Fundamentals of Programming Languages II

Model Checking

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Final Exam Policy

We will choose reports, instead of final exam!

Final Exam Policy

We will choose reports, instead of final exam!

A list of reports will be announced gradually throughout the course.

Final Exam Policy

We will choose reports, instead of final exam!

A list of reports will be announced gradually throughout the course.

Each report has maximal people requirements, first apply first achieve.

Final Exam Policy

We will choose reports, instead of final exam!

A list of reports will be announced gradually throughout the course.

Each report has maximal people requirements, first apply first achieve.

You can submit the report at any time before the end of 18th weekend.
Each one has at most TWO submission chance.

Final Exam Policy

We will choose reports, instead of final exam!

A list of reports will be announced gradually throughout the course.

Each report has maximal people requirements, first apply first achieve.

You can submit the report at any time before the end of 18th weekend. Each one has at most TWO submission chance.

You need to read at least THREE relative references, which should be listed at the end of the report.

Final Exam Policy

We will choose reports, instead of final exam!

A list of reports will be announced gradually throughout the course.

Each report has maximal people requirements, first apply first achieve.

You can submit the report at any time before the end of 18th weekend. Each one has at most TWO submission chance.

You need to read at least THREE relative references, which should be listed at the end of the report.

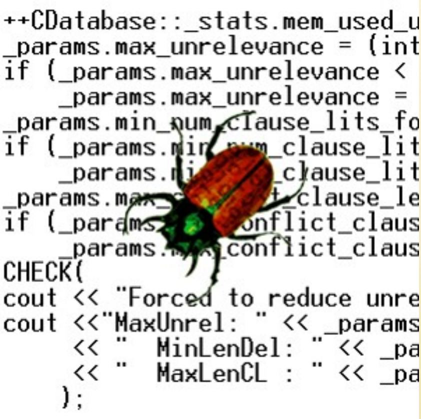
A sample section titles of a report:

- Introduction
- The problem description
- Key theorems/techniques/algorithms
- An application
- Conclusion

Assignment

Assignment 1 is announced!

Bugs in Software



Testing VS. Verification

Testing!

Testing VS. Verification

Testing!

Whenever no bugs are detected by testing, we cannot claim that there are NO bugs!

Testing VS. Verification

Testing!

Whenever no bugs are detected by testing, we cannot claim that there are NO bugs!

Hence, testing is a sound methodology, but not a complete one.

Testing VS. Verification

Testing!

Whenever no bugs are detected by testing, we cannot claim that there are NO bugs!

Hence, testing is a sound methodology, but not a complete one.

Can we gain a complete methodology?

Testing VS. Verification

Testing!

Whenever no bugs are detected by testing, we cannot claim that there are NO bugs!

Hence, testing is a sound methodology, but not a complete one.

Can we gain a complete methodology? The answer is YES!

Testing VS. Verification

Testing!

Whenever no bugs are detected by testing, we cannot claim that there are NO bugs!

Hence, testing is a sound methodology, but not a complete one.

Can we gain a complete methodology? The answer is YES!

This is so called formal verification.

Formal Verifications

Here are many formal verification techniques:

Formal Verifications

Here are many formal verification techniques:

- model checking
- theorem proving
- type systems
- SAT, SMT, and string solving …

Model Checking

Q: What is model checking?

Model Checking

Q: What is model checking?

Basically, model checking is a (non-trivial) search problem over a (non-trivial) data structure.

Model Checking

Q: What is model checking?

Basically, model checking is a (non-trivial) search problem over a (non-trivial) data structure.

Sometimes it is called algorithmic formal verification.

Search Problem

Search Problem

Binary Search

Search Problem

Binary Search

Search on Trees

Search Problem

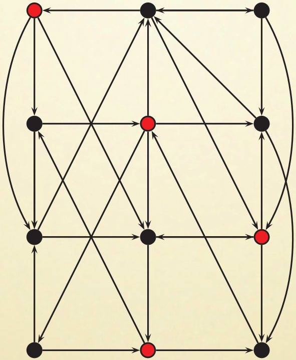
Binary Search

Search on Trees

Search on Graphs

The First Question

The First Question



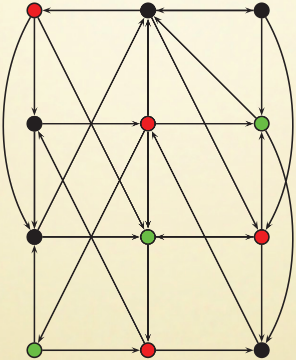
Safety as Reachability

Safety as Reachability

Bad things will never happen!

The Second Question

The Second Question



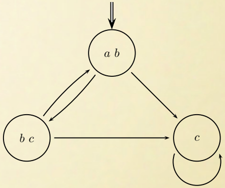
Liveness

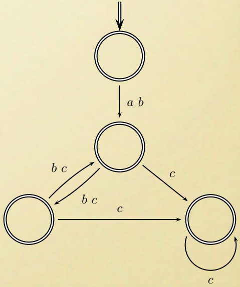
Liveness

Good things will eventually happen!

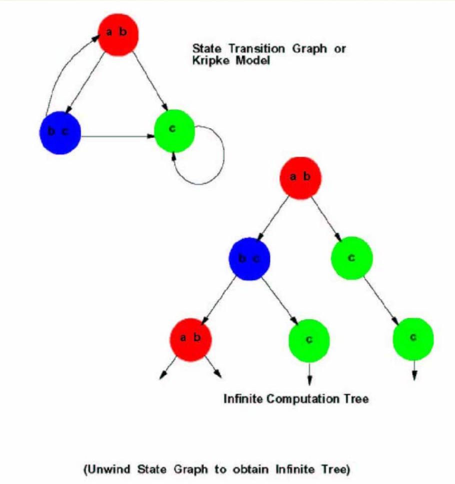
Data Structures

- Kripke structure: $M = (S, S_0, R, L)$
  - S, finite set of state
  - $S_0 \subseteq S$, initial state
  - $R \subseteq S \times S$, transition relations
  - $L : S \rightarrow 2^{AP}$, status label function
    (AP: atomic propositions)
- Finite automata: $\mathcal{A} = (\Sigma, Q, Q_0, F, \delta)$
  - A, finite set of input alphabet
  - Q, finite set of control location
  - $Q_0 \subseteq Q$, initial control locations
  - $F \subseteq Q$, final control locations
  - $\delta \subseteq Q \times \Sigma \times Q$, transitions

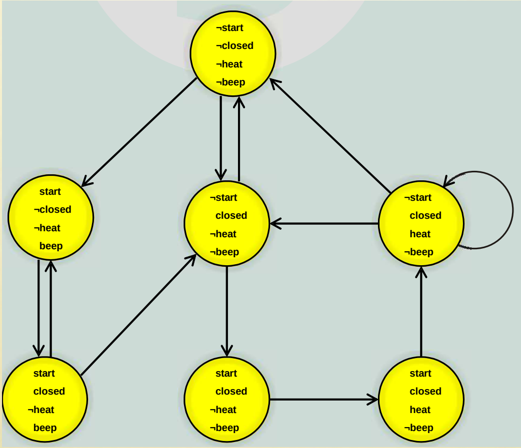




Finite Systems Vs. Infinite Computation Tree

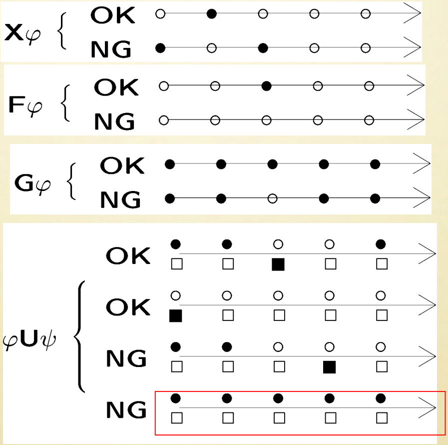


An Microwave Oven Example



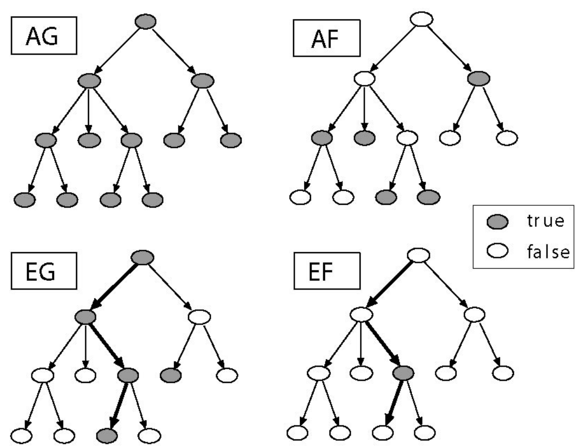
Logic-Based MC: Temporal Operators

- Next
- Finally
- Globally
- Until



● : φ, ■ : ψ, ○ : $\neg\varphi$, □ : $\neg\psi$

Logic-Based MC: Path Operators, *A*, *E*



- *AG*: safety, bad things will never happen.
- *AF*: liveness, good things will eventually happen.

State Formula & path formulas

Let AP be the set of atomic proposition names. The syntax of state formulas is given by the following rules:

- If $p \in AP$, then p is a state formula.
- If f and g are state formulas, then $\neg f$, $f \vee g$ and $f \wedge g$ are state formulas.
- If f is a path formula, then $E f$ and $A f$ are state formulas.
- If f is a state formula, then f is also a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $X f$, $F f$, $G f$ and $f\ U\ g$ are path formulas.

Formal Description

1.	$M, s \models p$	$\Leftrightarrow$	$p \in L(s)$.
2.	$M, s \models \neg f_1$	$\Leftrightarrow$	$M, s \not\models f_1$.
3.	$M, s \models f_1 \vee f_2$	$\Leftrightarrow$	$M, s \models f_1$ or $M, s \models f_2$.
4.	$M, s \models f_1 \wedge f_2$	$\Leftrightarrow$	$M, s \models f_1$ and $M, s \models f_2$.
5.	$M, s \models \mathbf{E}\ g_1$	$\Leftrightarrow$	there is a path π from s such that $M, \pi \models g_1$.
6.	$M, s \models \mathbf{A}\ g_1$	$\Leftrightarrow$	for every path π starting from s, $M, \pi \models g_1$.
7.	$M, \pi \models f_1$	$\Leftrightarrow$	s is the first state of π and $M, s \models f_1$.
8.	$M, \pi \models \neg g_1$	$\Leftrightarrow$	$M, \pi \not\models g_1$.
9.	$M, \pi \models g_1 \vee g_2$	$\Leftrightarrow$	$M, \pi \models g_1$ or $M, \pi \models g_2$.
10.	$M, \pi \models g_1 \wedge g_2$	$\Leftrightarrow$	$M, \pi \models g_1$ and $M, \pi \models g_2$.
11.	$M, \pi \models \mathbf{X}\, g_1$	$\Leftrightarrow$	$M, \pi^1 \models g_1$.
12.	$M, \pi \models \mathbf{F}\, g_1$	$\Leftrightarrow$	there exists a $k \geq 0$ such that $M, \pi^k \models g_1$.
13.	$M, \pi \models \mathbf{G}\, g_1$	$\Leftrightarrow$	for all $i \geq 0$, $M, \pi^i \models g_1$.
14.	$M, \pi \models g_1\, \mathbf{U}\, g_2$	$\Leftrightarrow$	there exists a $k \geq 0$ such that $M, \pi^k \models g_2$ and for all $0 \leq j < k$, $M, \pi^j \models g_1$.

Example Specification

- $EF(\texttt{Start} \wedge \neg \texttt{Ready})$
- $AG(\texttt{Req} \rightarrow AF\ \texttt{Ack})$
- $AG(AF\ \texttt{DeviceEnabled})$
- $AG(EF\ \texttt{Restart})$

Example Specification

- $EF(\texttt{Start} \wedge \neg \texttt{Ready})$
  - It is possible to get to a state where `Start` holds but `Ready` does not hold.
- $AG(\texttt{Req} \to AF\,\texttt{Ack})$
- $AG(AF\,\texttt{DeviceEnabled})$
- $AG(EF\,\texttt{Restart})$

Example Specification

- $EF(\texttt{Start} \wedge \neg \texttt{Ready})$
  - It is possible to get to a state where `Start` holds but `Ready` does not hold.
- $AG(\texttt{Req} \rightarrow AF\ \texttt{Ack})$
  - If a request occurs, then it will be eventually acknowledged.
- $AG(AF\ \texttt{DeviceEnabled})$
- $AG(EF\ \texttt{Restart})$

Example Specification

- $EF(\texttt{Start} \land \neg \texttt{Ready})$
  - It is possible to get to a state where `Start` holds but `Ready` does not hold.
- $AG(\texttt{Req} \to AF\,\texttt{Ack})$
  - If a request occurs, then it will be eventually acknowledged.
- $AG(AF\,\texttt{DeviceEnabled})$
  - The proposition `DeviceEnabled` holds infinitely often on every computation path.
- $AG(EF\,\texttt{Restart})$

Example Specification

- $EF(\texttt{Start} \wedge \neg \texttt{Ready})$
  - It is possible to get to a state where `Start` holds but `Ready` does not hold.
- $AG(\texttt{Req} \rightarrow AF\,\texttt{Ack})$
  - If a request occurs, then it will be eventually acknowledged.
- $AG(AF\,\texttt{DeviceEnabled})$
  - The proposition `DeviceEnabled` holds infinitely often on every computation path.
- $AG(EF\,\texttt{Restart})$
  - From any state it is possible to get to the `Restart`.

Example Specification

- $AG\,(\texttt{request} \rightarrow F\,\texttt{grant})$
- $AG(\neg(\neg\texttt{request}\ U\ \texttt{grant}))$
- $AGF\ \texttt{request}$

Example Specification

- $AG\,(\texttt{request} \rightarrow F\ \texttt{grant})$
  - each `request` will be finally `grant(ed)`.
- $AG(\neg(\neg\texttt{request}\ U\ \texttt{grant}))$
- $AGF\ \texttt{request}$

Example Specification

- $AG\,(\texttt{request} \rightarrow F\,\texttt{grant})$
  - each `request` will be finally `grant(ed)`.
- $AG(\neg(\neg\texttt{request}\; U\; \texttt{grant}))$
  - each `grant` follows some `request`.
- $AGF\;\texttt{request}$

Example Specification

- $AG\,(\texttt{request} \rightarrow F\ \texttt{grant})$
  - each `request` will be finally `grant(ed)`.
- $AG(\neg(\neg\texttt{request}\ U\ \texttt{grant}))$
  - each `grant` follows some `request`.
- $AGF\ \texttt{request}$
  - `request` occurs infinitely often.

CTL and LTL

- CTL: temporal operators must be immediately followed by path quantifiers.
    - e.g., $AF\varphi, EG\varphi, AXEG\varphi, EXA(\varphi U \psi)$
- LTL: path quantifiers are allowed only at the outermost position.
    - e.g., $AGF\varphi, EX(\varphi U \psi), A(F\varphi \vee G\psi)$
- Except for fairness, most properties are expressed in CTL $\cap$ LTL.

CTL Vs. LTL

- LTL can, but CTL cannot.
- CTL can, but LTL cannot.

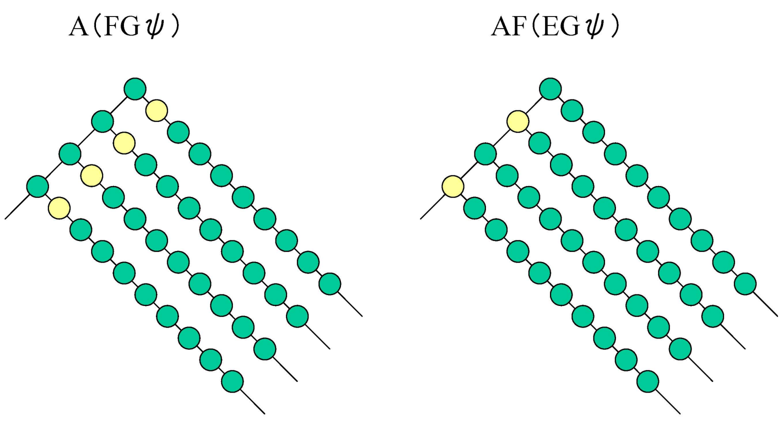
CTL Vs. LTL

- LTL can, but CTL cannot.
  - $A(GF\varphi)$ infinitely often (fairness)
  - $A(FG\varphi)$ almost everywhere
- CTL can, but LTL cannot.

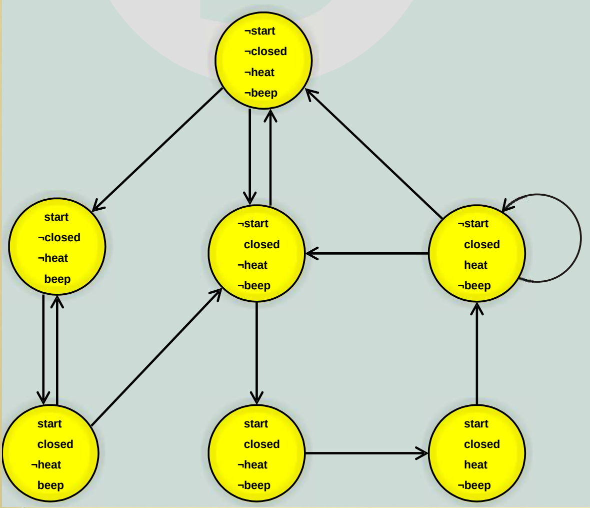
CTL Vs. LTL

- LTL can, but CTL cannot.
  - $A(GF\varphi)$ infinitely often (fairness)
  - $A(FG\varphi)$ almost everywhere
- CTL can, but LTL cannot.
  - $AG(EF\varphi)$
  - $AF(EG\varphi)$

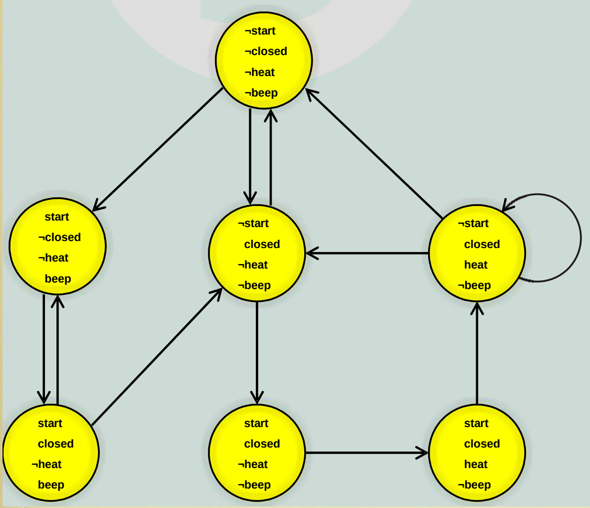
CTL Vs. LTL



Fairness

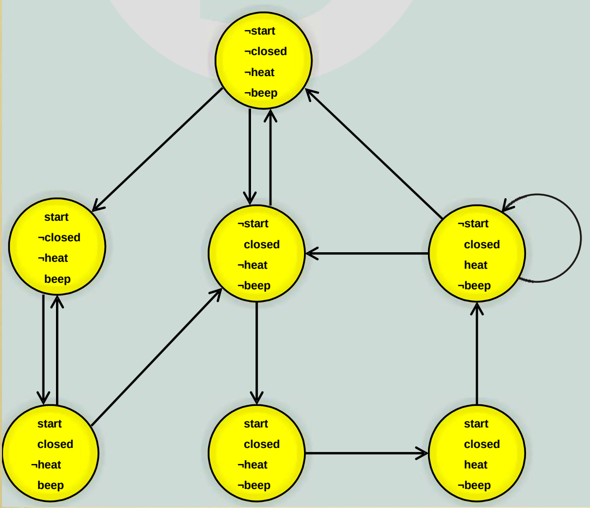

¬start ¬closed ¬heat ¬beep
start ¬closed ¬heat beep
¬start closed ¬heat ¬beep
¬start closed heat ¬beep
start closed ¬heat beep
start closed ¬heat ¬beep
start closed heat ¬beep

Fairness



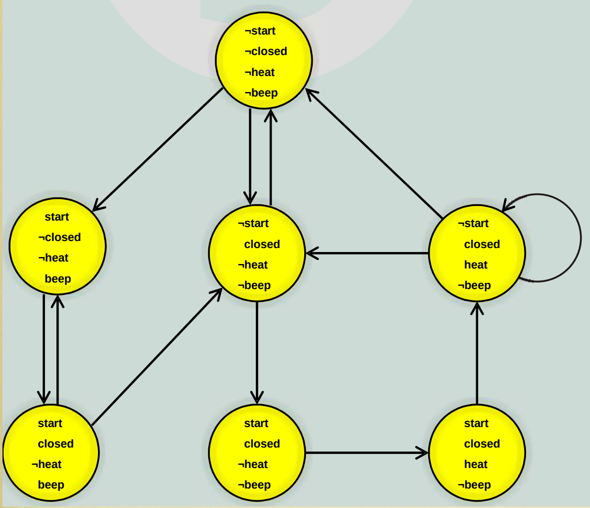
- $AG(start \rightarrow AF\ heat)$

Fairness



- $AG(start \rightarrow AF\ heat)$
  - NG!

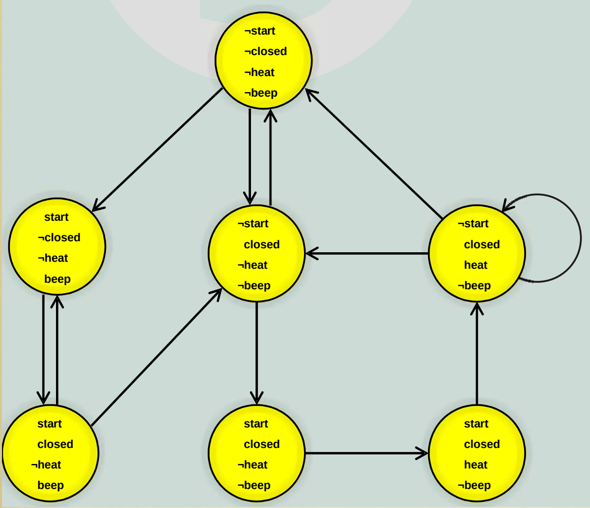
Fairness



- $AG(start \rightarrow AF\ heat)$
  - NG!
- Constraint: $AGF start \land close \land \neg beep$

  (operate correctly infinitely often)

Fairness



- $AG(start \rightarrow AF\ heat)$
  - NG!
- Constraint: $AGF start \wedge close \wedge \neg beep$ (operate correctly infinitely often)
- $AG(start \rightarrow AF\ heat)$
  - OK!

Fairness

- More Examples...
  - Protocols operated over reliable channels, to check no message is ever transmitted but never received.
  - Scheduler that schedules released tasks, to check all released tasks will be finally scheduled.
- How to check fairness
  - LTL: $A(GF\varphi)$

    e.g. $AG(start \rightarrow AF\ heat)\ \wedge\ A(GF\ start \wedge close \wedge \neg beep)$
  - CTL: NG!

Quiz I: Crossing River

- Group {Man, Sheep, Wolf, Cabbage} trying across river.
- Constraints:

Quiz I: Crossing River

- Group {Man, Sheep, Wolf, Cabbage} trying across river.
- Constraints:
    - Man can carry one item at a time by boat.
    - If Sheep and Wolf only, Wolf will eat Sheep.
    - If Sheep and Cabbage only, Sheep will eat Cabbage.

Quiz I: Crossing River

- Group {Man, Sheep, Wolf, Cabbage} trying across river.
- Constraints:
  - Man can carry one item at a time by boat.
  - If Sheep and Wolf only, Wolf will eat Sheep.
  - If Sheep and Cabbage only, Sheep will eat Cabbage.
- Find way by model checking!

Quiz II. Hamilton Path

- Find out whether a graph occurs a Hamilton path.

Quiz II. Hamilton Path

- Find out whether a graph occurs a Hamilton path.
- $M, q_0 \models E(F\, p_1 \wedge \ldots \wedge F\, p_n \wedge G(P_1 \rightarrow X\, G\, \neg p_1) \wedge \ldots \wedge G(P_n \rightarrow X\, G\, \neg p_n))$
- $M = (\{q_0, q_f\} \cup V(G), \{q_0\}, \{(q_0, v), (v, q_f), (q_f, q_f) \mid v \in V(G)\} \cup E(G), L)$
- $L(v_i) = \{p_i\}$

CTL Model Checking

CTL Formula

- AX and EX
- AF and EF
- AG and EG
- AG and EG

Properties

$$AX\phi = \neg EX(\neg\phi)$$
$$EF\phi = E(\mathit{True}\ U\ \phi)$$
$$AG\phi = \neg EF(\neg\phi)$$
$$AF\phi = \neg EG(\neg\phi)$$
$$A(\phi\ U\ \psi) = \neg E[\neg\psi\ U\ (\neg\phi \wedge \neg\psi)] \wedge \neg EG\neg\phi$$

Properties

$$
\begin{aligned}
AX\phi &= \neg EX(\neg\phi) \\
EF\phi &= E(True\ U\ \phi) \\
AG\phi &= \neg EF(\neg\phi) \\
AF\phi &= \neg EG(\neg\phi) \\
A(\phi\ U\ \psi) &= \neg E[\neg\psi\ U\ (\neg\phi \wedge \neg\psi)] \wedge \neg EG\neg\phi
\end{aligned}
$$

- EX, EG, EU are enough!

EX

EX

- Trivial!

EU

EU

```
procedure CheckEU(f1, f2)
    T := { s | f2 ∈ label(s) };
    for all s ∈ T do label(s) := label(s) ∪ { E[f1 U f2] };
    while T ≠ Ø do
        choose s ∈ T;
        T := T \ {s};
        for all t such that R(t, s) do
            if E[f1 U f2] ∉ label(t) and f1 ∈ label(t) then
                label(t) := label(t) ∪ { E[f1 U f2] };
                T := T ∪ {t};
            end if;
        end for all;
    end while;
end procedure
```

EG

EG

```
procedure CheckEG(f1)
    S' := { s | f1 ∈ label(s) };
    SCC := { C | C is a nontrivial SCC of S' };
    T := ⋃_{C∈SCC} { s | s ∈ C };
    for all s ∈ T do label(s) := label(s) ∪ { EG f1 };
    while T ≠ ∅ do
        choose s ∈ T;
        T := T \ {s};
        for all t such that t ∈ S' and R(t, s) do
            if EG f1 ∉ label(t) then
                label(t) := label(t) ∪ { EG f1 };
                T := T ∪ {t};
            end if;
        end for all;
    end while;
end procedure
```

LTL Model Checking

Complicity of LTL Model Checking

Tableau method: $O((|S| + |R|) \times 2^{O(|\varphi|)})$

Complicity of LTL Model Checking

Tableau method: $O((|S| + |R|) \times 2^{O(|\varphi|)})$

At least NP-hard: consider Hamilton path of G

- $M, q_0 \models E(F\, p_1 \wedge \ldots \wedge F\, p_n \wedge G(P_1 \rightarrow X\, G \neg p_1) \wedge \ldots \wedge G(P_n \rightarrow X\, G \neg p_n))$
- $M = (\{q_0, q_f\} \cup V(G), \{q_0\}, \{(q_0, v), (v, q_f), (q_f, q_f) \mid v \in V(G)\} \cup E(G), L)$
- $L(v_i) = \{p_i\}$

Scenario of LTL Model Checking

- $A\varphi$ is a LTL, then the only state sub-formulas in φ are atomic propositions.
- $M, s \models A\varphi \iff M, s \models \neg E \neg\varphi$
- $M, s \models F\varphi \iff M, s \models true\, U\, \varphi$
- $M, s \models G\varphi \iff M, s \models \neg F \neg\varphi$
- It is sufficient to only consider the temporal operators X, U with $\neg, \vee$ wrapped by E.

Scenario of LTL Model Checking

- $A\,\varphi$ is a LTL, then the only state sub-formulas in φ are atomic propositions.
- $M, s \models A\,\varphi \iff M, s \models \neg E\,\neg\varphi$
- $M, s \models F\,\varphi \iff M, s \models \mathit{true}\,U\,\varphi$
- $M, s \models G\,\varphi \iff M, s \models \neg F\,\neg\varphi$
- It is sufficient to only consider the temporal operators X, U with $\neg, \vee$ wrapped by E.
- Construct closure $cl(\varphi)$ of φ, which is the set of formulae related to the truth value of φ.
- Construct graph of atoms (transition graph on truth table of $cl(\varphi)$)
- $M, s \models E\,\varphi$ is equivalent to existence of an eventuality sequence, which is detected as a SCC.

Closure of LTL formula

- The smallest set of formulae containing φ, where
  - $\neg\phi \in cl(\varphi)$ iff $\phi \in cl(\varphi)$;
  - if $\psi \vee \phi \in cl(\varphi)$, then $\psi, \phi \in cl(\varphi)$;
  - if $X\,\psi \in cl(\varphi)$, then $\psi \in cl(\varphi)$;
  - if $\neg X\,\psi \in cl(\varphi)$, then $X\,\neg\psi \in cl(\varphi)$;
  - if $\psi\,U\,\phi \in cl(\varphi)$, then $\psi, \phi, X(\psi\,U\,\phi) \in cl(\varphi)$.
- To keep finite (linear to $|\varphi|$), $\neg\neg$ is eliminated.
- By construction, at most one X would be added.
- Size of $cl(f)$ is linear in the size of f.
- e.g. $cl(\delta\,U\,\psi) =$

Closure of LTL formula

- The smallest set of formulae containing φ, where
  - $\neg\phi \in cl(\varphi)$ iff $\phi \in cl(\varphi)$;
  - if $\psi \vee \phi \in cl(\varphi)$, then $\psi, \phi \in cl(\varphi)$;
  - if $X\,\psi \in cl(\varphi)$, then $\psi \in cl(\varphi)$;
  - if $\neg X\,\psi \in cl(\varphi)$, then $X\,\neg\psi \in cl(\varphi)$;
  - if $\psi\,U\,\phi \in cl(\varphi)$, then $\psi, \phi, X(\psi\,U\,\phi) \in cl(\varphi)$.
- To keep finite (linear to $|\varphi|$), $\neg\neg$ is eliminated.
- By construction, at most one X would be added.
- Size of $cl(f)$ is linear in the size of f.
- e.g. $cl(\delta\,U\,\psi) =$
  - $\{\delta, \neg\delta, \psi, \neg\psi,$
  - $\delta\,U\,\psi, \neg(\delta\,U\,\psi), X(\delta\,U\,\psi), \neg X(\delta\,U\,\psi),$
  - $X\neg(\delta\,U\,\psi)\}$

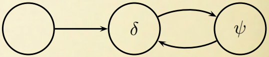
Atoms (wrt. φ)

- Atom (s, K) with $s \in S$ and $K \subseteq cl(\varphi) \cup AP$, where
  - for each $p \in AP$, $p \in K$, iff $p \in L(s)$;
  - for every $\delta \in cl(\varphi)$, $\delta \in K$, iff $\neg\delta \notin K$;
  - for every $\delta \vee \psi \in cl(\varphi)$, $\delta \vee \psi \in K$ iff $\delta \in K$ or $\psi \in K$;
  - for every $\neg X\, \delta \in cl(\varphi)$, $\neg X\, \delta \in K$ iff $X\,(\neg\delta) \in K$;
  - for every $\delta\, U\, \psi \in cl(\varphi)$, $\delta\, U\, \psi \in K$ iff $\psi \in K$ or $\delta, X(\delta\, U\, \psi) \in K$.
- Intuitively, K is the maximum consistent truth valuation at s.

Atoms (wrt. φ)

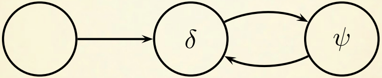
- Atom (s, K) with $s \in S$ and $K \subseteq cl(\varphi) \cup AP$, where
  - for each $p \in AP$, $p \in K$, iff $p \in L(s)$;
  - for every $\delta \in cl(\varphi)$, $\delta \in K$, iff $\neg\delta \notin K$;
  - for every $\delta \vee \psi \in cl(\varphi)$, $\delta \vee \psi \in K$ iff $\delta \in K$ or $\psi \in K$;
  - for every $\neg X\,\delta \in cl(\varphi)$, $\neg X\,\delta \in K$ iff $X\,(\neg\delta) \in K$;
  - for every $\delta\, U\, \psi \in cl(\varphi)$, $\delta\, U\, \psi \in K$ iff $\psi \in K$ or $\delta, X(\delta\, U\, \psi) \in K$.
- Intuitively, K is the maximum consistent truth valuation at s.

e.g.



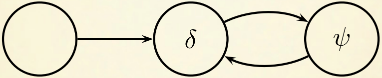
$cl(\delta\, U\, \psi) = \{\delta, \psi, \delta\, U\, \psi, X(\delta\, U\, \psi), \neg\delta, \neg\psi, \neg(\delta\, U\, \psi), \neg X(\delta\, U\, \psi), X\neg(\delta\, U\, \psi)\}$

Example of Atoms



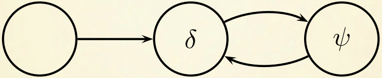
- Tableau:
  - [T,T,T,T], [T,T,T,F], [T,T,F,T], [T,T,F,F]
  - [T,F,T,T], [T,F,T,F], [T,F,F,T], [T,F,F,F]
  - [F,T,T,T], [F,T,T,F], [F,T,F,T], [F,T,F,F]
  - [F,F,T,T], [F,F,T,F], [F,F,F,T], [F,F,F,F]

Example of Atoms



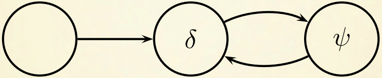
- Tableau:
    - [T,T,T,T], [T,T,T,F], [T,T,F,T], [T,T,F,F]
    - [T,F,T,T], [T,F,T,F], [T,F,F,T], [T,F,F,F]
    - [F,T,T,T], [F,T,T,F], [F,T,F,T], [F,T,F,F]
    - [F,F,T,T], [F,F,T,F], [F,F,F,T], [F,F,F,F]
- $s_0 : (L(s_0) = \neg\delta \wedge \neg\psi)$
    - [F,F,T,T], [F,F,T,F], [F,F,F,T], [F,F,F,F]

Example of Atoms



- Tableau:
  - [T,T,T,T], [T,T,T,F], [T,T,F,T], [T,T,F,F]
  - [T,F,T,T], [T,F,T,F], [T,F,F,T], [T,F,F,F]
  - [F,T,T,T], [F,T,T,F], [F,T,F,T], [F,T,F,F]
  - [F,F,T,T], [F,F,T,F], [F,F,F,T], [F,F,F,F]
- $s_0 : (L(s_0) = \neg\delta \wedge \neg\psi)$
  - [F,F,T,T], [F,F,T,F], [F,F,F,T], [F,F,F,F]
- $s_1 : (L(s_1) = \delta \wedge \neg\psi)$
  - [T,F,T,T], [T,F,T,F], [T,F,F,T], [T,F,F,F]

Example of Atoms

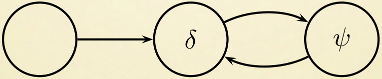


- Tableau:
  - [T,T,T,T], [T,T,T,F], [T,T,F,T], [T,T,F,F]
  - [T,F,T,T], [T,F,T,F], [T,F,F,T], [T,F,F,F]
  - [F,T,T,T], [F,T,T,F], [F,T,F,T], [F,T,F,F]
  - [F,F,T,T], [F,F,T,F], [F,F,F,T], [F,F,F,F]
- $s_0 : (L(s_0) = \neg\delta \wedge \neg\psi)$
  - [F,F,T,T], [F,F,T,F], [F,F,F,T], [F,F,F,F]
- $s_1 : (L(s_1) = \delta \wedge \neg\psi)$
  - [T,F,T,T], [T,F,T,F], [T,F,F,T], [T,F,F,F]
- $s_2 : (L(s_2) = \neg\delta \wedge \psi)$
  - [F,T,T,T], [F,T,T,F], [F,T,F,T], [F,T,F,F]

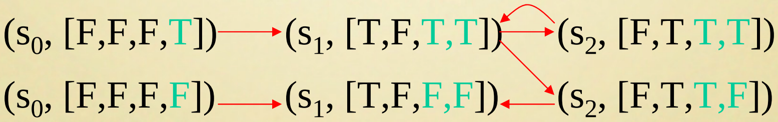
Graph of Atoms

- For Kripke structure $M = (S, S_0, R, L)$, formula φ, define a graph of atoms where nodes are atoms and edged are:

$$\{((s,K),(s',K')) \mid (s,s') \in R \wedge \forall (X\,\delta) \in cl(\varphi), X\,\delta \in K \iff \delta \in K'\}$$

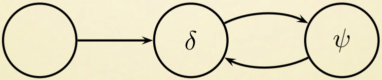


$\delta, \psi, \delta\, U\, \psi, X(\delta\, U\, \psi)$

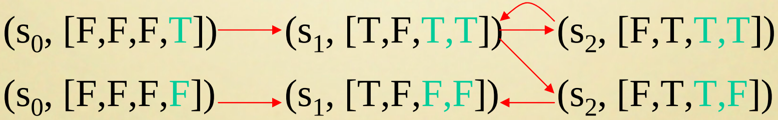


Eventuality Sequence

- An eventuality sequence is an infinite path π in a graph of atoms, satisfying:
  - If $\delta\, U\, \psi \in K$ for an atom (s, K) on π, then there exists an atom (s', K') on π after (s, K) with $\psi \in K'$.

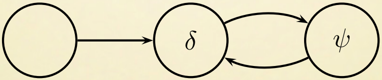


$\delta, \psi, \delta\, U\, \psi, X(\delta\, U\, \psi)$



Eventuality Sequence

- An eventuality sequence is an infinite path π in a graph of atoms, satisfying:
  - If $\delta\, U\, \psi \in K$ for an atom (s, K) on π, then there exists an atom (s', K') on π after (s, K) with $\psi \in K'$.
  - Don't care on δ between (s, K) and (s', K'), why?



$\delta, \psi, \delta\, U\, \psi, X(\delta\, U\, \psi)$

$(s_0, [F,F,F,T]) \rightarrow (s_1, [T,F,T,T]) \rightarrow (s_2, [F,T,T,T])$

$(s_0, [F,F,F,F]) \rightarrow (s_1, [T,F,F,F]) \leftarrow (s_2, [F,T,T,F])$

Key Lemma

Lemma:
$M, s \models E\varphi$ iff there exists an eventuality sequence starting from an atom (s, K) with $\varphi \in K$.

Proof Sketch ($\Longrightarrow$)

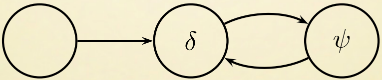
- $M, s_0 \models E\,\varphi$, if there exists an eventuality sequence $\pi = (s_0, K_0), (s_1, K_1), (s_2, K_2) \ldots$ with $\varphi \in K_0$.
- let $\pi^i = (s_i, K_i), (s_{i+1}, K_{i+1}), (s_{i+2}, K_{i+2}) \ldots$, we will prove "$\pi^i \models \delta \iff \delta \in K_i$, *for each* $\delta \in cl(\varphi)$" by induction on the structure of formula.
  - Case $\delta = X\,\gamma$: By construction of a graph of atoms, $((s_i, K_i), (s_{i+1}, K_{i+1}))$ implies $X\,\gamma \in K_i \iff \gamma \in K_{i+1}$. Thus, $X\,\gamma \in K_i \iff \gamma \in K_{i+1} \iff \pi^{i+1} \models \gamma \iff \pi^i \models X\,\gamma$.
  - Case $\delta = \gamma\,U\,\psi$:
    - By definition of π, there exists (first) $j \geq i$ with $\psi \in K_j$.
    - Then $\delta \in K_j$ (by definition of atom), and $\pi^j \models \psi$ (by induction hypothesis); thus $\pi^j \models \delta$.
    - Note that $\psi \notin K_i \wedge \ldots \wedge \psi \notin K_{j-1}$; then $\gamma, X\,\delta \in K_i$ $\iff \gamma \in K_i \wedge \delta \in K_{i+1} \iff \gamma \in K_i \wedge \ldots \wedge \gamma \in K_{j-i} \iff \pi^i \models \gamma \wedge \ldots \pi^{j-1} \models \gamma \iff \pi^i \models \delta$.

Proof Sketch ($\Longleftarrow$)

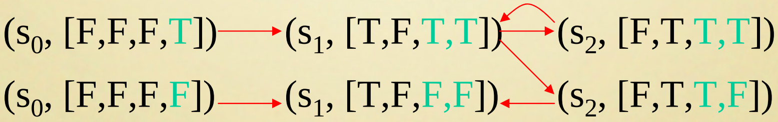
- $M, s_0 \models E\,\varphi$ only if there exists an eventuality sequence starting from an atom (s, K) with $\varphi \in K$.
- Let $\pi = s_0, s_1, s_2, \ldots$, s.t. $M, \pi \models \varphi$. Then, $(s_0, K_0), (s_1, K_1), (s_2, K_2), \ldots$ is an eventuality sequence where $K_i = \{\delta \mid \delta \in cl(\varphi) \wedge M, \pi^i \models \delta\}$ for $\pi^i = s_i, s_{i+1}, s_{i+2}, \ldots$.

Self-fulfilling SCC in Graph of Atoms

A non-trivial SCC C in a graph of atoms is self-fulfilling iff, for every atom (s, K) in C with $\delta\, U\, \psi \in K$, there exists an atom (s', K') in C such that $\psi \in K'$.
(i.e., there is an eventuality sequence that covers SCC C).

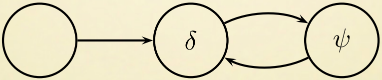


$\delta, \psi, \delta\, U\, \psi, X(\delta\, U\, \psi)$



Self-fulfilling SCC in Graph of Atoms

A non-trivial SCC C in a graph of atoms is self-fulfilling iff, for every atom (s, K) in C with $\delta \, U \, \psi \in K$, there exists an atom (s', K') in C such that $\psi \in K'$.
(i.e., there is an eventuality sequence that covers SCC C).



Lemma:
There exists an eventuality sequence starting at an atom (s, K) iff there exists a path from (s, K) to a self-fulfilling SCC.

Proof

$\Rightarrow$: Assume that there is an eventuality sequence starting at (s, K). Consider the set C' of all atoms that appear infinitely often in this sequence. The set C' is a subset of a (maximal) strongly connected component C of G. Consider a subformula $\delta U \varphi$, and an atom $(s, K) \in C$ such that $\delta U \varphi \in K$. Because C is strongly connected, there is a finite path in C from (s, K) into C'. If φ appears on the path, we are done! Otherwise, since C' comes from an eventuality sequence, and φ is in some atom of C'.
$\Leftarrow$: Trivial.

LTL Model Checking

$M, s \models E\,\phi$ iff there exists atom $A = (s, K)$ such that $\phi \in K$ and there exists a path from A to a self-fulfilling strongly connected component.

Summary of Algorithm

- Construct a graph of atoms for a formula φ, and compute self-fulfilling SCCs.
- Finding an eventuality sequence to self-fulfilling SCC by depth-first search.
- Atoms may multiplicand at most the exponential of the size of closure, (which is linear to $|\varphi|$).
- Complexity: $O((|S| + |R|) \times 2^{O(|\varphi|)})$

The Reality of Model Checking

The Reality of Model Checking

State explosion!

The Reality of Model Checking

State explosion!

The target system is huge!

The Reality of Model Checking

State explosion!

The target system is huge!

The software model checking is infinite!

The Reality of Model Checking

State explosion!

The target system is huge!

The software model checking is infinite!

The search algorithm itself is exponential!

Milestones

Milestones

- symbolic model checking SMV

Milestones

- symbolic model checking SMV
- partial reduction Spin

Milestones

- symbolic model checking SMV
- partial reduction Spin
- on-the-fly model checking SMV v.2

Milestones

- symbolic model checking SMV
- partial reduction Spin
- on-the-fly model checking SMV v.2
- bounded model checking NuSMV

Milestones

- symbolic model checking SMV
- partial reduction Spin
- on-the-fly model checking SMV v.2
- bounded model checking NuSMV
- counter-example guided abstract refinement (CEGAR) BLAST

Milestones

- symbolic model checking SMV
- partial reduction Spin
- on-the-fly model checking SMV v.2
- bounded model checking NuSMV
- counter-example guided abstract refinement (CEGAR) BLAST
- Craig interpolation NuSMV v.?

Milestones

- symbolic model checking SMV
- partial reduction Spin
- on-the-fly model checking SMV v.2
- bounded model checking NuSMV
- counter-example guided abstract refinement (CEGAR) BLAST
- Craig interpolation NuSMV v.?
- antichain

Further Topics

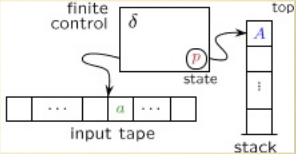
Infinite Structures: Unbounded Stack

```
function parse ( $handle )
{
        // Get the file
        $contents = $this->FILES[$handle];
        // If there's no template variables in the file, don't bother
        if ( strpos($contents, OPEN_VAR) === false )
        {
                echo $contents;
                return;
        }

        // Substitute global vars. This is the easy part
        foreach ( $this->VARS as $var_name => $var_value )
        {
                $contents = str_replace( OPEN_VAR . $var_name . CLOSE_VAR,
        }

        // If there's no block vars, don't bother processing them
        if ( strpos($contents, '<!-- BEGIN ') === false )
        {
                echo $contents;
                return;
        }

        // Now the tricky part: Substituting an HTML code block for multip
        foreach ( $this->BLOCK_VARS as $block_name => $block_array )
        {
                // Get all the blocks matching $block_name
                $count = preg_match_all("#<!-- BEGIN $block_name -->(.*?)<
```

Pushdown Automata

A pushdown system $\mathcal{P} = (Q, q_0, \Gamma, w_0, \Delta)$ is a transition system with carrying an unbounded stack.

- Q is a set of control locations, and $q_0 \in Q$ is the initial location.
- Γ is a finite set of stack alphabet, and $w_0 \in \Gamma^*$ is the initial stack contents.
- $\Delta : (Q \times \Gamma) \times (Q \times \Gamma^*)$ is a finite subset of transitions with the form $\langle q, \gamma \rangle \hookrightarrow \langle q', w \rangle$, where $q, q' \in Q$, $\gamma \in \Gamma$ and $w \in \Gamma^*$.



Infinite Structures: Real-Time

Timed Automata

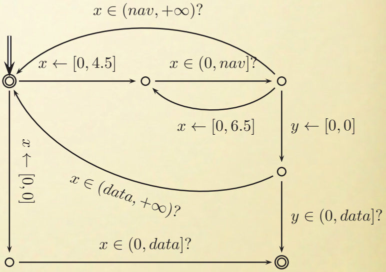
A TA (Q, q_0, F, X, Δ), where

- Q is a finite set of locations,
- initial location $q_0 \in Q$,
- $F \subseteq Q$ is the set of final locations,
- X is a finite set of clocks,
- $\Delta \subseteq Q \times \mathcal{O} \times Q$. A transition $q_1 \xrightarrow{\phi} q_2$, where ϕ is either of

  Local ϵ,

  Test $x \in I?$,

  Assignment $x \leftarrow I$.



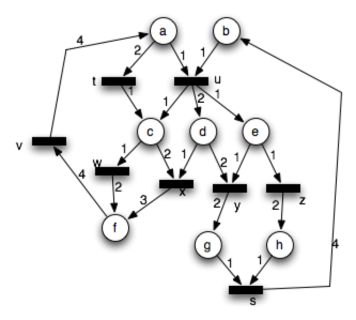
Infinite Structures: Multi-Threads

```
/// <summary>
/// This method will always run in a thread separate from the main thread.
/// </summary>
private void doStuffAsync()
{
    //this if statement makes sure that this method is running in a thread
    //separate from the main thread.
    if (Dispatcher.Thread == System.Threading.Thread.CurrentThread)
    {
        System.Threading.ThreadStart threadStart = new System.Threading.ThreadStart(doStuffAsync);
        System.Threading.Thread newThread = new System.Threading.Thread(threadStart);
        newThread.Start();
        return;
    }

    //code beyond here is running in a thread separate from the main thread

    setText("I can count to 10!");
    System.Threading.Thread.Sleep(1000);
    for (int x = 1; x <= 10; x++)
    {
        System.Threading.Thread.Sleep(500);
        setText(x.ToString());
    }

    System.Threading.Thread.Sleep(1000);
    setText("yay me.");
}
```

Petri Net

A Petri net is a triple $N = (P, T, F)$ where:

- P and T are disjoint finite sets of places and transitions, respectively.
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs.



Reports

Rep1. Probabilistic model checking (Maximal 3 students).

Rep2. Stochastic model checking (Maximal 3 students).

Rep3. Model checking in a specific field (Maximal 5 students).