Temporal Logics & Model Checking

Farn Wang
Dept. of Electrical Engineering
National Taiwan University

Specifications, descriptions, & verification

- specification:
  - The user's requirement
- description (implementation):
  - The user's description of the systems
  - No strict line between description and specification.
- verification:
  - Does the description satisfy the specification?

Formal specification & automated verification

- formal specification:
  - specification with rigorous mathematical notations
- automated verification:
  - verification with support from computer tools.

Why formal specifications?

- to make the engineers/users understand the system to design through rigorous mathematical notations.
- to avoid ambiguity/confusion/misunderstanding in communication/discussion/reading.
- to specify the system precisely.
- to generate mathematical models for automated analysis.

But according to Goedel’s incompleteness theorem, it is impossible to come up with a complete specification.
Why automated verification?

- to somehow be able to verify complexer & larger systems
- to liberate human from the labor-intensive verification tasks
  - to set free the creativity of human
- to avoid the huge cost of fixing early bugs in late cycles.
- to compete with the core verification technology of the future.

Specification & Verification?

- Specification $\rightarrow$ Complete & sound.
- Verification
  $\rightarrow$ Reducing bugs in a system.
  $\rightarrow$ Making sure there are very few bugs.

Very difficult!

Competitiveness of high-tech industry!
A way to survive for the students!
A way to survive for Taiwan!

400 horses
100 microprocessors

$4$ billion development effort
$>50\%$ system integration & validation cost
$2,500,000-1,500,000$ lines of codes (most in Ada)
Bugs in complex software

- They take effects only with special event sequences.
  - the number of event sequences is factorial and super astronomical!
- It is impossible to check all traces with test/simulation.

Budget appropriation

<table>
<thead>
<tr>
<th>The rest</th>
<th>VERIFICATION 40%-60%</th>
<th>Design &amp; Coding 10%-20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERIFICATION 1%</td>
<td>Design &amp; Coding 99%</td>
<td></td>
</tr>
</tbody>
</table>

Training in Taiwan College

Three technologies in verification

- Testing (real wall for real cars)
  - Expensive
  - Low coverage
  - Late in development cycles
- Simulation (virtual wall for virtual cars)
  - Economic
  - Low coverage
  - Don't know what you haven't seen.
- Formal Verification (virtual car checked)
  - Expensive
  - Functional completeness
    - 100% coverage
  - Automated!
    - With algorithms and proofs.

Sum of the 3 angles = 180°

- Testing (check all Δs you see)
  - Expensive
  - Low coverage
  - Late in development cycles
- Simulation (check all Δs you draw)
  - Economic
  - Low coverage
  - Don't know what you haven't seen.
- Formal Verification (we prove it.)
  - Expensive
  - Functional completeness
    - 100% coverage
  - Automated!
    - With algorithms and proofs.
Model-checking - a general framework for verification of sequential systems

![Diagram showing model checking process](image)

**Answer**

- Yes if the model is equivalent to the specification
- No if not.

---

Models & Specifications - formalism

Whenever a baby cries, it is hungry.

- Logics: $\Box (\text{crying} \rightarrow \text{hungry})$
- Graphs:

![Graph showing model checking](image)

---

Models & Specifications - fairness assumptions

Some properties are almost impossible to verify without assumptions.

Example: $\Box (\text{start} \rightarrow \Diamond \text{finish})$

To verify that a program halts, we assume:

- CPU does not burn out.
- OS gives the program a *fair* share of CPU time.
- All the drivers do not stuck.
- ........

---

Model-checking - frameworks in our lecture

![Table showing model checking frameworks](image)

- **Spec**
- **Logics**
  - traces
  - Trees
  - Linear
  - Branching

<table>
<thead>
<tr>
<th>Model</th>
<th>Spec</th>
<th>Logics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>traces</td>
<td>F=∅</td>
</tr>
</tbody>
</table>

- ✓: known;
- ☐: discussed in the lecture
History of Temporal Logic

- Designed by philosophers to study the way that time is used in natural language arguments
- Reviewed by Prior [PR57, PR67]
- Brought to Computer Science by Pnueli [PN77]
- Has proved to be useful for specification of concurrent systems

Framework

- Temporal Logic is a class of Modal Logic
- Allows qualitatively describing and reasoning about changes of the truth values over time
- Usually implicit time representation
- Provides variety of temporal operators (sometimes, always)
- Different views of time (branching vs. linear, discrete vs. continuous, past vs. future, etc.)

Outline

- Linear
  - LPTL (Linear time Propositional Temporal Logics)
- Branching
  - CTL (Computation Tree Logics)
  - CTL* (the full branching temporal logics)

Kripke structure

A = (S, S₀, R, L)

- S
  - a set of all states of the system
- S₀ ⊆ S
  - a set of initial states
- R ⊆ S×S
  - a transition relation between states
- L : S ↦ 2ᵖ
  - a function that associates each state with set of propositions true in that state
Kripke Model

- Set of states \( S \)
  - \( \{q_1, q_2, q_3\} \)
- Set of initial states \( S_0 \)
  - \( \{q_1\} \)
- Set of atomic propositions \( AP \)
  - \( \{a, b\} \)

Example of Kripke Structure

Suppose there is a program

- Initially \( x=1 \) and \( y=1; \)
- While true do
  - \( x := (x+y) \mod 2; \)
- Endwhile

where \( x \) and \( y \) range over \( D=\{0,1\} \)

Example of Kripke Structure

- \( S=D \times D \)
- \( S_0=\{(1,1)\} \)
- \( R=\{(1,1),(0,1),(0,1),(1,1),(1,0),(1,0),(0,0),(0,0)\} \)
- \( L((1,1))=\{x=1,y=1\}, L((0,1))=\{x=0,y=1\}, L((1,0))=\{x=1,y=0\}, L((0,0))=\{x=0,y=0\} \)

BNF, syntax definitions

Note!

Be sure how to read BNF!
- Used for defining syntax of context-free languages
- Important for the definition of
  - Automata predicates and
  - Temporal logics
- Used throughout the lectures!
- In exam: violate the syntax rules \( \Rightarrow \) no credit.

\[
A ::= c \mid x \mid (M) \mid A_1+A_2 \mid A_1-A_2 \\
M ::= c \mid x \mid (A) \mid M_1*M_2 \mid M_1/M_2 \\
\]

- \( c \) is an integer
- \( x \) is a variable name.
BNF, syntax definitions

A ::= c | x | (M) | A₁+A₂ | A₁–A₂
M ::= c | x | (A) | M₁*M₂ | M₁/M₂

- BNF, syntax definitions - derivation trees (from top down)

A ::= c | x | (M) | A₁+A₂ | A₁–A₂
M ::= c | x | (A) | M₁*M₂ | M₁/M₂

c is an integer
x is a variable name.

used in string generation.

- BNF, syntax definitions - parsing trees (from bottom up)

A ::= c | x | (M) | A₁+A₂ | A₁–A₂
M ::= c | x | (A) | M₁*M₂ | M₁/M₂

c is an integer
x is a variable name.

used in compiler.

- Temporal Logics : Catalog

propositional ↔ first-order
  global ↔ compositional
  branching ↔ linear-time
  points ↔ intervals
  discrete ↔ continuous
  past ↔ future
Temporal Logics

- Linear
  - LPTL (Linear time Propositional Temporal Logics)
    - LTL, PTL, PLTL

- Branching
  - CTL (Computation Tree Logics)
  - CTL* (the full branching temporal logics)

LPTL (PTL, LTL)
Linear-Time Propositional Temporal Logic

Conventional notation:
- propositions: \( p, q, r, \ldots \)
- sets: \( A, B, C, D, \ldots \)
- states: \( s \)
- state sequences: \( S \)
- formulas: \( \varphi, \psi \)
- Set of natural number: \( N = \{0, 1, 2, 3, \ldots \} \)
- Set of real number: \( R \)

LPTL

Given \( P \): a set of propositions,
a linear-time structure: state sequence
\[
S = s_0 s_1 s_2 s_3 s_4 \ldots s_k \ldots
\]

\( s_k \) is a function of \( P \) where \( P \subseteq \{\text{true, false}\} \)
or \( s_k \in 2^P \)

example: \( P = \{a, b\} \)
\{a\}{a,b}{a}{a}{b}...


Syntax definitions

Note!

Be sure how to read BNF!

- used for define syntax of context-free language
- important for the definition of
  - automata predicates and
  - temporal logics
- Used throughout the lectures!
- In exam: violate the syntax rules → no credit.

\[ A ::= (M) \mid A1 + A2 \mid A1 - A2 \]
\[ M ::= (A) \mid M1 \cdot M2 \mid M1 / M2 \]

---

LPTL

- syntax

\[ \psi ::= \text{true} \mid p \mid \neg \psi \mid \psi_1 \lor \psi_2 \mid \Box \psi \mid \psi_1 \U \psi_2 \]

abbreviation

\[ \text{false} \equiv \neg \text{true} \]
\[ \psi_1 \land \psi_2 \equiv \neg ((\neg \psi_1) \lor (\neg \psi_2)) \]
\[ \psi_1 \rightarrow \psi_2 \equiv (\neg \psi_1) \lor \psi_2 \]
\[ \Diamond \psi \equiv \text{true} \lor \psi \]
\[ \Box \psi \equiv \neg \Diamond \neg \psi \]

---

LPTL

- syntax

Exam. Symbol in CMU

\[ \Box p \quad Xp \quad p \text{ is true on next state} \]
\[ p \lor q \quad p \lor q \quad \text{From now on, } p \text{ is always true until } q \text{ is true} \]
\[ \Diamond p \quad Fp \quad \text{From now on, there will be a state where } p \text{ is eventually (sometimes) true} \]
\[ \Box p \quad Gp \quad \text{From now on, } p \text{ is always true} \]


? : don’t care
From now on, \( p \) is always true until \( q \) is true.

\[ \Diamond p \quad Fp \]

From now on, there will be a state where \( p \) is eventually (sometimes) true.

\[ \Box p \quad Gp \]

From now on, \( p \) is always true.

Two operator for Fairness:

- \( \Diamond \infty p \equiv \Box \Diamond p \); \( p \) will happen infinitely many times infinitely often
- \( \Box \infty p \equiv \Diamond \Box p \); \( p \) will be always true after some time in the future almost everywhere
LPTL

- semantics

Given a state sequence

\[ S = s_0 s_1 s_2 s_3 s_4 \ldots s_k \ldots \]

We define \( S \models \psi \) (\( S \) satisfies \( \psi \)) inductively as:

- \( S \models \text{true} \)
- \( S \models p \iff s_0(p) = \text{true} \), or equivalently \( p \in s_0 \)
- \( S \models \neg \psi \iff S \models \psi \) is false
- \( S \models \psi_1 \lor \psi_2 \iff S \models \psi_1 \) or \( S \models \psi_2 \)
- \( S \models \Omega \psi \iff S^{(1)} \models \psi \)
- \( S \models \psi_1 \U \psi_2 \iff \exists k \geq 0 (S^{(k)} \models \psi_2 \land \forall 0 \leq j < k (S^{(j)} \models \psi_1)) \)

Branching Temporal Logic

Basic assumption of tree-like structure

- Every node is a function of \( P \rightarrow \{ \text{true}, \text{false} \} \)
- Every state may have many successors

It can accommodate infinite and dense state successors

- In CTL and CTL*, it can’t tell
  - Finite and infinite
    - Is there infinite transitions?
  - Dense and discrete
    - Is there countable (\( \omega \)) transitions?
Branching Temporal Logic

Get by flattening a finite state machine

\[
\begin{array}{c}
s_0 \\
\downarrow \\
s_1 \\
\downarrow \\
s_2 \\
\downarrow \\
s_0 \\
\end{array}
\]

CTL (Computation Tree Logic)
- syntax

\[\varphi ::= \text{true} \mid p \mid \neg \varphi \mid \varphi_1 \lor \varphi_2 \mid \exists \varphi_1 \lor \varphi_2 \mid \forall \varphi_1 \lor \varphi_2\]

abbreviation:

\[
\begin{align*}
\text{false} & \equiv \neg \text{true} \\
\varphi_1 \land \varphi_2 & \equiv \neg (\neg \varphi_1) \lor (\neg \varphi_2) \\
\varphi_1 \to \varphi_2 & \equiv (\neg \varphi_1) \lor \varphi_2 \\
\exists \lozenge \varphi & \equiv \exists \text{true} \lor \varphi \\
\forall \Box \varphi & \equiv \neg \exists \lozenge \neg \varphi \\
\forall \lozenge \varphi & \equiv \forall \text{true} \lor \varphi \\
\exists \Box \varphi & \equiv \neg \forall \lozenge \neg \varphi
\end{align*}
\]

CTL (Computation Tree Logic)
- semantics

example symbol in CMU

\[
\begin{align*}
\exists \lozenge p & \quad \text{EX} p \\
\exists p U q & \quad p \text{EU} q \\
\forall \lozenge p & \quad \text{AX} p \\
\forall p U q & \quad p \text{AU} q
\end{align*}
\]

there exists a path where \( p \) is true on next state
from now on, there is a path where \( p \) is always true until \( q \) is true
for all path where \( p \) is true on next state
from now on, for all path where \( p \) is always true until \( q \) is true
CTL - semantics

\( \exists \varphi \) EX\( \varphi \) there exists a path where \( \varphi \) is true on next state

\( \forall \varphi \) AX\( \varphi \) for all path where \( \varphi \) is true on next state

CTL - semantics

\( \exists \varphi \bigcup q \) pEUq from now on, there is a path where \( \varphi \) is always true until \( q \) is true

\( \forall \varphi \bigcup q \) pAUq from now on, for all path where \( \varphi \) is always true until \( q \) is true
CTL
- semantic
Assume there are
- a tree structure $M$,
- one state $s$ in $M$, and
- a CTL formula $\varphi$

$M,s \models \varphi$ means $s$ in $M$ satisfy $\varphi$

---

CTL
- semantics

$s$-path : a path in $M$ which starts from $s$

$s_0$-path:
$s_0s_1s_2s_3s_5 \ldots \ldots$

$s_1$-path:
$s_1s_2s_3s_5 \ldots \ldots$

$s_2$-path:
$s_2s_3s_5 \ldots \ldots$

$s_{13}$-path:
$s_{13}s_{15} \ldots \ldots$

---

CTL
- examples (I)

$P_0: (p_0:=0 \mid p_0 := p_0 \lor p_1 \lor p_2)$
$P_1: (p_1:=0 \mid p_1 := p_0 \lor p_1)$
$P_2: (p_2:=0 \mid p_2 := p_1 \lor p_2)$

If $P_0$ is true, it is possible that $P_2$ can be true after the next two cycles.

$\forall (p_0 \rightarrow \exists \bigcirc \exists \bigcirc p_2)$
**CTL**

- examples (II)

1. If there are dark clouds, it will rain.
   \[\forall \square (\text{dark-clouds} \rightarrow \forall \Diamond \text{rain})\]

2. If a butterfly flaps its wings, the New York stock could plunder.
   \[\forall \square (\text{butterfly-flap-wings} \rightarrow \exists \Diamond \text{NY-stock-plunder})\]

3. If I win the lottery, I will be happy forever.
   \[\forall \square (\text{win-lottery} \rightarrow \forall \square \text{happy})\]

4. In an execution state, if an interrupt occurs in the next cycle, the interrupt handler will execute at the 2nd next cycle.
   \[\forall \square (\text{exec} \rightarrow \forall \Diamond (\text{intrpt} \rightarrow \forall \Diamond \text{intrpt-handler}))\]

- examples (III)

In an execution state, if an interrupt occurs in the next cycle, the interrupt handler will execute at the 2nd next cycle.

\[\forall \square (\text{exec} \rightarrow \forall \Diamond (\text{intrpt} \rightarrow \forall \Diamond (\text{intrpt-handler})))\]

Some possible mistakes:

\[\forall \square (\text{exec} \rightarrow ((\forall \Diamond \text{intrpt}) \rightarrow \forall \Diamond \text{intrpt-handler}))\]

\[\forall \square (\text{exec} \rightarrow ((\forall \square \text{intrpt}) \rightarrow \forall \Diamond \forall \Diamond \text{intrpt-handler}))\]

**CTL**

- important classes

- \(\forall \square \eta\): safety properties
  - \(\eta\) is always true in all computations from now.

- \(\exists \Diamond \eta\): reachability properties
  - \(\eta\) is eventually true in some computation from now.
    - \(\forall \square \eta = \neg \exists \Diamond \neg \eta\)

- \(\forall \Diamond \eta\): inevitabilities
  - \(\eta\) is eventually true in all computations from now.

- \(\exists \Diamond \eta\)
  - \(\forall \Diamond \eta \equiv \neg \exists \square \neg \eta\)
**CTL**
- syntax

- **CTL** * formula (state-formula)
  \[ \varphi ::= \text{true} | p | \neg \varphi_1 | \varphi_1 \lor \varphi_2 | \exists \psi | \forall \psi \]

- path-formula
  \[ \psi ::= \varphi | \neg \psi_1 | \psi_1 \lor \psi_2 | \diamond \psi_1 | \psi_1 \mathcal{U} \psi_2 \]

**CTL** * is the set of all state-formulas!

---

**CTL**
- examples (1/4)

In a fair concurrent environment, jobs will eventually finish.

\[ \forall(((\Box \diamond \text{execute}_1) \land (\Box \diamond \text{execute}_2)) \rightarrow \diamond \text{finish}) \]

or

\[ \forall(((\diamond \diamond \text{execute}_1) \land (\diamond \diamond \text{execute}_2)) \rightarrow \diamond \text{finish}) \]

---

**CTL**
- examples (2/4)

No matter what, infinitely many comets will hit earth.

\[ \forall \Box \diamond \Box \diamond \text{comet-hit-earth} \]

*What CTL?*

- \[ \forall \Box \forall \Box \forall \Box \Box \text{comet-hit-earth} \]
- \[ \forall \Box \forall \Box \Box \Box \Box \text{comet-hit-earth} \]

Exercise, please construct a model that tells the last from the first.

*Why not CTL?*

- \[ \forall \Box \forall \Box \forall \Box \Box \text{comet-hit-earth} \]
- \[ \forall \Box \exists \Box \text{comet-hit-earth} \]

---

**CTL**
- examples (2/4)

No matter what, infinitely many comets will hit earth.

\[ \forall \Box \diamond \text{comet-hit-earth} \]

*What is the difference?*

- \[ \forall \Box \diamond \text{comet-hit-earth} \]
- \[ \forall \Diamond \diamond \text{comet-hit-earth} \]

*Why not CTL?*

- \[ \forall \Box \forall \Box \diamond \text{comet-hit-earth} \]
- \[ \forall \Box \exists \Box \text{comet-hit-earth} \]

true
Please draw Kripke structures that tell (1) from (2) and (3)
(2) from (1) and (3)
(3) from (1) and (2)

Why not CTL ?

Please draw trees that tell (1) from (2)
(2) from (3)
(3) from (4)
(4) from (1)

If I buy lottery tickets infinitely many times, eventually I will win the lottery.

∀(((□∞buy-lottery) → ◇win-lottery)
**CTL**
- semantics

**suffix path:**

\[
S = s_0s_1s_2s_3s_5 \ldots \quad \quad S(0) = s_0s_1s_2s_3s_5 \ldots \quad \quad S(1) = s_1s_2s_3s_5 \ldots \quad \quad S(2) = s_2s_3s_5 \ldots \quad \quad S(3) = s_3s_5 \ldots \quad \quad S(4) = s_5 \ldots \\
S = s_0s_1s_6s_7s_8 \ldots \quad \quad S(2) = s_6s_7s_8 \ldots \quad \quad S = s_0s_{14}s_{12}s_{13}s_{15} \ldots \quad \quad S(3) = s_{13}s_{15} \ldots \\
\]

**CTL**
- semantics

**state-formula**

\[
\varphi ::= \text{true} \mid p \mid \neg \varphi_1 \mid \varphi_1 \lor \varphi_2 \mid \exists \psi \mid \forall \psi \\
\]

- \( M,s \models \text{true} \)
- \( M,s \models p \iff p \in s \)
- \( M,s \models \neg \varphi \iff M,s \models \varphi \text{ is false} \)
- \( M,s \models \varphi_1 \lor \varphi_2 \iff M,s \models \varphi_1 \text{ or } M,s \models \varphi_2 \)
- \( M,s \models \exists \psi \iff \exists \text{s-path } = S \ (S \models \psi) \)
- \( M,s \models \forall \psi \iff \forall \text{s-path } = S \ (S \models \psi) \)

**Expressiveness**

Given a language \( L \)

- what model sets \( L \) can express?
- what model sets \( L \) cannot?

**model set:** a set of behaviors

A formula = a set of models (behaviors)

- for any \( \varphi \in L, [\varphi] \overset{\text{def}}{=} \{ M \mid M \models \varphi \} \)

A language = a set of formulas.

**Expressiveness:** Given a model set \( F \), \( F \) is expressible in \( L \) iff \( \exists \varphi \in L ([\varphi] = F) \)
Expressiveness

Comparison in expressiveness:
Given two languages $L_1$ and $L_2$

**Definition:** $L_1$ is *more expressive than* $L_2$ ($L_2 < L_1$) iff $\forall \varphi \in L_2 \ (\varphi \text{ is expressible in } L_1)$

**Definition:** $L_1$ and $L_2$ are *expressively equivalent* ($L_1 \equiv L_2$) iff $(L_2 < L_1) \land (L_1 < L_2)$

**Definition:** $L_1$ and $L_2$ are *expressively incomparable* iff $\neg ((L_2 < L_1) \lor (L_1 < L_2))$

---

Expressiveness

- branching-time logics

What to compare with?
- finite-state automata on infinite trees.
- 2nd-order logics with monadic predicate and many successors (SnS)
- 2nd-order logics with monadic and partial-order

*Very little known at the moment,*
the fine difference in semantics of branching-structures

---

Expressiveness

- CTL*, example (I)

A tree that distinguishes the following two formulas.

$\forall((\Diamond \text{eat}) \rightarrow \Diamond \text{full})$
- Negation: $\exists((\Diamond \text{eat}) \land \Box \neg \text{full})$
- $(\forall \Diamond \text{eat}) \rightarrow (\forall \Diamond \text{full})$

---

Expressiveness

- CTL*, example (II)

A tree that distinguishes the following two formulas.

$\forall((\square \text{eat}) \rightarrow \Diamond \text{full})$
- $\forall \square (\text{eat} \rightarrow \forall \Diamond \text{full})$
- Negation: $\exists \Diamond (\text{eat} \land \exists \Diamond \neg \text{full})$
Expressiveness - CTL*

With the abundant semantics in CTL*, we can compare the subclasses of CTL*.

With restrictions on the modal operations after \(\exists, \forall\), we have many CTL* subclasses.

Example:
- \(B(\neg, \lor, O, U)\) : only \(\neg, \lor, O, U\) after \(\exists, \forall\)
- \(B(\neg, \lor, O, \diamond)\) : only \(\neg, \lor, O, \diamond\) after \(\exists, \forall\)
- \(B(O, \diamond)\) : only \(O, \diamond\) after \(\exists, \forall\)

Some theorems:
- \(B(\neg, \lor, O, \diamond, U) \equiv B(O, \diamond, U)\)
- \(\exists \diamond^\infty p\) is inexpressible in \(B(O, \diamond, U)\).

Expressiveness - CTL*

CTL* subclass expressiveness heirarchy
- \(\text{CTL}^* > B(\neg, \lor, O, \diamond, U, \diamond^\infty)\)
- \(\text{CTL}^* > B(O, \diamond, U, \diamond^\infty)\)
- \(\text{CTL}^* > B(\neg, \lor, O, \diamond, U)\)
- \(\text{CTL}^* = B(O, \diamond, U)\)
- \(\text{CTL}^* > B(\neg, \lor, O, \diamond)\)
- \(\text{CTL}^* > B(O, \diamond)\)
- \(\text{CTL}^* > B(\diamond)\)

Comparing PLTL with CTL*

assumption, all \(\phi \in \text{PLTL}\) are interpreted as \(\forall \phi\)

Intuition: PLTL is used to specify all runs of a system.
Verification

- LPTL, validity checking \( \psi \models \phi \)
  - instead, check the satisfiability of \( \psi \land \neg \phi \)
  - construct a tableau for \( \psi \land \neg \phi \)
- model-checking \( M \models \phi \)
  - LPTL: \( M \): a Büchi automata, \( \phi \): an LPTL formula
  - CTL: \( M \): a finite-state automata, \( \phi \): a CTL formula
- simulation & bisimulation checking \( M \models M' \)

Satisfiability-checking framework

Answer
Yes if the model is equivalent to the specification
No if not.

LPTL - tableau for satisfiability checking

Tableau for \( \phi \)
- a finite Kripke structure that fully describes the behaviors of \( \phi \)
- exponential number of states
- An algorithm can explore a fulfilling path in the tableau to answer the satisfiability.
  - nondeterministic
  - without construction of the tableau
  - PSPACE.

\[
\neg (\psi_1 \land \psi_2) \equiv (\neg \psi_1) \lor (\neg \psi_2)
\]
\[
\neg (\psi_1 \lor \psi_2) \equiv (\neg \psi_1) \land (\neg \psi_2)
\]
\[
\neg \Box \psi \equiv \Diamond \neg \psi
\]
\[
\neg \Diamond \psi \equiv \neg \Box \psi
\]
\[
\neg (\psi_1 U \psi_2) \equiv (\neg \psi_2) \lor ((\neg \psi_2) U ((\neg \psi_1) \land (\neg \psi_2)))
\]
\[
\neg \Box \psi \equiv \Diamond \neg \psi
\]
LPTL
- tableau for satisfiability checking

Tableau construction

CL(\(\phi\)) (closure) is the smallest set of formulas containing \(\phi\) with the following consistency requirement.

- \(\neg p \in CL(\phi)\) iff \(p \not\in CL(\phi)\)
- If \(\psi_1 \lor \psi_2, \psi_1 \land \psi_2 \in CL(\phi)\), then \(\psi_1, \psi_2 \in CL(\phi)\)
- If \(\Box \psi \in CL(\phi)\), then \(\psi \in CL(\phi)\)
- If \(\psi_1 U \psi_2 \in CL(\phi)\), then \(\psi_1, \psi_2, \Box (\psi_1 U \psi_2) \in CL(\phi)\)
- If \(\Diamond \psi \in CL(\phi)\), then \(\psi \in CL(\phi)\)

**Example:** \((p U q)\) tableau 

\(\text{CL}(p U q) = \{p U q, \Box p U q, p, \neg p, q, \neg q\}\)

Tableau \((V, E)\), node consistency condition:

- A tableau node \(v \in V\) is a set \(v \subseteq CL(\phi)\) such that
- \(p \in v\) iff \(\neg p \not\in v\)
- If \(\psi_1 \lor \psi_2 \in v\), then \(\psi_1 \lor \psi_2 \in v\)
- If \(\psi_1 \land \psi_2 \in v\), then \(\psi_1, \psi_2 \in v\)
- If \(\Box \psi \in v\), then \(\psi \in v\) and \(\Box \psi \in v\)
- If \(\Diamond \psi \in v\), then \(\psi \in v\) or \(\Box \Diamond \psi \in v\)
- If \(\psi_1 U \psi_2 \in v\), then \(\psi_2 \in v\) or \(\psi_1 \in v\) and \(\Box (\psi_1 U \psi_2) \in v\)

Tableau \((V, E)\), arc consistency condition:

Given an arc \((v, v') \in E\), if \(\Box \psi \in v\), then \(\psi \in v'\)

A node \(v\) in \((V, E)\) is initial for \(\phi\) if \(\phi \in v\).
Please use tableau method to show that \( pUq \not\models \Box q \) is false.

1) Convert to negation: \( (pUq) \land \Box \lnot q \)

2) \( CL((pUq) \land \Box \lnot q) = \{ (pUq) \land \Box \lnot q, pUq, \lor pUq, p, q, \Box \lnot q, \lor \Box \lnot q \} \)

Pf: In each path that is a model of \( (pUq) \land \Box \lnot q \), \( q \) must always be satisfied. Thus, \( pUq \) is never fulfilled in the model.

QED
CTL model-checking framework

Model Checker

Answer
Yes if the model is equivalent to the specification
No if not.

specification in logics
\[\exists, \forall, \Box, \neg, \lor, \bigotimes, \bigoplus\]

CTL - model-checking

Given a finite Kripke structure \( M \) and a CTL formula \( \varphi \), is \( M \) a model of \( \varphi \)?

- usually, \( M \) is a finite-state automata.
- PTIME algorithm.
- When \( M \) is generated from a program with variables, its size is easily exponential.

CTL - model-checking algorithm

- state-space exploration
  - state-spaces represented as finite Kripke structure
    - directed graph
    - nodes: states or possible worlds
    - arcs: state transitions
  - regular behaviors
- Usually the state count is astronomical.

Kripke structure
- Least fixpoint in modal logics

Dark-night murder, strategy I:
A suspect will be in the 2nd round iff

- He/she lied to the police in the 1st round; or
- He/she is loyal to someone in the 2nd round

What is the minimal solution to \( 2nd[i] \) ?

\[2nd[i] \equiv \text{Liar}[i] \lor \exists j \neq i (2nd[j] \land \text{Loyal-to}[i,j])\]
Kripke structure

- **Least fixpoint in modal logics**

In a dark night, there was a cruel murder.
- n suspects, numbered 0 through n-1.
- \( \text{Liar}[i] \) iff suspect i has lied to the police in the 1st round investigation.
- \( \text{Loyal-to}[i,j] \) iff suspect i is loyal to suspect j in the same criminal gang.
- \( \text{2nd}[i] \) iff suspect i to be in 2nd round investigation.

What is the minimal solution to \( \text{2nd}[i] \)?

**Safety analysis**

Given M and p (safety predicate), do all states reachable from initial states in M satisfy p?
- In model-checking: Is M a model of \( \forall \Box p \)?
- Or in risk analysis: Is there a state reachable from initial states in M satisfy p?

\( \forall \Box p \equiv \neg \exists \Diamond \neg p \equiv \neg \exists \text{true} U \neg p \)

Kripke structure

- **Greatest fixpoint in modal logics**

In a dark night, there was a cruel murder.
- n suspects, numbered 0 through n-1.
- \( \neg \text{Liar}[i] \) iff the police cannot prove suspect i has lied to the police in the 1st round investigation.
- \( \text{Loyal-to}[i,j] \) iff suspect i is loyal to j are in the same criminal gang.
- \( \text{2nd}[i] \) iff suspect i to be in 2nd round investigation.

What is the maximal solution to \( \neg \text{2nd}[i] \)?

In comparison:

\( \neg \text{2nd}[i] \equiv \neg \text{Liar}[i] \land \exists j \neq i (\neg \text{2nd}[j] \land \text{Loyal-to}[i,j]) \)

\( \neg \text{2nd}[i] \equiv \neg \text{Liar}[i] \land \forall j \neq i (\neg \text{2nd}[j] \land \text{Loyal-to}[i,j]) \)

\( \neg \text{2nd}[i] \equiv \neg \text{Liar}[i] \land \forall j \neq i (\neg \text{Loyal-to}[i,j] \rightarrow \neg \text{2nd}[j]) \)

\( \neg \text{2nd}[i] \equiv \neg \text{Liar}[i] \land \forall j \neq i (\text{Loyal-to}[i,j] \rightarrow \neg \text{2nd}[j]) \)
Reachability analysis: $\exists \Diamond \eta$

Is there a state $s$ reachable from another state $s'$?
- Encode risk analysis
- Encode the complement of safety analysis
- Most used in real applications

Kripke structure - safety analysis

Reachability algorithm in graph theory
Given
- a Kripke structure $A = (S, S_0, R, L)$
- a safety predicate $\eta$

find a path from a state in $S_0$ to a state in $[\neg \eta]$

Solutions in graph theory
- Shortest distance algorithms
- spanning tree algorithms

Kripke structure - safety analysis

/* Given $A = (S, S_0, R, L)$*/
safety_analysis($\eta$) /* using least fixpoint algorithm */ {
  for all $s$, if $\neg \eta \in L(s)$, $L(s) = L(s) \cup \{\exists \Diamond \neg \eta\}$;
  repeat {
    for all $s$, if $\exists (s, s') (\exists \Diamond \neg \eta \in L(s'))$
    $L(s) = L(s) \cup \{\exists \Diamond \neg \eta\}$;
  } until no more changes to $L(s)$ for any $s$.
  if there is an $s_0 \in S_0$ with $\exists \Diamond \neg \eta \in L(s_0)$,
  return 'unsafe,'
  else return 'safe.'
}

The procedure terminates since $S$ is finite in the Kripke structure.
Kripke structure - liveness analysis: \( \forall \Diamond \eta \)

Given
- a Kripke structure \( A = (S, S_0, R, L) \)
- a liveness predicate \( \eta \)
  can \( \eta \) be true eventually?

Example:
Can the computer be started successfully?
Will the alarm sound in case of fire?

---

Kripke structure - liveness analysis

Strongly connected component algorithm in graph theory

Given
- a Kripke structure \( A = (S, S_0, R, L) \)
- a liveness predicate \( \eta \)
  find a cycle such that
  - all states in the cycle are \( \neg \eta \)
  - there is a \( \neg \eta \) path from a state in \( S_0 \) to the cycle.

Solutions in graph theory
- strongly connected components (SCC)

---

Kripke structure - liveness analysis

\[ \text{liveness}(\eta) \] /* using greatest fixpoint algorithm */
\[
\begin{align*}
\text{for all } s, & \text{ if } \neg \eta \in L(s), \ L(s) = L(s) \cup \{ \exists \Box \neg \eta \}; \\
\text{repeat } \{ & \\
\text{for all } s, \text{ if } \exists \Box \neg \eta \in L(s) \text{ and } \forall (s,s') (\exists \Box \neg \eta \notin L(s)), \\
& L(s) = L(s) - \{ \exists \Box \neg \eta \}; \\
\} & \text{until no more changes to } L(s) \text{ for any } s. \\
\text{if there is an } s_0 \in S_0 \text{ with } \exists \Box \neg \eta \in L(s_0), \\
& \text{return 'liveness not true,'} \\
\text{else return 'liveness true.'}
\end{align*}
\]

The procedure terminates since \( S \) is finite in the Kripke structure.
Kripke structure
- liveness analysis

Greatest fixpoint in modal logics
iterative elimination

CTL model-checking

The NORMAL form needed in CTL model-checking:

1. only modal operators
   \( \exists \Box \varphi, \exists \psi_1 U \psi_2, \exists \Diamond \varphi \)

2. No modal operators
   \( \forall \Box \varphi, \forall \psi_1 U \psi_2, \forall \Diamond \varphi, \forall \Box \varphi, \exists \Diamond \varphi \)

3. No double negation: \( \neg \neg \varphi \)

4. No implication: \( \psi_1 \Rightarrow \psi_2 \)

CTL - model-checking algorithm (1/6)

Given \( M \) and \( \varphi \),

1. Convert \( \varphi \) to NORMAL form.
2. list the elements in \( \text{Cl}(\varphi) \) according to their sizes
   \[ \varphi_0 \varphi_1 \varphi_2 \ldots \varphi_n \]
   for all \( 0 \leq i \leq n \), \( \varphi_i \) is not a subformula of \( \varphi \)
3. for i=0 to n,
   label (\( \varphi_i \))
4. return `No!'

CTL - model-checking algorithm (2/6)

\[
\text{label(}\varphi\text{ )} \{
\text{case } p, \text{ return;}
\text{case } \neg \varphi, \text{ for all } s, \text{ if } \varphi \notin L(s), L(s) = L(s) \cup \{\neg \varphi\}
\text{case } \varphi \lor \psi, \text{ for all } s, \text{ if } \varphi \in L(s) \text{ or } \psi \in L(s),
L(s) = L(s) \cup \{\varphi \lor \psi\}
\text{case } \exists c \varphi, \text{ for all } s, \text{ if } \exists (s,s') \text{ with } \varphi \in L(s'),
L(s) = L(s) \cup \{\exists c \varphi\}
\text{case } \exists \psi_1 U \psi_2, \text{ lfp}(\psi_1, \psi_2); \text{ if } \psi \in L(s'),
L(s) = L(s) \cup \{\exists \Box \varphi\}
\text{case } \exists \Box \varphi, \text{ gfp}(\varphi); \text{ if } \exists (s,s') \text{ with } \varphi \in L(s'),
L(s) = L(s) \cup \{\exists \Box \varphi\}; \text{ return `Yes!'}
\}
**CTL - model-checking algorithm (3/6)**

\[ \text{lp}(\psi_1, \psi_2) \] /* least fixpoint algorithm */ {
  for all \( s \), if \( \psi_2 \in L(s) \), \( L(s) = L(s) \cup \{ \exists \psi_1 U \psi_2 \} \); 
  repeat {
    for all \( s \), if \( \psi_1 \in L(s) \) and \( \exists (s, s') (\exists \psi_1 U \psi_2 \in L(s')) \), 
    \( L(s) = L(s) \cup \{ \exists \psi_1 U \psi_2 \} \); 
  } until no more changes to \( L(s) \) for any \( s \).
}

The procedure terminates since \( S \) is finite in the Kripke structure.

---

**CTL - model-checking algorithm (4/6)**

Least fixpoint in modal logics

iterative expansion

---

**CTL - model-checking algorithm (5/6)**

\[ \text{gfp}(\psi) \] /* greatest fixpoint algorithm */ {
  for all \( s \), if \( \psi \in L(s) \), \( L(s) = L(s) \cup \{ \exists \Box \psi \} \); 
  repeat {
    for all \( s \), if \( \exists \Box \psi \in L(s) \) and \( \forall (s, s') (\exists \Box \psi \notin L(s')) \), 
    \( L(s) = L(s) - \{ \exists \Box \psi \} \); 
  } until no more changes to \( L(s) \) for any \( s \).
}

The procedure terminates since \( S \) is finite in the Kripke structure.

---

**CTL - model-checking algorithm (6/6)**

Greatest fixpoint in modal logics

iterative elimination

---
(∃O∃pUq) ∧ ∃□p

Labeling function:
label the subformulae true in each state.

Evaluating ∃pUq using least fixpoint

Iteration 0

Iteration 1

Iteration 2
\[(\exists c \exists pUq) \land \exists \Box p\]

Evaluating \(\exists c \exists pUq\)

Iteration 0

Evaluating \(\exists \Box p\) using greatest fixpoint

Iteration 0

Iteration 1

Result:
\[(\exists c \exists p \ U q) \land \exists p \]

Finally, evaluating \[(\exists c \exists p \ U q) \land \exists p \]

\[(\exists c \exists p \ U q, \exists c \exists p \ U p) \land \exists p \]

\[(\exists c \exists p \ U q, \exists c \exists p \ U p) \land \exists p \]

\[(\exists c \exists p \ U q, \exists c \exists p \ U p) \land \exists p \]

\[(\exists c \exists p \ U q, \exists c \exists p \ U p) \land \exists p \]

**CTL**

- model-checking problem complexities

- The PLTL model-checking problem is PSPACE-complete.
  - definition: Is there a run that satisfies the formula?
- The PLTL without ◯ (modal operator “next”) model-checking problem is NP-complete.
- The model-checking problem of CTL is PTIME-complete.
- The model-checking problem of CTL* is PSPACE-complete.

**CTL**

- symbolic model-checking with BDD

- System states are described with binary variables.
  - \(n\) binary variables \(\rightarrow 2^n\) states
  - \(x_1, x_2, \ldots, x_n\)
- we can use a BDD to describe legal states.
  - a Boolean function with \(n\) binary variables
  - \(\text{state}(x_1, x_2, \ldots, x_n)\)
**CTL - symbolic model-checking with Propositional logics**

*Example:*

\[ x_1 \ x_2 \ x_3 \]

\[
\begin{array}{c}
101 \\
001 \\
010
\end{array}
\]

\[
\text{state}(x_1, x_2, x_3) = (x_1 \land \neg x_2 \land x_3) \\
\lor (\neg x_1 \land x_2 \land x_3) \\
\lor (\neg x_1 \land x_2 \land \neg x_3)
\]

**CTL - symbolic model-checking with Propositional logics**

State transition relation as a logic function with \(2n\) parameters

\[
\text{transition}(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)
\]

\[
\begin{array}{c}
x_1, x_2, \ldots, x_n \\
y_1, y_2, \ldots, y_n
\end{array}
\]

**CTL - symbolic model-checking with Propositional logics**

Path relation also as a logic function with \(2n\) parameters

\[
\text{reach}(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n)
\]

\[
\begin{array}{c}
x_1, x_2, \ldots, x_n \\
y_1, y_2, \ldots, y_n
\end{array}
\]
CTL - symbolic model-checking with Propositional logics

\[ \text{reach}(x_1, x_2, x_3, y_1, y_2, y_3) = \]

\[
\begin{align*}
(x_1 \land \neg x_2 \land x_3 \land \neg y_1 \land y_2 \land y_3) \\
\lor (x_1 \land \neg x_2 \land x_3 \land \neg y_1 \land y_2 \land \neg y_3) \\
\lor (\neg x_1 \land \neg x_2 \land x_3 \land \neg y_1 \land y_2 \land y_3) \\
\lor (\neg x_1 \land \neg x_2 \land x_3 \land \neg y_1 \land y_2 \land \neg y_3) \\
\lor (\neg x_1 \land \neg x_2 \land \neg x_3 \land \neg y_1 \land y_2 \land y_3) \\
\lor (\neg x_1 \land \neg x_2 \land \neg x_3 \land \neg y_1 \land y_2 \land \neg y_3)
\end{align*}
\]

Symbolic safety analysis

- \( I \): initial condition with parameters \( x, x_2, \ldots, x_n \)
- \( \eta \): safe condition with parameters \( y_1, y_2, \ldots, y_n \)
- If \( I \land \neg \eta \land \text{reach}(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_n) \) is not false,
  - a risk state is reachable.
  - the system is not safe.

Symbolic safety analysis (backward)

Encode the states with variables \( x_0, x_1, \ldots, x_n \).
- the state set as a proposition formula: \( s(x_0, x_1, \ldots, x_n) \)
- the risk state set as \( r(x_0, x_1, \ldots, x_n) \)
- the initial state set as \( i(x_0, x_1, \ldots, x_n) \)
- the transition set as \( t(x_0, x_1, \ldots, x_n, x'_0, x'_1, \ldots, x'_n) \)

\[ b_0 = r(x_0, x_1, \ldots, x_n) \land s(x_0, x_1, \ldots, x_n); k = 1; \]

repeat
\[
\begin{align*}
b_k &= b_{k-1} \lor \exists x'_0 \exists x'_1 \ldots \exists x'_n t(x_0, \ldots, x_n, x'_0, x'_1, \ldots, x'_n) \land (b_{k-1} \uparrow) \\
k &= k + 1;
\end{align*}
\]

until \( b_k \equiv b_{k-1} \);

if \( (b_k \land i(x_0, x_1, \ldots, x_n)) \equiv \text{false} \), return ‘safe’; else return ‘risky’;

Kripke structure

- symbolic safety analysis

states: \( s(x, y, z) \equiv (\neg x \land \neg y \land \neg z) \lor (\neg x \land \neg y \land z) \lor (\neg x \land y \land \neg z) \lor (\neg x \land y \land z) \lor (\neg y \land x \land \neg z) \lor (\neg y \land x \land z) \lor (y \land x \land \neg z) \lor (y \land x \land z) \)

initial state: \( i(x, y, z) = x \land \neg y \land \neg z \)

risk state: \( r(x, y, z) \equiv x \land \neg y \land \neg z \)
Symbolic safety analysis (backward)

One assumption for the correctness!

- Two states cannot be with the same proposition labeling.
- Otherwise, the collapsing of the states may cause problem.

may need a few propositions for the names of the states.

Symbolic safety analysis (forward)

Encode the states with variables $x_0, x_1, \ldots, x_n$.

- the state set as a proposition formula: $s(x_0, x_1, \ldots, x_n)$
- the risk state set as $r(x_0, x_1, \ldots, x_n)$
- the initial state set as $i(x_0, x_1, \ldots, x_n)$
- the transition set as $t(x_0, x_1, \ldots, x_n, x', y, z)$

$f_0 = i(x_0, x_1, \ldots, x_n) \land s(x_0, x_1, \ldots, x_n); k = 1$

repeat

$f_k = f_{k-1} \lor (\exists x_0 \exists x_1 \exists x_n (t(x_0, x_1, \ldots, x_n, x', y, z) \land f_{k-1}))$

$k = k + 1$

until $f_k \equiv f_{k-1}$

if $(f_k \land r(x_0, x_1, \ldots, x_n)) \equiv \text{false}$, return 'safe'; else return 'risky';
Symbolic safety analysis (forward)

\[ f_0 = i(x,y,z) \equiv \neg x \land \neg y \land \neg z \]

\[ f_1 = f_0 \lor (\exists x \exists y \exists z(t(x,y,z,x',y',z') \land f_0)) \]
\[ = (\neg x \land \neg y \land \neg z) \lor (\exists x \exists y \exists z(t(x,y,z,x',y',z') \land \neg x \land \neg y \land \neg z)) \]
\[ = (\neg x \land \neg y \land \neg z) \lor (\exists x \exists y \exists z(\neg x' \land \neg y' \land \neg z' \land \neg x \land \neg y \land \neg z)) \]
\[ = (\neg x \land \neg y \land \neg z) \lor (\neg x' \land \neg y' \land \neg z') \]
\[ = (\neg x \land \neg y \land \neg z) \land (\neg x' \land \neg y' \land \neg z') \]

Symbolic liveness analysis

\[ f_0 = i(x,y,z) \equiv \neg x \land \neg y \land \neg z \]

\[ f_1 = f_0 \lor (\exists x \exists y \exists z(t(x,y,z,x',y',z') \land f_0)) \]
\[ = (\neg x \land \neg y \land \neg z) \lor (\exists x \exists y \exists z(t(x,y,z,x',y',z') \land \neg x \land \neg y \land \neg z)) \]
\[ = (\neg x \land \neg y \land \neg z) \lor (\exists x \exists y \exists z(\neg x' \land \neg y' \land \neg z' \land \neg x \land \neg y \land \neg z)) \]
\[ = (\neg x \land \neg y \land \neg z) \lor (\neg x' \land \neg y' \land \neg z') \]
\[ = (\neg x \land \neg y \land \neg z) \land (\neg x' \land \neg y' \land \neg z') \]

Bounded model-checking

Encode the states with variables \( x_{0,k}, x_{1,k}, \ldots, x_{n,k} \).

- the state set as a proposition formula: \( s(x_{0,k}, x_{1,k}, \ldots, x_{n,k}) \)
- the risk state set as \( r(x_{0,k}, x_{1,k}, \ldots, x_{n,k}) \)
- the initial state set as \( i(x_{0,0}, x_{1,0}, \ldots, x_{n,0}) \)
- the transition set as \( t(x_{0,k}, x_{1,k}, \ldots, x_{n,k}, x_{0,k+1}, x_{1,k+1}, \ldots, x_{n,k+1}) \)

\[ f_0 = i(x_{0,0}, x_{1,0}, \ldots, x_{n,0}) \land s(x_{0,0}, x_{1,0}, \ldots, x_{n,0}); k = 1; \]
repeat
\[ f_k = t(x_{0,k}, x_{1,k}, \ldots, x_{n,k}, x_{0,k+1}, x_{1,k+1}, \ldots, x_{n,k+1}) \land f_{k-1}; \]
\[ k = k + 1; \]
until \( f_k \land r(x_{0,k}, x_{1,k}, \ldots, x_{n,k}) \neq \text{false} \)

The value of \( x_i \) at state \( k \).

When to stop?
1. diameter of the state graph
2. explosion up to tens of steps

Symbolic liveness analysis

Encode the states with variables \( x_0, x_1, \ldots, x_n \).

- the state set as a proposition formula: \( s(x_0, x_1, \ldots, x_n) \)
- the non-liveness state set as \( b(x_0, x_1, \ldots, x_n) \)
- the initial state set as \( i(x_0, x_1, \ldots, x_n) \)
- the transition set as \( t(x_0, x_1, \ldots, x_n, x'_0, x'_1, \ldots, x'_n) \)

\[ b_0 = b(x_0, x_1, \ldots, x_n) \land s(x_0, x_1, \ldots, x_n); k = 1; \]
repeat
\[ b_k = b_{k-1} \land \exists x'_0 \exists x'_1 \ldots \exists x'_n(t(x_0, x_1, \ldots, x_n, x'_0, x'_1, \ldots, x'_n) \land b_{k-1}) \]
\[ k = k + 1; \]
until \( b_k \land i(x_0, x_1, \ldots, x_n) \neq \text{false} \);
if \( (b_k \land i(x_0, x_1, \ldots, x_n)) \equiv \text{false} \), return 'live'; else return 'not live';

change all unprimed variable in \( b_{k-1} \) to primed.
Kripke structure
- symbolic liveness analysis

states: s(x,y,z) \equiv (\neg x \land y \land \neg z) \lor (\neg x \land y \land z) \lor (x \land y \land z) \\
\lor (\neg x \land y) \lor (x \land y) \lor (x \land y) \\
\lor (\neg x) \lor (x \land y)

initial state: i(x,y,z) \equiv \neg x \land y \land \neg z

non-liveness state: b(x,y,z) \equiv (\neg x) \land (x \land y \land z)

Symbolic liveness analysis

b0 = b(x,y,z) \equiv (\neg x) \land (x \land y \land z)

b1 = b0 \lor \exists x \exists y \exists z' (T(x,y,z,x',y',z') \land b0')

= ((\neg x) \land (x \land y \land z))
\lor \exists x \exists y \exists z' (T(x,y,z,x',y',z') \land (\neg x') \land (x' \land y' \land z'))
\lor ((\neg x) \land (x \land y \land z))
\lor \exists x \exists y \exists z' (T(x,y,z,x',y',z') \land (\neg x') \land (x' \land y' \land z'))
\lor ((\neg x) \land (x \land y \land z))
\lor \exists x \exists y \exists z' (T(x,y,z,x',y',z') \land (\neg x') \land (x' \land y' \land z'))

b2 = b1 \lor \exists x \exists y \exists z' (T(x,y,z,x',y',z') \land b1')

= (\neg x \land y \land z) \land (\neg x' \land y \land z)

b3 = b2 \lor \exists x \exists y \exists z' (T(x,y,z,x',y',z') \land b2')

= (\neg x \land y \land z) \land (\neg x' \land y \land z)

transitions: T(x,y,z,x',y',z') \equiv

(\neg x \land y \land z \land \neg x' \land y' \land z') \lor (\neg x \land y \land z \land \neg x' \land y' \land z')
\lor (\neg x \land y \land z \land \neg x' \land y' \land z')
\lor (\neg x \land y \land z \land \neg x' \land y' \land z')

CTL
- symbolic model-checking algorithm

Assume program with rules r_1, r_2, \ldots, r_n

label(\varphi) \{
\text{case } p, \text{ return } p; \\
\text{case } \neg \varphi, \text{ return } \neg \text{label(}\varphi\text{);} \\
\text{case } \varphi \lor \psi, \text{ return } \text{label(}\varphi\text{)} \lor \text{label(}\psi\text{);} \\
\text{case } \exists \varphi, \text{ return } \lor_{i=1}^{n} \text{pred}(r_i, \text{label(}\varphi\text{);} \\
\text{case } \exists \psi_1 \text{ U } \psi_2, \text{ return } \text{lfp}(l \text{abel(}\psi_1\text{)}, \text{label(}\psi_2\text{)}); \\
\text{case } \exists \Box \varphi, \text{ return } \text{gfp}(\text{label(}\varphi\text{)}; \\
\}\
Symbolic model-checking
- with real-world programs

Consider guarded commands with modes (GCM)
Guard → Actions
- Guard is a propositional formula of state variables.
- Actions is a command of the following syntax.

GCM

\[ \text{Guard} = \text{a propositional formula of state variables.} \]
\[ \text{Actions} = \text{a command of the following syntax.} \]

\[
C ::= \text{ACT} | \{C\} | C \cdot C | \text{if } (B) \text{ C else } C | \text{while } (B) \text{ C} \\
\text{ACT} ::= ; | \text{goto name;} | x = E \\
\]

Guarded commands with modes (GCM)

1: \( w = 0; \)
2: \( x = 0; \)
3: \( y = z^2; \)
4: \( \text{while } (x < y) \{ \)
5: \( w = w + x^2; \)
6: \( x = x + 1; \)
7: \( \}
8: \( \text{if } (w > z^2 z^2) w = z^2 z^2; \)

A state-transition
- represented as a GCM

8 rules in total:

(a1) \( \rightarrow w = 0; \) goto a2;
(a2) \( \rightarrow x = 0; \) goto a3;
(a3) \( \rightarrow y = z^2; \) goto a4;
(a4) \( \land x \geq y \rightarrow \) goto a8;
(a4) \( \land x < y \rightarrow \) goto a5;
(a5) \( \rightarrow w = w + x^2; \) goto a6;
(a6) \( \rightarrow x = x + 1; \) goto a4;
(a8) \( \rightarrow \text{if } (w > z^2 z^2) w = z^2 z^2; \)

A state-transition
- represented as a GCM

\[
\begin{align*}
a_0 & \rightarrow \text{a1} \\
a_1 & \rightarrow (a1) \rightarrow w = 0; \\
a_2 & \rightarrow (a2) \rightarrow x = 0; \\
a_3 & \rightarrow (a3) \rightarrow y = z^2; \\
a_4 & \rightarrow (a4) \land x < y \rightarrow ; \\
a_5 & \rightarrow (a5) \rightarrow w = w + x^2; \\
a_6 & \rightarrow (a6) \rightarrow x = x + 1; \\
a_7 & \rightarrow (a7) \land x \geq y \rightarrow ; \\
a_8 & \rightarrow (a8) \rightarrow \text{if } (w > z^2 z^2) w = z^2 z^2; \\
a_0 & \rightarrow \end{align*}
\]
Transition relation from GCM rules.

Given a set of rules for $X=\{x,y,z\}$

\[ r_1: (x<y && y>2) \rightarrow y=x+y; x=3; \]
\[ r_2: (z>=2) \rightarrow y=x+1; z=0; \]
\[ r_3: (x<2) \rightarrow x=0; \]

\[ t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \]
\[ \equiv (x<y \land y>2 \land y'=x+y \land x'=3 \land z'=z) \]
\[ \lor (z>=2 \land y'=x+1 \land z'=0 \land x'=x) \]
\[ \lor (x<2 \land x'=0 \land y'=y \land z'=z) \]

On-the-fly precondition calculation with GCM rules.

Given a set of rules $r_1, r_2, \ldots, r_m$ of the form

\[ r_k: (\tau_k) \rightarrow y_{k,0}=d_0; y_{k,1}=d_1; \ldots; y_{k,n_k}=d_{n_k}; \]

\[ \exists x'_0 \exists x'_1 \ldots \exists x'_n (t(x_0,x_1,\ldots,x_n,x'_0,x'_1,\ldots,x'_n) \land (b_{k-1} \uparrow)) \]

directly with the GCM rules?

Yes, **on-the-fly state space construction.**

However, GCM rules are more complex than that.
On-the-fly precondition calculation with GCM rules.

Given a set of rules for \( X = \{x, y, z\} \)
\[
\begin{align*}
  r_1: & \quad (x<y && y>2) \Rightarrow y=z; \ x=3; \\
  r_2: & \quad (z>=2) \Rightarrow y=x+1; \ z=7; \\
  r_3: & \quad (x<2) \Rightarrow z=0;
\end{align*}
\]

\[
\exists x_0 \exists x_1 \ldots \exists x_n \exists x_{n}^'(t(x_0,x_1,\ldots,x_n,x_{n}^0,x_{n}^1,\ldots,x_{n}^n) \land (x<4 \land z>5))
\]

\[
\equiv (x<y \land y>2 \land \exists y \exists x (x<4 \land z>5 \land y==z \land x==3)) \\
\lor (z>=2 \land \exists y \exists z (x<4 \land z>5 \land y==x+1 \land z==7)) \\
\lor (x<2 \land \exists z (x<4 \land z>5 \land z==0))
\]

\[
\equiv (x<y \land y>2 \land z>5) \lor (z>=2 \land x<4) \lor (x<2 \land \exists z (false))
\]

\[
\equiv (x<y \land y>2 \land z>5) \lor (z>=2 \land x<4)
\]

What is \( \text{pre}(s, b) \)?

- \( \text{pre}(x = E; \ b) \equiv b[x/E] \)

  Ex 1. the precondition to \( x=x+z \);
  \[
  (x==y+2 \land x<4 \land z>5) \quad [x+x+z] \equiv x+z==y+2 \land x+z<4 \land z>5
  \]

  Ex 2. the precondition to \( x=5 \);
  \[
  (x==y+2 \land x<4 \land z>5) \quad [x+x+z] \equiv 5==y+2 \land 5<4 \land z>5
  \]

  Ex 3. the precondition to \( x=2*x+1 \);
  \[
  (x==y+2 \land x<4 \land z>5) \quad [x+x+z] \equiv 2*x+1==y+2 \land 2*x+1<4 \land z>5
  \]

On-the-fly precondition calculation with GCM rules.

Given a set of rules \( r_1, r_2, \ldots, r_m \) of the form
\[
r_k: (\tau_k) \Rightarrow s_k;
\]

What is \( \text{pre}(s, b) \)?

- \( \text{pre}(x = E; \ b) \equiv b[x/E] \)

  Ex. the precondition to \( x=x+z \);
  \[
  (x==y+2 \land x<4 \land z>5) \quad [x+x+z] \equiv x+z==y+2 \land x+z<4 \land z>5
  \]

  \[
  \text{pre}(s_1s_2, b) \equiv \text{pre}(s_1, \ \text{pre}(s_2, b))
  \]

  \[
  \text{pre}(\text{if}(B) \ s_1\text{else} \ s_2) \equiv (B \land \text{pre}(s_1, b)) \lor (\neg B \land \text{pre}(s_2, b))
  \]

  \[
  \text{pre}(\text{while}(B) \ s, b) \equiv \ldots
  \]
On-the-fly precondition calculation with GCM rules.

Given a set of rules $r_1, r_2, \ldots, r_m$ of the form

$r_k$: $(\tau_k) \rightarrow s_k$

What is $\text{pre}(s, b)$?

$\text{pre}(\text{while } (B) s, b) \equiv \text{formula } L_1 \vee L_2$ for

$L_1$: those states that reach $\neg B \wedge b$ with finite steps of $s$ through states in $B$; and

$L_2$: those states that never leave $B$ with steps of $s$.

---

On-the-fly precondition calculation with GCM rules.

$L_1$: those states that reach $\neg B \wedge b$ with finite steps of $s$ through states in $B$

$w_0 = \neg B \wedge b; k = 1$;

repeat

$w_k = w_{k-1} \vee (B \wedge \text{pre}(s, w_{k-1}))$;
$k = k + 1$;
until $w_k \equiv w_{k-1}$;
return $w_k$;

Also a least fixpoint procedure

---

Precondition to $b$ through while $(B) s$;

Example: $b \equiv x == 2 \wedge y == 3$

while ( $x < y$ ) $x = x + 1$;

$L_1$ computation.

$w_0 = x >= y \wedge x == 2 \wedge y == 3 \equiv \text{false}$ ; $k = 1$;

$w_1 \equiv \text{false} \lor (x < y \wedge \text{pre}(x = x + 1, \text{false}))$

$\equiv \text{false} \lor (x < y \wedge \text{false})$

$\equiv \text{false}$;

---

On-the-fly precondition calculation with GCM rules.

$L_2$: those states that never leave $B$ with steps of $s$.

$w_0 = B; k = 1$;

repeat

$w_k = w_{k-1} \wedge \text{pre}(s, w_{k-1})$;
$k = k + 1$;
until $w_k \equiv w_{k-1}$;
return $w_k$;

A greatest fixpoint procedure
Precondition to b through while (B) s;

Example:

while ( x<y && x>0) x = x+1;

L2 computation.

w_0 ≡ x<y ∧ x>0 ; k = 1;

w_1 ≡ x<y ∧ x>0 ∧ pre(x=x+1, x<y ∧ x>0)
≡ x<y ∧ x>0 ∧ x+1<y ∧ x+1>0
≡ x>0 ∧ x+1<y

w_2 ≡ x+1<y ∧ x>0 ∧ pre(x=x+1, x+1<y ∧ x>0)
≡ x+1<y ∧ x+2<y ∧ x+2>0
≡ x>0 ∧ x+1<y

non-terminating for algorithms and protocols!

Precondition to b through while (B) s;

Example:

while ( x>y && x>0) x = x+1;

L2 computation.

w_0 ≡ x>y ∧ x>0 ; k = 1;

w_1 ≡ x>y ∧ x>0 ∧ pre(x=x+1, x>y ∧ x>0)
≡ x>y ∧ x+1>y ∧ x+1>0
≡ x>y ∧ x>0

terminating for algorithms and protocols!

Precondition to b through while (B) s;

Example: b ≡ x==2 ∧ y==3

while (x>y && x>0) x = x+1;

L1 computation.

w_0 ≡ x==y ∧ x<=0) ∧ x==2 ∧ y==3 ≡ x==2 ∧ y==3;

w_1 ≡ (x==2 ∧ y==3) ∨ (x>y ∧ x>0 ∧ pre(x=x+1, x==2 ∧ y==3));
≡ (x==2 ∧ y==3) ∨ (x>y ∧ x>0 ∧ x==1 ∧ y==3);
≡ (x==2 ∧ y==3) ∨ false
≡ x==2 ∧ y==3

Symbolic weakest precondition

Assume program with rules

- x=3 ∧ y=6 → x:=2; z:=7;
- x, y, z are discrete variables with range declarations

What is the weakest precondition of η for those states before the transitions?
Symbolic weakest precondition

Assume program with rules:
- \( r: x=3 \land y=6 \rightarrow x:=2; z:=7; \)

What is the weakest precondition of \( \eta \) for those states before the transitions?

\[
pre(r, \eta) \overset{\text{def}}{=} x=3 \land y=6 \land \exists x \exists z (x=2 \land z=7 \land \eta)
\]

Symbolic safety analysis
- with Kripke structures as programs

Assume program with rules \( r_1, r_2, \ldots, r_n \)

What characterizes all states that can reach \( \neg \eta \)?

\[
lfp (\phi, \psi) \forall \exists \phi \psi/\{ \\
Z' := false; Z := \psi; \\
\text{while } (Z \neq Z') \{ \\
Z' := Z; \\
Z := Z \lor (\phi \land \lor_{i=n} \text{pred}(r_i, Z)); \\
\}
\}
\]

Symbolic liveness analysis
- with Kripke structures as programs

Assume program with rules \( r_1, r_2, \ldots, r_n \)

What is the characterization of all states that may not reach \( \eta \)?

\[
gfp (\phi) \forall \exists \phi/\{ \\
Z' := false; Z := \phi; \\
\text{while } (Z \neq Z') \{ \\
Z' := Z; \\
Z := \phi \land \lor_{i=n} \text{pred}(r_i, Z); \\
\}
\}
\]

Bisimulation Framework

Answer

Yes if the model is equivalent to the specification

No if not.
Bisimulation-checking

- \( K = (S, S_0, R, AP, L) \)
  \( K' = (S', S'_0, R', AP, L') \)
- Note \( K \) and \( K' \) use the same set of atomic propositions \( AP \).
- \( B \in S \times S' \) is a bisimulation relation between \( K \) and \( K' \) iff for every \( B(s, s') \):
  - \( L(s) = L'(s') \) (BSIM 1)
  - If \( R(s, s_1) \), then there exists \( s_1' \) such that \( R'(s', s_1') \) and \( B(s_1, s_1') \). (BISIM 2)
  - If \( R'(s', s_2') \), then there exists \( s_2 \) such that \( R(s, s_2) \) and \( B(s_2, s_2') \). (BISIM 3)

Bisimulations
Bisimulations

Unwinding preserves bisimulation
Examples

Examples

Bisimulations

The Preservation Property.

- $K = (S, S_0, R, AP, L)$
- $K' = (S', S_0', R', AP, L')$
- $K$ and $K'$ are bisimilar (bisimulation equivalent) iff there exists a bisimulation relation $B \subseteq S \times S'$ between $K$ and $K'$ such that:
  - For each $s_0$ in $S_0$ there exists $s_0'$ in $S_0'$ such that $B(s_0, s_0')$.
  - For each $s_0'$ in $S_0'$ there exists $s_0$ in $S_0$ such that $B(s_0, s_0')$.

- $K = (S, S_0, R, AP, L)$
- $K' = (S', S_0', R', AP, L')$
- $B \subseteq S \times S'$, a bisimulation.
- Suppose $B(s, s')$.
- FACT: For any CTL formula $\psi$ (over $AP$), $K, s \models \psi$ iff $K', s' \models \psi$.
- If $K'$ is smaller than $K$ this is worth something.
**Simulation Framework**

- **Design Implementation**
- **Model Construction**
- **Model Checker**

**Answer**
- Yes if the model satisfies the specification
- No if not.

**Simulation-checking**

- \( K = (S, S_0, R, AP, L) \)
- \( K' = (S', S'_0, R', AP, L') \)
- Note \( K \) and \( K' \) use the same set of atomic propositions \( AP \).
- \( B \in S \times S' \) is a simulation relation between \( K \) and \( K' \) iff for every \( B(s, s') \):
  - \( L(s) = L'(s') \) (BSIM 1)
  - If \( R(s, s_1) \), then there exists \( s_1' \) such that \( R'(s', s_1') \) and \( B(s_1, s_1') \). (BISIM 2)

**Simulations**

- \( K = (S, S_0, R, AP, L) \)
- \( K' = (S', S'_0, R', AP, L') \)
- \( K \) is simulated by (implements or refines) \( K' \) iff there exists a simulation relation \( B \subseteq S \times S' \) between \( K \) and \( K' \) such that for each \( s_0 \) in \( S_0 \) there exists \( s_0' \) in \( S'_0 \) such that \( B(s_0, s_0') \).

**Bisimulation Quotients**

- \( K = (S, S_0, R, AP, L) \)
- There is a maximal simulation \( B \subseteq S \times S \).
  - Let \( R \) be this bisimulation.
  - \([s] = \{s' \mid s \ R s'\}\).
- \( R \) can be computed “easily”.
- \( K' = K / R \) is the bisimulation quotient of \( K \).
Bisimulation Quotient

- $K = (S, S_0, R, AP, L)$
- $[s] = \{s' \mid s R s'\}$.
- $K' = K / R = (S', S'_0, R', AP, L')$.
  - $S' = \{[s] \mid s \in S\}$
  - $S'_0 = \{[s_0] \mid s_0 \in S_0\}$
  - $R' = \{([s], [s']) \mid R(s_1, s_1') , s_1 \in [s], s_1' \in [s']\}$
  - $L'([s]) = L(s)$.
Facts About a (Bi)Simulation

- The empty set is always a (bi)simulation
- If R, R' are (bi)simulations, so is R U R'
- Hence, there always exists a maximal (bi)simulation:
  - Checking if DB_1=DB_2: compute the maximal bisimulation R, then test (root(DB_1),root(DB_2)) in R

198

Kripke structure - simulation-checking

/* Given model A = (S, S_0, R, L), spec. A'=(S', S'_0, R', L') */
Simulation-checking(A,A') /* using greatest fixpoint algorithm */ {
  Let B={(s,s') | s∈S, s'∈S', L(s)=L'(s')} ;
  repeat {
    B = B - {(s,s') | (s,s')∈B, ∃(s,t)∈R ∀(s',t')∈R'((t,t')∉B)};
  } until no more changes to B.
  if there is an s_0∈S_0 with ∀s'_0∈S'_0((s_0,s'_0)∉ B),
    return 'no simulation,'
  else return 'simulation exists.'
}
The procedure terminates since B is finite in the Kripke structure.

199

Kripke structure - bisimulation-checking

/* Given model A = (S, S_0, R, L), spec. A'=(S', S'_0, R', L') */
Bisimulation-checking(A,A') /* using greatest fixpoint algorithm */ {
  Let B={(s,s') | s∈S, s'∈S', L(s)=L'(s')} ;
  repeat {
    B = B - {(s,s') | (s,s')∈B, ∃(s,t)∈R ∀(s',t')∈R'((t,t')∉B)};
    B = B - {(s,s') | (s,s')∈B, ∃(s',t')∈B ∀(s,t)∈R((t,t')∉B)};
  } until no more changes to B.
  if there is an s_0∈S_0 with ∀s'_0∈S'_0((s_0,s'_0)∉ B),
    return 'no simulation,'
  else return 'simulation exists.'
}

200

(Bi)Simulation - complexities

- Bisimulation: O((m+n)log(m+n))
- Simulation: O(m n)
- In contrast, finding a graph homeomorphism is NP-complete.

201
Symbolic simulation-checking

- Encode the states with variables $x_0, x_1, \ldots, x_n$ (for the model) and $y_0, y_1, \ldots, y_m$ (for the spec.)
- Usually there are shared variables between $\{x_0, x_1, \ldots, x_n\}$ and $\{y_0, y_1, \ldots, y_m\}$. $L(s) = L'(s')$ means that the shared variables are of the same values.

- the state sets as proposition formulas:
  - $s(x_0, x_1, \ldots, x_n) \& s(y_0, y_1, \ldots, y_m)$
- the initial state set as
  - $i(x_0, x_1, \ldots, x_n) \& i'(y_0, y_1, \ldots, y_m)$
- the transition set as
  - $R(x_0, x_1, \ldots, x_n, x'_0, x'_1, \ldots, x'_n) \& R'(y_0, y_1, \ldots, y_n, y'_0, y'_1, \ldots, y'_m)$

Symbolic simulation-checking - an example

- $s(x, y) \equiv \text{true}$, $s'(x, y, z) \equiv \neg y \lor (\neg x \land y \land z)$
- $i(x, y) \equiv \neg x \land \neg y$, $i'(x, y, z) \equiv \neg x \land \neg y \land \neg z$
- $R(x, y, x', y') \equiv \ldots$, $R'(x, y, z, x', y', z') \equiv \ldots$

Symbolic simulation-checking

- $B_0 = \bigwedge_{i=1, i\neq i'} L(x_0, x_1, \ldots, x_n) \land L(y_0, y_1, \ldots, y_m) \land L'(x_0, x_1, \ldots, x_n) \land L'(y_0, y_1, \ldots, y_m)$
- for $(k = 1, B_1 = \text{false}; B_k \neq B_{k-1}, k = k + 1)$
  - $B_k = B_{k-1} \land \exists x_0 \exists x'_1 \ldots \exists x'_n (R(x_0, x_1, \ldots, x_n, x'_0, x'_1, \ldots, x'_n) \land \neg \exists y_0 \exists y'_1 \ldots \exists y'_m (R'(y_0, y_1, \ldots, y_n, y'_0, y'_1, \ldots, y'_m) \land (B_{k-1} \uparrow))$;
  - change all unprimed variable in $B_{k-1}$ to primed.

Symbolic simulation-checking - an example

- $R(x, y, x', y') \equiv (\neg x \land \neg y \land \neg x' \land y') \lor (\neg x \land y \land x' \land \neg y') \lor (x \land y \land x' \lor y')$
Symbolic simulation-checking
- an example

$B_0 = s(x,y) \& s'(x,y,z) = \neg z \vee (\neg x \& y \& z)$

$B_1 = (\neg z \vee (\neg x \& y \& z)) \land \neg \exists x' \exists y' \exists z' (\neg x \& y \& z) \land (\neg x \& y \& z' \land (\neg x \& y' \& z') \land (\neg x \& y \land z'))

$\land \neg \exists x' \exists y' \exists z' (\neg x \& y \& z) \land (\neg x \& y \& z') \land (\neg x \& y' \& z') \land (\neg x \& y \land z')$

$\land (\neg z \vee (\neg x \& y \& z))$)

$= (\neg z \vee (\neg x \& y \& z)) \land \neg \exists x' \exists y' \exists z' ((\neg x \& y \& z) \land (\neg x \& y' \& z') \land (\neg x \& y \land z'))$

$\land (\neg x \& y \land z))$

Symbolic simulation-checking
- an example

$B_2 = ((\neg x \& y \& z) \vee (x \& x' \& y) \vee (x \& x' \& z) \vee (x \& y \& x' \& y') \vee (x \& y \& x' \& z))$

$\land \neg \exists x' \exists y' \exists z' (\neg x \& y \& z) \land (\neg x \& y \& z') \land (\neg x \& y' \& z') \land (\neg x \& y \land z')$

$\land (\neg z' \vee (\neg x \& y \& z))$)

$= ((\neg x \& y \& z) \vee (x \& x' \& y) \vee (x \& x' \& z) \vee (x \& y \& x' \& y') \vee (x \& y \& x' \& z))$

$\land \neg \exists x' \exists y' \exists z' (\neg x \& y \& z) \land (\neg x \& y \& z') \land (\neg x \& y' \& z') \land (\neg x \& y \land z')$

$\land (\neg z' \vee (\neg x \& y \& z))$)

$= (x \& y \& z) \vee (x \& y \& z)$

Here, the initial statepair has been eliminated.