3.2 Linear-Time Temporal Logic

Model Checking

- Proof-based (syntactic): $\Gamma \vdash \phi$
- Represent system behaviour using set of formulas Γ

Theorem Proving

- Model-based (semantic): $M \models \phi$
- Represent system behaviour using an abstraction (model) M

Temporal Logic

Can reason about time using predicate logic:

> Use a variable t to represent time in predicates

Alternatively: try to build in the notion of time into the logic

Linear Temporal Logic (LTL):

- Start with atoms(p,q,r,...), just like in propositional logic
- Reason about the points in time these atoms hold
- Do this by considering execution paths of a system

Syntax

LTL Formulas:

$$\phi ::= \top|\bot|p|\neg\phi|\phi \land \phi|\phi \lor \phi|\phi \implies \phi|\mathbf{X}\phi|\mathbf{F}\phi|\mathbf{G}\phi|\phi\mathbf{U}\phi|\phi\mathbf{W}\phi|\phi\mathbf{R}\phi$$

Operator precedence:

- $\neg$ (highest)
- **X**
- **F**
- **G**
- **U**
- **R**
- **W**
- $\land$
- $\lor$
- $\implies$ (lowest)

Bold letters are temporal operators

- **U** is the only one you need for LTL. It is called the until.
- Can derive the rest from **U**

Transition Systems

A transition system (model) M = (S,→,L) is a set of states S, a (binary) transition relation → such that every s ∈ S has some s' ∈ S with s → s', and a labeling function L:S → P (Atoms)

Example:

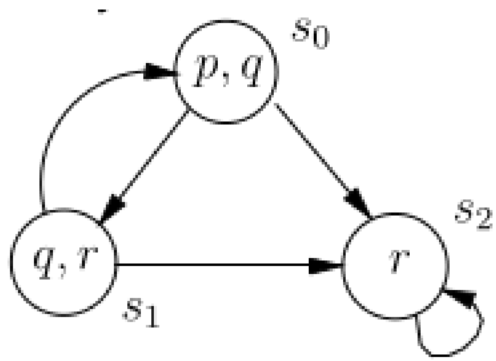


Figure 1: A representation of a transition system M = (S,→L) as a directed graph. We label state s with l iff l ∈ L(s)

A **path** in a model M = (S,→,L) is an infinite sequence of states in S such that $s_i \rightarrow s_{i+1}$.

- Write the path as $s_1 \rightarrow s_2$

Formulas over Paths

Let M = (S,→,L) be a model

Let $\pi = s_1 \rightarrow \ldots$ be a path in M

Path semantics:

- $\pi \models \top$
- $\pi \not\models \bot$
- $\pi \models$ p if and only if p ∈ $L(s_1)$ (This p is for the first set in the trace)
- $\pi \models \neg\phi$ if and only if $\pi \not\models \phi$
- $\pi \models \phi \wedge \psi$ if and only if $\pi \models \phi$ and $\pi \models \psi$
- $\pi \models \phi \vee \psi$ if and only if $\pi \models \phi$ or $\pi \models \psi$
- $\pi \models \phi \implies \psi$ if and only if $\pi \models \phi$ whenever $\pi \models \psi$
- $\phi \models \mathbf{X}\ \phi$ if and only if $\pi^2 \models \phi$
- $\phi \models \phi\ \mathbf{U}\ \psi$ if and only if there is some $k \geq 1$ where $\pi^k \models \psi$ and $\pi^j \models \phi$ for all $1 \leq j < k$

LTL Semantics

Let M = (S,→,L) be a model

Let s ∈ S

Let ϕ be an LTL formula

We write M, s $\models \phi$ to mean:

> For every path π of M starting at s, we have $\pi \models \phi$

If the model M is clear from context, we simply write s $\models \phi$

Talks about a single initial state

LTL Equivalences

Future operator: At some point ϕ needs to hold

$\mathbf{F}\phi \equiv \top \ \mathbf{U} \ \phi$

$\{\dots\} \rightarrow \{\dots\} \rightarrow \{\phi\} \rightarrow \{\dots\} \rightarrow \dots$

Globally operator: Need to always hold at all points in the path (complement of future operator)

$\mathbf{G}\phi \equiv \neg(\mathbf{F}\neg\phi)$

$\{\phi\} \rightarrow \{\phi\} \rightarrow \{\phi\} \rightarrow \{\phi\} \rightarrow \dots$

Weak Until operator: Second holds until first

$\phi\mathbf{W}\psi \equiv (\phi\mathbf{U}\psi) \lor \mathbf{G} \ \phi$

$\{\psi\} \rightarrow \dots \ \rightarrow \{\psi\} \rightarrow \{\phi\} \rightarrow \{\dots\} \rightarrow \dots$

Release operator: Second formula needs to hold till the point the first formula holds with it, then do whatever (like inverse until)

$\phi\mathbf{R}\psi \equiv \psi\mathbf{W}(\phi \land \psi)$

$\{\psi\} \rightarrow \dots \ \rightarrow \{\psi\} \rightarrow \{\phi, \psi\} \rightarrow \{\dots\} \rightarrow \dots$

LTL Examples

1. "For any state, if a request (of some resource) occurs, then it will eventually be acknowledged"

   $\mathbf{G}(\text{requested} \implies F \text{ acknowledged})$

2. "A certain process is enabled infinitely often on every computation path:"

   $\mathbf{GF}$ enabled

3. "A certain process will eventually be permanently deadlocked"

   $\mathbf{FG}$ deadlock

4. "If the process is enabled infinitely often, then it runs infinitely often"

   $\mathbf{GF} \text{ enabled} \implies \mathbf{GF} \text{ running}$

5. "An upwards traveling lift at the second floor does not change its direction when it has passengers wishing to go to the fifth floor"

   $\mathbf{G} \ (\text{floor2} \land \text{directionup} \land \text{ButtonPressed5} \implies (\text{directionup } \mathbf{U} \text{ floor5}))$

   (Does the above only imply for floor 2, wouldnt it break for any other floor)

Inexpressible in LTL

Cannot assert the **existence** of a path

- Example: "From any state it is possible to get to a restart state" (there is a path from all states to a state satisfying restart
- To do this "quantification" over paths, we need Computation Tree Logic (CTL)

3.3 Model Checking

Modeling Example: Mutual Exclusion

Mutual Exclusion model:

- Each process has **critical sections**
- Only one critical section can execute at a time (no interleaving of critical sections)
- Need a mutual exclusion protocol
- Basic requirements:
  - Safety: at most one critical section can execute at any given time
  - Liveness: request to enter critical section will eventually be granted
  - Non-blocking: a process can always request to enter critical section

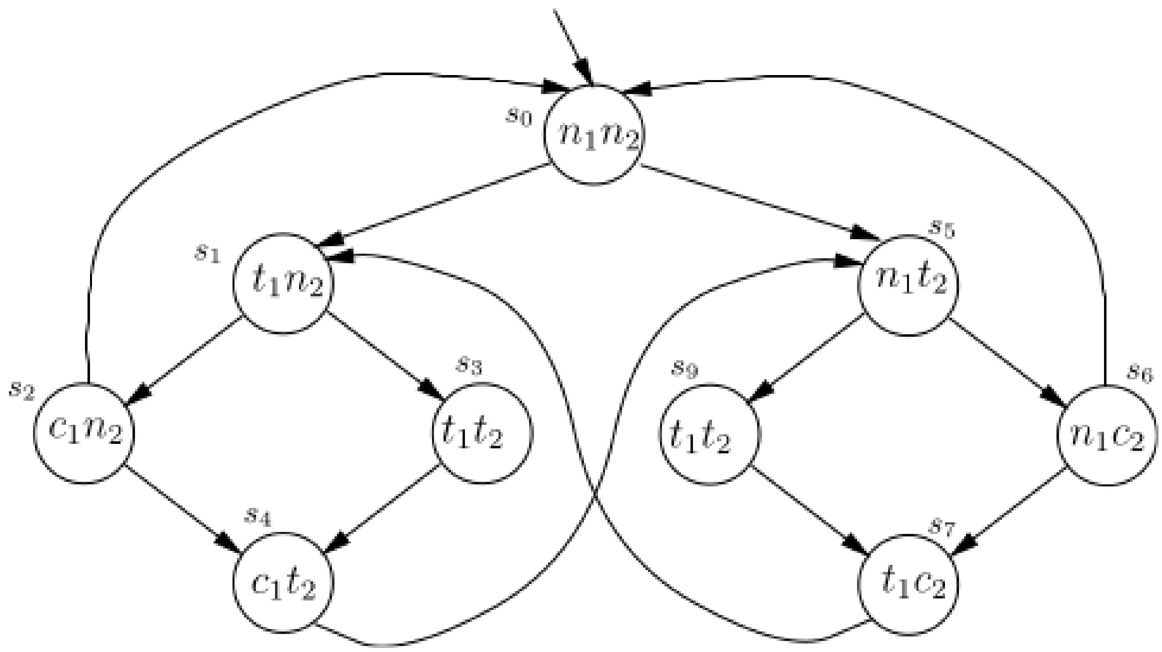


Figure 2: Model of mutual exclusion

- Safety: $\mathbf{G}\ \neg(c_1 \wedge c_2)$
- Liveness: $\mathbf{G}(t_1 \implies \mathbf{F}c_1)$

NuSMV Model Checker

```
MODULE main
VAR
    request : boolean;
    state : {ready, busy}
ASSIGN
    init(state) := ready;
    next(state) := case
            request : busy;
            TRUE : {ready,busy}
        ;esac
LTLSPEC G(request -> F state = busy)
```

3.4 Branch-Time Logic

Quantifiers in temporal logic: quantify over paths

- "There exists a path where p eventually holds"

  $\neg(\mathbf{G}\neg \mathrm{p})$

- "For all paths, p eventually holds"

  $\mathbf{GF}\mathrm{p}$

- This won't work if we have multiple quantifiers ($\forall$ $\exists$)
- Solution: Computation Tree Logic (CTL)

Interpreting formulas over trees

- In LTL, formulas are given meaning with respect to *traces*
- In CTL, formulas are given meaning with respect to a *tree*

Computation Tree Logic (CTL)

- Start with atoms (p,q,r,...), just like LTL
- Operators **U**, **F**, **G**, **X** are quantified by prefixing either **E** or **A**
- We reason about tree of states produced by executing system
- Transition Systems are the same as LTL

CTL Syntax

CTL formulas:

$$\phi ::= \top|\bot|\mathrm{p}|\neg\phi|\phi \wedge \phi|\phi \vee \phi|\phi \implies \phi|\ \mathbf{AX}\phi \mid \mathbf{EX}\phi \mid \mathbf{AF}\phi \mid \mathbf{EF}\phi \mid \mathbf{AG}\phi \mid \mathbf{EG}\phi \mid \mathbf{A}[\phi\mathbf{U}\phi] \mid \mathbf{E}[\phi\mathbf{U}\phi]$$

Operator precedence:

- $\neg$ (highest)
- *$\mathbf{X}$
- *$\mathbf{F}$
- *$\mathbf{G}$
- *$\mathbf{U}$
- $\wedge$
- $\vee$
- $\implies$ (lowest)

CTL Semantics

Let M = (S,→,L) be a model

Let s ∈ S

Let ϕ be an CTL formula

M, s $\models \phi$ means:

- If ϕ is atomic, satisfaction is determined by L
- If the top-level connective of ϕ (the connective occurring top-most in the parse tree of ϕ) is boolean connective ($\wedge$, $\vee$, $\neg$, $\top$, etc.) then the satisfaction question is answered by the usual truth-table definition and further recursion down ϕ
- If the top level connective is an operator beginning A, then satisfaction holds if all paths from s satisfy the "LTL formula" resulting from removing the A symbol.
- Similarly, if the top level connective begins with E, then satisfaction holds if some path from s satisfy the 'LTL formula' resulting from removing the E.

Formulas over Trees

Let M = (S,→,L) be a model

Tree semantics:

- M, s $\models \top$ and M, s $\not\models \bot$
- M, s $\models$ p iff p ∈ L(s)
- M, s $\models \neg\phi$ iff M, s $\not\models \phi$
- M, s $\models \phi_1 \wedge \phi_2$ iff M, s $\models \phi_1$ and M, s $\models \phi_2$
- M, s $\models \phi_1 \vee \phi_2$ iff M, s $\models \phi_1$ or M, s $\models \phi_2$
- M, s $\models \phi_1 \rightarrow \phi_2$ iff M, s $\not\models \phi_1$ or M, s $\models \phi_2$
- M, s $\models$ AXϕ iff for all s_1 such that s → s_1 we have M, s $\models \phi$. Thus, AX says: "in every state"
- M, s $\models$ EXϕ iff for some s_1 such that s → s_1 we have M, s $\models \phi$. Thus, EX says: "in some next state". E is dual to A - in exactly the same way that ∃ is dual to ∀ in predicate logic.
- M, s $\models$ AGϕ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \ldots$, where s_1 equals s, and all s_i along the path, we have M, $s_i \models \phi$. Mnemonically: for all computation paths beginning in s the property ϕ holds globally. Note that 'along the path' includes the path's initial state.
- M, s $\models$ EGϕ holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and for all s_i along the path, we have M,$s_i \models \phi$. Mnemonically: there exists a path beginning in s such that ϕ holds globally along the path
- M, s $\models$ AFϕ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, there is some s_i such that M,$s_i \models \phi$. Mnemonically: for all computation paths beginning in s there will be some future state where ϕ holds
- M, s $\models$ EFϕ holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and for some s_i along the path, we have M,$s_i \models \phi$. Mnemonically: these exists a computation path beginning in s such that ϕ holds in some future state
- M, s $\models$ A[ϕ_1 U ϕ_2] holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, that path satisfies ϕ_1 U ϕ_2 ie. there is some s_i along the path, such that M,$s_i \models \phi_2$ and, for each j < i, we have M,$s_j \models \phi_1$. Mnemonically: All computation paths beginning in s satisfy that ϕ_1 Until ϕ_2 holds on it
- M, s $\models$ E[ϕ_1 U ϕ_2] holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and that path satisfies ϕ_1 U ϕ_2 as specified in the previous rule. Mnemonically: there exists a computation path beginning in s such that ϕ_1 Until ϕ_2 holds on it

CTL Examples

- It is possible to get to state where started holds, but ready doesn't
  - **EF** (started $\wedge$ ¬ready)
- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged:
  - **AG** (requested $\implies$ **AF** acknowledged)
- The process is enabled infinitely often on every computation path:
  - **AG** (**AF** enabled)
- Whatever happens, the process will eventually be permanently deadlocked:
  - **AF** (**AG** deadlock)
- From any state it is possible to get to a restart state:
  - **AG** (**EF** restart)

CTL Equivalences

- $\neg\mathbf{AF}\phi \equiv \mathbf{EG}\ \neg\phi$
- $\neg\mathbf{EF}\phi \equiv \mathbf{AG}\ \neg\phi$
- $\neg\mathbf{AX}\phi \equiv \mathbf{EX}\ \neg\phi$
- $\mathbf{AF}\phi \equiv \mathbf{A}[\top\mathbf{U}\phi]$
- $\mathbf{EF}\phi \equiv \mathbf{E}[\top\mathbf{U}\phi]$

Adequate Sets of Operators

All the CTL operators can be defined using only **AU**, **EU**, and **EX**

3.6.1 Model Checking Algorithms

CTL Model Checking

As humans its easy to reason about all traces, but hard for a computer to reason.

LTL property: $\mathrm{G}\neg(c_1 \wedge c_2)$

CTL property: $\mathrm{AG}\neg(c_1 \wedge c_2)$

CTL Model Checking Problem

- Let M = (S,→,L) be a model
- Model checking: M, s $\models \phi$
- Given M, ϕ, we will find a set S' $\subseteq$ S such that M, s $\models \phi$ for all s $\in$ S'

Solving the third bullet will allow you to solve the second. (Find all states s where property is satisfied)

Adequate Sets of Operators

Recall: all the CTL operators can be defined using only **AU**, **EU**, and **EX**

More generally: Set of temporal operators is adequate and only if it contains at least one of {**AX**, **EX**}, at least one of {**EG**, **AF**, **AU**}, and **EU**

Our CTL model-checking algorithm only needs to handle: **AF**, **EU**, **EX**, $\wedge$, $\neg$, $\perp$

- Need three to represent the other operators. (This makes it adequate)

If the CTL formula is not in this form, first translate it

Label-Based CTL Model Checking Algorithm

- $\bot$: then no states are labelled with $\bot$
- p: then label s with p if $p \in L(s)$
- $\psi_1 \wedge \psi_2$: label s with $\psi_1 \wedge \psi_2$ 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
- AF ψ_1:
  - If any state s is labelled with ψ_1, label it with AFψ_1
  - Repeat: label any state with AFψ_1 if all successor states are labelled with AF ψ_1, until there is no change. (SHOWN IN FIGURE 3)
- E[ψ_1 U ψ_2]:
  - If any state s is labelled with ψ_2, label it with E[ψ_1 U ψ_2]
  - Repeat: label any state with E[ψ_1 U ψ_2], if it is labelled with ψ_1 and at least one of its successors is labelled with E[ψ_1 U ψ_2], until there is no change. (SHOWN IN FIGURE 4)
- EXψ_1: label any state with EXψ_1 if one of its successors is labelled with ψ_1

Complexity: linear in size of the formula, quadratic in the size of the model

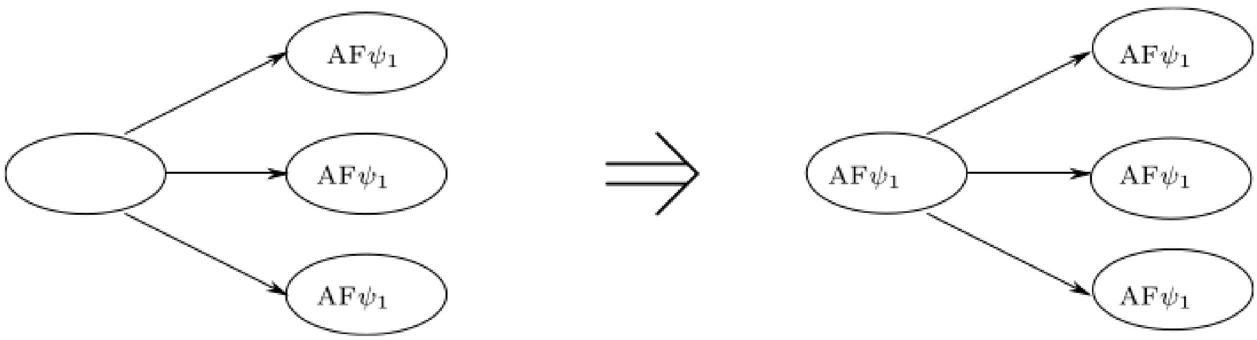


Figure 3: Example of AF Rule

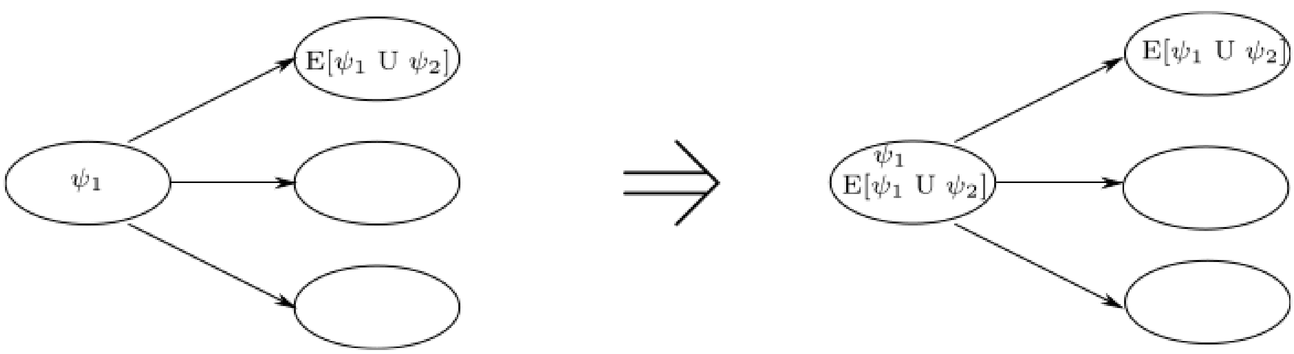


Figure 4: Example of E Rule

Handling EG Directly

EGψ_1:

- Label all the states with EGψ_1
- If any state s is not labelled with psi_1, delete the label EGψ_1
- Repeat: delete the label EGψ_1 from any state if none of its successors is labelled with EGψ_1; until there is no change

Translating formula is expensive. Size of the model is big, this will be terrible for the algorithm

Handling EG (Alternative):

- Restrict the graph to states satisfying ψ (delete all other states and their transitions)
- Find the maximal strongly connected components (SCCs); these are maximal regions of the state space in which every state is linked with (= has a finite path to) every other one in that region
- Use backwards breadth-first search on the restricted graph to find any state that can reach an SCC

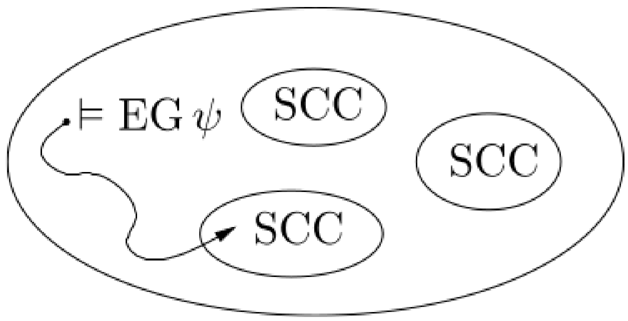


Figure 5: Example of EG alternative and the SCC

Complexity: linear in both the size of the formula, and the size of the model

Running label algorithm

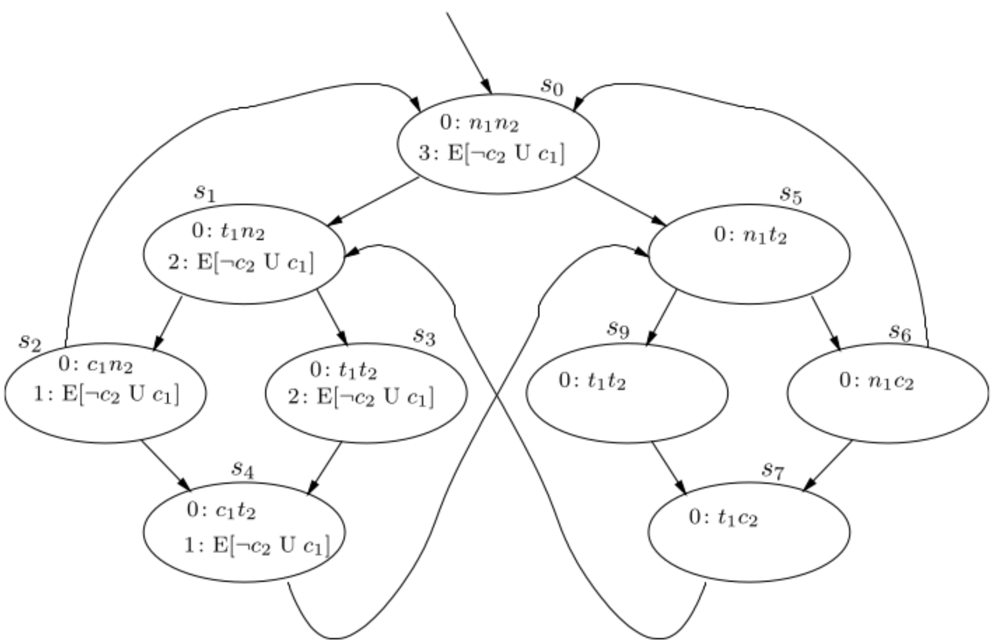


Figure 6: Example graph to demonstrate the labeling algorithm

0 1 3 4 $\leftarrow$ path that satisfy property

With the labeling, if the starting node is labeled, then the property is satisfied

5 6 9 7 $\leftarrow$ doesn't fit property so it doesnt get label

Pre operators

Helper functions for the labeling algorithm

- $\text{pre}_\exists(Y) = \{s \in S: \exists s', (s \to s') \land \in Y\}$

  (set of states that **may** transition into Y)
- $\text{pre}_\forall(Y) = \{s \in S: \forall s', (s \to s') \implies \in Y\}$

  (set of states that **only** transition into y

CTL Labeling: EX

function $\texttt{SAT}_{\texttt{EX}}\,(\phi)$
 /* determines the set of states satisfying EX ϕ */
local var X, Y
begin
  $X := \texttt{SAT}\,(\phi)$;
  $Y := \text{pre}_\exists(X)$;
  return Y
end

CTL Labeling: AF

function $\texttt{SAT}_{\texttt{AF}}\,(\phi)$
 /* determines the set of states satisfying AF ϕ */
local var X, Y
begin
  $X := S$;
  $Y := \texttt{SAT}\,(\phi)$;
  repeat until $X = Y$
  begin
    $X := Y$;
    $Y := Y \cup \text{pre}_\forall(Y)$
  end
  return Y
end

CTL Labeling: EU

function $\texttt{SAT}_{\texttt{EU}}\,(\phi, \psi)$
/* determines the set of states satisfying E[ϕ U ψ] */
local var W, X, Y
begin
    $W := \texttt{SAT}\,(\phi)$;
    $X := S$;
    $Y := \texttt{SAT}\,(\psi)$;
    repeat until $X = Y$
    begin
        $X := Y$;
        $Y := Y \cup (W \cap \mathrm{pre}_{\exists}(Y))$
    end
    return Y
end

CTL Labeling Algorithm

function $\texttt{SAT}\,(\phi)$
/* determines the set of states satisfying ϕ */
begin
    case
        ϕ is $\top$: **return** S
        ϕ is $\bot$: **return** $\emptyset$
        ϕ is atomic: **return** $\{s \in S \mid \phi \in L(s)\}$
        ϕ is $\neg\phi_1$: **return** $S - \texttt{SAT}\,(\phi_1)$
        ϕ is $\phi_1 \wedge \phi_2$: **return** $\texttt{SAT}\,(\phi_1) \cap \texttt{SAT}\,(\phi_2)$
        ϕ is $\phi_1 \vee \phi_2$: **return** $\texttt{SAT}\,(\phi_1) \cup \texttt{SAT}\,(\phi_2)$
        ϕ is $\phi_1 \rightarrow \phi_2$: **return** $\texttt{SAT}\,(\neg\phi_1 \vee \phi_2)$
        ϕ is $\mathrm{AX}\,\phi_1$: **return** $\texttt{SAT}\,(\neg\mathrm{EX}\,\neg\phi_1)$
        ϕ is $\mathrm{EX}\,\phi_1$: **return** $\texttt{SAT}_{\texttt{EX}}(\phi_1)$
        ϕ is $\mathrm{A}[\phi_1 \text{ U } \phi_2]$: **return** $\texttt{SAT}(\neg(\mathrm{E}[\neg\phi_2 \text{ U } (\neg\phi_1 \wedge \neg\phi_2)] \vee \mathrm{EG}\,\neg\phi_2))$
        ϕ is $\mathrm{E}[\phi_1 \text{ U } \phi_2]$: **return** $\texttt{SAT}_{\texttt{EU}}(\phi_1, \phi_2)$
        ϕ is $\mathrm{EF}\,\phi_1$: **return** $\texttt{SAT}\,(\mathrm{E}(\top \text{ U } \phi_1))$
        ϕ is $\mathrm{EG}\,\phi_1$: **return** $\texttt{SAT}(\neg\mathrm{AF}\,\neg\phi_1)$
        ϕ is $\mathrm{AF}\,\phi_1$: **return** $\texttt{SAT}_{\texttt{AF}}\,(\phi_1)$
        ϕ is $\mathrm{AG}\,\phi_1$: **return** $\texttt{SAT}\,(\neg\mathrm{EF}\,\neg\phi_1)$
    end case
end function

State Explosion

CTL model checking can be fast (linear), but...

- State space can be huge
- Possible solutions:
  - Binary decision diagrams (BDDs): represent sets of states
  - Partial order reduction: exploit the fact that some traces can be equivalent with respect to a temporal logic property (can ignore prefixes if two traces end up in the same spot)
  - Composition: decompose problem into easier/smaller subproblems

3.6.3 Labeling-Based Algorithm for LTL

Below is an example of why we **CANT** make a labeling-based algorithm for LTL

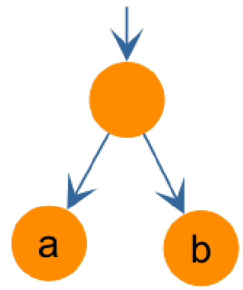


Figure 7: LTL fail, formula is satisfied, but neither subformula are satisfied

Basic Idea for LTL Model Checking

LTL model checking:

1. Construct automaton $A_{\neg\phi}$ for $\neg\phi$ (this encodes exactly the traces that don't satisfy ϕ)
2. Combine A$\neg\phi$ with the model M of the system (resulting in transition system whose paths are both paths of automaton and system)

- If there is a path in the combined automaton from step 2, output = true
  - path = infinite trace
- If there exists a path (return it as a counter example)a

Step 1 is tricky

Also, you can combine both steps 1 and 2 into a single step

Example LTL Model Checking

EXAMPLE PROBLEM: LTL ¬(a **U** b)

```
init(a) := 1;
init(b) := 0;
next(a) := case
             !a : 0;
             b  : 1;
             1  : {0,1};
           esac;
next(b) := case
             a & next(a) : !b;
             !a : 1;
             1  : {0,1};
           esac;
```

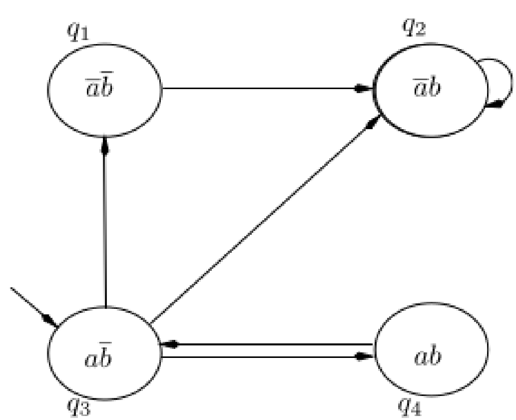
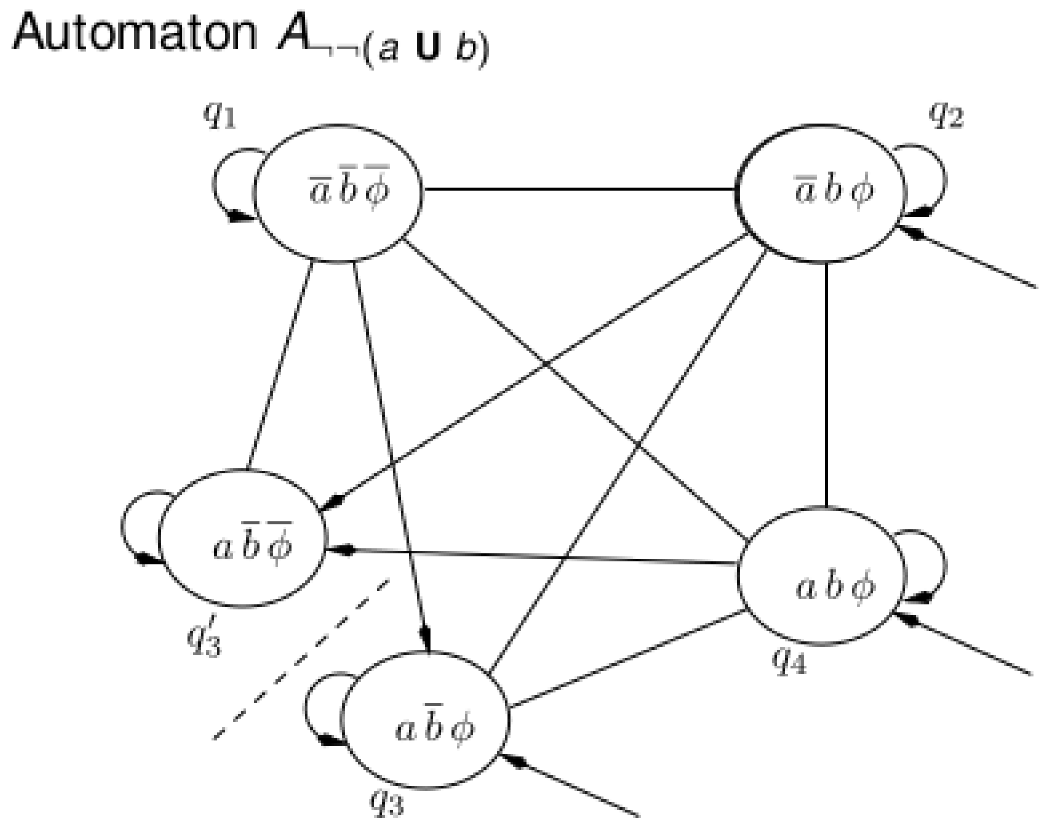


Figure 8: Example Model



{width=

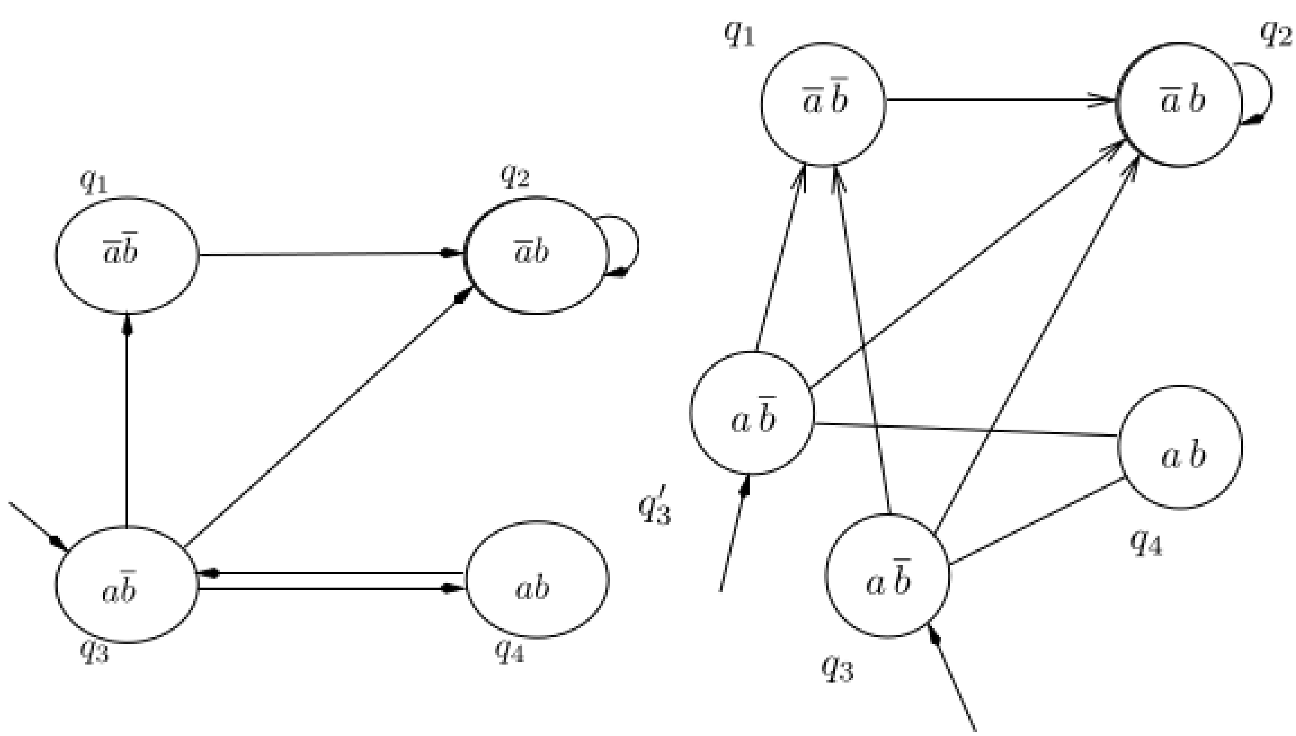


Figure 9: Example Transform Model. Take in the model and transform it so that there are as many states as automaton

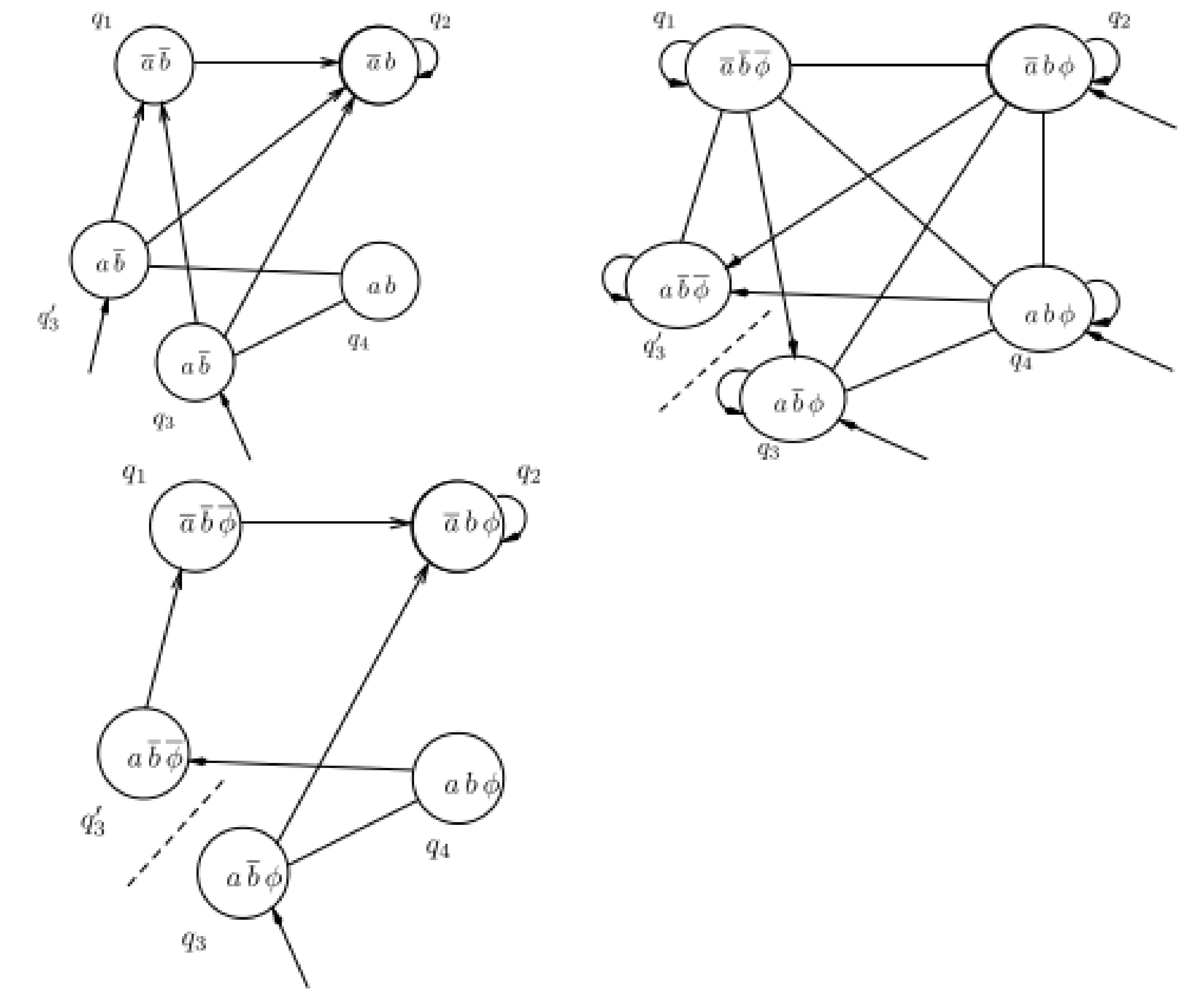


Figure 10: Example of combining the two models

Combining Model with Automaton

If directed edge exist in both, it exists in combined

If initial state exist in both, it exists in combined