

The SPIN Model Checker

Metodi di Verifica del Software

Andrea Corradini – GianLuigi Ferrari

Lezione 6

2011

Slides per gentile concessione di Gerard J. Holzmann

help with properties

OVERVIEW

- [ABOUT](#)
- [PEOPLE](#)
- [FUNDING](#)
- [RELATED PROJECTS](#)

DOCUMENTATION

- THE PATTERNS
- [PROPERTY SPECIFICATIONS](#)

COLLABORATIONS

- [PAPERS](#)

The Patterns

The information in the patterns can be presented in a variety of ways. One organization, illustrated below, is based on classifying the patterns in terms of the kinds of system behaviors they describe.

```
graph TD;
  PP[Property Patterns] --> Occurrence;
  PP --> Order;
  Occurrence --> Absence;
  Occurrence --> Universality;
  Occurrence --> Existence;
  Occurrence --> BoundedExistence[Bounded Existence];
  Order --> Precedence;
  Order --> Response;
  Order --> ChainPrecedence[Chain Precedence];
  Order --> ChainResponse[Chain Response];
```

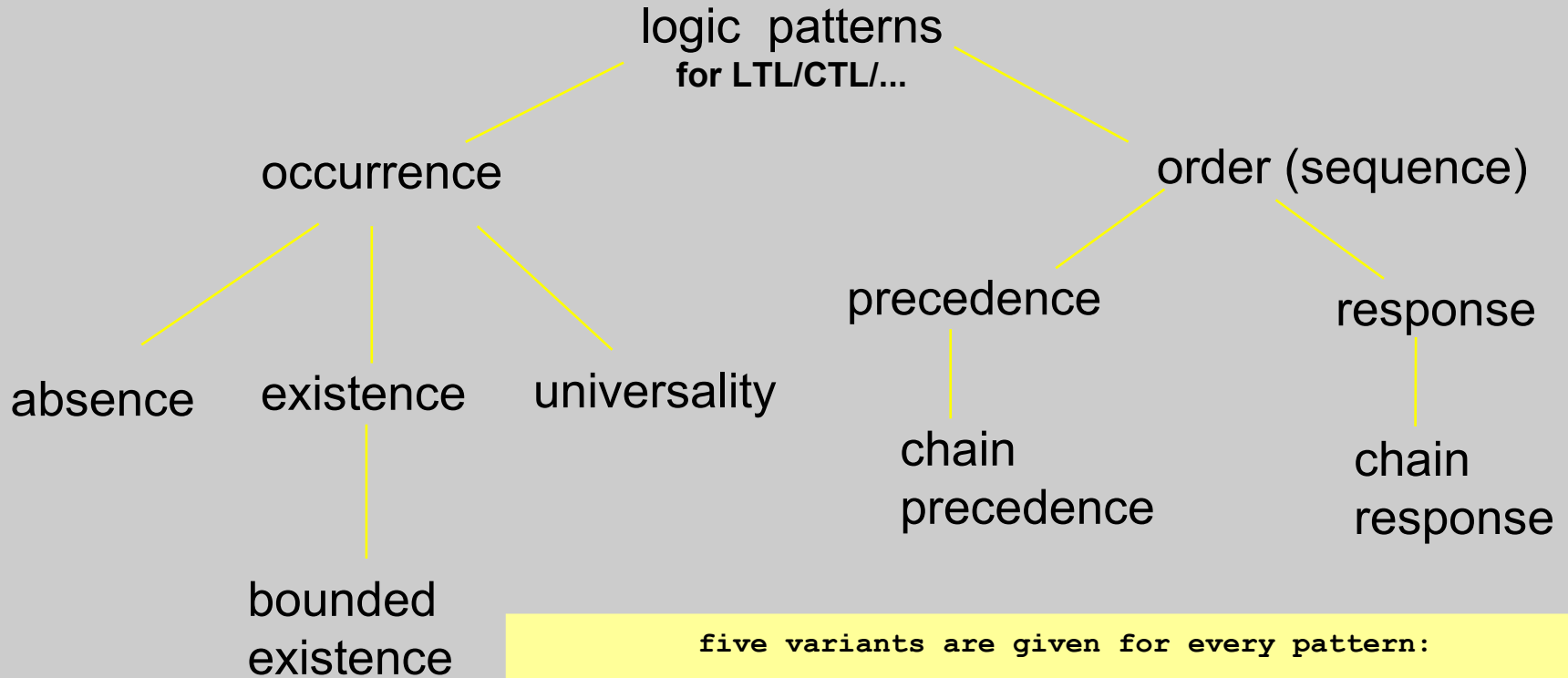
- **Occurrence Patterns** talk about the occurrence of a given event/state during system execution.
- **Order Patterns** talk about relative order in which multiple events/states occur during system execution.
- While not themselves patterns, **Pattern Notes** discuss common ways to vary the existing patterns to suite your needs.

An alternative organization for this information is to group pattern to formalism mappings by specification formalism. The supported formalisms are listed below. Clicking on the formalism will bring you to pages with mappings for each property pattern in that formalisms. We supply the mappings on these formalism-specific pages and you are referred to the complete patterns for information about relationships and example uses.

- **Linear Temporal Logic** (LTL)
- **Computation Tree Logic** (CTL)
- **Graphical Interval Logic** (GIL)

the temporal logic patterns database

<http://patterns.projects.cis.ksu.edu/>



five variants are given for every pattern:

name	example for 'absence' and LTL	#states
globally !p	<code>[](!p)</code>	1
before r	<code><>r -> (!p U r)</code>	4
after q	<code>[](q -> [](!p))</code>	2
between r and q	<code>[]((r && !q && <>q) -> (!p U q))</code>	4
after r until q	<code>[](r && !q -> ((!p U q) []!p))</code>	4



expressiveness of LTL

compared to never claims

(cf. book p. 151)

- never-claims can define all ω -regular word-automata
- propositional linear temporal logic (without quantifiers) defines a *subset* of of this language
 - anything expressable in LTL can be expressed as a never claim
 - but, never claims can also express properties that *cannot* be expressed in LTL
- adding a single existential quantifier over 1 propositional symbol to LTL suffices to extend its expressiveness to all ω -regular word-automata:

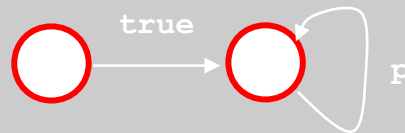
$$\exists p, [] (p \rightarrow \langle \rangle q)$$

- Kousha Etessami's 'temporal message parlor' TMP:
<http://www.bell-labs.com/projects/TMP>

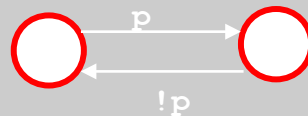
omega-regular properties

(~p. 150 book)

- something not expressible in pure LTL:
 - (p) *can* hold after an even number of execution steps, but *never* holds after an odd number of steps
 - $\square X(p)$ certainly does not capture it:



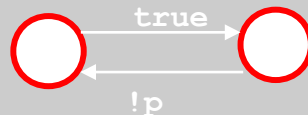
- $p \ \&\& \ \square(p \rightarrow X!p) \ \&\& \ \square(!p \rightarrow Xp)$ does not capture it either (because now p *must* always hold after all even steps):



(1t12ba -f)

$\exists t, !t \ \&\& \ \square(t \rightarrow X!t) \ \&\& \ \square(!t \rightarrow Xt) \ \&\& \ \square(p \rightarrow !t)$

this formula expresses it correctly



LTL compared to other logics

- an LTL formula states a property that must be satisfied for *all* runs starting in the initial system state

```
operators:
!    logical negation
&&  logical and
||   logical or
X  in the next state
U  strong until
U    weak until
<>  eventually
[]   always
```

basic temporal operators are **red**
gray operators can be derived:

```
 $\varphi \ || \ \varphi == !( \ !\varphi \ \&\& \ !\varphi )$ 
 $\langle \> \ \varphi == \text{true} \ U \ \varphi$ 
 $[\ ] \ \varphi == !\langle \> \ !\varphi$ 
 $\varphi_1 \ U \ \varphi_2 == [\ ] \ \varphi_1 \ || \ (\varphi_1 \ U \ \varphi_2)$ 
```

grammar:

propositional formulae:

```
p
!f
(f)
f && f
f || f
```

temporal formulae:

```
f
!φ
(φ)
φ && φ
φ || φ
X φ
φ U φ
<> φ
[] φ
```

p is an arbitrary propositional symbol
f is an arbitrary propositional formula
φ is an arbitrary temporal formula

the logic CTL*

- CTL* is a *branching* time logic
 - introduces explicit path quantifiers \forall and \exists
 - often used in hardware verification
 - by convention, one often uses F for $\langle \rangle$ and G for $[]$

!	logical negation
&&	logical and
	logical or
E	there exists a path
A	for all paths
X	in the next state
U	until (strong)
$\langle \rangle$ F	finally (eventually)
$[]$ G	generally (always)

the **red** operator is new
gray operators can be derived:
 $\varphi || \varphi == !(!\varphi \&\& !\varphi)$
 $A \varphi == !E ! \varphi$
 $F \varphi == true U \varphi$
 $G \varphi == !F !\varphi$
 $\varphi_1 U \varphi_2 == G \varphi_1 || (\varphi_1 U \varphi_2)$

```
state formulae:  p
                  !f
                  (f)
                  f && f
                  f || f
                  E  $\varphi$ 

path formulae:   f
                  ! $\varphi$ 
                  ( $\varphi$ )
                   $\varphi \&\& \varphi$ 
                   $\varphi || \varphi$ 
                  X  $\varphi$ 
                   $\varphi U \varphi$ 
                  F  $\varphi$ 
                  G  $\varphi$ 
```

p is a propositional symbol
f is an arbitrary state formula
 φ is an arbitrary path formula

the subset CTL

- CTL is the fragment of CTL* in which at most one occurrence of the operators X and U can occur within the scope of a path quantifier (A or E):

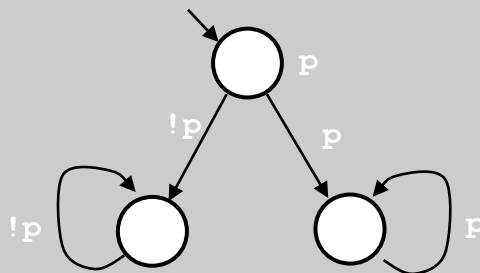
valid CTL formula:

p
 $!\varphi$
 $\varphi_1 \ \&\& \ \varphi_2$
 $E \ X \ f$
 $E(f_1 \ U \ f_2)$
 $A(f_1 \ U \ f_2)$

p a propositional symbol
 f is a state formula
 φ a path formula

derivable:

$EF \ f == E(\text{true} \ U \ f)$
 $AF \ f == A(\text{true} \ U \ f)$
 $EG \ f == !AF \ !f$
 $AG \ f == !EF \ !f$
 $AX \ f == !EX \ !f$



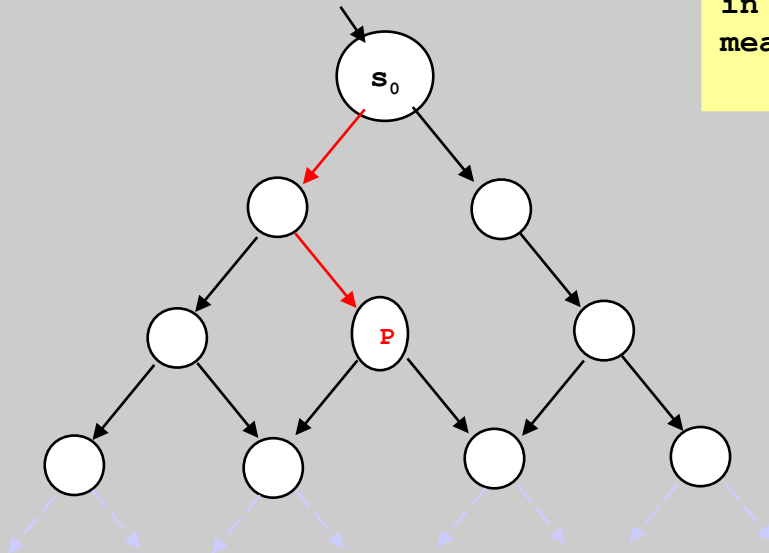
$EG(p)$

satisfied

$EXG(!p)$

satisfied

comparison



in LTL: $\langle \rangle p$
means: $A \langle \rangle p$ for *all* computations starting
at initial state s_0 $\langle \rangle p$ holds

in CTL one can say:

$EF(p)$ there *exists* a computation
 where $\langle \rangle p$ holds

$AF(p)$ in *all* computations $\langle \rangle p$ holds

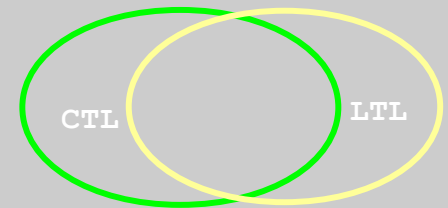
$AG(p)$ always invariantly p

$EG(p)$ there *exists* a computation
 where p is invariantly true
etc.

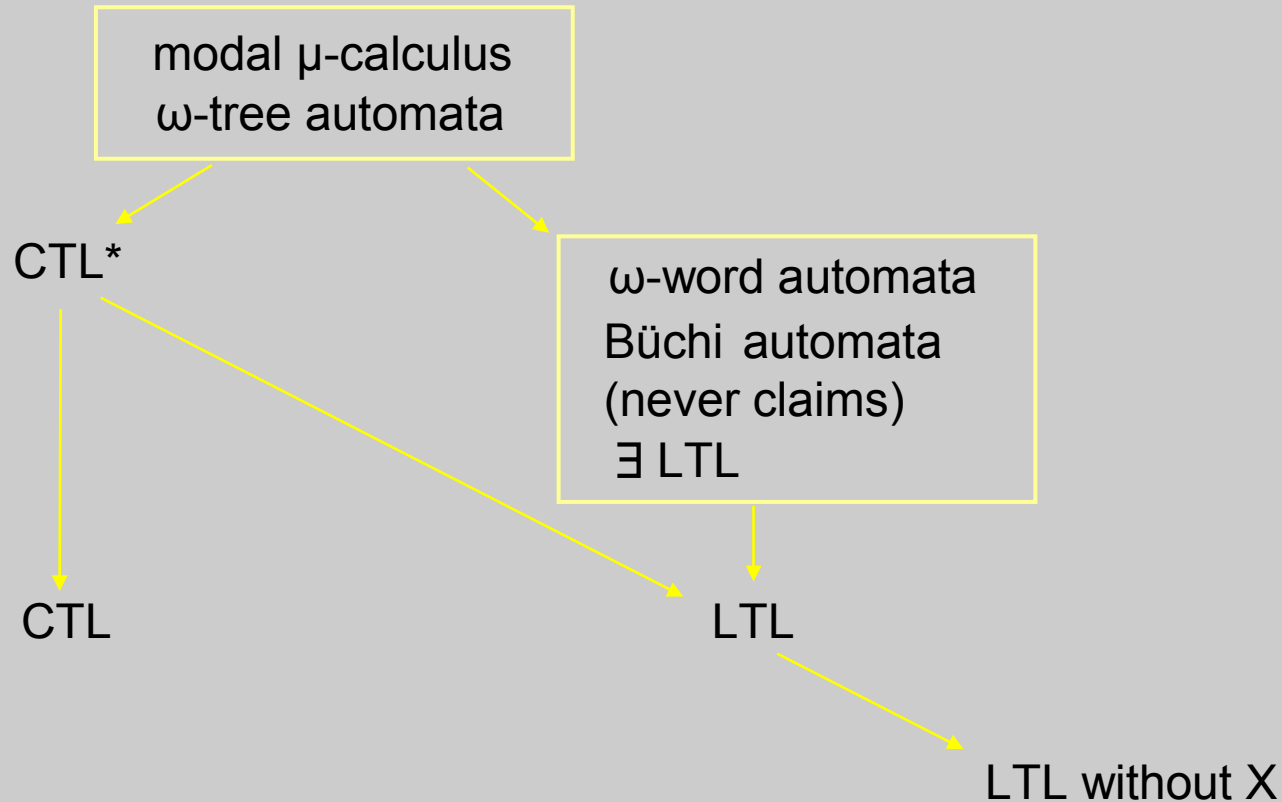
the expressiveness of LTL compared with CTL* and CTL

(cf. Appendix B)

- CTL* and CTL define subsets of ω -regular *tree* automata
 - *tree* automata are more expressive than *word* automata
 - a CTL formula is generally satisfied by a *tree* of possible runs, not a single run
 - both LTL and CTL can be defined as subsets of CTL*
 - but, LTL and CTL are *not* comparable in expressiveness
 - they overlap, but neither includes the other
 - LTL can express properties that CTL cannot
 - CTL cannot express properties of the type $\square\langle\rangle(p)$
 - $\square\langle\rangle p$ can formalize *fairness* constraints in LTL
 - CTL can express properties that LTL cannot
 - LTL cannot express properties of the type AGEF(p)
 - AGEF(p) can formalize *reset* properties in CTL:
from every system state it is *possible* to return to the initial state



expressiveness



- same box means 'equally expressive'
- single arrow means 'more expressive than'
- no arrow means 'expressiveness is not comparable'