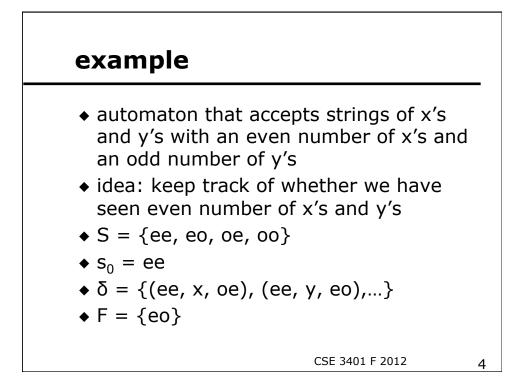




- a fsa accepts an input sequence from an alphabet Σ if, starting in the designated starting state, scanning the input sequence leaves the automaton in a final state
- sometimes called recognition
- e.g. automaton that accepts strings of x's and y's with an even number of x's and an odd number of y's

CSE 3401 F 2012



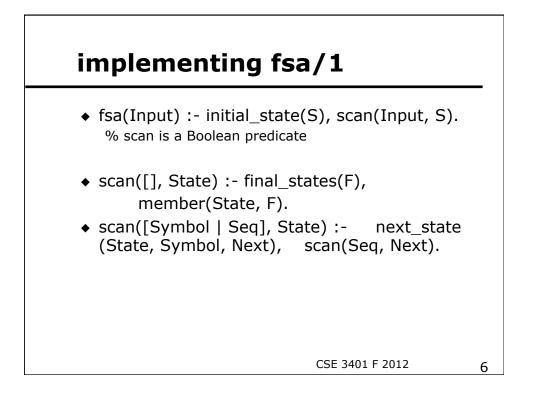
implementation

- fsa(Input) succeeds if and only if the fsa accepts or recognizes the sequence (list) Input.
- initial state represented by a predicate

 initial_state(State)
- final states represented by a predicate

 final_states(List)
- state transition table represented by a predicate
 - next_state(State, InputSymbol, NextState)
- note: next_state need not be a function

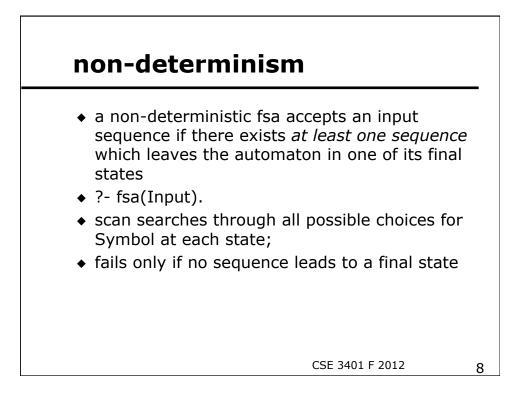
CSE 3401 F 2012

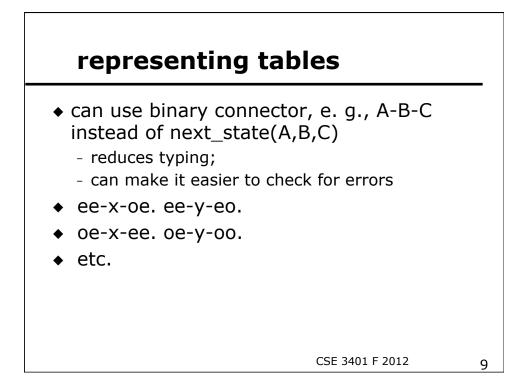


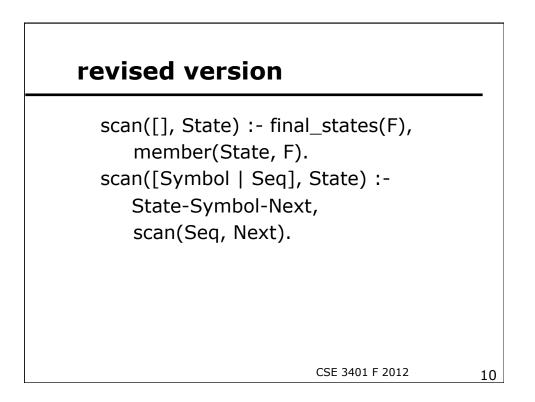


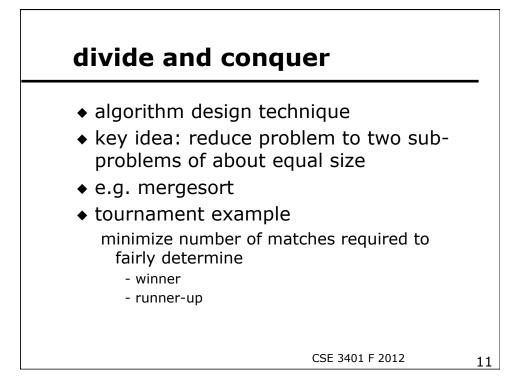
- scan uses pumping/result propagation
- carries around current state and remainder of input sequence
- if FSA is deterministic, when end of input is reached, can make an accept/reject decision immediately; tail recursion optimization can be applied
- if FSA is nondeterministic, may have to backtrack; must keep track of remaining alternatives on execution stack

CSE 3401 F 2012

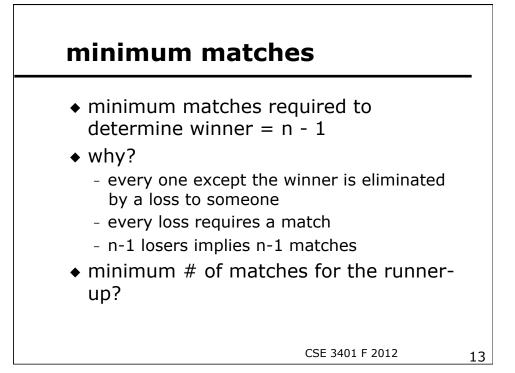


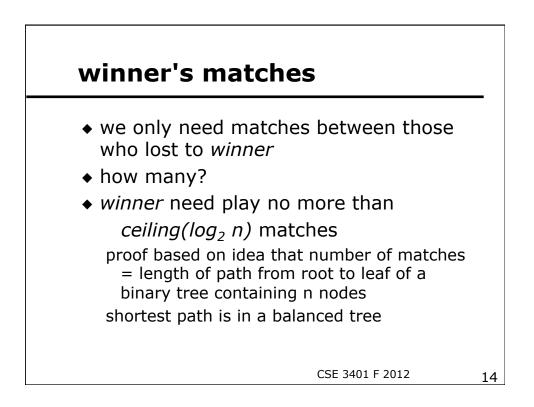


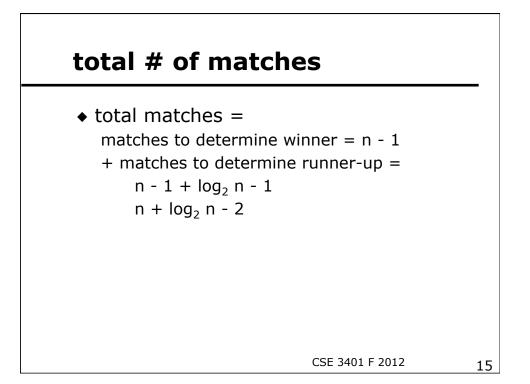


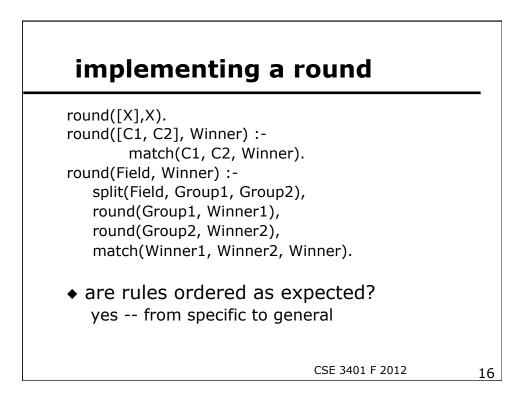


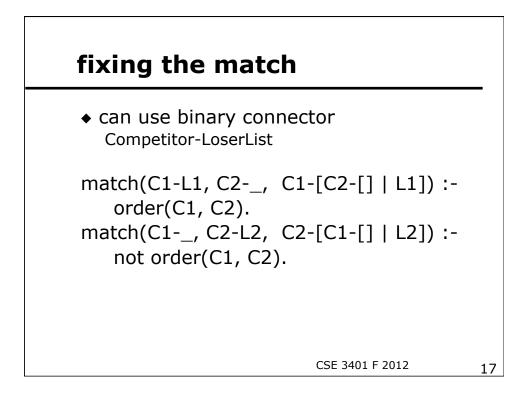


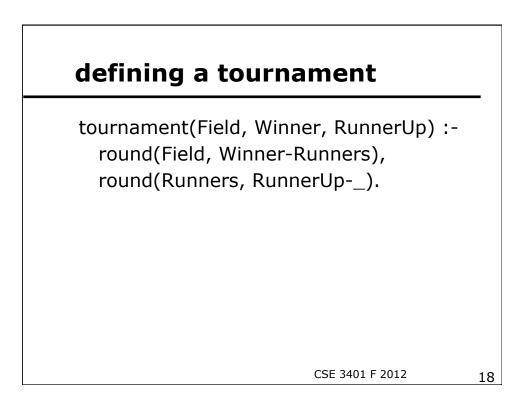


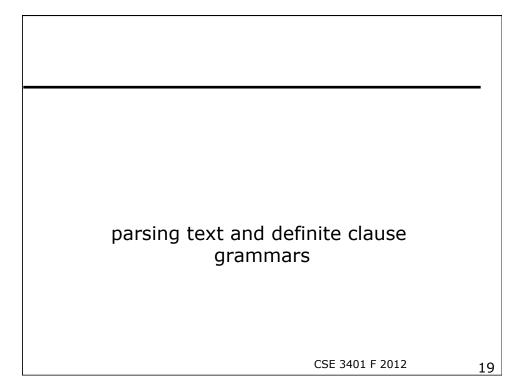




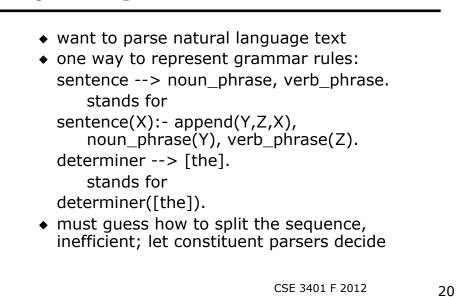








Prolog representation for parsing text





- sentence(S0,S):noun_phrase(S0,S1), verb_phrase(S1,S).
- determiner([the | S],S).
- 1st argument is sequence to parse and 2nd argument is what is left after removing it
- Rule means "there is a sentence between S0 and S if ..."
- ?-sentence([the, boy, drinks, the, juice], []). succeeds
- ?-noun_phrase([the, boy, drinks, the, juice], R). succeeds with R = [drinks, the, juice]

CSE 3401 F 2012

21

definite clause grammar (DCG) notation

```
sentence --> noun_phrase,verb_phrase.
    stands for
sentence(S0,S):- noun_phrase(S0,S1),
    verb_phrase(S1,S).
determiner --> [the].
    stands for
determiner([the|S],S).
```

CSE 3401 F 2012

enforcing constraints between constituents

- suppose we want to enforce number agreement
- can add extra argument to pass this info between constituents
- noun_phrase(N) --> determiner(N), noun(N).
- noun(singular) --> [boy].
- noun(plural) --> [boys].
- determiner(singular) --> [a].
- ?- noun_phrase(N,[a, boys],[]). fails
- ?- noun_phrase(N,[a, boy],[]). succeeds with N = singular

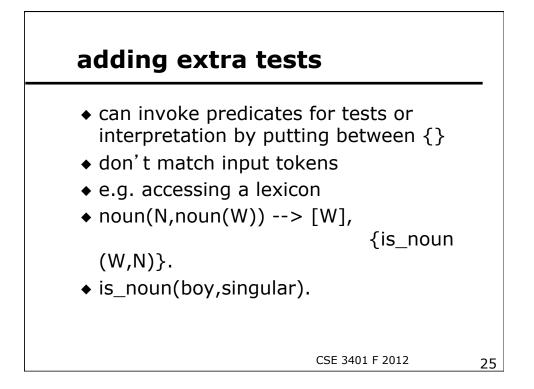
CSE 3401 F 2012

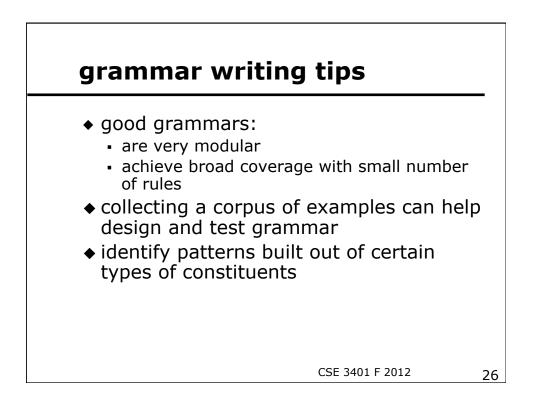
23

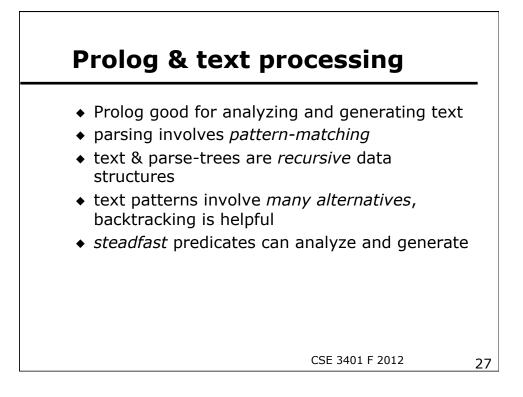
returning a parse tree or interpretation

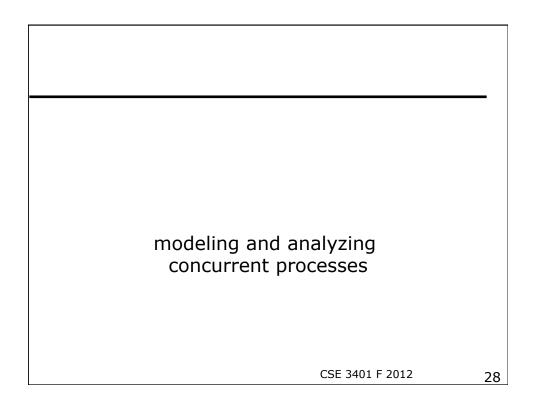
- Extra arguments can also be used to return a parse tree or interpretation
- noun_phrase(np(D,N)) --> determiner(D), noun(N).
- determiner(determiner(a)) --> [a].
- noun(noun(boy)) --> [boy].
- ?- noun_phrase(PT,[a, boy],[]). succeeds with PT = np(determiner(a),noun(boy))

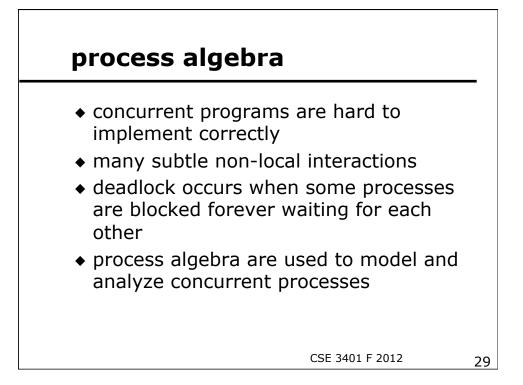
CSE 3401 F 2012

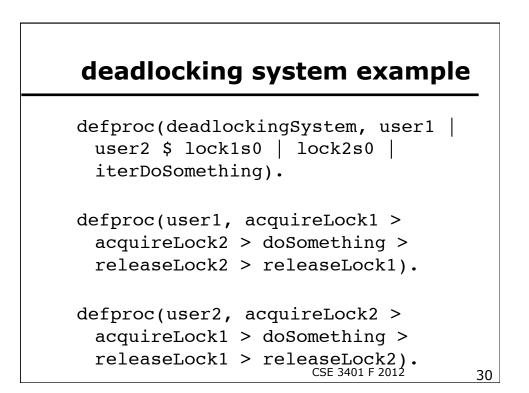






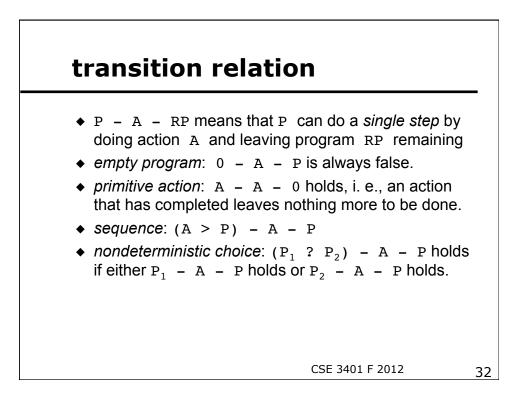






deadlocking system example

```
defproc(lock1s0,
    acquireLock1 > lock1s1 ? 0).
defproc(lock1s1, releaseLock1 > lock1s0).
defproc(lock2s0,
    acquireLock2 > lock2s1 ? 0).
defproc(lock2s1,releaseLock2 > lock2s0).
defproc(iterDoSomething,
    doSomething > iterDoSomething ? 0).
CSE 3401 F 2012 31
```



transition relation

- interleaved concurrency: $(P_1 | P_2) A P$ holds if either $P_1 - A - P_{11}$ holds and $P = (P_{11} | P_2)$, or $P_2 - A - P_{21}$ holds and $P = (P_1 | P_{21})$
- synchronized concurrency: (P₁ \$ P₂) A P holds if both P₁ - A - P₁₁ holds and P₂ - A - P₂₁ holds and P = (P₁₁ \$ P₂₁)
- recursive procedures: ProcName A P holds if ProcName is the name of a procedure that has body B and B - A - P holds.

CSE 3401 F 2012

33

can check properties by searching process graph a process has an *infinite execution* if there is a cycle in its configuration graph e.g. defproc(aloop, a > aloop) has_infinite_run(P):- P - _ - PN, has_infinite_run(PN,[P]).

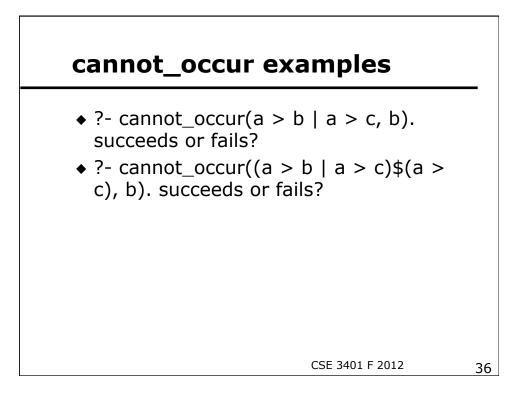
- has_infinite_run(P,V):- member(P,V), !.
- has_infinite_run(P,V):- P _ PN, has_infinite_run(PN,[P|V]).

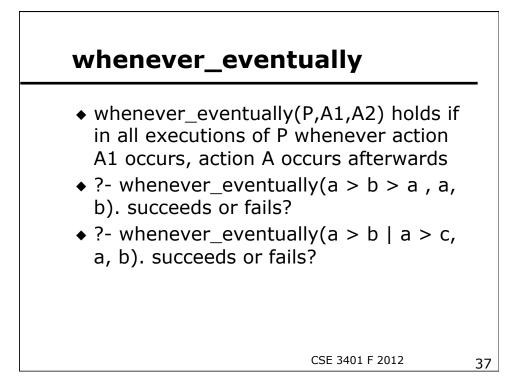
CSE 3401 F 2012

checking properties by searching process graph

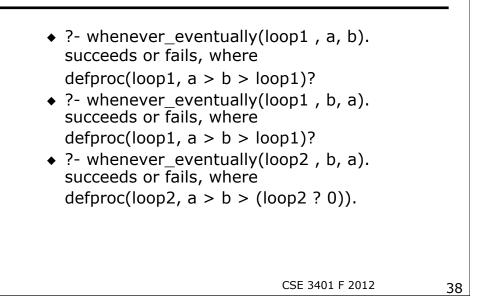
- cannot_occur(P,A) holds if no execution of P where action A occurs
- ◆ search graph for a transition P1 A P2
- useful built-in predicate: forall(+Cond, +Action) holds iff for all bindings of Cond, Action succeeds
- e.g. forall(member(C,[8,3,9]), C >= 3) succeeds

CSE 3401 F 2012





whenever_eventually examples



deadlock_free

- deadlock_free(P) holds if process P cannot reach a deadlocked configuration, i.e. one where the remaining process is not final, but no transition is possible
- ?- deadlock_free(a \$ a). succeeds or fails?
- ?- deadlock_free(a > a \$ a). succeeds or fails?

CSE 3401 F 2012

