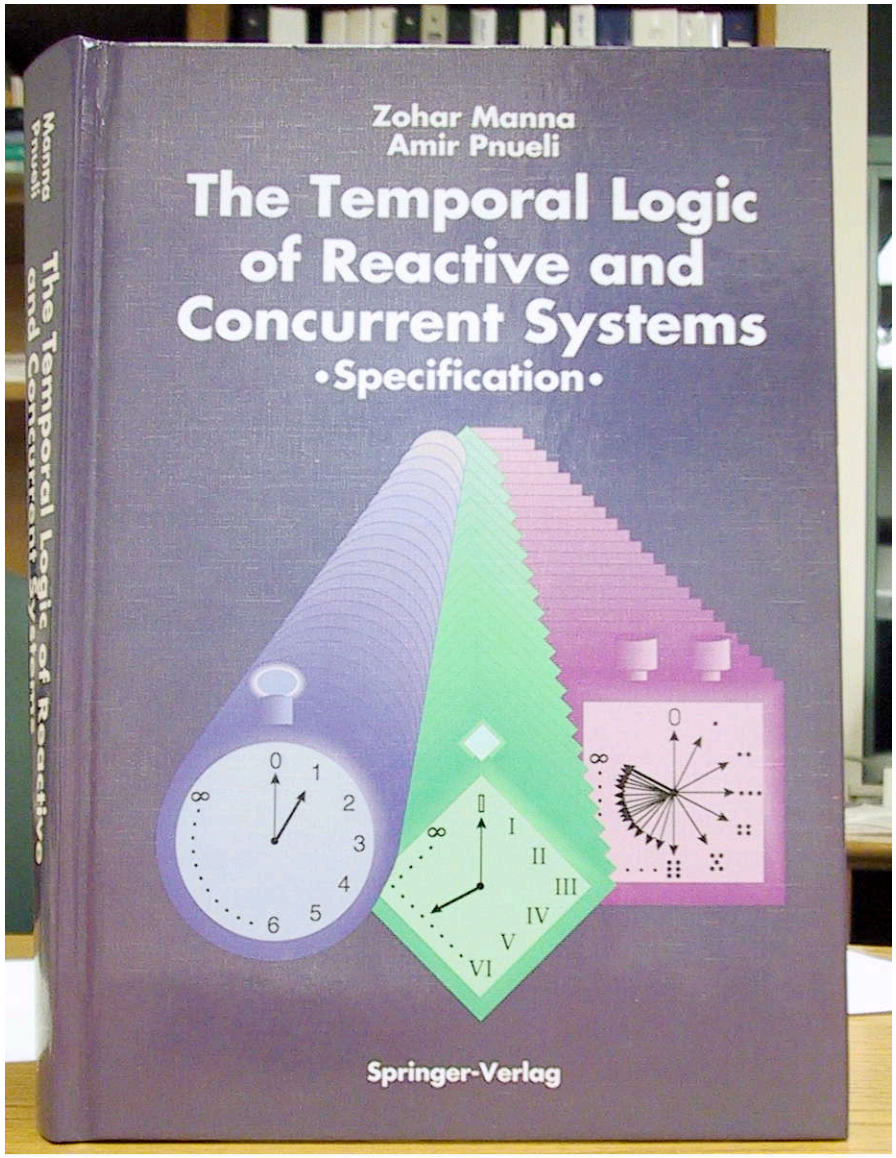
temporal logic

CS 119

includes slides
by Grigore Rosu

propositional logic extended with
temporal operators referring to
past and future

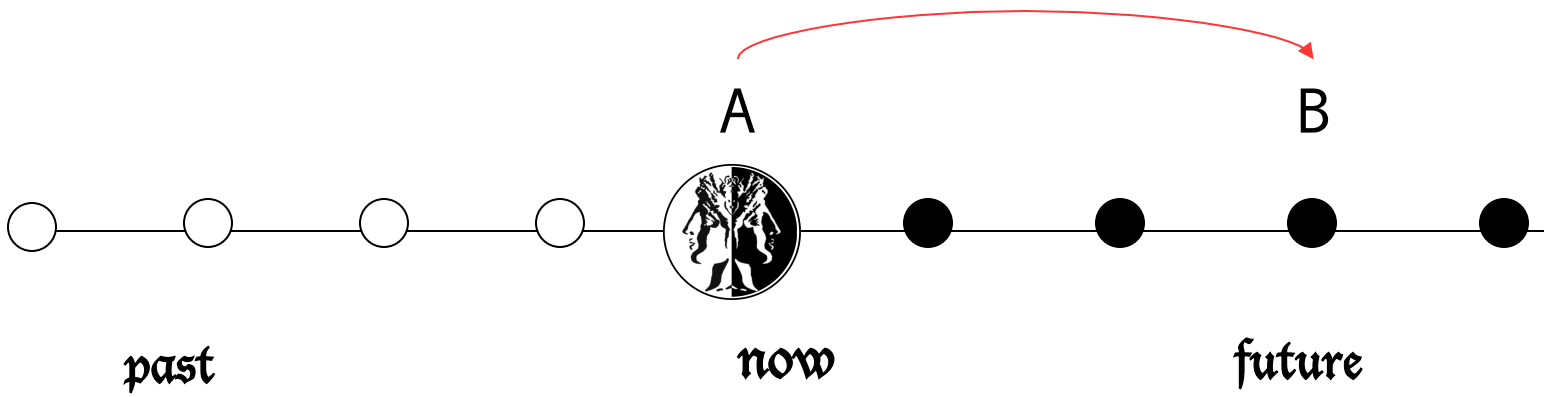

Zohar Manna
Amir Pnueli
The Temporal Logic of Reactive and Concurrent Systems
•Specification•
Springer-Verlag

future time properties

- If A happens now

  B must happen

$$\Box(A \rightarrow \Diamond B)$$

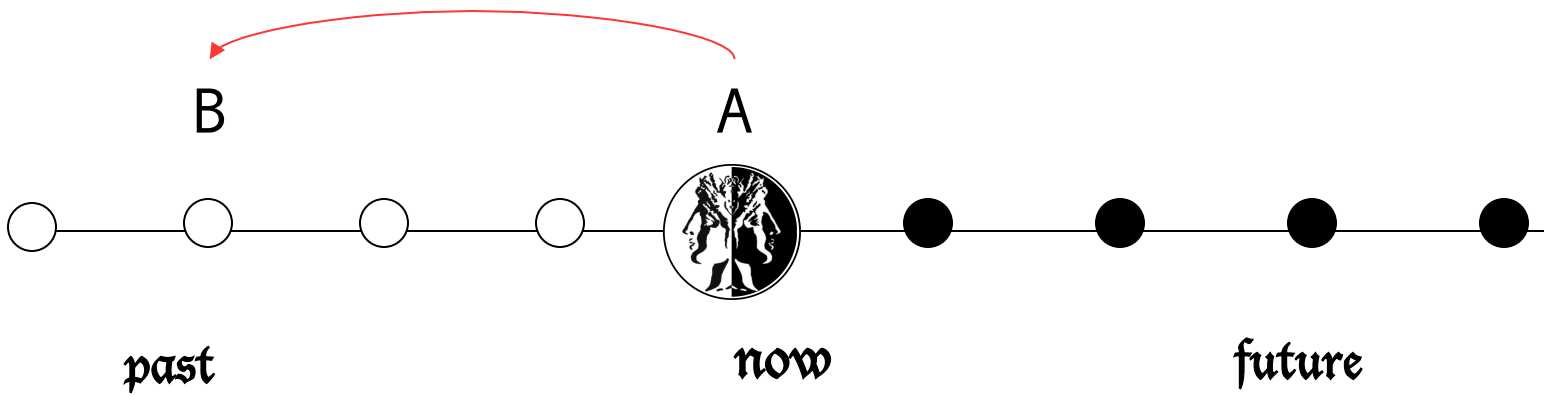


past time properties

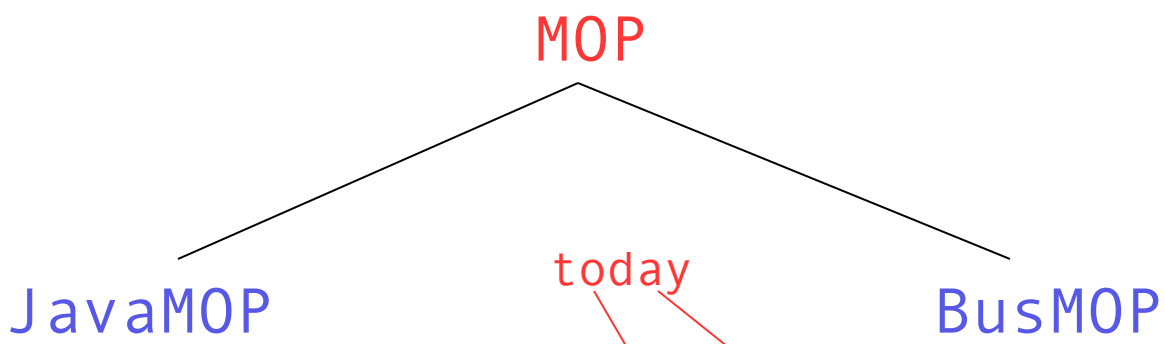
- If A happens now

  B must have happened

$$\Box(A \rightarrow \blacklozenge B)$$



instances of MOP



MOP (languages \ logic plugins)	FSM	ERE	CFG	PTLTL	FTLTL	PTCaRet	...
JavaMOP	JavaFSM	JavaERE	JavaCFG	JavaPTLTL	JavaFTLTL	JavaPTCaRet	...
BusMOP	BusFSM	BusERE	...	BusPTLTL	...	...	...
...	...	...	...	...	...	...	...

MOP Matrix: a clickable map of MOP pages.

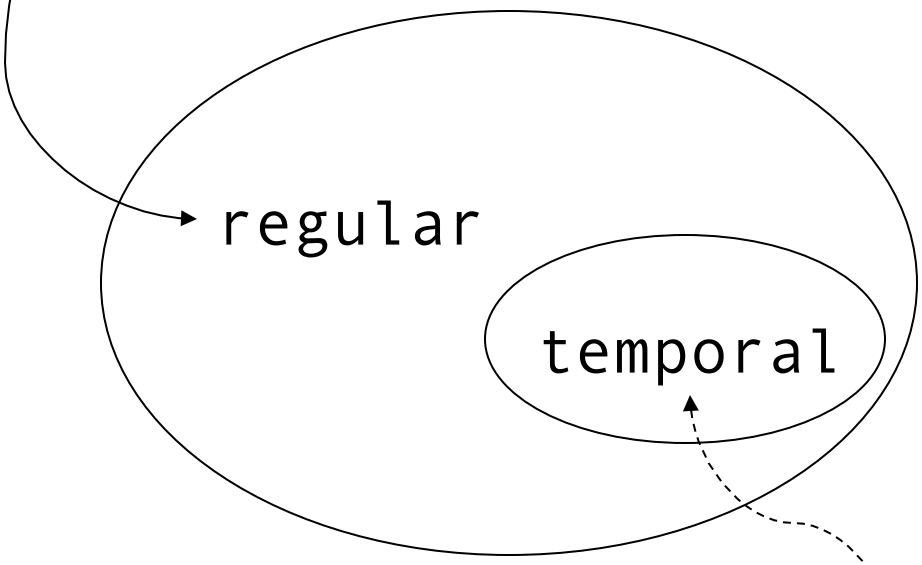
Chomsky's language hierarchy

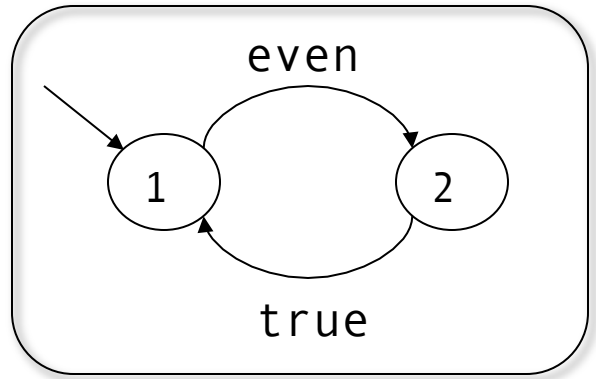
Grammar	Languages	Automaton	Production rules (constraints)
Type-0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$ (no restrictions)
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \gamma$
Type-3	Regular	Finite state automaton	$A \rightarrow a$ and $A \rightarrow aB$

http://en.wikipedia.org/wiki/Chomsky_hierarchy

temporal logic for finite traces: subset of regular languages

p is even in every other state





even /\ always(even implies (next next even))

2 1 2 1 2 1 2 2 2 1 2 1 2 1

does not work

properties of temporal logic

- some properties can be stated more succinctly in TL.
- but TL properties can be hard to write and read for non-trivial scenarios.

- now, many scenarios **are** trivial.

- it is of course a debate at a practical engineering level what notation is most suitable in practice. Number of characters is one possible measure.

future and past time temporal logic

semantics and algorithms

future time

semantics and algorithm

monitoring Future Time LTL

Syntax – Propositional Calculus plus

o F (next) □ F (always) ◇ F (eventually) F U F' (until)

Executable Semantics – Rewriting

_{_} : Formula x Event -> Formula ("consume" event e)

F{e} formula that should hold after processing e

p{e}	⇨	is the atomic predicate p true on e ?
(F *op* F′){e}	⇨	F{e} *op* F′{e}
(o F){e}	⇨	F
(□ F){e}	⇨	F{e} ∧ (□ F)
(◇ F){e}	⇨	F{e} ∨ (◇ F)
(F U F′){e}	⇨	F′{e} ∨ (F{e} ∧ (F U F′))

Future Time LTL - example

Event stream: **x** red yellow **x** green yellow **x** green **x** red ...

$\Box(\text{green} \rightarrow \neg\text{red U yellow})$ {yellow}

$\Downarrow *$

$\Box(\text{green} \rightarrow \neg\text{red U yellow})$ {green}

$\Downarrow *$

$((\neg\text{red U yellow}) \wedge \Box(\text{green} \rightarrow \neg\text{red U yellow}))$ {yellow}

$\Downarrow *$

$\Box(\text{green} \rightarrow \neg\text{red U yellow})$ {green}

$\Downarrow *$

$((\neg\text{red U yellow}) \wedge \Box(\text{green} \rightarrow \neg\text{red U yellow}))$ {red}

$\Downarrow *$

$(\text{yellow}\{\text{red}\} \vee (\neg\text{red}\{\text{red}\} \wedge \neg\text{red U yellow})) \wedge \ldots \overset{*}{\Rightarrow} \text{false} \wedge \ldots \overset{*}{\Rightarrow} \text{false}$

Formula was violated!

timed temporal logic

- add real time (RTL, MiTL, timed automata, etc.)

$$\Box(\text{start} \rightarrow \Diamond_{5}\, \text{stop})$$

$$(\Diamond_{t} F)\{e{:}\delta\} \Rightarrow (\delta \leq t) \wedge (F\{e{:}\delta\} \vee \Diamond_{t-\delta} F)$$

rewriting using Maude

- implemented the algorithm above in PaX
  - Maude as rewriting engine
    - 15 lines of obviously correct code!
- monitored 100 million events on 1.7GHz PC
  - 185 seconds, 220 million rewrites
  - Faster than modified Büchi automaton in Java (1,500 lines of code)
    - Is this 1,500 LOC Java program correct?
- I/O + buffering take longer than rewriting …

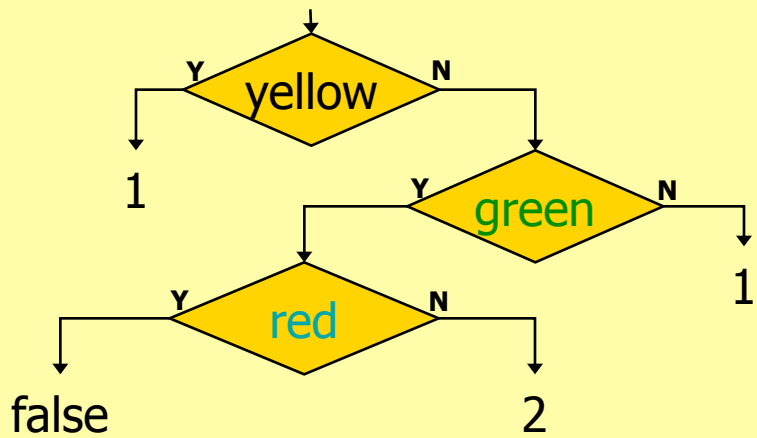
building a minimal BTT_FSM

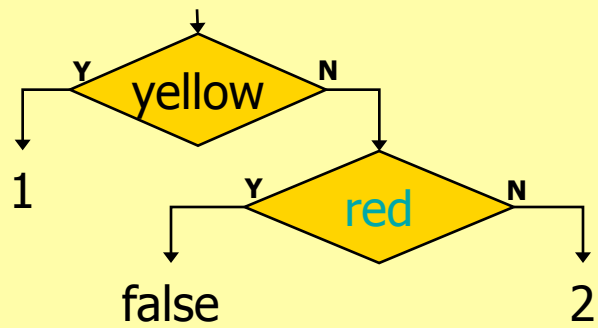
- idea
  - do the rewrites for *all* possible values of predicates
  - get a finite state machine
    - nodes are LTL formulae
    - optimize using a validity checker (F ↔ F' : one state)
    - edges are propositions
    - assign numbers to states
    - replace edges by *Binary Transition Trees*

Binary Transition Tree FSM

- We can build minimal FSMs *statically* for LTL

Formula	$\Box$(green $\rightarrow$ $\neg$red U yellow)	
State	1	2
BTT	yellow ? 1 : green ? (red ? false : 2) : 1	yellow ? 1 : (red ? false : 2)





MOP does not operate with "the end of a trace"

- so we cannot even write a formula of the form: $\Box$ (F → $\Diamond$ F′) and expect it to fail on for example the trace: F F′ F.

- instead, we have to assume an end event **E** and write: $\Box$ (F → !**E** U F′), meaning: if we see an F then we should see no **E** before an F′.

in MOP F U F' does not mean that F' eventually must happen. This is in contrast to traditional interpretations.

statistical approach
just a thought

Formula: $\varphi = \Box (A \rightarrow !\mathbf{E}\ U\ B)$

Trace: A A A A A A A B **E** satisfies φ

Trace: A A A A A A A A A **E** violates φ

Trace: A B A B A B A B A **E** violates φ

is this trace better?

was ok 4 out of 5 times

`maintain statistical information?`

past time

semantics and algorithm

monitoring safety

- Example:

<u>Safe Landing</u>

Land the space craft only after approval from ground and only if, since then, the radio signal has not been lost

or formally

$\uparrow$Landing $\rightarrow$ [Approved, $\downarrow$Radio}

where (MAC)

$\uparrow$F means <u>start F</u>,

$\downarrow$F means <u>end F</u>,

[F,F'} means <u>F butnot F'</u> since then

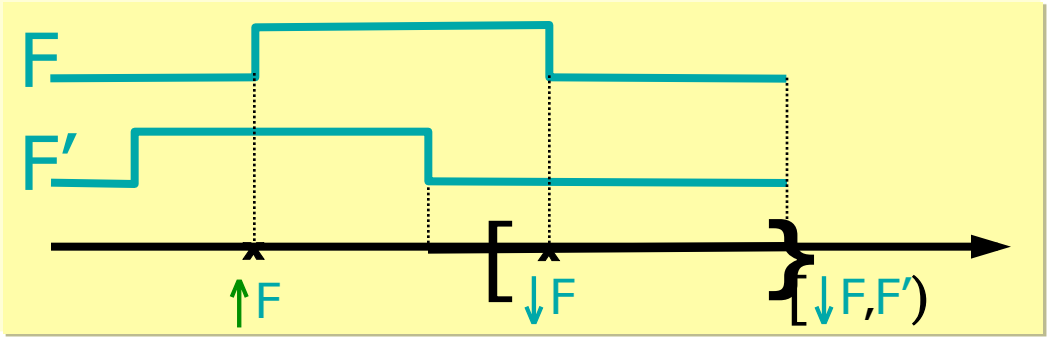
past time operators

Basic – Propositional Calculus plus

$\odot F$ (prev.) $F \; S \; F'$ (since) $\boxdot F$ (always) $\diamondminus F$ (eventually) in past

Special – Suitable for monitoring (MaC)

- $\uparrow F$ - start of F
- $\downarrow F$ - end of F
- $[F,F'\}$ - F butnot F'



<u>Theorem:</u> $\uparrow, \downarrow, [\_,\_\}$) and $\odot, \_S\_$ defined in terms of each other! F.ex:

$[F,F'\} = (\neg F') \; S \; F$,

$\uparrow F = F \wedge \odot \neg F$

Safety property: $\Box F$, where F is a past time LTL formula

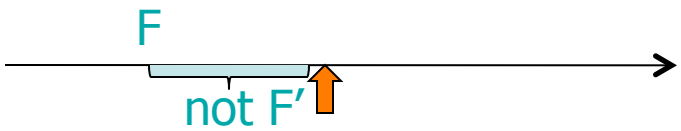
semantics

- standard semantics

$e_1\, e_2 \ldots e_{n-1}\, e_n \models [F, F')$ iff
there is $1 \leq i \leq n$ such that $e_1\, e_2 \ldots e_i \models F$ and
for all $i \leq j \leq n$, $e_1\, e_2 \ldots e_j \not\models F'$

- recursive Semantics

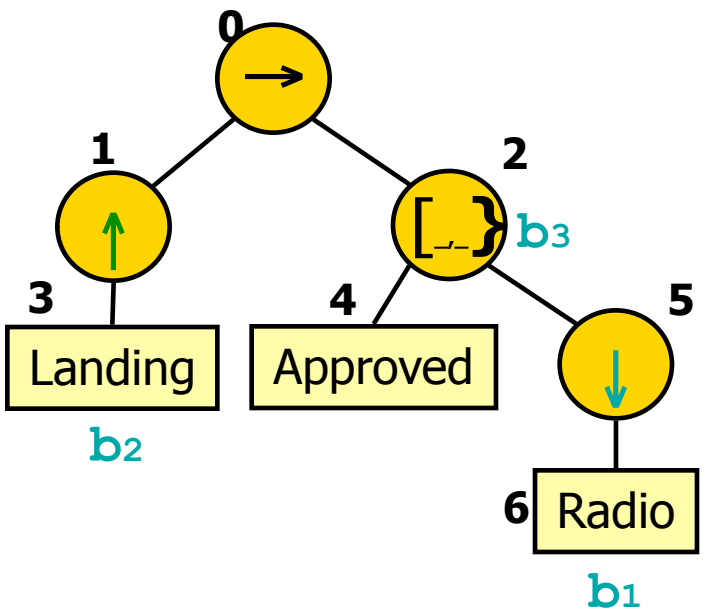
$t\, e \models [F, F')$ iff
$t\, e \not\models F'$ and ($t\, e \models F$ or $t \models [F, F')$)



dynamic programming algorithm

Formula: ↑Landing → [Approved, ↓Radio}

Trace: $e_1\, e_2 \ldots e_{n-1}\, e_n$



Step 4: Further Optimization

Global bits b_1, b_2, b_3

Temporary bits t_1, t_2, t_3

```
t1 = holds(Radio)
t2 = holds(Landing)
t3 = (not b1 or t1) and
     (holds(Approved) or b3)
if (t2 and not (b2 or t3))
     then "error"
(b1,b2,b3) = (t1,t2,t3)
```

Time: ≤ 6 CPU clocks!

the monitoring matrix

	MATCH/VALIDATE	FAIL/VIOLATE
TOTAL		
SUFFIX		

comparing ERE with LTL

```
property:
An p should be followed by a q
```

```
[](p -> <>q)
```

	TOTAL + FAIL	SUFFIX + MATCH
FTLTL	[](p -> !E U q)	
PTLTL	E implies ((! <*>p) or ((! p) S q))	
ERE	(q* (ε + (p p* q)))* E	
ERE		p p* E

Past Time LTL syntax

```
BNF below is extended with {p} for zero or more and [p] for zero or one repetitions of p
```

```
<PTLTL Plugin Input> ::=  {"event" <Event Name>} "ptltl:" <PTLTL>

<PTLTL> ::= "true" | "false" | <Event Name> | <PTLTL> "and" <PTLTL>
   <PTLTL> "or" <PTLTL> | "not" <PTLTL> | <PTLTL> "implies" <PTLTL>
    <!-- Normal boolean constants and operators>

    | "(*)" <PTLTL> | <PTLTL> "S" <PTLTL>
    | "<*>" <PTLTL> | "[*]" <PTLTL> | [<PTLTL>,<PTLTL>}
    <!-- temporal operators>

 <LOGIC State> ::= "violation" | "validation"
```

for writing handlers, for example:
@violation{…}

fomula	explanation
true	true
false	false
a	event a occurs
P and Q	Boolean conjunction
P or Q	Boolean disjunction
not P	Boolean negation
P implies Q	Boolean implication
(*) P	in previous state P
P S Q	P since Q
<*> P	P sometime in the past
[*] P	always in the past P
[P,Q}	P sometime in the past and Q never since then

Future Time LTL syntax

```
<FTLTL Syntax>  ::= <Event Header> <FTLTL>
        <FTLTL> ::= "ftltl:" <FTLTL Formula>
<FTLTL Formula> ::= "true" | "false"
                  | <Event Name>
                  | "!" <FTLTL Formula>
                  | <FTLTL Formula> "/\" <FTLTL Formula>
                  | <FTLTL Formula> "\/" <FTLTL Formula>
                  | <FTLTL Formula> "++" <FTLTL Formula>
                  | <FTLTL Formula> "->" <FTLTL Formula>
                  | <FTLTL Formula> "<->" <FTLTL Formula>
                  | "[]" <FTLTL Formula>
                  | "<>" <FTLTL Formula>
                  | "o" <FTLTL Formula>
                  | <FTLTL Formula> "U" <Formula>

 <FTLTL State> ::= "violation" | "validation"
```

for writing handlers, for example:
@violation{…}

true	true
false	false
a	event a occurs
! P	Boolean negation
P /\ Q	Boolean conjunction
P \/ Q	Boolean disjunction
P ++ Q	Boolean exclusive disjunction
P -> Q	Boolean implication
P <-> Q	Boolean bi-implication
[] P	always P
<> P	eventually P
o P	next P
P U Q	P until Q

MOP Past Time Linear Temporal Logic (PTLTL) Plugin - FSL

http://fsl.cs.uiuc.edu/index.php/Special:PTLTLPlugin

Joergen Ingman | Eventseer.net - Home | Free Website Polls | Programming with C/C++ | Castle for Sale | Costa Rica, ...Estate Blogs | BASH Help - ...sh Tutorial | DNS Alliance

MOP Past Time Linear T... | Untitled

Log in / create account

Formal Systems Laboratory

navigation

- Main Page
- People
- Projects
- Publications
- News
- Events

search

Go Search

MOP Past Time Linear Temporal Logic (PTLTL) Plugin

Authors:
Feng Chen
Patrick Meredith
Dongyun Jin
Grigore Rosu

MOP (languages \ logic plugins)	FSM	ERE	CFG	PTLTL	FTLTL	PTCaRet	...
JavaMOP	JavaFSM	JavaERE	JavaCFG	JavaPTLTL	JavaFTLTL	JavaPTCaRet	...
BusMOP	BusFSM	BusERE	...	BusPTLTL	...	...	...
...	...	...	...	...	...	...	...

MOP Matrix: a clickable map of MOP pages.

Links:
MOP
PTLTL Plugin
PTLTL Plugin Input Syntax
PTLTL Plugin Output Syntax
PTLTL Monitoring Algorithm

This page discusses the MOP past-time linear temporal logic (PTLTL) plugin. It also allows to type in an PTLTL specification and then generate a monitor for it. PTLTL is a logic for specifying properties of reactive and concurrent systems. PTLTL provides temporal operators that refer to the past states of an execution trace relative to a current point of reference. The logic plugin here is based on an rewriting-based algorithm for generating an optimized monitoring program from an PTLTL formula.

- MOP_PTLTL_Plugin.zip

Note: if there are any technical difficulties please alert pmeredit@cs.uiuc.edu

Choose an example:

- HasNext
- NoDisableWhileConverting
- Resource
- SafeDivrModify
- SafeIterator

Run Reset

Please press the Run button once and wait; it may take a few seconds to run PTLTLPlugin; the execution of PTLTLPlugin using this web interface is limited to 2 minutes of CPU time and 500 MB of RAM.

Powered By MediaWiki

MOP Past Time Linear Temporal Logic (PTLTL) Plugin – FSL

http://fsl.cs.uiuc.edu/index.php/Special:PTLTLPlugin

Joergen Ingman | Eventseer.net – Home | Free Website Polls | Programming with C/C++ | Castle for Sale | Costa Rica, ...Estate Blogs | BASH Help – ...sh Tutorial | DNS Alliance

MOP Past Time Linear T... | Untitled

MOP_PTLTL_Plugin.zip

Note: if there are any technical difficulties please alert pmeredit@cs.uiuc.edu

Choose an example:

- HasNext
- NoDisableWhileConverting
- Resource
- SafeDivrModify
- SafeIterator

```
event p
event q
event E

ptltl : E implies ((! <*>p) or ((! p) S q))
```

Run | Reset

Number of generated monitors (since 14 December 2008): 1044 (MOP), 92 (PTLTL), 49 (Language-neutral PTLTL)

The textbox below shows the generated monitor; see PTLTL Plugin Output Syntax (opens new window)

Done.

Text Output:

```
// 2 bits
// initialization of monitor state (nil
b[0] := false
b[1] := false) ;
// sequential monitoring code (
b[0] := 'q or not 'p and b[0] ;
b[1] := 'p or b[1]) ;
output(not 'E or not b[1] or b[0])
```

Powered By MediaWiki

MOP Future Time Linear Temporal Logic (FTLTL) Plugin - FSL

http://fsl.cs.uiuc.edu/index.php/Special:FTLTLPlugin

Joergen Ingman | Eventseer.net - Home | Free Website Polls | Programming with C/C++ | Castle for Sale | Costa Rica, ...Estate Blogs | BASH Help - ...sh Tutorial | DNS Alliance

MOP Future Time Linear... | Untitled

Log in / create account

Formal Systems Laboratory

navigation

- Main Page
- People
- Projects
- Publications
- News
- Events

search

Go Search

MOP Future Time Linear Temporal Logic (FTLTL) Plugin

Authors:
Feng Chen
Patrick Meredith
Dongyun Jin
Grigore Rosu

MOP (languages \ logic plugins)	FSM	ERE	CFG	PTLTL	FTLTL	PTCaRet	...
JavaMOP	JavaFSM	JavaERE	JavaCFG	JavaPTLTL	JavaFTLTL	JavaPTCaRet	...
BusMOP	BusFSM	BusERE	...	BusPTLTL	...	...	...
...	...	...	...	...	...	...	...

MOP Matrix: a clickable map of MOP pages.

Links:
MOP
FTLTL Plugin
FTLTL Plugin Input Syntax
FTLTL Plugin Output Syntax
FTLTL Monitoring Algorithm

This page discusses the MOP future-time linear temporal logic (FTLTL) plugin. It also allows to type in an FTLTL specification and then generate a monitor for it. FTLTL is a logic for specifying properties of reactive and concurrent systems. FTLTL provides temporal operators that refer to the future states of an execution trace relative to a current point of reference. The logic plugin here is based on a rewriting-based algorithm for generating an optimized monitoring program from an FTLTL formula.

- MOP_FTLTL_Plugin.zip

Note: if there are any technical difficulties please alert pmeredit@cs.uiuc.edu

Choose an example:

- HashSet
- TrafficLight

Run Reset

Please press the Run button once and wait; it may take a few seconds to run FTLTLPlugin; the execution of FTLTLPlugin using this web interface is limited to 2 minutes of CPU time and 500 MB of RAM.

Powered By MediaWiki

MOP Future Time Linear Temporal Logic (FTLTL) Plugin - FSL

http://fsl.cs.uiuc.edu/index.php/Special:FTLTLPlugin

This page discusses the MOP future-time linear temporal logic (FTLTL) plugin. It also allows to type in an FTLTL specification and then generate a monitor for it. FTLTL is a logic for specifying properties of reactive and concurrent systems. FTLTL provides temporal operators that refer to the future states of an execution trace relative to a current point of reference. The logic plugin here is based on a rewriting-based algorithm for generating an optimized monitoring program from an FTLTL formula.

- MOP_FTLTL_Plugin.zip

Note: if there are any technical difficulties please alert pmeredit@cs.uiuc.edu

Choose an example:

- HashSet
- TrafficLight

```
event p
event q
event E

ftltl : [](p -> !E U q)
```

Run Reset

Number of generated monitors (since 14 December 2008): 1026 (MOP), 49 (FTLTL), 23 (Language-neutral FTLTL)

The textbox below shows the generated monitor; see FTLTL Plugin Output Syntax (opens new window)

Done.

Text Output:

```
Starting state: 1
Transitions:
switch (state) {
case 1: state <- q ? 1 : p ? E ?  false  : 2 : 1
case 2: state <- q ? 1 : E ?     false  : 2
}
```

properties of Java library APIs

Iterator (Java 2 Platform SE 5.0)

http://java.sun.com/j2se/1.5.0/docs/api/

Joergen Ingman | TimeLife.com | Micro Site | Eventseer.net - Home | Free Website Polls | Richest Cou...n the World | Castle for Sale | Programming with C/C++ | Costa Rica, ...Estate Blogs | BASH Help - ...sh Tutorial

Try JavaMOP Online - FSL | Iterator (Java 2 Platform...

Java™ 2 Platform Standard Ed. 5.0

All Classes

Packages
- java.applet
- java.awt
- java.awt.color
- java.awt.datatransfer
- java.awt.dnd
- java.awt.event
- java.awt.font

IOException
IOR
IORHelper
IORHolder
IORInfo
IORInfoOperations
IORInterceptor
IORInterceptor_3_0
IORInterceptor_3_0Helper
IORInterceptor_3_0Holder
IORInterceptor_3_0Operations
IORInterceptorOperations
IRObject
IRObjectOperations
IstringHelper
ItemEvent
ItemListener
ItemSelectable
Iterable
Iterator
IvParameterSpec
JApplet
JarEntry
JarException
JarFile
JarInputStream
JarOutputStream
JarURLConnection
JButton
JCheckBox
JCheckBoxMenuItem
JColorChooser
JComboBox
JComboBox.KeySelectionManager

Overview | Package | **Class** | Use | Tree | Deprecated | Index | Help

PREV CLASS | NEXT CLASS | FRAMES | NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD — DETAIL: FIELD | CONSTR | METHOD

Java™ 2 Platform Standard Ed. 5.0

java.util

Interface Iterator<E>

All Known Subinterfaces:
ListIterator<E>

All Known Implementing Classes:
BeanContextSupport.BCSIterator, Scanner

public interface **Iterator<E>**

An iterator over a collection. Iterator takes the place of Enumeration in the Java collections framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

This interface is a member of the Java Collections Framework.

Since:
1.2

See Also:
Collection, ListIterator, Enumeration

Method Summary	
boolean	hasNext() — Returns true if the iteration has more elements.
E	next() — Returns the next element in the iteration.
void	remove() — Removes from the underlying collection the last element returned by the iterator (optional operation).

Method Detail

hasNext

R_1: There should be no two calls to next() without a call to hasNext() in between, on the same iterator.

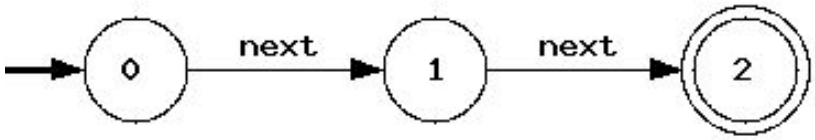
our hasNext example again

```
public class Test2 {
    public static void main(String[] args) {
        Vector<Integer> v1 = new Vector();
        Vector<Integer> v2 = new Vector();
        v1.add(1); v1.add(3); v2.add(5); v2.add(7);
        Iterator it1 = v1.iterator();
        Iterator it2 = v2.iterator();
        int sumFirstTwo_v2 = 0;
        if(it2.hasNext())
            sumFirstTwo_v2 += (Integer)it2.next();
        if(it1.hasNext())
            sumFirstTwo_v2 += (Integer)it2.next();
        System.out.println("sumFirstTwo_v2 = " + sumFirstTwo_v2);
    }
}
```

should have been:
`if(it2.hasNext())`

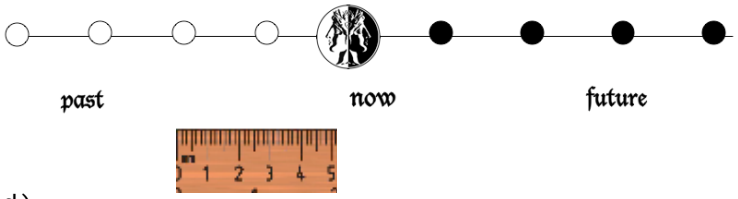
recall ERE 1

next next



i) suffix trace semantics

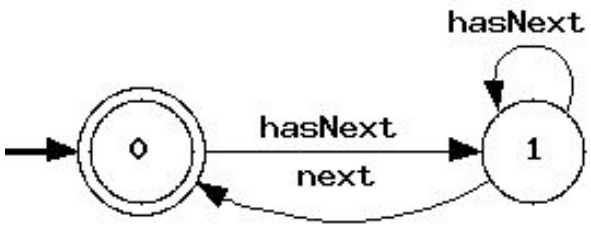
ii) looking for match



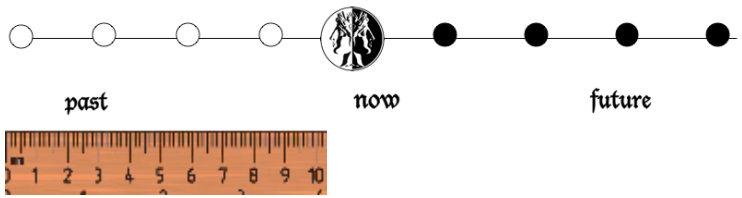
note: match = validation, fail = violation (MOP has changed)

recall ERE 2

(hasNext hasNext* next)*



i) total trace semantics
ii) looking for violation



LTL

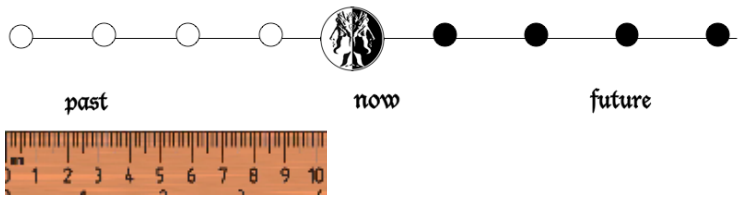
PTLTL

- next implies (*)(!next S hasnext)
- next implies (*)[hasnext,next}
- next implies (*)([hasnext,next} or !<*>next)

FTLTL

- [](next -> o(!next U hasnext))

i) total trace semantics
ii) looking for violation



```
package paramere.hasnext;

import java.io.*;
import java.util.*;

HasNext(Iterator i) {
    event hasnext before(Iterator i) : call(* Iterator.hasNext()) && target(i) {}
    event next before(Iterator i) : call(* Iterator.next()) && target(i) {}

    ptltl : next implies (*)(!next S hasnext)

    @violation {
        System.err.println("! hasNext not called before next");
        __RESET;
    }
}
```

```
package paramere.hasnext;

import java.io.*;
import java.util.*;

HasNext(Iterator i) {
  event hasnext before(Iterator i) : call(* Iterator.hasNext()) && target(i) {}
  event next before(Iterator i) : call(* Iterator.next()) && target(i) {}

  ftltl : [](next -> o(!next U hasnext))

  @violation{
    System.err.println("! hasNext not called before next");
    __RESET;
  }
}
```

properties of Java library APIs

Enumeration (Java 2 Platform SE 5.0)

http://java.sun.com/j2se/1.5.0/docs/api/

Overview Package Class Use Tree Deprecated Index Help

Java™ 2 Platform Standard Ed. 5.0

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Java™ 2 Platform Standard Ed. 5.0

All Classes

Packages

java.applet

EntityReference
EntityResolver
EntityResolver2
Enum
EnumConstantNotPrese
EnumControl
EnumControl.Type
Enumeration
EnumMap
EnumSet
EnumSyntax
Environment
EOFException
Error
ErrorHandler
ErrorListener
ErrorManager
EtchedBorder
Event

java.util

Interface Enumeration<E>

All Known Subinterfaces:

NamingEnumeration<T>

All Known Implementing Classes:

StringTokenizer

R_2: An enumeration should not be propagated after the underlying vector has been changed .

Method Summary

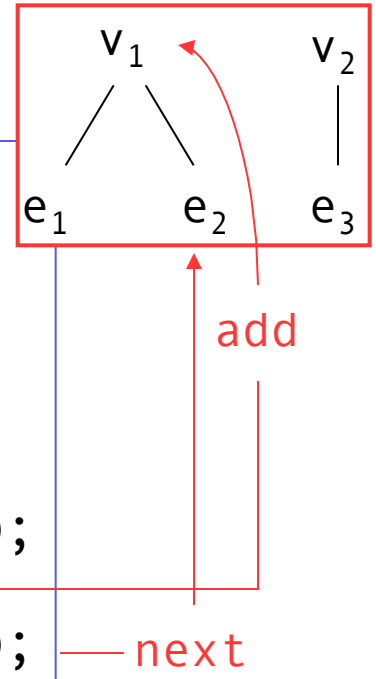
boolean	hasMoreElements() Tests if this enumeration contains more elements.
E	nextElement() Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

Find: 97 Next Previous Highlight all Match case

Downloads

many vectors and enumerators

```
Vector v1 = new Vector();
Vector v2 = new Vector();
v1.add(1); v1.add(2); v2.add(4); v2.add(5);
Enumeration e1 = v1.elements();
Enumeration e2 = v1.elements();
Enumeration e3 = v2.elements();
while(e1.hasMoreElements())print(e1.nextElement());
v1.add(99);
while(e2.hasMoreElements())print(e2.nextElement());
while(e3.hasMoreElements())print(e3.nextElement());
```



events:

```
create(v,e)
next(e)
updatesource(v)
```

recall ERE

```
package paramere.safeenum;

import java.io.*;
import java.util.*;

suffix SafeEnumeration(Vector v, Enumeration e) {
  event create after(Vector v) returning(Enumeration e) :
    call(Enumeration Vector.elements()) && target(v) {}

  event updatesource before(Vector v) :
    (call(* Vector.remove*(..)) || call(* Vector.add*(..)) ) && target(v) {}

  event next before(Enumeration e) :
    call(* Enumeration.nextElement()) && target(e) {}

  ere : create next* updatesource updatesource* next

  @match {
     System.out.println("improper enumerator usage");
  }
}
```

FTLTL

```
package paramere.safeenum;

import java.io.*;
import java.util.*;

SafeEnumeration(Vector v, Enumeration e) {
  event create after(Vector v) returning(Enumeration e) :
    call(Enumeration Vector.elements()) && target(v) {}

  event updatesource before(Vector v) :
    (call(* Vector.remove*(..)) || call(* Vector.add*(..)) ) && target(v) {}

  event next before(Enumeration e) :
    call(* Enumeration.nextElement()) && target(e) {}

  ftltl : [](create -> [](updatesource -> !<>next))

  @violation {
     System.out.println("improper enumerator usage");
  }
}
```

end