Human-machine interaction in invariance proofs

Ken McMillan
Tom Ball
Microsoft Research

Mooly Sagiv
Tel Aviv Univ.

Motivation

- Real-world verification efforts use little proof automation
  - CompCert, seL4, IronClad/IronFleet, Verdi, Intel, AMD
  - Most use proof assistants requiring detailed manual guidance
  - Use of model checking and other automation is the exception
- Why: Automated tools are brittle and opaque
  - Fail unpredictably and completely.
    - Unavoidable since problems intractable to undecidable
  - Diagnosing and correcting failures is hard because tools cannot effectively communicate their state.
- Can we make these tools fail visibly?
  - If they fail to prove a property, can they tell us what they *did* manage to prove in an intelligible way?
  - How does the user explain what is missing to the tool?

Inspiration: Foldit

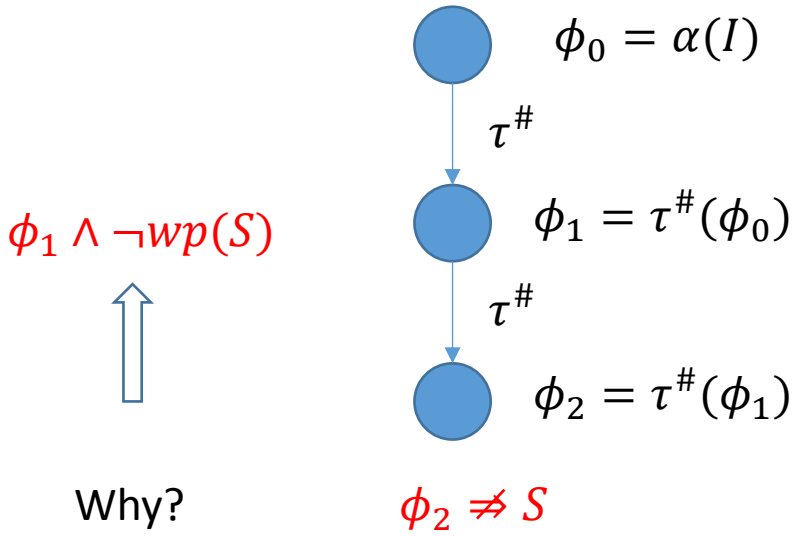
- A game allowing non-chemists to discover protein structures.
  - Old way: grow crystal in lab, shine X-rays through, examine diffraction pattern (very hard!)
  - New way: combine (brittle) optimization algorithms with human visual intuition.
- Foldit fails visibly:
  - State of the algorithms can visualized using 3D graphics
  - Users interact with the graphics to provide guidance, allowing the algorithm to escape local minima
  - The algorithms and users solve the structures collaboratively.

Failing visibly

- Notice that “wiggle” failed visibly
  - We examined the algorithm state visually
  - We used our visual abilities to analyze a large amount of information (image a column of numbers instead!)
  - We used the visualization to diagnose the problem and provide guidance to the algorithm
- Questions for this talk:
  - What would it mean for a model checker to fail visibly?
  - What is the algorithm state and how can we visualize it?
  - What sort of guidance could the human user provide?

A concrete scenario

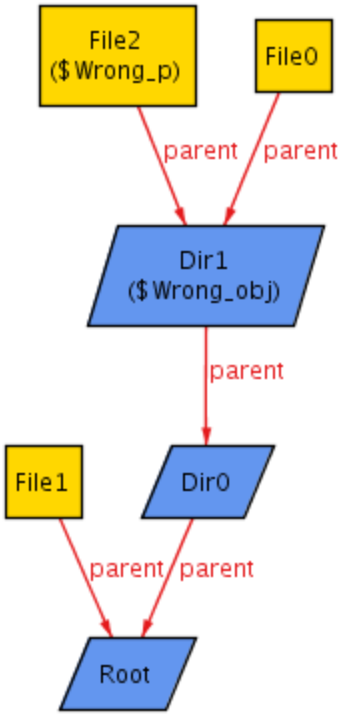
- Diagnosis of abstract interpretation failure



To refine abstraction, we need to form some generalization of $\phi_1 \wedge \neg wp(S)$ but this is a highly complex formula. How can we display it in a way a user can understand?

Visualizing formulas

- The *semantics* of a formula is a set of *models*.
  - These are relational structures that we readily visualize graphically (see the Alloy tool).

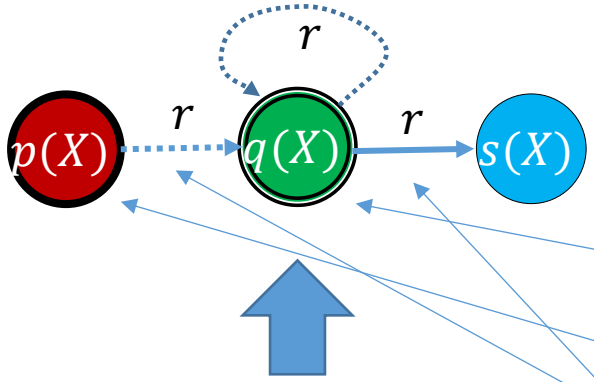


- A formula can have infinitely many models
- How can we represent them all finitely?

Concept graph

Use node to represent concepts rather than individuals

concept graph



Node represents the (possibly empty) set of individuals satisfying some predicate.

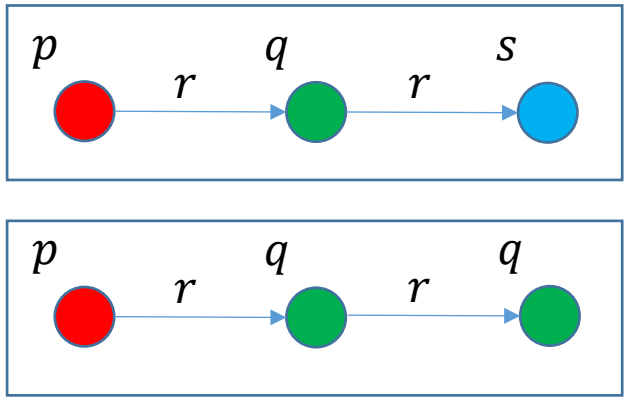
Summary node indicates possibly more than one.

Definite node indicates exactly one.

Indefinite arc indicates some pairs may be connected.

Definite arc indicates all pairs are connected.

models



Each graphical element represents a logical fact about the model set.

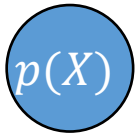
Presentation

- Use graphical conventions to represent facts about the set of models of a theory Γ.

$\Box_\Gamma \phi$	ϕ is true in *all* models of Γ
$\Diamond_\Gamma \phi$	ϕ is true in *some* model of Γ
$\lvert p \rvert$	cardinality of relation p

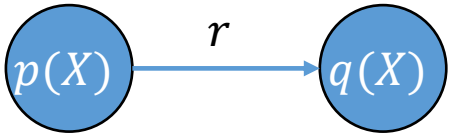
Our graphical elements (vertices, arcs) will stand for simple modal facts about the space of models of Γ.

Some possible conventions



$\diamond\ |p| > 0$

$\square\ |p| = 0$

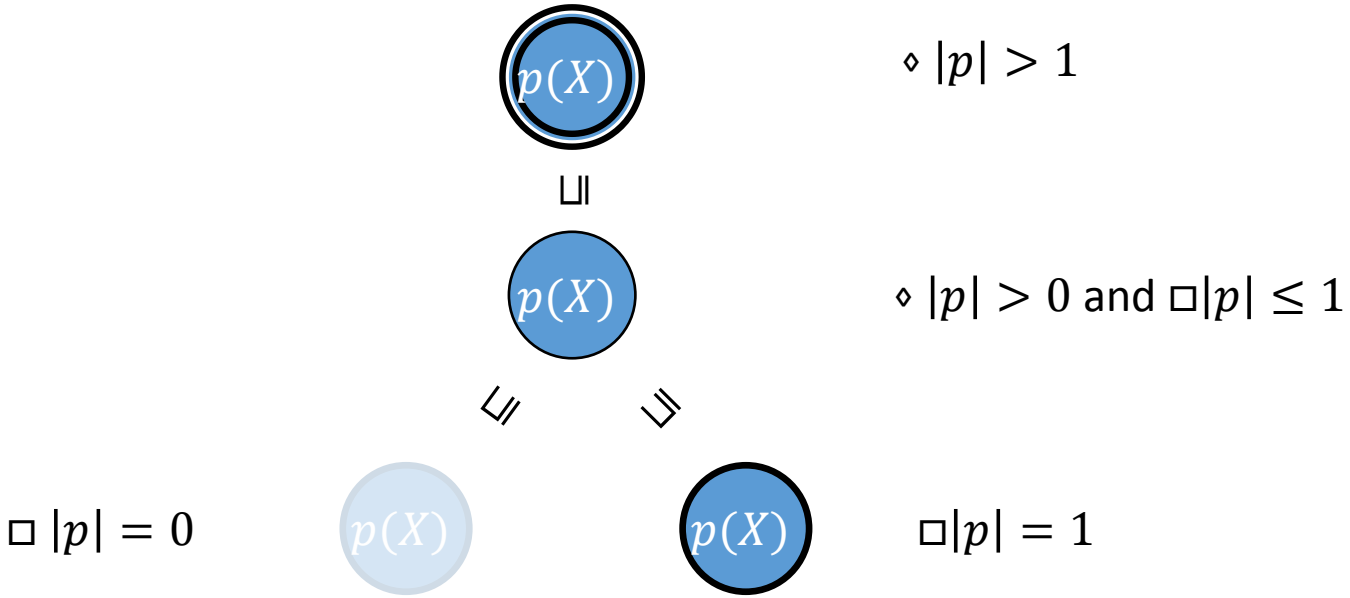


$\square \forall X, Y\colon\ p(X) \wedge q(Y) \Rightarrow r(X, Y)$

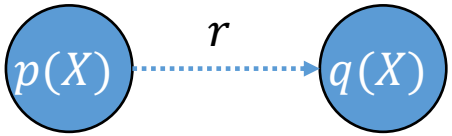
$\diamond\ \exists X, Y\colon\ p(X) \wedge q(Y) \wedge \neg r(X, Y)$

More detailed nodes…

We can provide more information about cardinality…



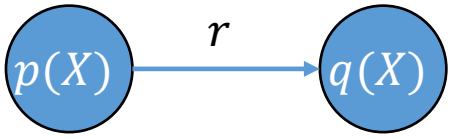
More detailed edges…



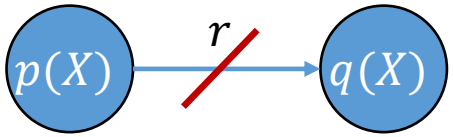
$\neg\Box\forall X, Y\colon\ p(X) \wedge q(Y) \Rightarrow r(X, Y)$

and

$\neg\Box\forall X, Y\colon\ p(X) \wedge q(Y) \Rightarrow \neg r(X, Y)$



$\Box\forall X, Y\colon\ p(X) \wedge q(Y) \Rightarrow r(X, Y)$



$\Box\forall X, Y\colon\ p(X) \wedge q(Y) \Rightarrow \neg r(X, Y)$

Focus

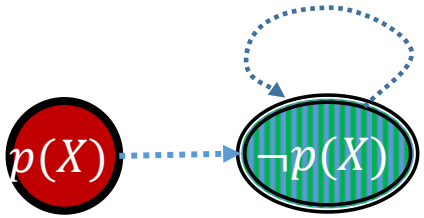
We can provide the user with various ways to focus the representation on relevant facts (as in Foldit we have zoom, pan and rotate)

- Refinement
  - Choose the concepts and relations to display
- Materialization
  - Narrow the space of models

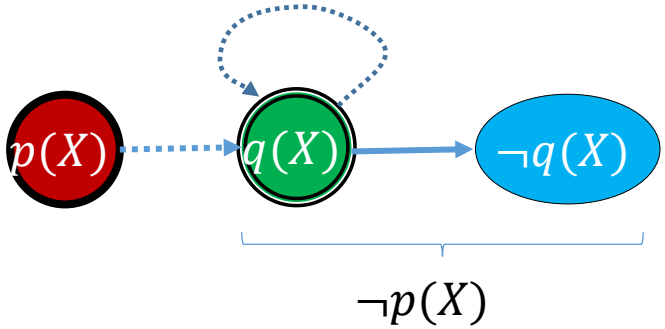
By applying these operations interactively, we want to expose the root cause of a proof failure.

Splitting predicates

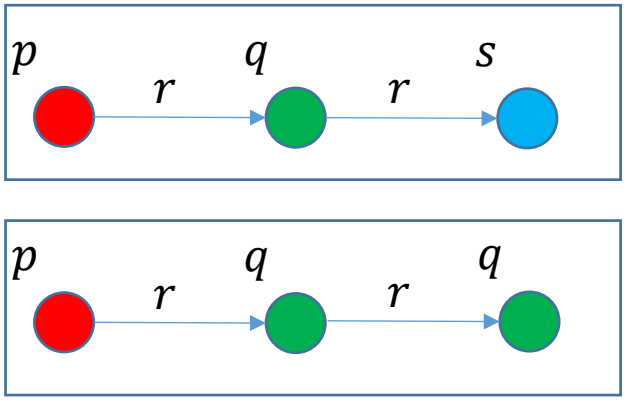
concept graph



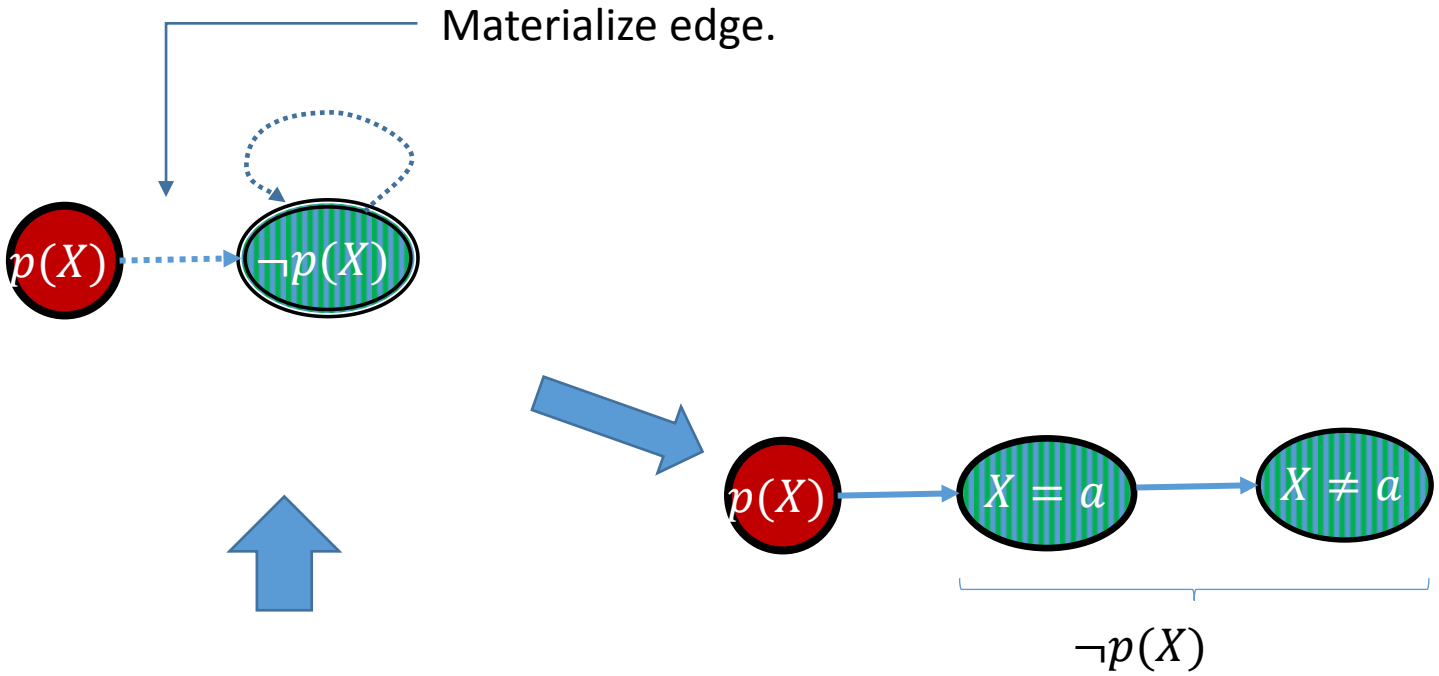
Split on q.



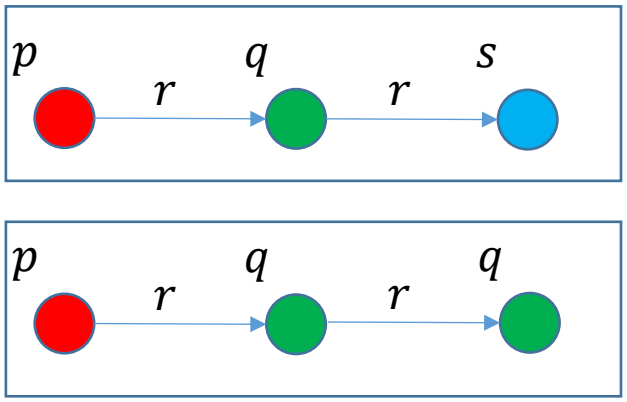
models



Materializing edges

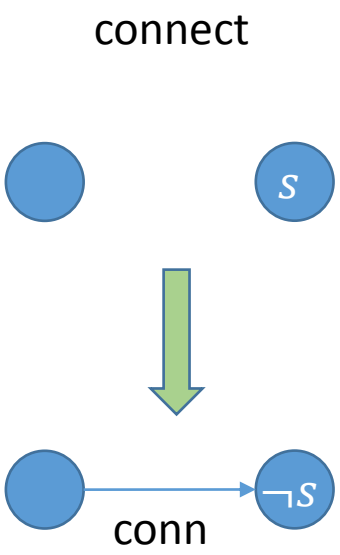


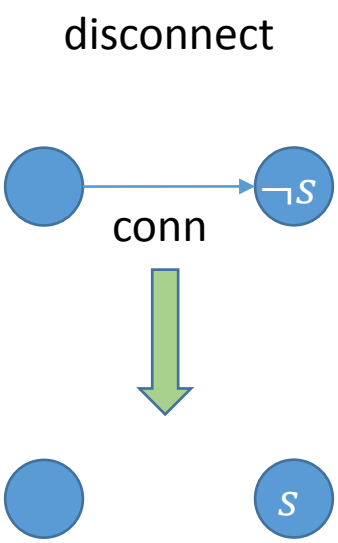
models



Client server example

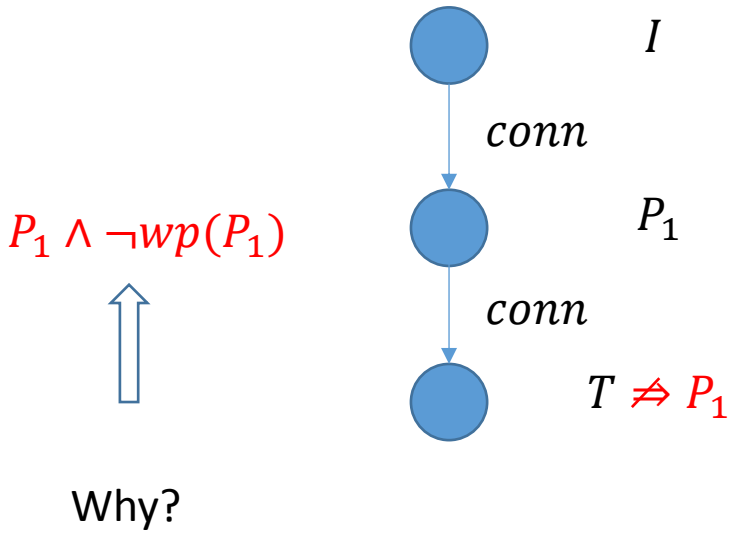
- A trivial parameterized protocol
  - N processes, each can be client or server
  - Each server has a semaphore
  - Prove each server always connected to at most one client





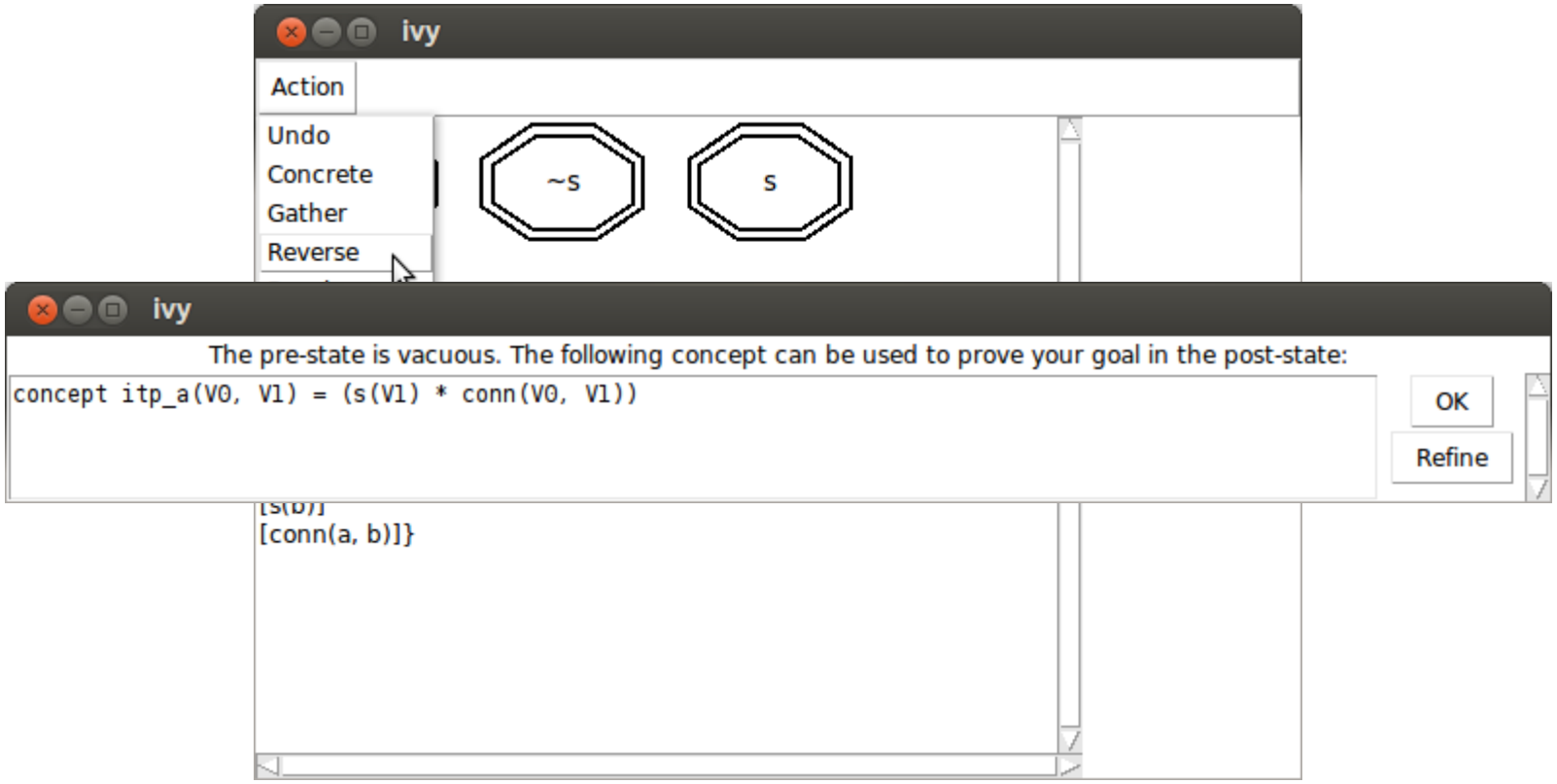
Failed abstract interpretation

- Try with just one predicate:
  - $P_1 \colon \forall x, y, z \colon x = y \vee \neg conn(x, z) \vee \neg conn(y, z)$



Demo: visualizing $P_1 \wedge \neg wp(P_1)$ to diagnose proof failure.

Exploring the bad states

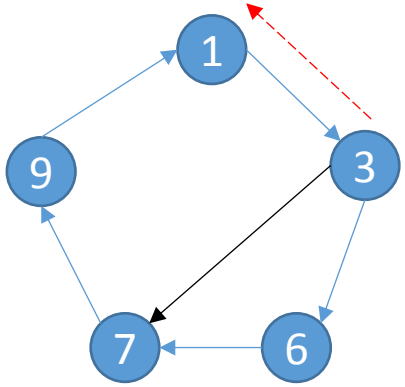


Effect

- With this refinement, abstract interpretation proves the property:
  - $\forall x, y: \neg(conn(x, y) \wedge s(y))$
- We used the concept graph to:
  - Compare the “bad states” to our intuition about the protocol.
  - Materialize a “bad pattern” that characterizes the failure.
  - Generalize this pattern by interpolation to yield abstraction refinement.
- In effect, we put the human in the loop in abstraction refinement.

Chord ring maintenance

- Each node has unique ID in range $0 \ldots N - 1$.
- Successor pointers for ring ordered by ID's mod N.

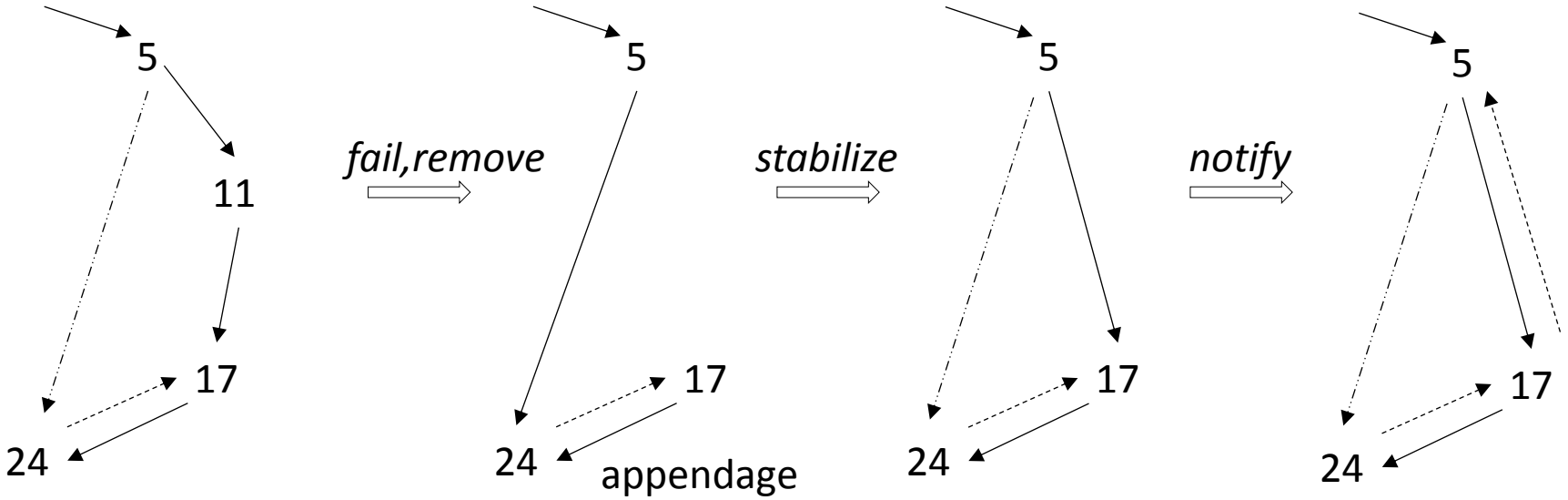


Predecessor used for repair of "appendages".

Second successor used for repair of gaps.

Peer-to-peer protocol, intended to be self-stabilizing when nodes fail or join.

Example: repair after node failure

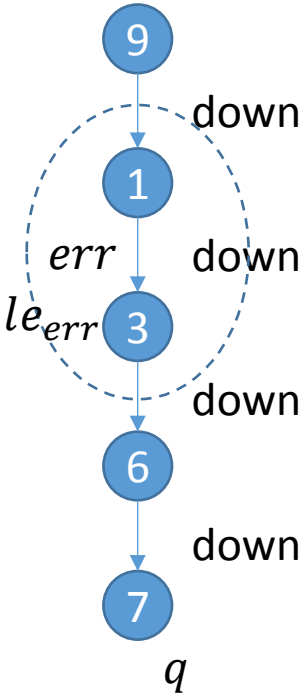


Proof goal [Zave,2014]:

- Assume: there is one node q that never fails.
- Assume: all successors of a node cannot fail.
- Prove: every node can reach q via best successors.
  - Best successor is nearest active successor
- Corollary: There is one cycle, and all active nodes can reach it.

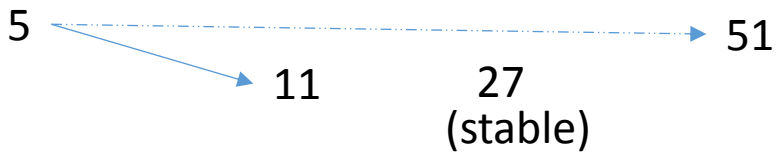
Main proof idea:

- Order all the nodes by distance to q.
  - We expect all best successor arcs to be downward in this order.
- Let err be the set of nodes not reaching q.
- Let le_{err} be the least element of err if err is non-empty.
- Use abstract interpretation to generate an invariant proving err is always empty.



Result

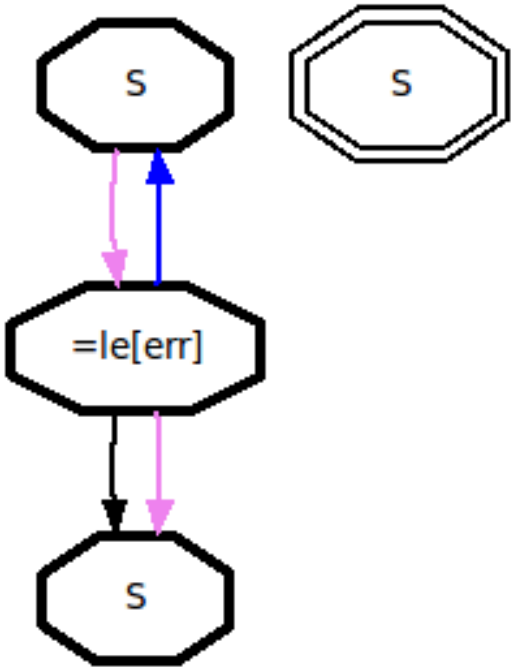
- This refinement is enough to allow abstract interpretation to finish the proof.
- The refinement is the key “non-skipping” invariant of Zave’s manual proof.



- First parameterized proof of this protocol
- Shows that visualization can be used to diagnose failures in fairly subtle proofs.

Comparing text to graphics

(err(x) & ~btw(X, X, Y) & ~btw(X, Y, Y) & (a(X) | a(Y) | s1(X, Y) | ~p(Y, X)) & (a(X) | p(Y, X) | ~s1(X, Y)) & (a(X) | ~p(Y, X) | ~s1(X, Y)) & (a(X) | ~s2(X, Y)) & (a(X) | X ~= q) & (a(Y) | a(Z) | ~s1(X, Y) | ~s2(X, Z)) & (a(Y) | bs(X, Z) | ~a(Z) | ~s1(X, Y) | ~s2(X, Z)) & (a(Y) | dom[s2](X) | ~a(X) | ~s1(X, Y)) & (a(Y) | ~a(X) | ~p(Y, X) | ~s1(X, Y)) & (bs(X, Y) | ~a(Y) | ~s1(X, Y)) & (btw(W, X, Y) | ~btw(W, X, Z) | ~btw(X, Y, Z)) & (btw(W, X, Z) | ~btw(W, X, Y) | ~btw(W, Y, Z)) & (btw(W, Y, Z) | ~btw(W, X, Z) | ~btw(X, Y, Z)) & (btw(X, Y, Z) | ~btw(W, X, Y) | ~btw(W, Y, Z)) & (dom[p](X) | ~p(X, Y)) & (dom[s1](X) | ~a(X)) & (dom[s1](X) | ~s1(X, Y)) & (dom[s2](X) | ~s2(X, Y)) & (err(le[err]) | ~err(X)) & (rch[q](X) | ~bs(X, q)) & (rch[q](X) | ~bs(X, Y) | ~rch[q](Y)) & (s1(le[err], s1[le[err]]) | ~dom[s1](le[err])) & (s1(X, Y) | ~a(X) | ~p(Y, X)) & (s2(le[err], s2[le[err]]) | ~dom[s2](le[err])) & (W = X | btw(W, X, W)) & (Y = X | btw(X, Y, Z) | btw(Y, X, Z)) & (Y = Z | btw(X, Y, Z) | btw(X, Z, Y)) & (Y = Z | ~p(X, Y) | ~p(X, Z)) & (Y = Z | ~s1(X, Y) | ~s1(X, Z)) & (Y = Z | ~s2(X, Y) | ~s2(X, Z)) & (~btw(le[err], X, q) | ~err(X)) & (~btw(X, q, Y) | ~s1(X, Y)) & down(X, Y) <-> (X = Y | (X ~= q & ~btw(X, q, Y))) & err(X) <-> (a(X) & ~rch[q](X)))



Discussion

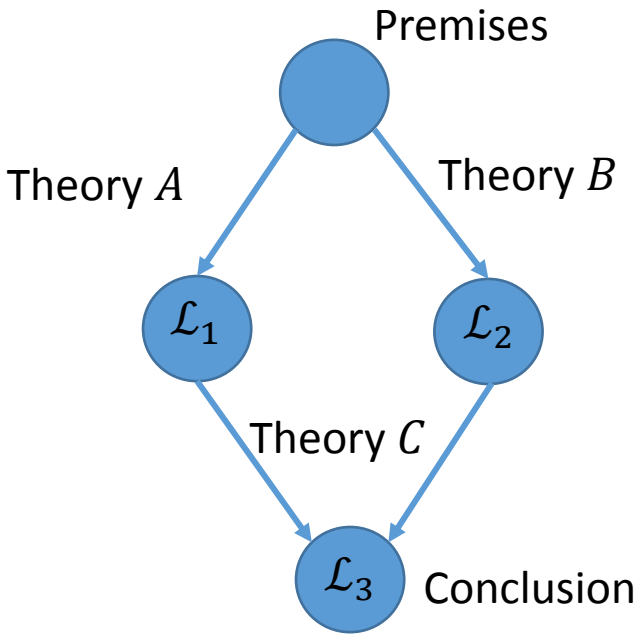
- By visualizing the *semantics* of formulas, we have allowed abstract interpretation to fail visibly.
  - This is necessary since the state of the algorithm is represented by formulas.
  - A simple graphic can capture in a clear way the relevant content of a very complex formula.
  - We needed user interaction to focus on relevant facts.
  - Since each graphical element represents the truth of a logical formula, we needed a decidable logic and a relatively efficient decision procedure to draw the graphs.

Conclusion

- If you are developing verification algorithms, consider:
  - How well would the technique integrate into a large and complex proof effort?
  - Does it fail visibly? Can the user diagnose and correct the inevitable algorithm failures?
  - Would a user building a larger proof consider the algorithm more effective than a fully manual but predictable approach?
- Put the human in the loop
  - Interactive visualization allows user to collaborate with the algorithm by diagnose failures and suggesting relevant generalizations.
  - Perhaps in this way we can convince people attempting large scale proofs of real systems to use some automation.

Future: abstract proofs?

- An ARG might represent other sorts of proofs.



User specifies an abstract proof outline and diagnoses failures visually.

Proof diagnosis approach

- Suppose we want to prove ψ
- Prover proves ϕ, where $\phi \not\models \psi$
- Steps:
  - Visualize the models of $\phi \wedge \neg\psi$.
  - Isolate a sub-model characterizing failure (abduction)
  - Proof goal: refute sub-model.

We'll use this approach to diagnose a failed proof of Chord's ring maintenance protocol.

Proof using Ivy

- Model protocol, safety condition.
- Proof by abstract interpretation
  - Abstract domain = universally quantified templates
  - Z3 used as decision procedure
- Users can interactively…:
  - construct abstract reachability graphs
  - visualize sets of states
  - perform backward analysis to find root causes

Approach

- Use interactive graphical visualization
  - Present a large set of facts in a comprehensible way
    - Not a long list of formulas!
  - Allow user to focus on relevant facts, and narrow to root causes.
  - Focus on semantics rather than syntax.
    - Visualize spaces of logical models that characterize proof failures.

I'll describe a tool called Ivy for visualizing sets of models. We'll use it to diagnose and repair a failed proof of the Chord distributed hash table protocol using abstract interpretation.