The SPIN Model Checker

Metodi di Verifica del Software

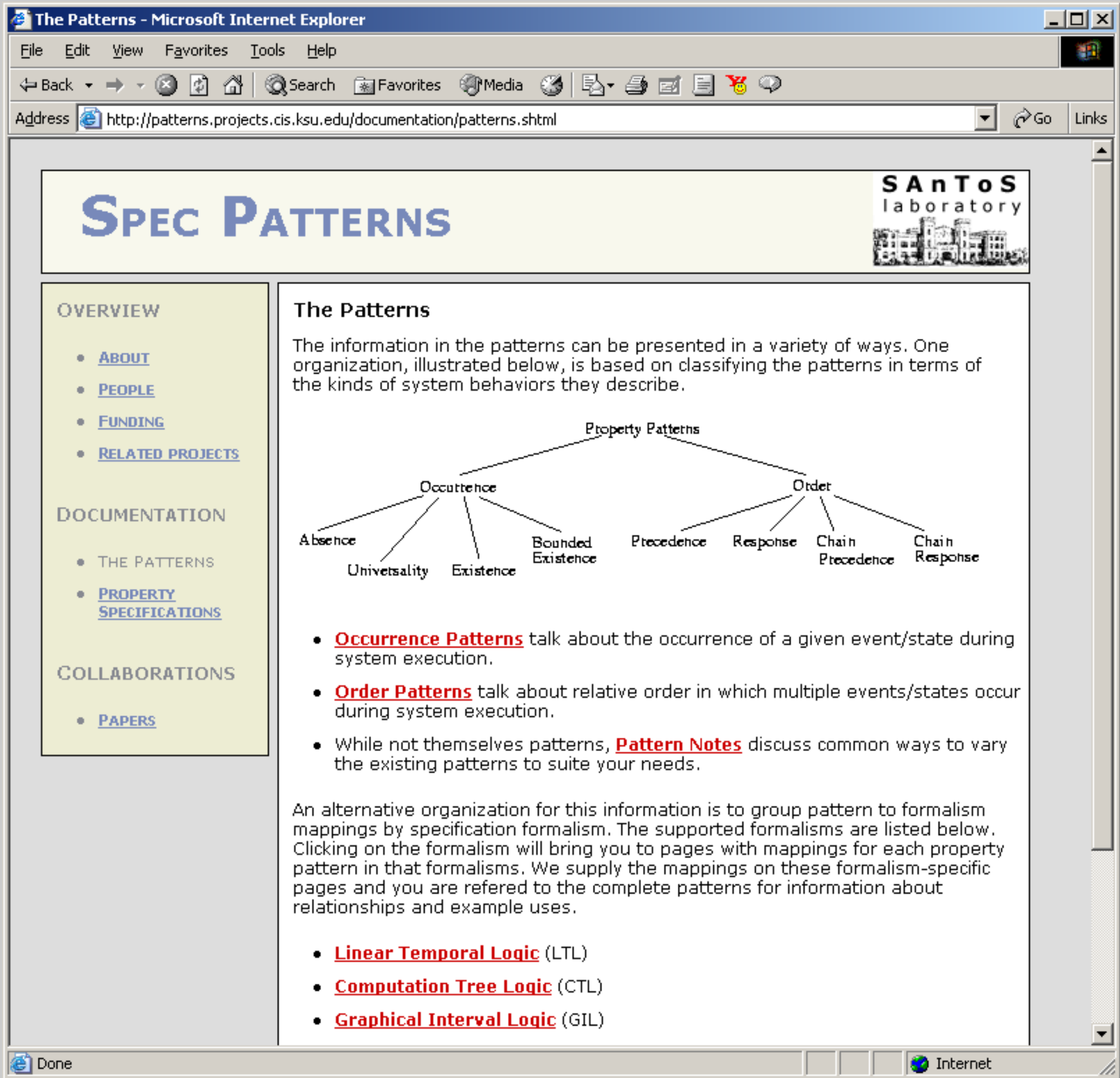
Andrea Corradini – GianLuigi Ferrari

Lezione 6

2011

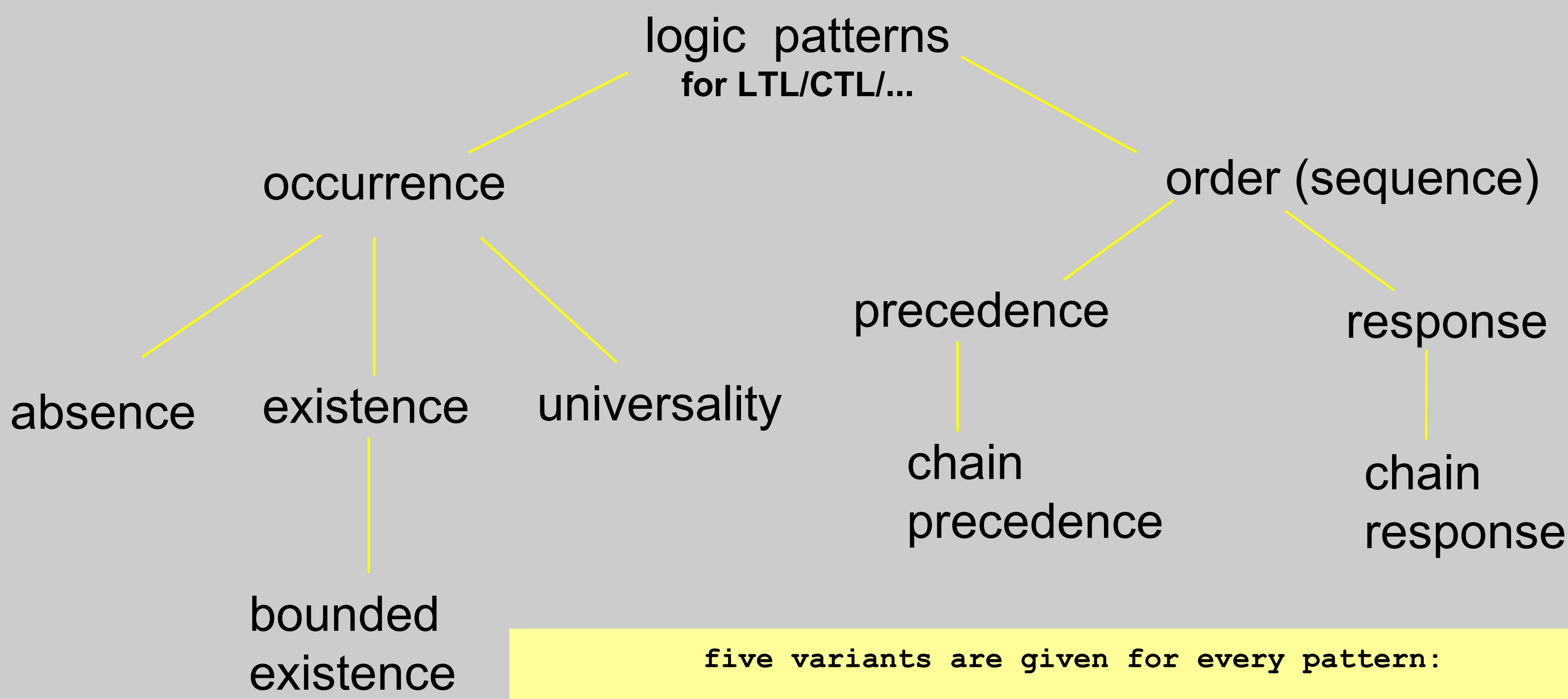
Slides per gentile concessione di Gerard J. Holzmann

help with properties



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	`[](!p)`	1
before r	`<>r -> (!p U r)`	4
after q	`[](q -> [](!p))`	2
between r and q	`[]((r && !q && <>q) -> (!p U q))`	4
after r until q	`[](r && !q -> ((!p U q) || []!p))`	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:

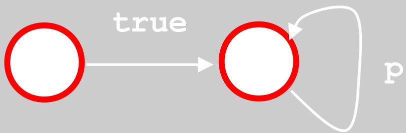
  ∃p, [ ](p -> <> q)

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

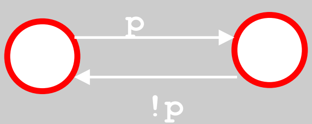
omega-regular properties

(~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
  - [] X (p) certainly does not capture it:



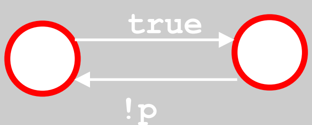
  - p && [](p -> X!p) && [](!p -> Xp) does not capture it either (because now p *must* always hold after all even steps):



(ltl2ba -f)

∃t, !t && [] (t -> X !t) && [](!t -> Xt) && [](p -> !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:

φ || φ == !(!φ && !φ)
<> φ   == true U φ
[] φ   == !<> !φ
φ1 U φ2 == [] φ1 || (φ1 U φ2)
```

```
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 ∀ and ∃
  - often used in hardware verification
  - by convention, one often uses F for <> 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)
<>	F	finally (eventually)
[]	G	generally (always)

the red operator is new
gray operators can be derived:
```
φ || φ == !(!φ && !φ)
A φ == !E ! φ
F φ == true U φ
G φ == !F !φ
```
φ_1 *U* φ_2 == G φ_1 || (φ_1 U φ_2)

```
state formulae:  p
                 !f
                 (f)
                 f && f
                 f || f
                 E φ
path formulae:   f
                 !φ
                 (φ)
                 φ && φ
                 φ || φ
                 X φ
                 φ U φ
                 F φ
                 G φ
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):

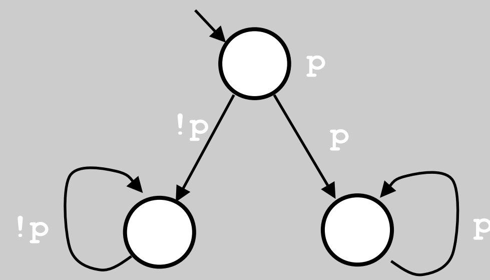
valid CTL formula:

p
!φ
φ_1 && φ_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

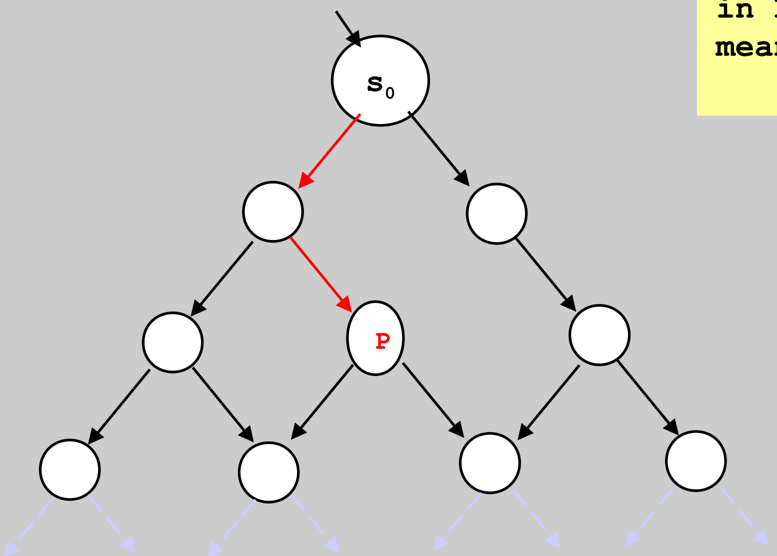
derivable:

EF f== E(true U f)
AF f== A(true U f)
EG f== !AF !f
AG f== !EF !f
AX f== !EX !f



EG(p)	satisfied
EXG(!p)	satisfied

comparison



in LTL: **<>p**
means: **A <>p for *all* computations starting at initial state s_0 <>p holds**

in CTL one can say:

EF(p) **there *exists* a computation where <>p holds**

AF(p) **in *all* computations <>p holds**

AG(p) **always invariantly p**

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

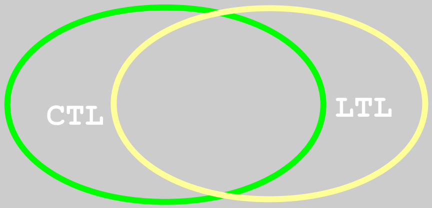
etc.

the expressiveness of LTL compared with CTL* and CTL

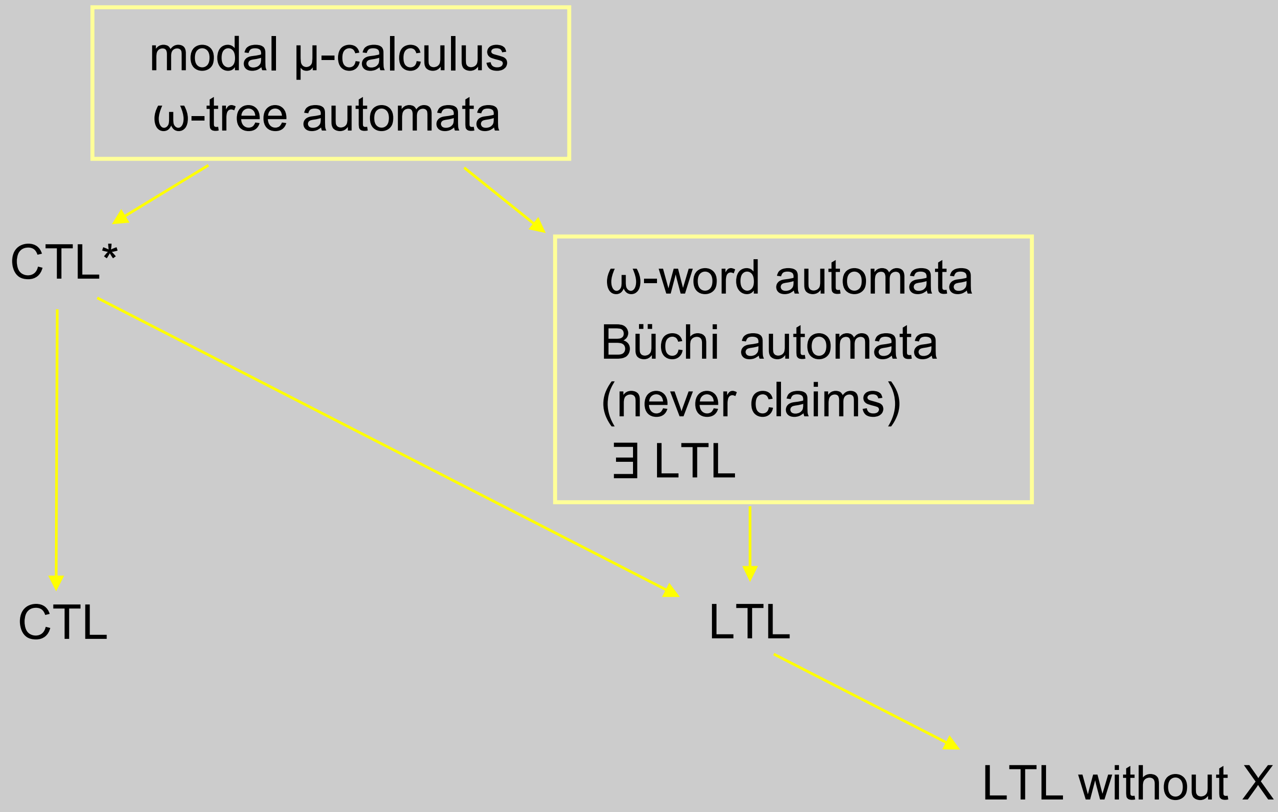
(cf. Appendix B)

- 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 []<>(p)
    - []<>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'