
  - reach(X,Y) :- edge(X,Y).
  - reach(X,Y) :- edge(X,Z), reach(Z,Y).
  - http://fmv.jku.at/biere/talks/Biere-SATSMTAR18-talk.pdf

Unrolling

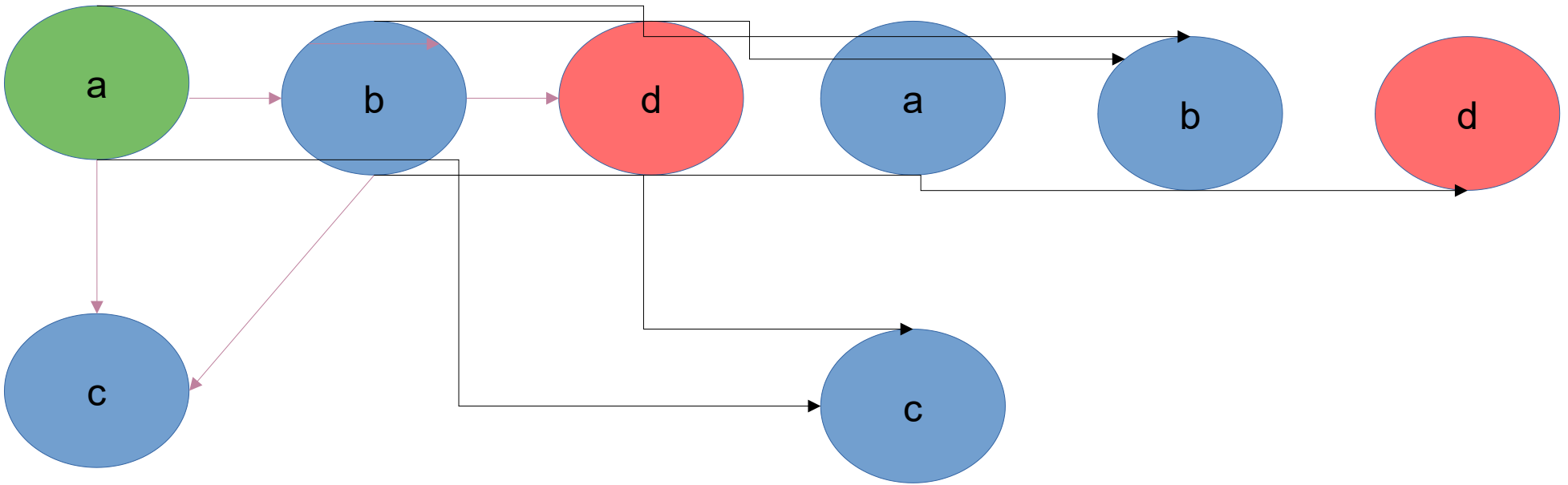


Unrolling

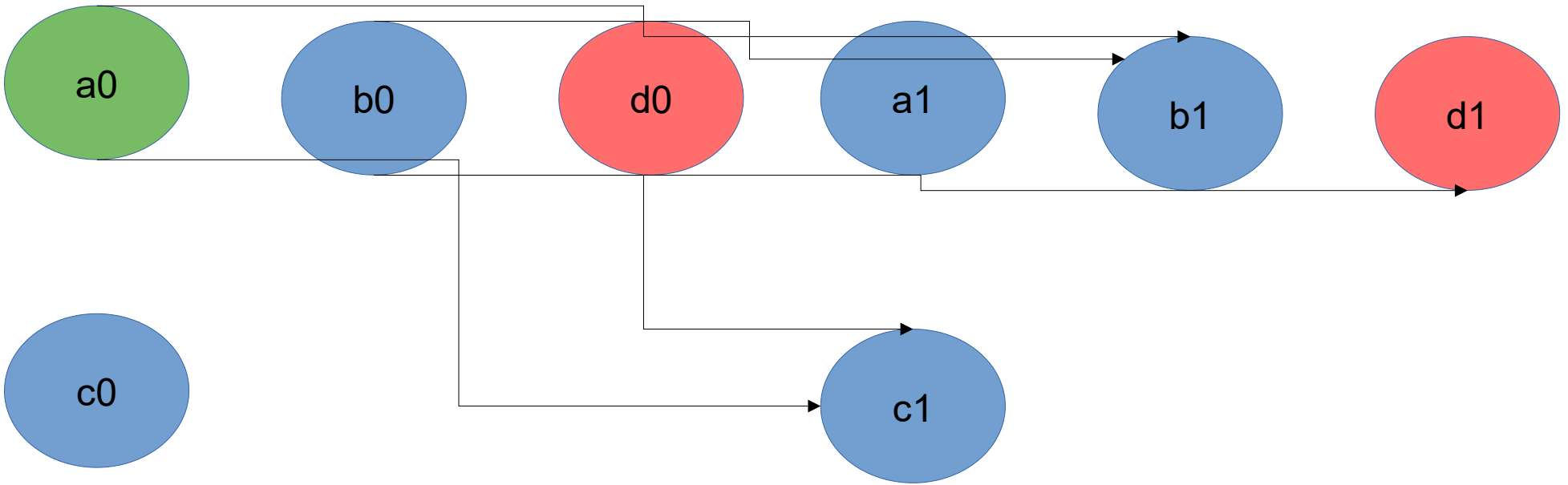




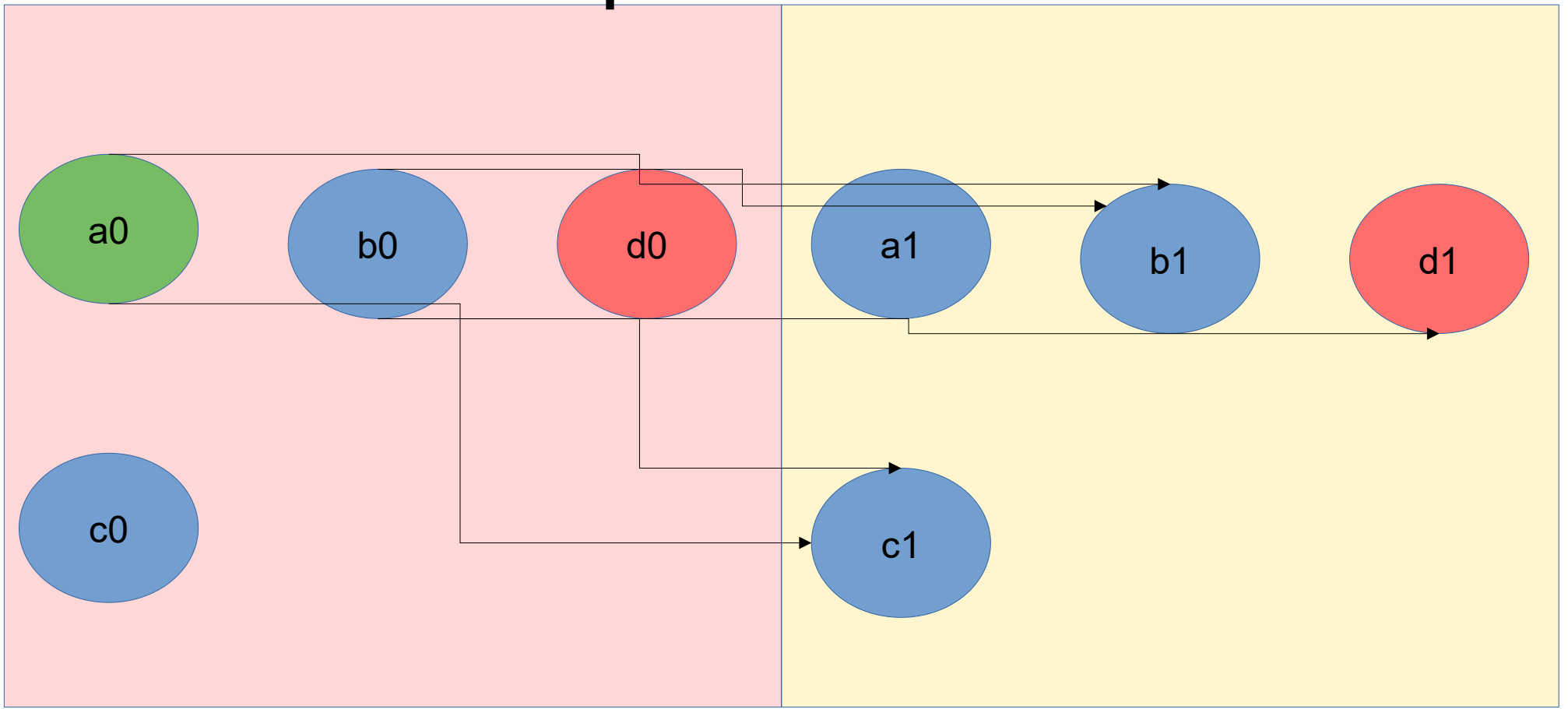
Unrolling



Unrolling



Bipartite DAG



Unrolling

- Given N nodes, and S steps, we will have N nodes in each step.
- Generate new node number n’=n * s where n is the old node number and s is the step we’re on
- If we can detect that we’re dealing with a DAG, we can skip this step

Reachability to Sat

- Unrolling
  - Removes cycles
- **Prolog reachability**
  - **reach(X,Y) :- edge(X,Y).**
  - **reach(X,Y) :- edge(X,Z), reach(Z,Y).**
  - http://fmv.jku.at/biere/talks/Biere-SATSMTAR18-talk.pdf

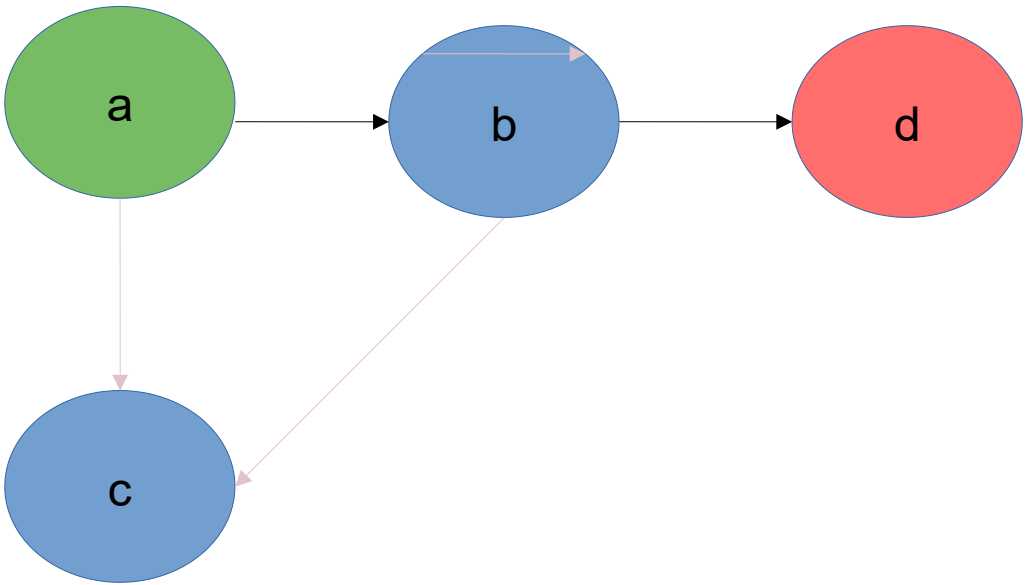
General Idea

- A node is reachable if it is in the initial state.
- A node is reachable if there is an edge between itself and a node in the initial state
- A node is reachable if there is a node between itself and a node that is reachable from the initial state.

Converting to SAT

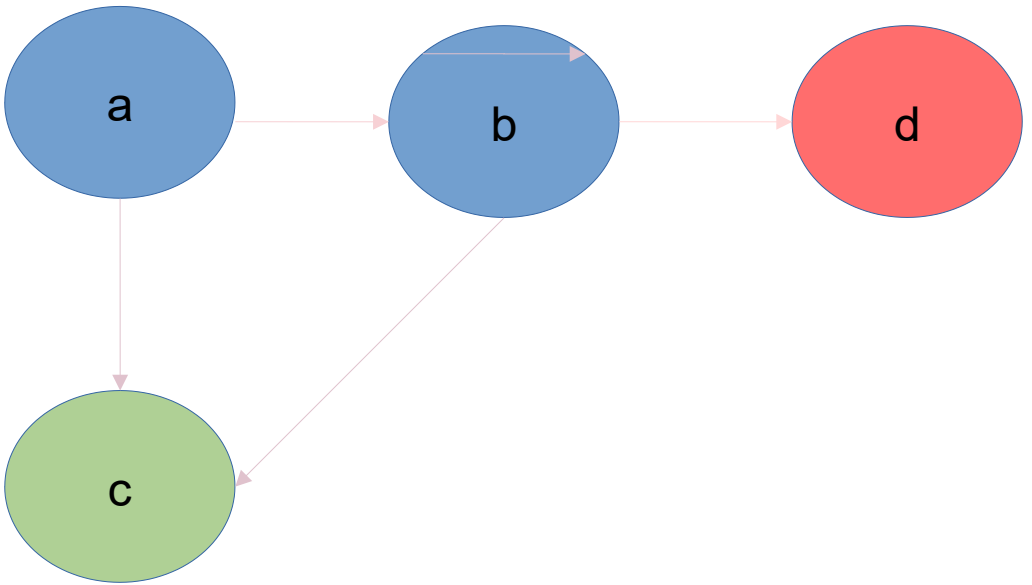
- Want variable assignment to be an encoding of the path from the initial state to the end state
- If there is no possible path to the bad state we would like the SAT solver to say that the reachability is unsatisfiable
- Bijection between nodes/variables

Desired Output



Desired output: [A,B,D] or [A_0, B_1, D_2]

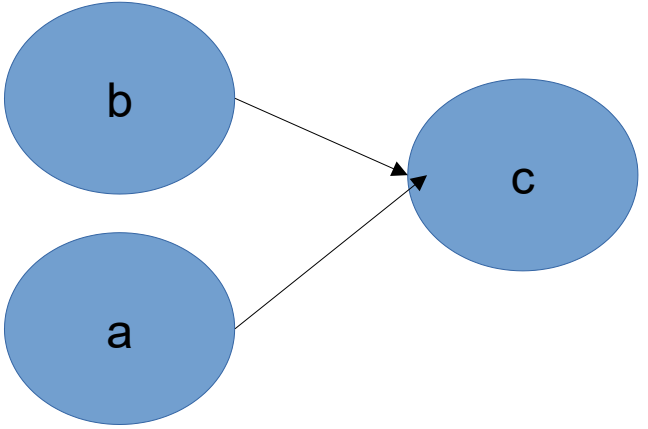
Desired Output



Desired output: UNSAT

Encoding Reachability

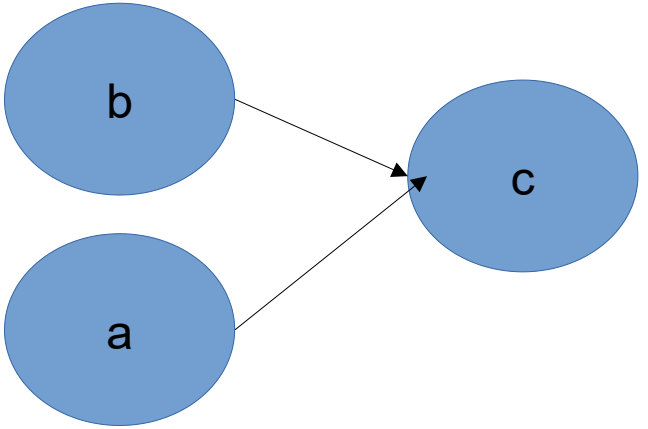
- A node is reachable if any of its parents are reachable
  - C if a or b
  - Need DAG to avoid circular dependencies



- Need CNF for sat solver
- False start: need to find a/b's ancestors, and make one big clause

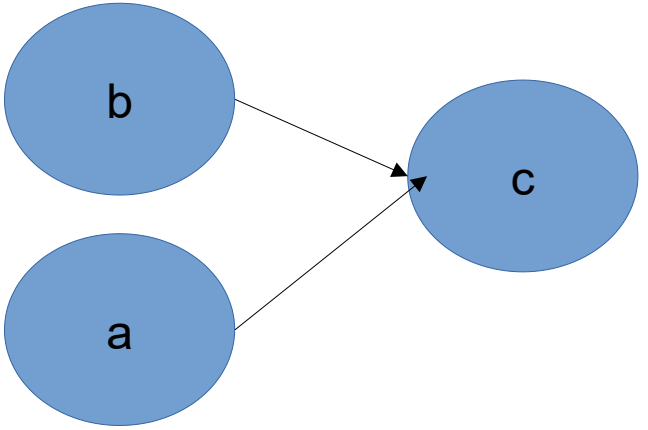
Encoding Reachability

- Breakthrough
  - A ->B == A + -B
    - I TA'd the logic class for 3 sem.
    - Somehow it took me a while to realize that I could use this trick for this project
  - A node is unreachable or at least one of its parents is reachable
  - -C + A + B



Encoding Reachability

- Need to keep track of where nodes are reachable from
  - Keep an adjacency list
- One clause per node per step
  - (~c1+a0+b0)
  - If the node has no possible parents, that node is unreachable, so clause is just (~a0)



- And all these clauses, along with boundary and property clauses

Encoding graph boundaries

- All nodes not in the initial state are not reachable in step 0
- All Nodes without in edges are not reachable.

Encoding the Property

Edges
(0, 2)
(2, 3)
(2, 4)
(2, 5)
(4, 5)
(3, 4)
(3, 5)
(5, 1)

“Reverse” edge list
[[], [5], [0], [2], [2, 3], [2, 4, 3]]

Unrolled Graph
(NOT me OR one of my parents)
[-6]
[-7, 5]
[-8, 0]
[-9, 2]
[-10, 2, 3]
[-11, 2, 4, 3]
[-12]
[-13, 11]
[-14, 6]
[-15, 8]
[-16, 8, 9]
[-17, 8, 10, 9]
[-18]
[-19, 17]
[-20, 12]
[-21, 14]
[-22, 14, 15]
[-23, 14, 16, 15]

Unreachable Step 0
[-1]
[-2]
[-3]
[-4]
[-5]
Bad State Reachable
Step 1-3
[7, 13, 19]
Initial State 0
[0]

Understanding the Answer

- UNSAT means you can’t reach node 1 from node 0 in the number steps
- Ignore FALSE variables in the solution (means that the path didn’t visit the given node)
- Variable encodes node number + step number
  - Sort variables to get sequential path
  - [0, 8, 17, 19]
- Variable mod number of variables is the varnum in init (rolled) graph
  - [0, 2, 5, 1]
  - To get a different path add a clause with one of the “time encoded” variables negated
  - This means “at step x, don’t accept y”

Final Ideas on Conversion

- Going through an intermediate representation (like Prolog) is useful
  - SAT Encoded Recursion
- Graph encoding is fairly mechanical
- No super fancy data structures or algorithms required
- SAT handles state space explosion
- Liveness – verify that it's impossible to do i iterations and not hit the desired node in each one
  - Ie: <unrolled>&&<init>&&(~1)&&(~(n+1))&&..(~(i*n+1))

My Sat Solvers

- First SAT solver based on BDDs
  - Build one BDD for each clause, do an AND of resulting BDDs
- Second one was simple DPLL
- Also used Minisat (which you can install with apt-get, so it’s standard)
- For small usecase, BDDs took 37 ms, DPLL <1 ms.
- For 100 node input graph unrolled 100 steps (1000 clauses), BDD had Java out of memory, DPLL took about .5 seconds

My Sat Solvers

- One usecase where BDD took 1.4s
- My DPLL didn’t terminate
- Minisat did it in under 1 second (so my DPLL algorithm is suboptimal)
- SAT is still NP complete, so there is always some usecase which will stump a solver
- Like compression – there is no perfect compression algorithm (and no algorithm that will make any instance of SAT terminate in polynomial time ...yet)
- Rely on patterns found in most SAT expressions – software, written by humans, has patterns, and these can be exploited by the solver.

References

- https://github.com/Scanteianu/FormalVerificationProject/tree/master/py - My code
- Refs:
- OG unrolling
- http://www.cs.cmu.edu/~emc/papers/Books%20and%20Edited%20Volumes/Bounded%20Model%20Checking.pdf
- without unrolling
- https://s3.amazonaws.com/academia.edu.documents/39846809/SAT-Based_Model_Checking_without_Unrolli20151109-1286-o6dd8v.pdf?response-content-disposition=inline%3B%20filename%3DSAT-Based_Model_Checking_without_Unrolli.pdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWOWYYGZ2Y53UL3A%2F20191211%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20191211T140012Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=b5ebd65b64e7b4f2ae2b163da8dfc200daee1caffbf39d0506857d2160da7cda
- transition relation simplification:
- https://www.cs.york.ac.uk/rts/docs/SIGDA-Compendium-1994-2004/papers/2004/iccad04/pdffiles/01c_2.pdf
- prolog
- http://fmv.jku.at/biere/talks/Biere-SATSMTAR18-talk.pdf
- sat and smt
- https://www.cs.rochester.edu/u/kautz/papers/ijcai07-numeric-rev.pdf
- circuitsat sat floyd warshall
- https://www.csie.ntu.edu.tw/~lyuu/complexity/2008a/20080327.pdf
- reminder for if equivalence
- http://www.cs.mun.ca/~kol/courses/2742-f09/studysheet-t1.pdf
- 2-sat: http://web.csulb.edu/~tebert/teaching/lectures/528/sat/sat.pdf
- how to turn biconditional into cnf?
- https://www.cs.jhu.edu/~jason/tutorials/convert-to-CNF.html