Design and Analysis of Algorithms (IV)

Algorithmic Verification Basics

Guoqiang Li
School of Software

SHANGHAI JIAO TONG UNIVERSITY

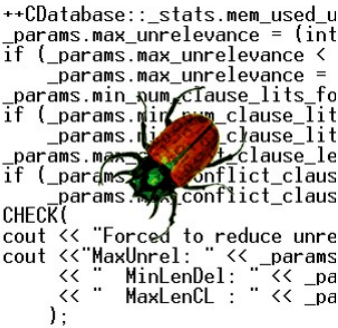
Category of Formal Methods

Formal modelling

Formal specification

Formal verification

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 ...

Formal Verifications

Here are many formal verification techniques:

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

This lecture will give a very brief introduction of **model checking**.

Model Checking

Q: What is model checking?

Q: What is model checking?

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

Model Checking

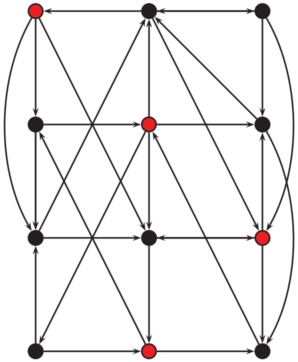
Q: What is model checking?

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

Sometimes it is called algorithmic formal verification.

The First Question

The First Question



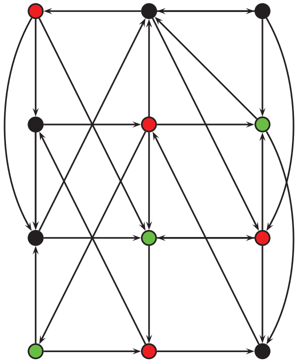
Safety as Reachability

Safety as Reachability

Bad things will never happen!

The Second Question

The Second Question



Liveness

Liveness

Good things will eventually happen!

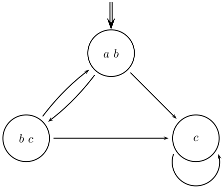
Data Structures

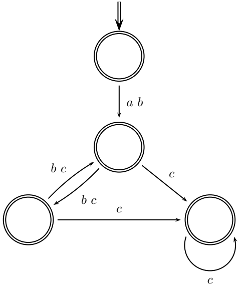
Kripke structure: $M = (S, S_0, R, L)$

- S, finite set of state
- $S_0 \subseteq S$, initial state
- $R \subseteq S \times S$, transition relations
- $L : S \to 2^{AP}$, status label function
  (AP: atomic propositions)

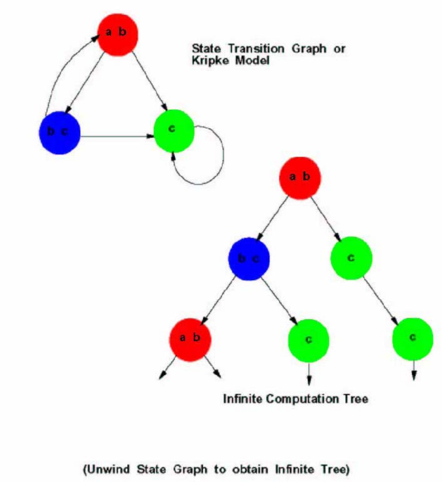
Finite automata: $\mathcal{A} = (\Sigma, Q, Q_0, F, \delta)$

- 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

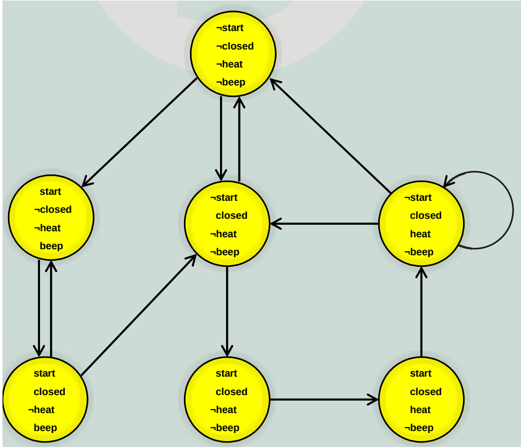




Finite Systems Vs. Infinite Computation Tree



An Microwave Oven Example



Logic-Based MC: Temporal Operators

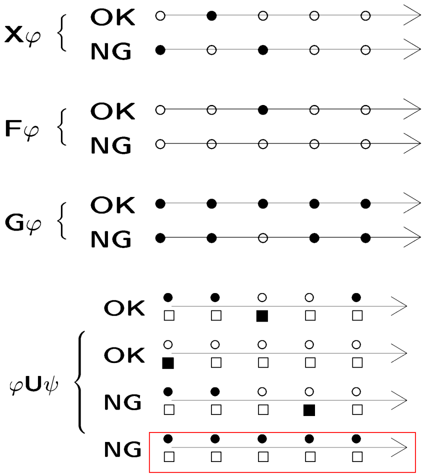
Shanghai Jiao Tong University

Next

Finally

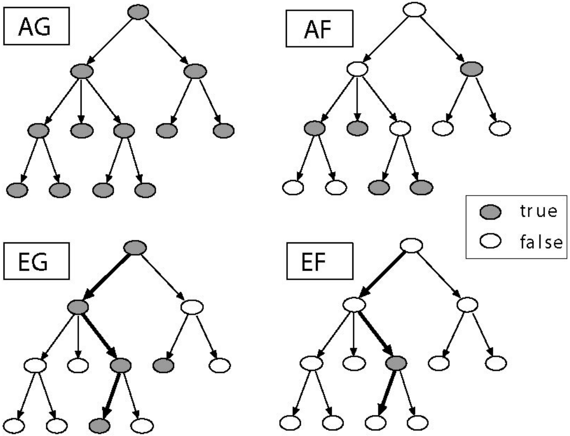
Globally

Until



$\bullet : \varphi$, $\blacksquare : \psi$, $\circ : \neg\varphi$, $\square : \neg\psi$

Logic-Based MC: Path Operators, A, E



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

Example Specification

$$EF(\texttt{Start} \wedge \neg \texttt{Ready})$$

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.

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.

$AG(\texttt{Req} \rightarrow AF\,\texttt{Ack})$

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.

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.

$AG(\texttt{Req} \rightarrow AF\,\texttt{Ack})$

- If a request occurs, then it will be eventually acknowledged.

$AG(AF\,\texttt{DeviceEnabled})$

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.

$AG(\texttt{Req} \rightarrow AF\,\texttt{Ack})$

- If a request occurs, then it will be eventually acknowledged.

$AG(AF\,\texttt{DeviceEnabled})$

- The proposition `DeviceEnabled` holds infinitely often on every computation path.

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.

$AG(\texttt{Req} \to AF\,\texttt{Ack})$

- 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

$$AG\ (\texttt{request} \rightarrow F\ \texttt{grant})$$

Example Specification

$AG\ (\texttt{request} \rightarrow F\ \texttt{grant})$

- Each `request` will be finally `grant(ed)`.

Example Specification

$AG\ (\texttt{request} \rightarrow F\ \texttt{grant})$

- Each `request` will be finally `grant(ed)`.

$AG(\neg(\neg\texttt{request}\ U\ \texttt{grant}))$

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`.

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.

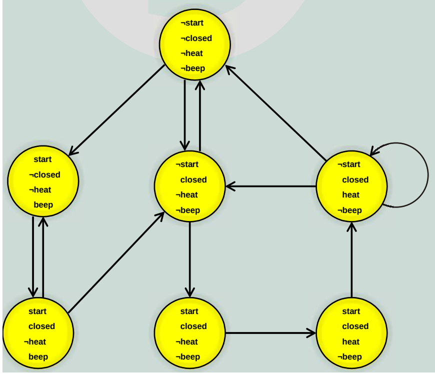
- e.g., $AF\varphi, EG\varphi, AXEG\varphi, EXA(\varphi U\psi)$

LTL: path quantifiers are allowed only at the outermost position.

- e.g., $AGF\varphi, EX(\varphi U\psi), A(F\varphi \vee G\psi)$

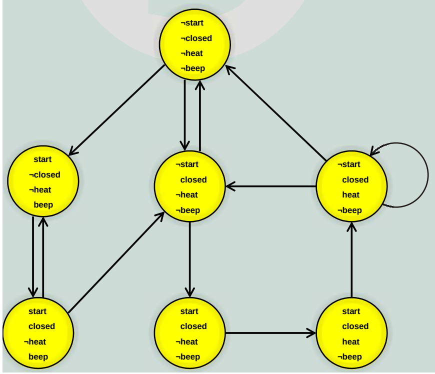
Except for fairness, most properties are expressed in CTL $\cap$ LTL.

Fairness



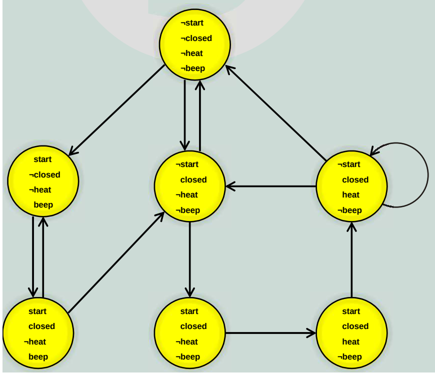
¬start ¬closed ¬heat ¬beep
start ¬closed ¬heat beep
¬start closed ¬heat ¬beep
¬start closed heat ¬beep
start closed ¬heat beep
start closed ¬heat ¬beep
start closed heat ¬beep

Fairness



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

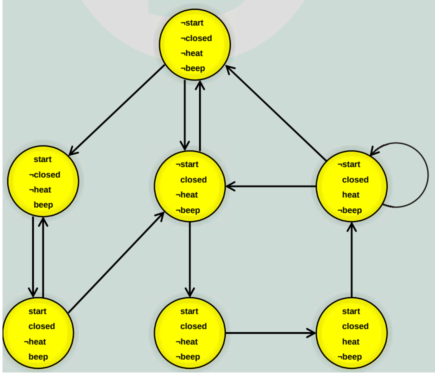
Fairness



$AG(start \rightarrow AF\ heat)$

- NG!

Fairness



$AG(start \rightarrow AF\ heat)$

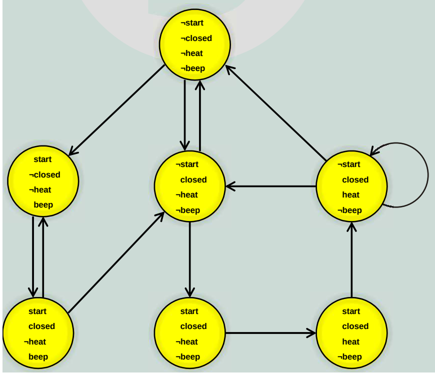
- NG!

Constraint:

$AGF start \wedge close \wedge \neg beep$

(operate correctly infinitely often)

Fairness



$AG(start \rightarrow AF\ heat)$

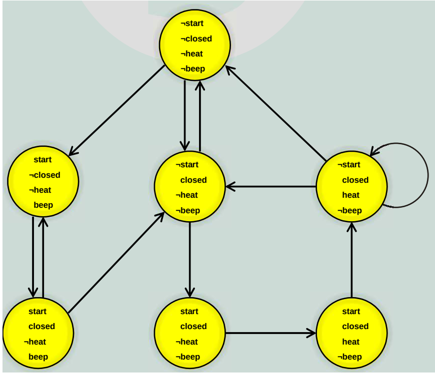
- NG!

Constraint:
$AGF start \wedge close \wedge \neg beep$
(operate correctly infinitely often)

$AG(start \rightarrow AF\ heat)$

Fairness

SHANGHAI JIAO TONG UNIVERSITY



$AG(start \rightarrow AF\ heat)$

- NG!

Constraint:
$AGF start \wedge close \wedge \neg beep$
(operate correctly infinitely often)

$AG(start \rightarrow AF\ heat)$

- OK!

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.

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:

- 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.

CTL Model Checking Algorithms

CTL Formula

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

Properties

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

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

procedure *CheckEU*(f_1, f_2)
    $T := \{\, s \mid f_2 \in label(s) \,\}$;
    for all $s \in T$ **do** $label(s) := label(s) \cup \{\, \mathbf{E}[f_1 \; \mathbf{U} \; f_2] \,\}$;
    while $T \neq \emptyset$ **do**
        choose $s \in T$;
        $T := T \setminus \{s\}$;
        for all t **such that** $R(t, s)$ **do**
            if $\mathbf{E}[f_1 \; \mathbf{U} \; f_2] \notin label(t)$ **and** $f_1 \in label(t)$ **then**
                $label(t) := label(t) \cup \{\, \mathbf{E}[f_1 \; \mathbf{U} \; f_2] \,\}$;
                $T := T \cup \{t\}$;
            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
```

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

SHANGHAI JIAO TONG UNIVERSITY

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

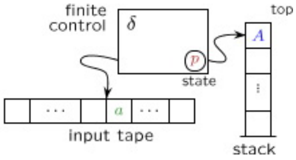
Infinite Structures: Unbounded Stack


```
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

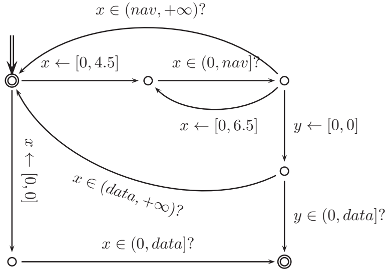
A TA (Q, q_0, F, X, Δ), where

- 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 $x \leftarrow I$.



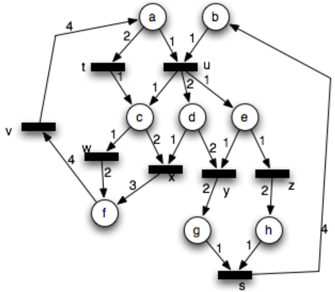
Infinite Structures: Multi-Threads


```
/// <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

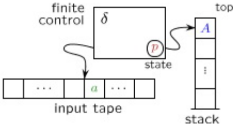
A Petri net is a triple $N = (P, T, F)$ where:

- 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.

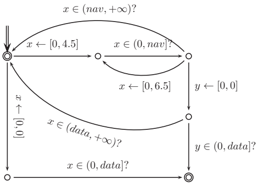


Infinite Structures

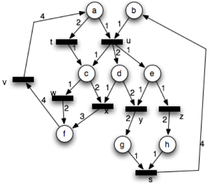
Recursion	Time	Concurrent



pushdown automata



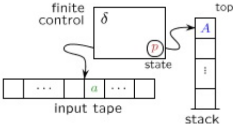
timed automata



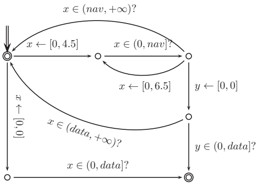
petri net

Infinite Structures

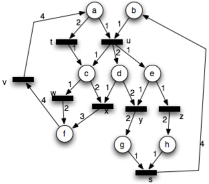
Recursion | Time | Concurrent



pushdown automata



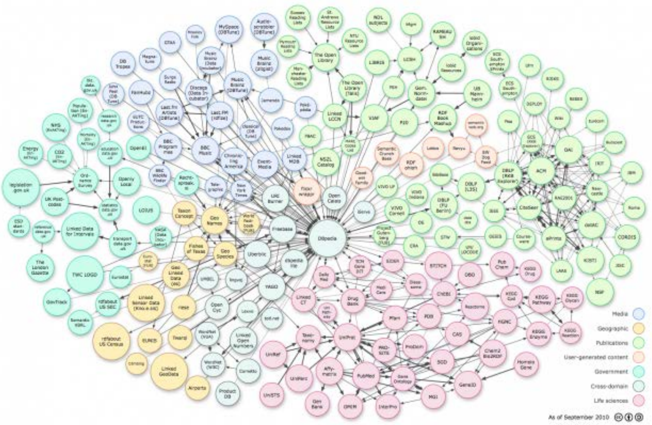
timed automata



petri net

What if combines several features of them?

Another Direction



What if structure is simple but the graph is much, much huge?