Model Checking

Aleksei Sholokhov, Yuan-Mao Chueh

Agenda

Formal reasoning:

1. Deductive Reasoning (like writing math proof)
2. Model checking (finite; run automatically)

Model Checking:

1. Language used: Temporal logic: ex: CTL* (which contains CTL, LTL) [Alex]
2. Explicit-state -> state explosion problem [Alex]
    a. Partial Order Reduction (X)
    b. BDD-based symbolic model checking (Yuan-Mao)
    c. SAT/SMT based model checking (X)
    d. Abstraction (<- our required reading) (Yuan-Mao)
3. Applications (<- our optional readings) [Alex] 2-3 pages

Model-Checking Overview

1. Property Specification Language
    - Typically expressed based on a temporal logic
2. Model Specification Language
    - Encoding the system (program, hardware) as a finite-state transition system
3. Verification Procedure
    - Algorithms that does an exhaustive search of the model state space
    - Provides a counterexample if it finds a state that breaks the specification

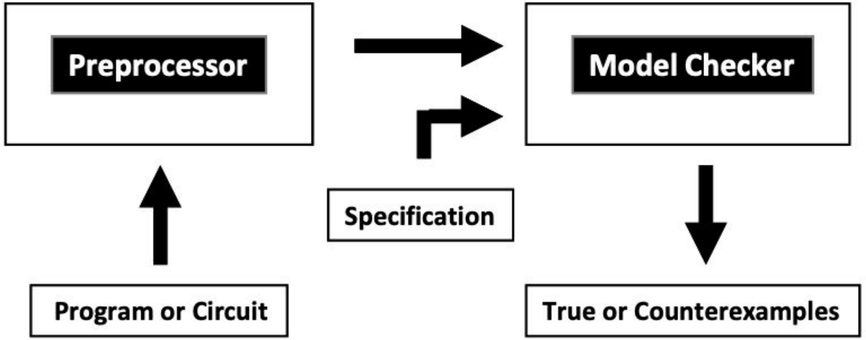
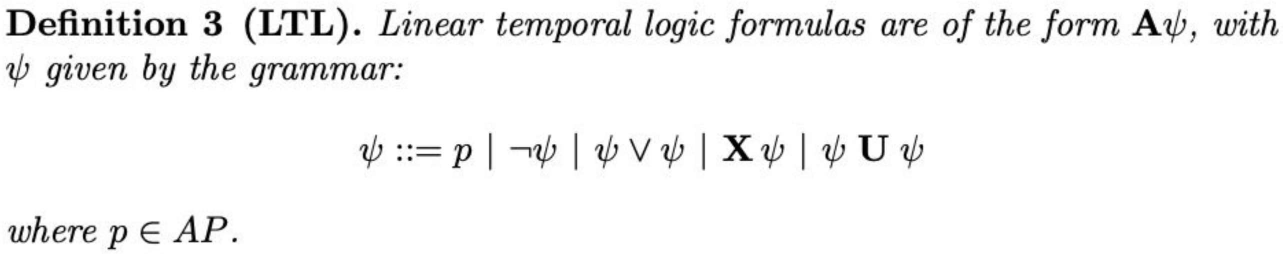


Fig. 4: A Model Checker with Counterexamples

LTL – Linear-time Temporal Logic



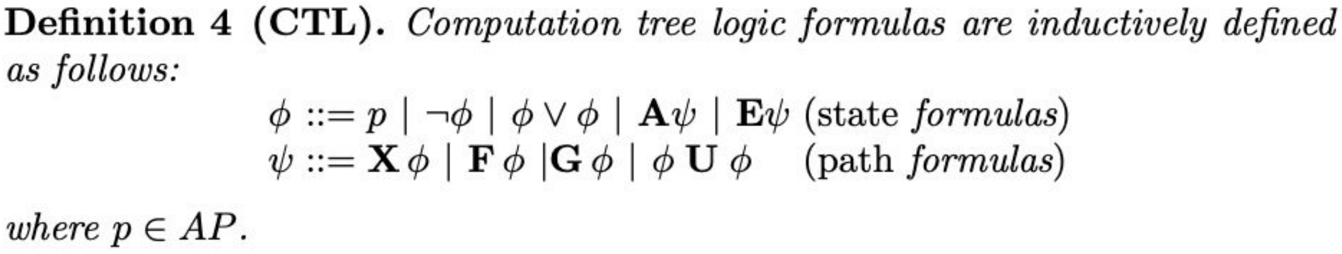
Definition 3 (LTL). *Linear temporal logic formulas are of the form* $\mathbf{A}\psi$, *with* ψ *given by the grammar:*

$$\psi ::= p \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\,\psi \mid \psi\,\mathbf{U}\,\psi$$

where $p \in AP$.

- Temporal Operators: Xφ | Gφ | Fφ | Rφ | Wφ | Mφ
  - **X** *φ* – Ne**x**t: *φ* has to hold at the next state (this operator is sometimes noted **N** instead of **X**).
  - **G** *φ* – **G**lobally: *φ* has to hold on the entire subsequent path.
  - **F** *φ* – **F**inally: *φ* eventually has to hold (somewhere on the subsequent path).
  - *ψ* **W** *φ* – **W**eak until: *ψ* has to hold *at least* until *φ*; if *φ* never becomes true, *ψ* must remain true forever.
  - *ψ* **U** *φ* – **U**ntil: *ψ* has to hold *at least* until *φ* becomes true, which must hold at the current or a future position.
  - *ψ* **R** *φ* - **R**elease: *φ* has to be true until and including the point where *ψ* first becomes true; if *ψ* never becomes true, *φ* must remain true forever
  - *ψ* **M** *φ* - Strong release: *φ* has to be true until and including the point where *ψ* first becomes true, which must hold at the current or a future position
- All formulas have an implicit A in front

Computation Tree Logic (CTL) – branching-time logic



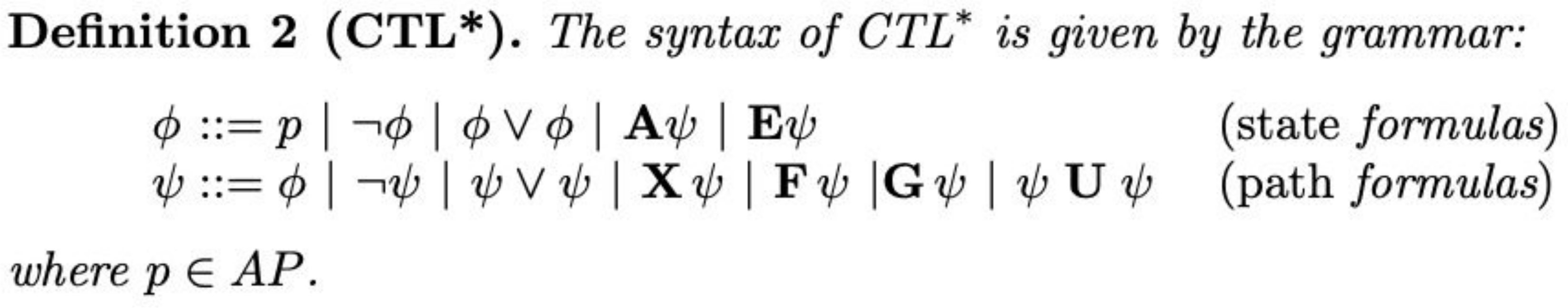
Definition 4 (CTL). *Computation tree logic formulas are inductively defined as follows:*

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{A}\psi \mid \mathbf{E}\psi \quad \text{(state } \textit{formulas}\text{)}$$
$$\psi ::= \mathbf{X}\,\phi \mid \mathbf{F}\,\phi \mid \mathbf{G}\,\phi \mid \phi\,\mathbf{U}\,\phi \quad \text{(path } \textit{formulas}\text{)}$$

where $p \in AP$.

- Each basic temporal (X, F, G, U) operator must be immediately preceded by a path quantifier (A or E)
- Quantifiers over paths
  - **A** Φ – **A**ll: Φ has to hold on all paths starting from the current state.
  - **E** Φ – **E**xists: there exists at least one path starting from the current state where Φ holds.
- Path-specific quantifiers
  - **X** *φ* – Ne**x**t: *φ* has to hold at the next state (this operator is sometimes noted **N** instead of **X**).
  - **G** *φ* – **G**lobally: *φ* has to hold on the entire subsequent path.
  - **F** *φ* – **F**inally: *φ* eventually has to hold (somewhere on the subsequent path).
  - *φ* **U** *ψ* – **U**ntil: *φ* has to hold *at least* until at some position *ψ* holds. This implies that *ψ* will be verified in the future.
  - *φ* **W** *ψ* – **W**eak until: *φ* has to hold until *ψ* holds. The **W** operator is sometimes called "unless".

CTL* – combines state- and path-specific qualifiers



Definition 2 (CTL*). *The syntax of CTL* is given by the grammar:*

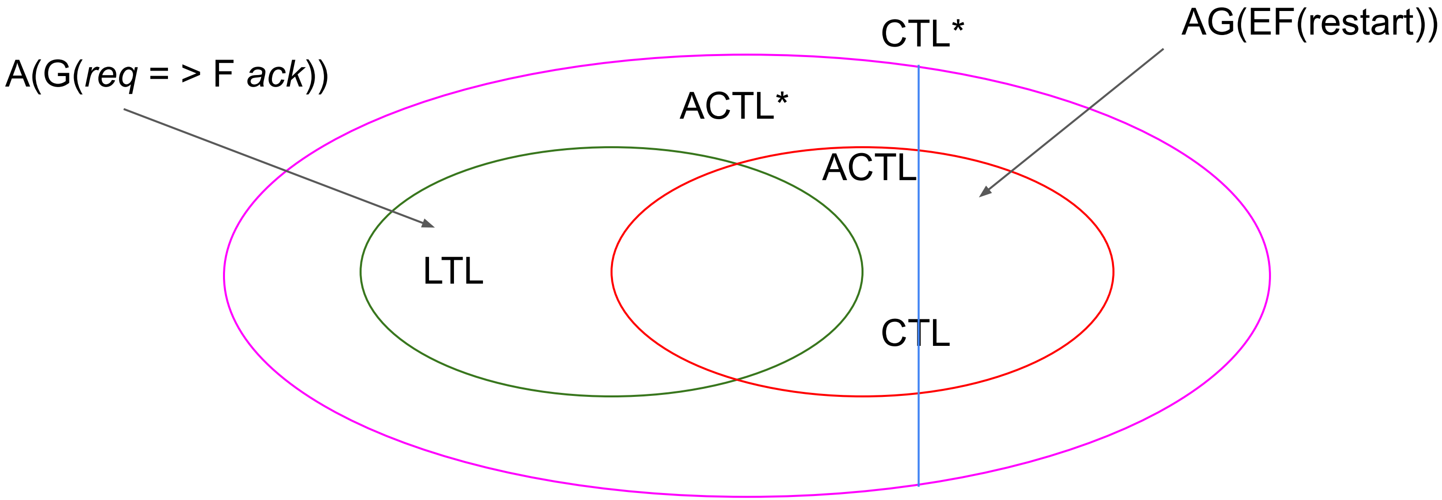
$$\begin{array}{ll} \phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{A}\psi \mid \mathbf{E}\psi & \text{(state \textit{formulas})} \\ \psi ::= \phi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\,\psi \mid \mathbf{F}\,\psi \mid \mathbf{G}\,\psi \mid \psi\,\mathbf{U}\,\psi & \text{(path \textit{formulas})} \end{array}$$

where $p \in AP$.

- φ is satisfied with respect to the state: s |= φ
- Ψ is satisfied with respect to the path: **π** |= ψ

- ACTL* – CTL* where the A (forall) qualifier is excluded and all formulas are in NNF.
  - Because of the latter, we can not define Eφ = ¬A¬φ. Thus ACTL* ⊂ CTL*

LTL vs CTL vs CTL* vs ACTL*



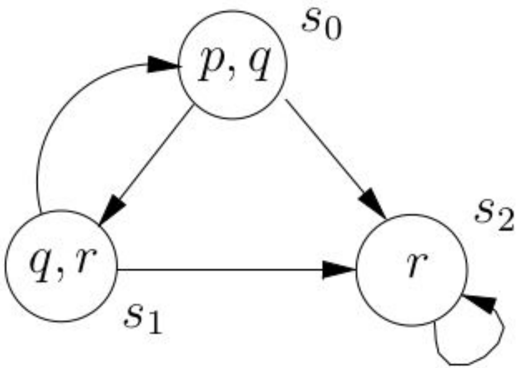
State Explosion Problem

- Each n-bit number has 2^n states
- k branch conditions give 2^k states
- m asynchronous processes with n states each have m^n states

Superposition of those quickly yields unmanageable number of states

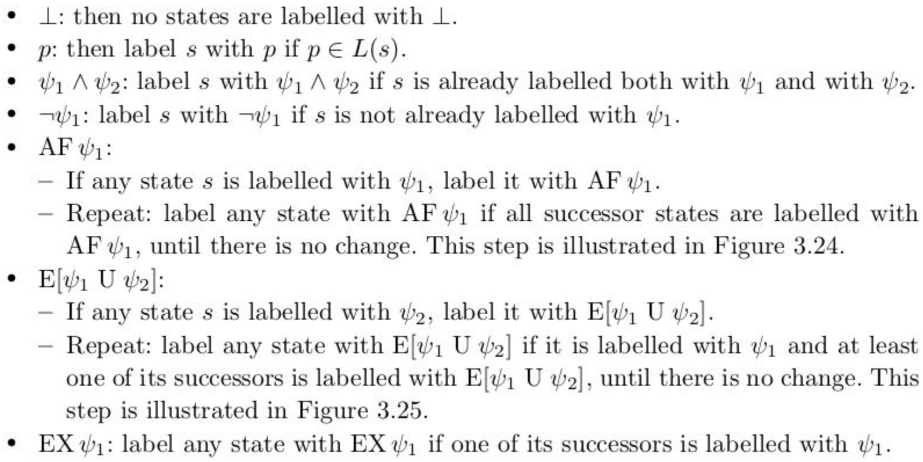
Model Checking

- System is modeled as transition system
  - M = (S, ->, L) with a set of atoms (p, q, r, … : either True or False)
    - S: States
    - ->: Transitions (rule: transitions are always possible) (paths are infinite)
    - L: which atoms are true in which states
- Problem: Is “M, s ⊨ φ” true?
- Input:
  - Model M = (S, ->, L)
  - Formula φ in a temporal logic



Model Checking Algorithm (CTL)

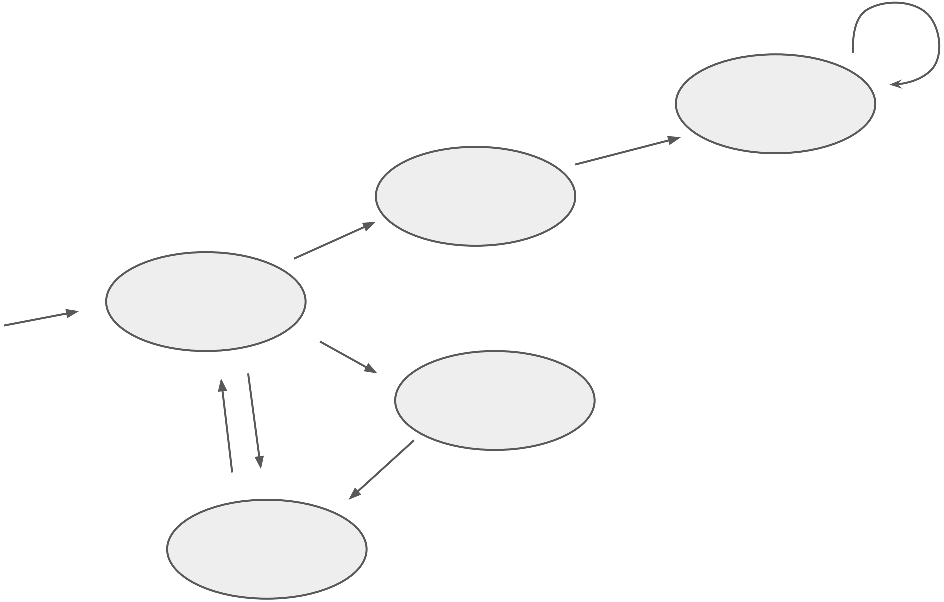
- Strategy: Starting from the smallest subformulas and working outward towards φ, label the states of M with the subformulas of φ that are satisfied there.
- Follow the following rules until the whole φ has been considered:



- $\bot$: then no states are labelled with $\bot$.
- p: then label s with p if $p \in L(s)$.
- $\psi_1 \land \psi_2$: label s with $\psi_1 \land \psi_2$ if s is already labelled both with ψ_1 and with ψ_2.
- $\neg\psi_1$: label s with $\neg\psi_1$ if s is not already labelled with ψ_1.
- $\mathrm{AF}\,\psi_1$:
  - If any state s is labelled with ψ_1, label it with $\mathrm{AF}\,\psi_1$.
  - Repeat: label any state with $\mathrm{AF}\,\psi_1$ if all successor states are labelled with $\mathrm{AF}\,\psi_1$, until there is no change. This step is illustrated in Figure 3.24.
- $\mathrm{E}[\psi_1\ \mathrm{U}\ \psi_2]$:
  - If any state s is labelled with ψ_2, label it with $\mathrm{E}[\psi_1\ \mathrm{U}\ \psi_2]$.
  - Repeat: label any state with $\mathrm{E}[\psi_1\ \mathrm{U}\ \psi_2]$ if it is labelled with ψ_1 and at least one of its successors is labelled with $\mathrm{E}[\psi_1\ \mathrm{U}\ \psi_2]$, until there is no change. This step is illustrated in Figure 3.25.
- $\mathrm{EX}\,\psi_1$: label any state with $\mathrm{EX}\,\psi_1$ if one of its successors is labelled with ψ_1.

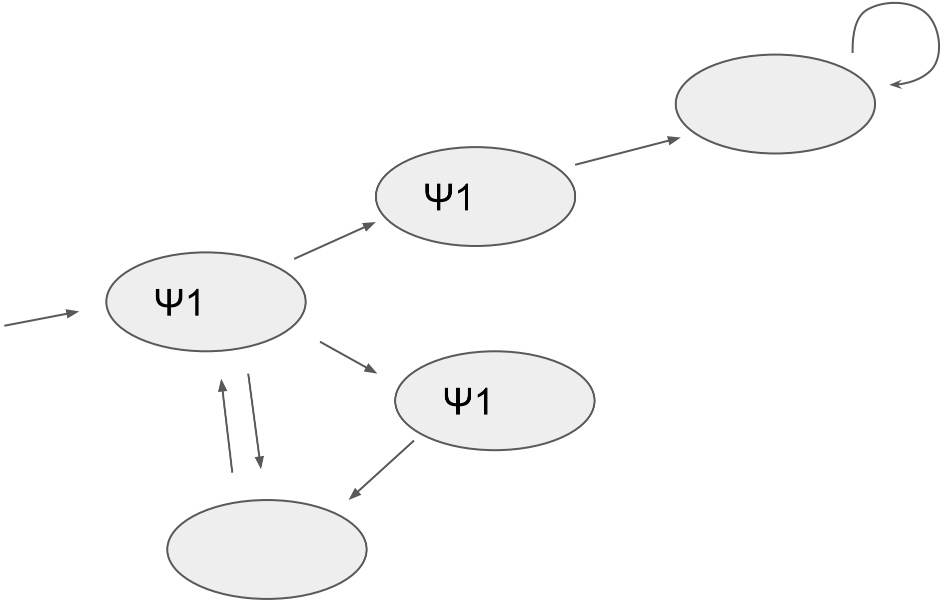
Example of CTL algorithm

Q: Is “E [Ψ1 U Ψ2]”
true in M?



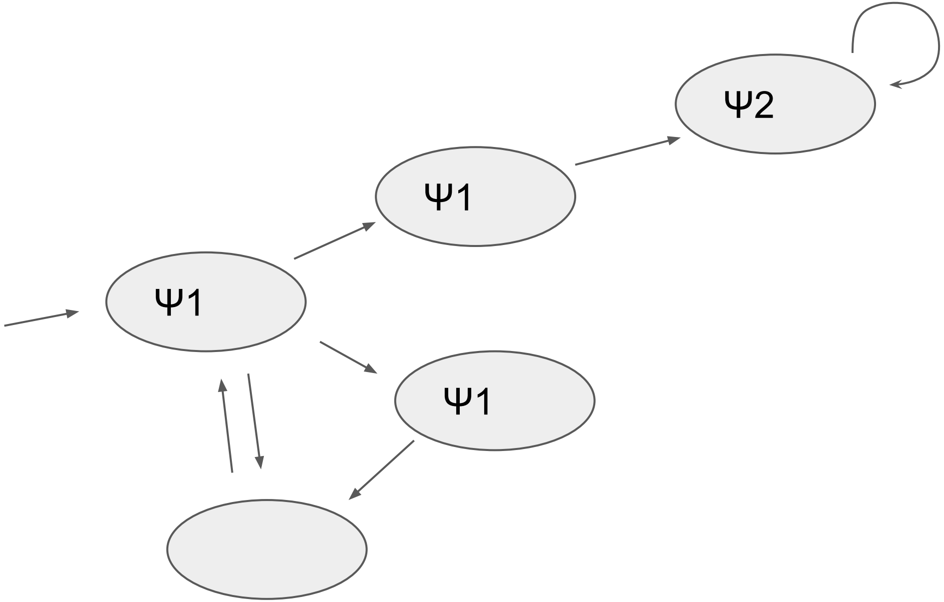
Example of CTL algorithm

Q: Is “E [Ψ1 U Ψ2]” true in M?



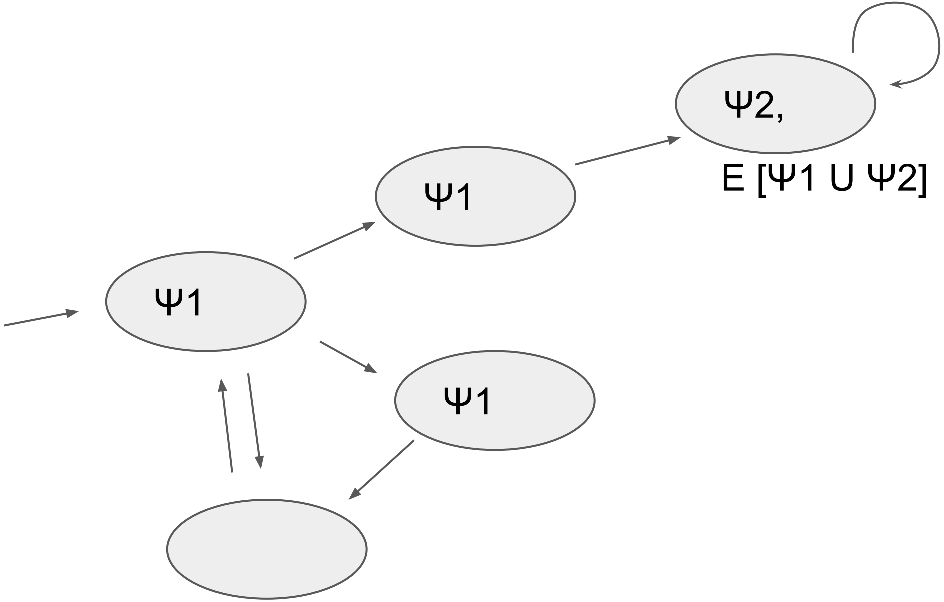
Example of CTL algorithm

Q: Is “E [Ψ1 U Ψ2]”
true in M?



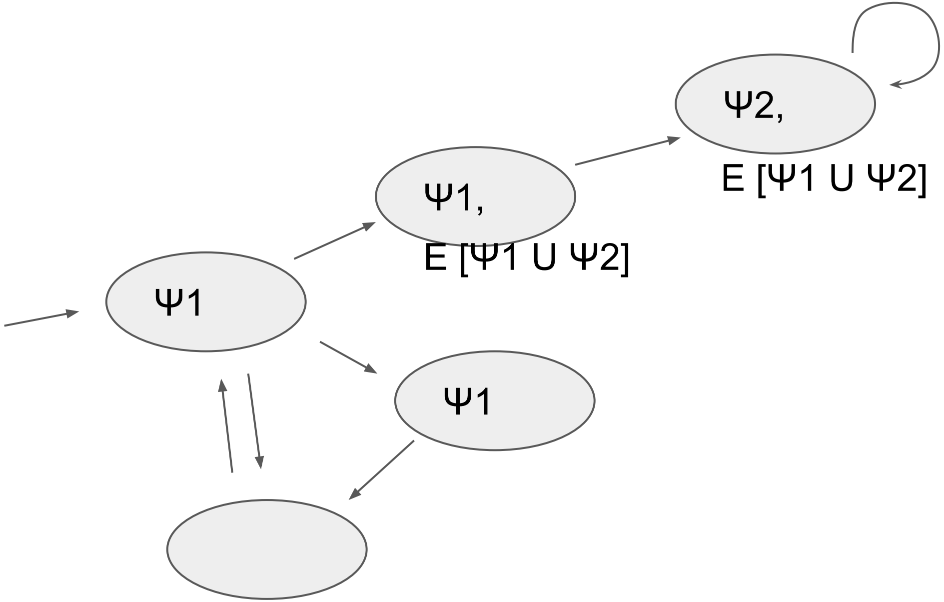
Example of CTL algorithm

Q: Is “E [Ψ1 U Ψ2]”
true in M?



Example of CTL algorithm

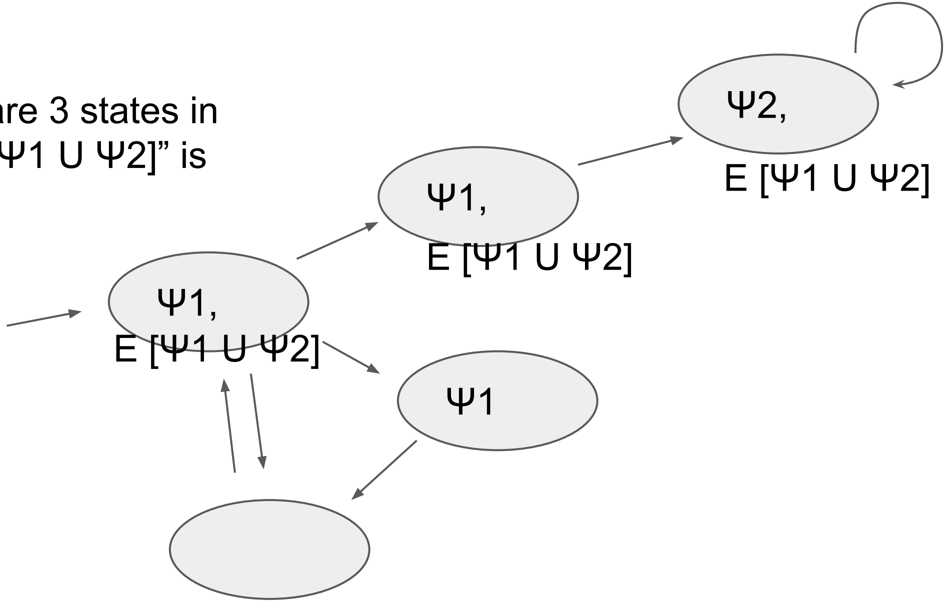
Q: Is “E [Ψ1 U Ψ2]”
true in M?



Example of CTL algorithm

Q: Is “E [Ψ1 U Ψ2]” true in M?

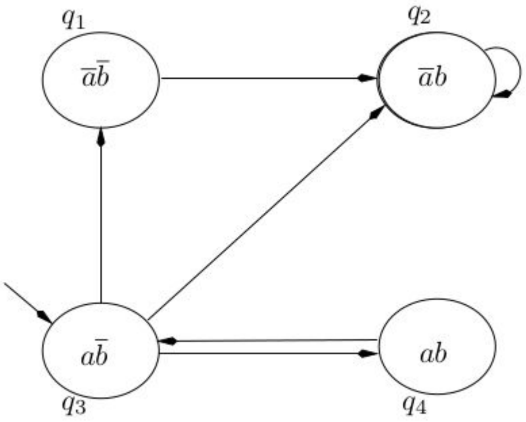
A: There are 3 states in which “E [Ψ1 U Ψ2]” is true.



Model Checking Algorithm (LTL)

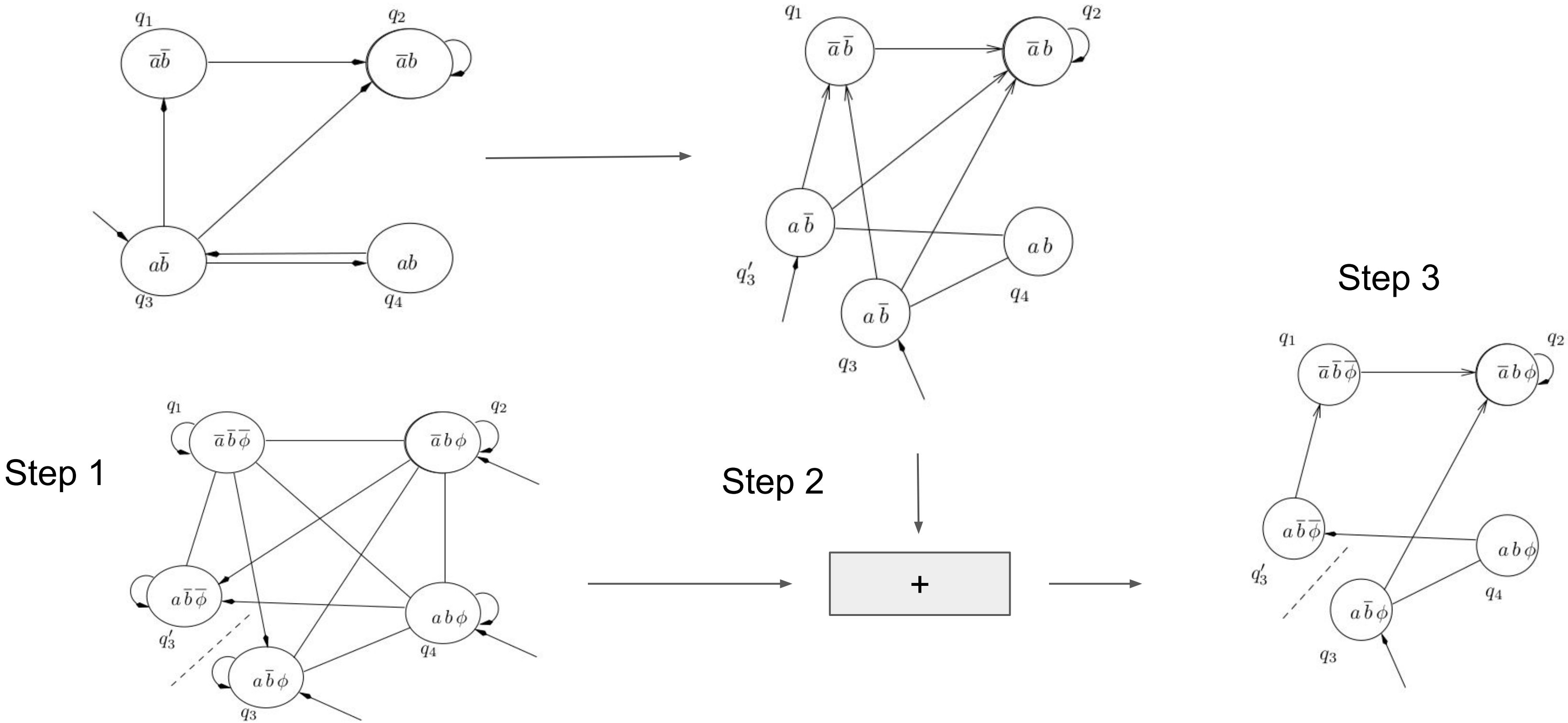
- Step 1: Construct an automaton A¬φ that accepts formula ¬φ
- Step 2: Combine A¬φ and model M into a new automaton.
- Step 3: Check if the new automaton accepts any path. If no, M, s ⊨ φ; if yes, the path is a counterexample.

Our model M:



Example of LTL algorithm

Our model M:



Step 1

Step 2

Step 3

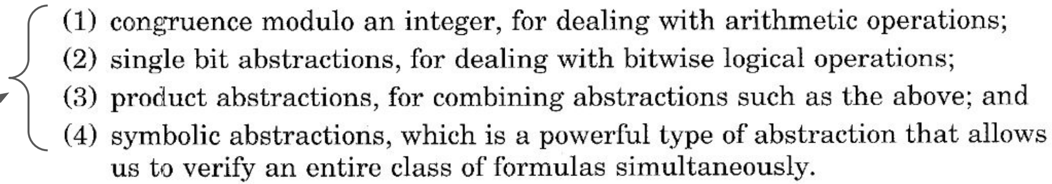
+

Ways to overcome State Explosion Problem

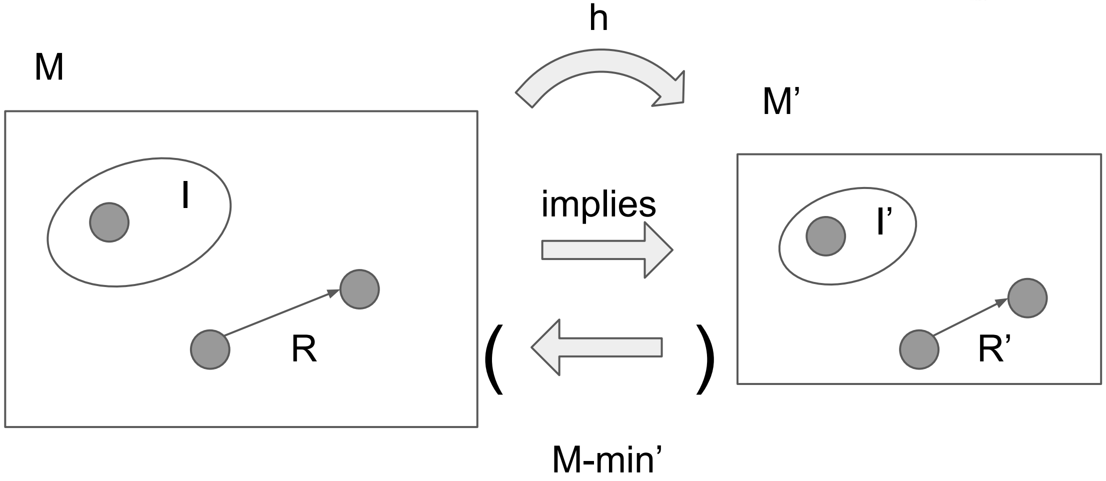
- Abstraction (<- our required reading)
- Partial order reduction
- BDD-based symbolic model checking
- Bounded model checking with SAT / SMT

Abstraction

- Approximation
    - M’ approximates M
    - Use M’ to deduce properties of M



(1) congruence modulo an integer, for dealing with arithmetic operations;
(2) single bit abstractions, for dealing with bitwise logical operations;
(3) product abstractions, for combining abstractions such as the above; and
(4) symbolic abstractions, which is a powerful type of abstraction that allows us to verify an entire class of formulas simultaneously.



Abstraction

- How to produce M-min’
  - Impractical to construct directly from M explicitly (what if b is 64-bit)
  - Solution: Compute it directly from the program text using relational semantics + approximation tricks

```
0: $p := 0$
1: while $b \neq 0$
        $p := p \oplus \mathrm{lsb}(b)$
        $b := b \gg 1$
   endwhile
2: end
```

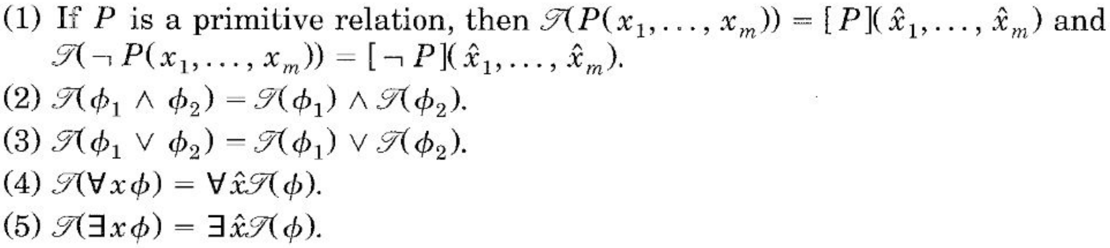
Symbolic Representation of the program:

$$
\begin{aligned}
&(\mathrm{PC} = 0 \land p' = 0 \land b' = b \land \mathrm{PC}' = 1) \\
&\quad \lor (\mathrm{PC} = 1 \land b = 0 \land p' = p \land b' = b \land \mathrm{PC}' = 2) \\
&\quad \lor (\mathrm{PC} = 1 \land b \neq 0 \land p' = p \oplus \mathrm{lsb}(b) \land b' = b \gg 1 \land \mathrm{PC}' = 1) \\
&\quad \lor (\mathrm{PC} = 2 \land p' = p \land b' = b \land \mathrm{PC}' = 2).
\end{aligned}
$$

Abstraction

- How to produce M-min’ from program text
  - Step 1. Derive formula *I* for initial condition and formula *R* for the transition relation using *relational semantic*. *I* and *R* can represent M.
  - Step 2. Try to compute *I-min’* and *R-min’* (which represent M-min’) directly from *I* and *R*.
  - Step 3. Step 2 is too difficult. Use approximation tricks to derive I-app’ and R-app’ for M-app’ instead. M-app’ somehow similar to M-min’.
- Result: we get M-app’ instead of M-min’

The tricks:



(1) If P is a primitive relation, then $\mathscr{T}(P(x_1, \ldots, x_m)) = [P](\hat{x}_1, \ldots, \hat{x}_m)$ and $\mathscr{T}(\neg P(x_1, \ldots, x_m)) = [\neg P](\hat{x}_1, \ldots, \hat{x}_m)$.

(2) $\mathscr{T}(\phi_1 \wedge \phi_2) = \mathscr{T}(\phi_1) \wedge \mathscr{T}(\phi_2)$.

(3) $\mathscr{T}(\phi_1 \vee \phi_2) = \mathscr{T}(\phi_1) \vee \mathscr{T}(\phi_2)$.

(4) $\mathscr{T}(\forall x \phi) = \forall \hat{x} \mathscr{T}(\phi)$.

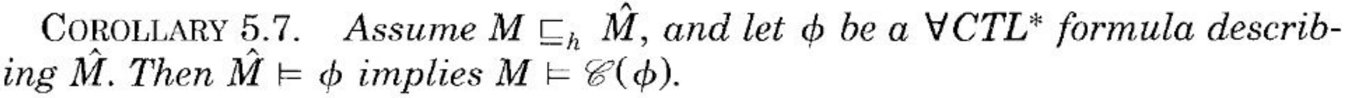
(5) $\mathscr{T}(\exists x \phi) = \exists \hat{x} \mathscr{T}(\phi)$.

How similar:

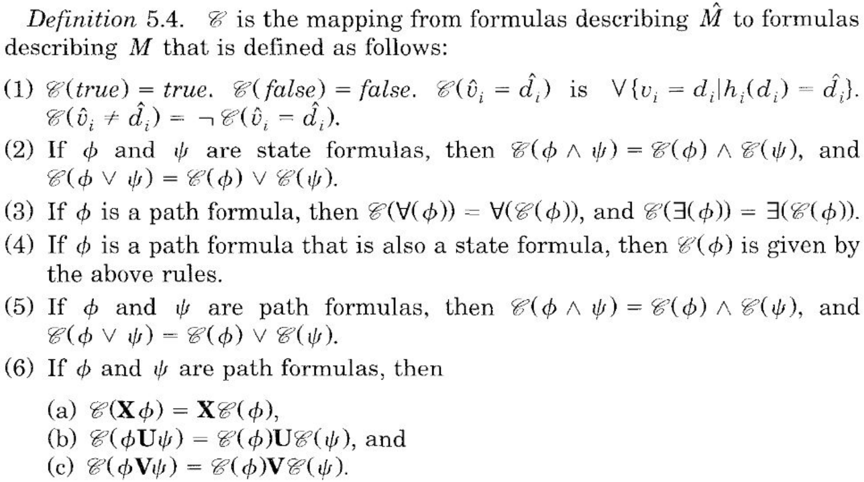
1.M-min’ transition -> M-app’ transition
2.M-min’ initial state -> M-app’ initial state

Abstraction

- Now we have M-app'. The paper shows M-app' also approximates M.
- Main result in the paper:



COROLLARY 5.7. *Assume $M \sqsubseteq_h \hat{M}$, and let ϕ be a $\forall$CTL* formula describing $\hat{M}$. Then $\hat{M} \models \phi$ implies $M \models \mathscr{C}(\phi)$.*



Definition 5.4. $\mathscr{C}$ is the mapping from formulas describing $\hat{M}$ to formulas describing M that is defined as follows:

(1) $\mathscr{C}(true) = true$. $\mathscr{C}(false) = false$. $\mathscr{C}(\hat{v}_i = \hat{d}_i)$ is $\bigvee\{v_i = d_i | h_i(d_i) = \hat{d}_i\}$. $\mathscr{C}(\hat{v}_i \neq \hat{d}_i) = \neg\mathscr{C}(\hat{v}_i = \hat{d}_i)$.

(2) If ϕ and ψ are state formulas, then $\mathscr{C}(\phi \wedge \psi) = \mathscr{C}(\phi) \wedge \mathscr{C}(\psi)$, and $\mathscr{C}(\phi \vee \psi) = \mathscr{C}(\phi) \vee \mathscr{C}(\psi)$.

(3) If ϕ is a path formula, then $\mathscr{C}(\forall(\phi)) = \forall(\mathscr{C}(\phi))$, and $\mathscr{C}(\exists(\phi)) = \exists(\mathscr{C}(\phi))$.

(4) If ϕ is a path formula that is also a state formula, then $\mathscr{C}(\phi)$ is given by the above rules.

(5) If ϕ and ψ are path formulas, then $\mathscr{C}(\phi \wedge \psi) = \mathscr{C}(\phi) \wedge \mathscr{C}(\psi)$, and $\mathscr{C}(\phi \vee \psi) = \mathscr{C}(\phi) \vee \mathscr{C}(\psi)$.

(6) If ϕ and ψ are path formulas, then

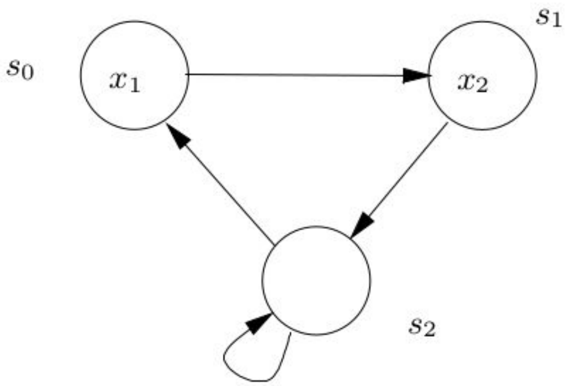
(a) $\mathscr{C}(\mathbf{X}\phi) = \mathbf{X}\mathscr{C}(\phi)$,
(b) $\mathscr{C}(\phi\mathbf{U}\psi) = \mathscr{C}(\phi)\mathbf{U}\mathscr{C}(\psi)$, and
(c) $\mathscr{C}(\phi\mathbf{V}\psi) = \mathscr{C}(\phi)\mathbf{V}\mathscr{C}(\psi)$.

Binary Decision Diagram

- Binary Decision Tree - - - some reduction rules - - > Reduced OBDD
- Characteristics of Reduced OBDD
    - Compact representation of boolean functions
    - Canonical: all semantically-equivalent boolean func have exactly the same BDD structures
    - Common operations (+, *, ^) have reasonable complexities (not exponential). The complexities depends on **the size of OBDD**.
    - Size of OBDD critically relies on the variable order. Worst case can be exponential. In some cases we can have only worst case (ex: integer multiplication function).

How is BDD useful?

- State space and transition relations in model M can be represented as Reduced OBDD.



States:

$$x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2 + \overline{x_1} \cdot \overline{x_2}$$

Transitions:

x_1	x_2	x'_1	x'_2	→
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

$$f^{\rightarrow} \overset{\text{def}}{=} \overline{x}_1 \cdot \overline{x}_2 \cdot \overline{x}'_1 \cdot \overline{x}'_2 + \overline{x}_1 \cdot \overline{x}_2 \cdot x'_1 \cdot \overline{x}'_2 + x_1 \cdot \overline{x}_2 \cdot \overline{x}'_1 \cdot x'_2 + \overline{x}_1 \cdot x_2 \cdot \overline{x}'_1 \cdot \overline{x}'_2.$$

Application 1: Model Checking of Linux TCP (2004)

- 50k lines of code
- Size of the system state – 250 KBs (~2^2048000 states)
    - The observable universe has ~2^273 atoms

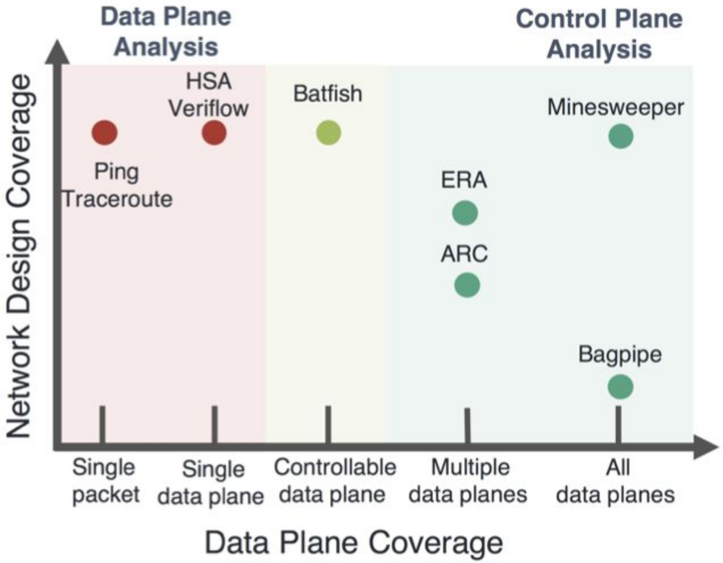
CMC System:

- Runs two Linux Kernels in parallel (two TCP peers)
- Containerised TCP code via an interface
- Compresses states efficiently to deal with state explosion
- Attempts to visit as many states as possible before running out of resources
- Checks for memory leaks, resource leaks, and protocol conformance

Results: Found **4 bugs** in implementation

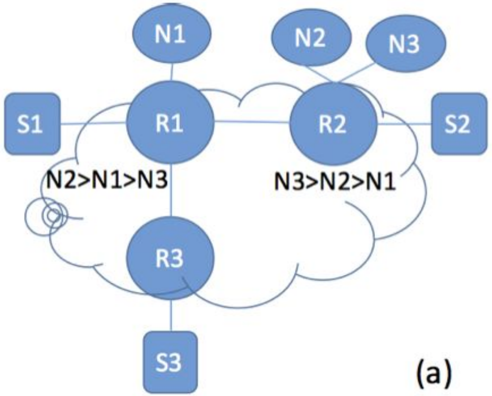
Application 2: Network Configuration Verification

- Most of the network outages happen due to misconfiguration
- We need tools that could verify all data planes for a given configuration

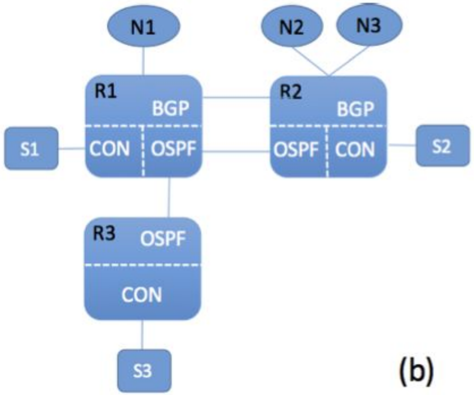


Minesweeper

- Reasoning about **networks as graphs**, not as paths
- Combinatorial search (**formal logic**) instead of message construction
- BGP as a **stable path problem**
- **Multiple optimizations** to scale to size of real networks



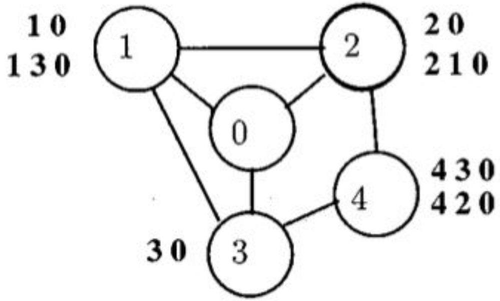
(a)

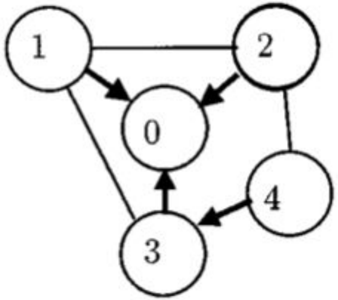


(b)

Stable Path Problem

- Graph (V, E) with a special node 0 that every other node is trying to reach.
- Paths to 0 from vk: P = (vk, vk-1, …, 0)
- Path value L(P)
- Stable path assignment s(v) = P if P maximizes the value
- A stable path problem is solvable if every node can have a stable assignment





Results

- Created formal system F that embeds network configuration and constraints
- Type of constraints supported:
    - Reachability and isolation
    - Waypoints, path length, equal paths, disjoint paths
    - Identifying forwarding loops and black holes
    - Load balancing, fault tolerance,
    - Full and partial equivalence
    - Many more

- Testing:
    - Applied to 152 real network, found **120 violations** of must-hold properties
    - Including one that possesses significant security threat