

The Specification Logic



Linear Time Temporal Logic (LTL)

- LTL is well known in the area of formal verification, i.e., model checking
- special kind of a modal logic
- intuitive, but formal semantics
- **easy-to-use human interfaces are developed in various research projects**
 - wave form editors (e.g., OFISS Oldenburg)
 - natural language interfaces (e.g., Prosper project)

LTL is explained in detail in

[E.A.Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B, Amsterdam, Elsevier Science Publishers, 1990.]

Elements of LTL Formulas



Atomic propositions

- a,b,c Represent (Boolean) signals
- oneHot, less Boolean functions

Boolean operators

- not Negation
- and, or Conjunction, disjunction
- xor Mutual exclusion
- \rightarrow Implication

Temporal operators

- $X_{[n]} f$ f holds at the **next** time step
- $F_{[m,n]} f$ f holds **eventually**
- $G_{[m,n]} f$ f **always** holds

Expressiveness of LTL (Examples)



Classes of specifiable properties:

➤ Safety properties

- "sig1 and sig2 are never true at the same time"
 $G(\text{not}(\text{sig1 and sig2}))$

➤ Liveness

- "Every request is acknowledged within m to n steps"
 $G(\text{req} \rightarrow F_{[m,n]} \text{ack})$

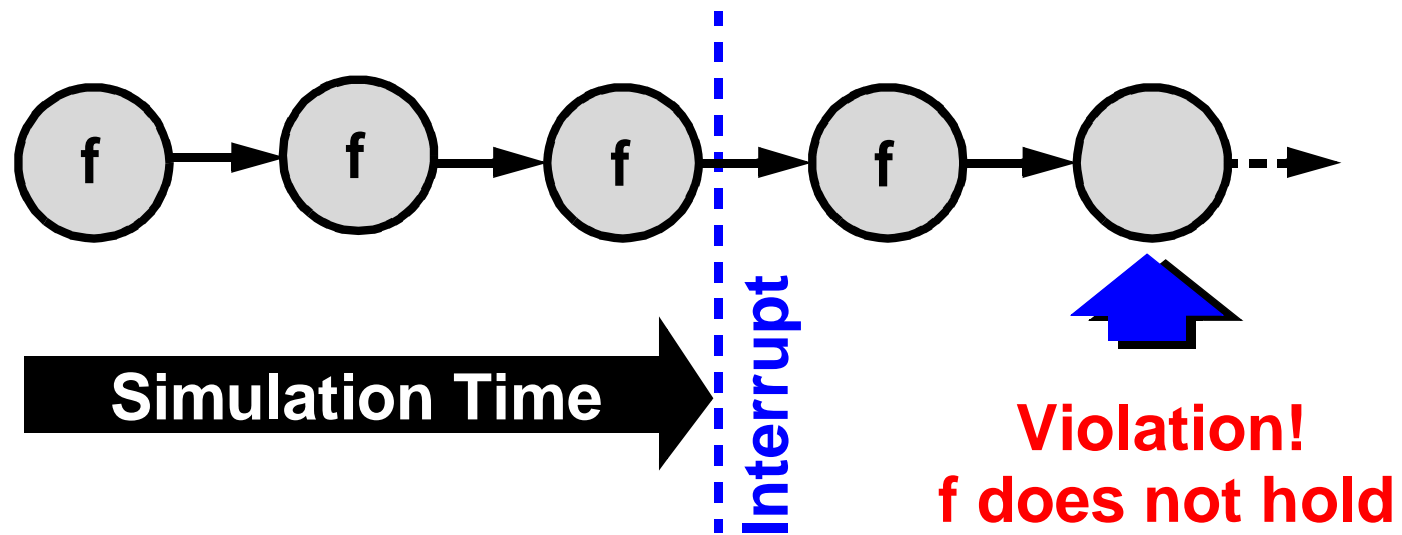
➤ Fairness

- "sig1 is true at least every n time steps"
 $G F_{[n]}(\text{sig1})$

New Semantics



G f "f always holds"



Validity of "G f" can not yet be decided!

=> Three-valued logic required!

Three cases must be distinguished



(three-values logic)

Case 1:

f is **true** for trace T if f is true for all path extensions T' of T

Case 2:

f is **false** for trace T if there is no path extensions T' of T for which f holds

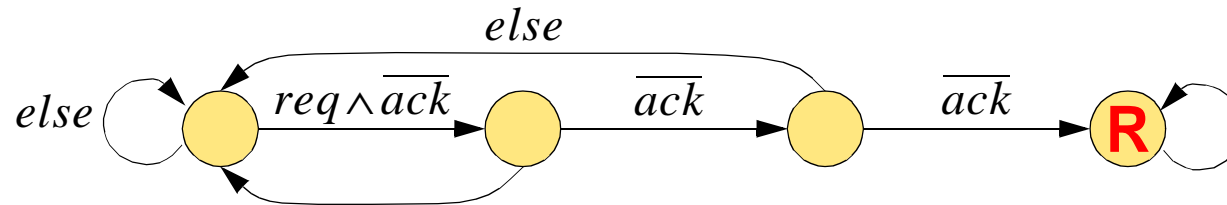
Case 3:

f is **pending**, otherwise

Approach



- ▶ start **simulation**
- ▶ given the LTL formula in the system description, e.g. $G(\text{req} \rightarrow F_{[2]} \text{ack})$
- ▶ translate the formula to an **AR-automaton** for accepting or rejecting in the future



- ▶ feeding the AR-Automaton with traces while **simulation** continues

Translation to AR-Automata



- ▶ an AR-automaton is a finite state machine:

$$A = (S, \rightarrow, A, R, s_0)$$

Translation to AR-Automata



- ▶ an AR-automaton is a finite state machine:

$$A = (S, \rightarrow, A, R, s_0)$$

- ▶ constructing **bottom-up** in the formula the corresponding AR-automaton

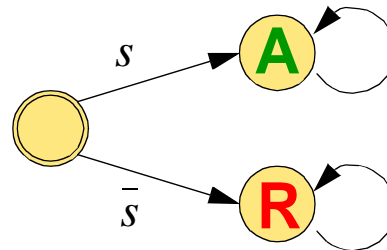
Translation to AR-Automata



- ▶ an AR-automaton is a finite state machine:

$$A = (S, \rightarrow, A, R, s_0)$$

- ▶ constructing **bottom-up** in the formula the corresponding AR-automaton
- ▶ the AR-automaton for **propositions**



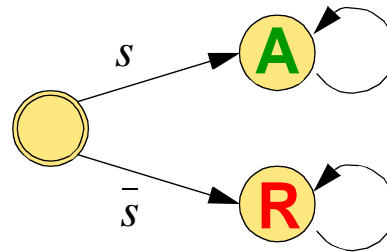
Translation to AR-Automata



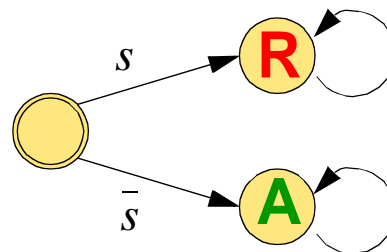
- ▶ an AR-automaton is a finite state machine:

$$A = (S, \rightarrow, A, R, s_0)$$

- ▶ constructing **bottom-up** in the formula the corresponding AR-automaton
- ▶ the AR-automaton for **propositions**



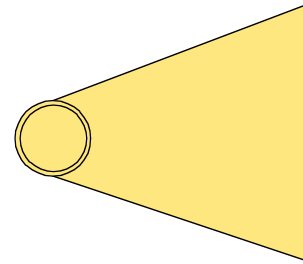
- ▶ **negation**: swapping accept and reject states



Translation to AR-Automata



- ▶ the AR-automaton for the **Xf** Operator

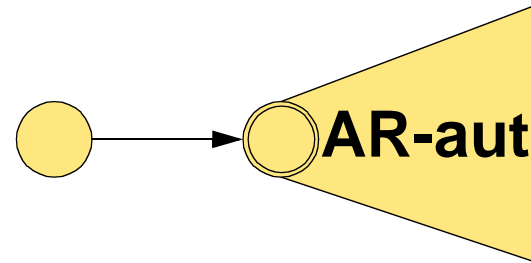


AR-automaton
for formula **f**

Translation to AR-Automata



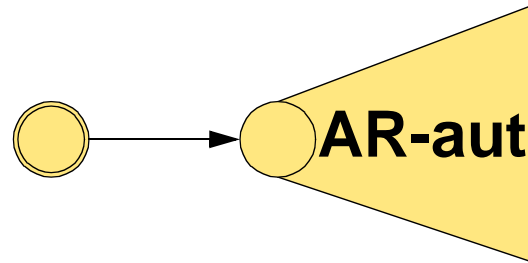
- ▶ the AR-automaton for the **X** Operator



Translation to AR-Automata



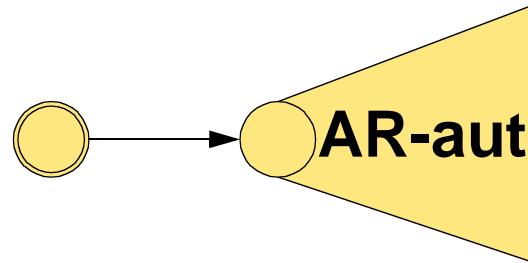
- ▶ the AR-automaton for the **X** Operator



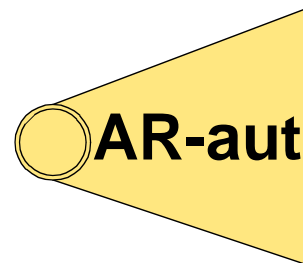
Translation to AR-Automata



- ▶ the AR-automaton for the **X** Operator



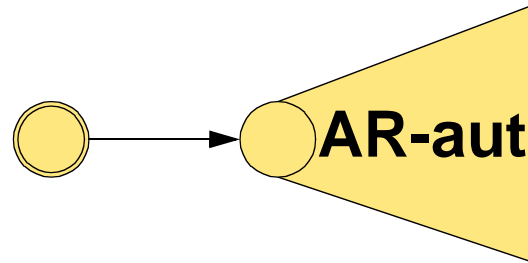
- ▶ the AR-automaton for the **F** and **G** operator



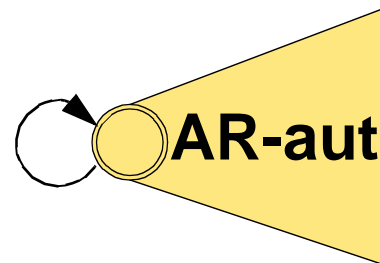
Translation to AR-Automata



- ▶ the AR-automaton for the **X** Operator



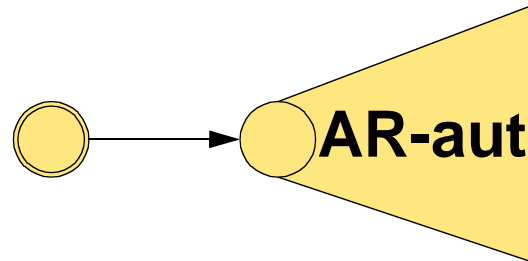
- ▶ the AR-automaton for the **F** and **G** operator



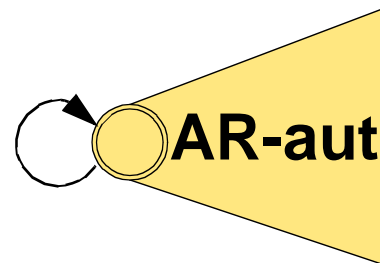
Translation to AR-Automata



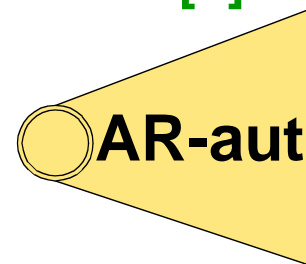
- ▶ the AR-automaton for the **X** Operator



- ▶ the AR-automaton for the **F** and **G** operator



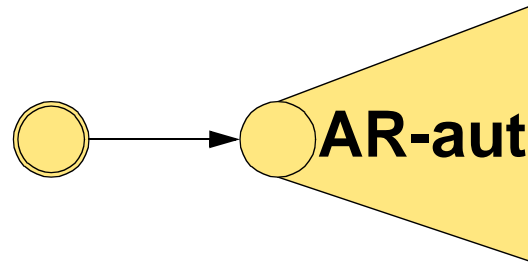
- ▶ the AR-automaton for **F**_[n] and **G**_[n]



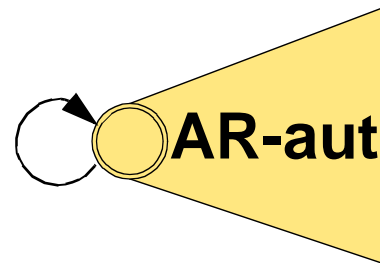
Translation to AR-Automata



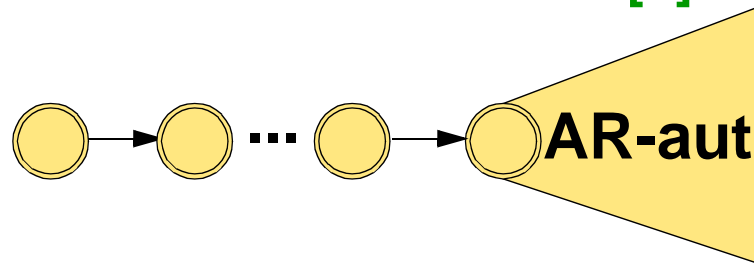
- ▶ the AR-automaton for the **X** Operator



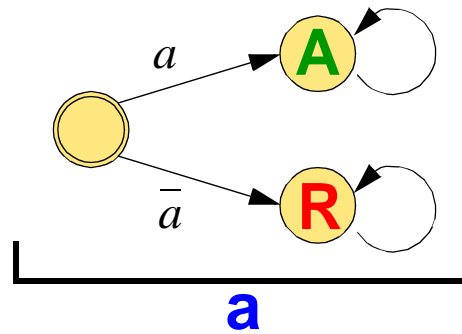
- ▶ the AR-automaton for the **F** and **G** operator



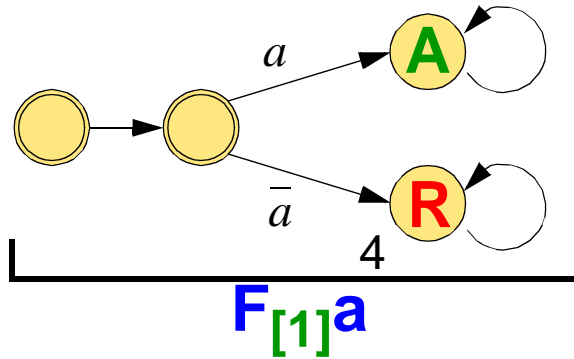
- ▶ the AR-automaton for **F**_[n] and **G**_[n]



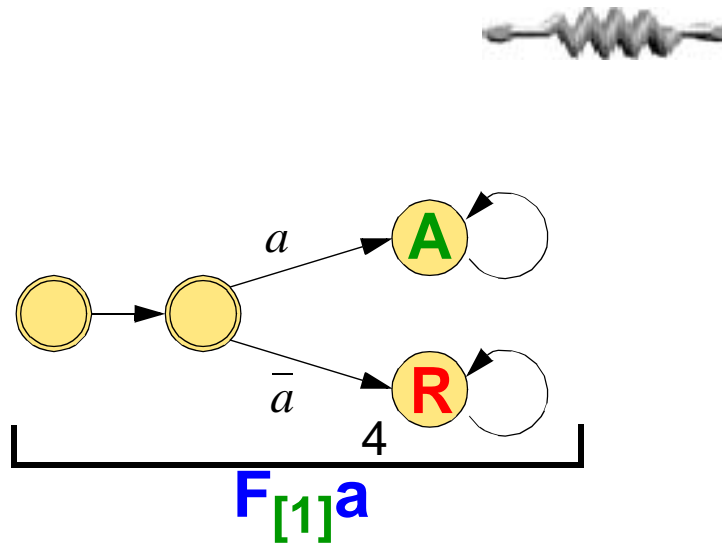
Example: $F_{[1]}a$



Example: $F_{[1]}a$

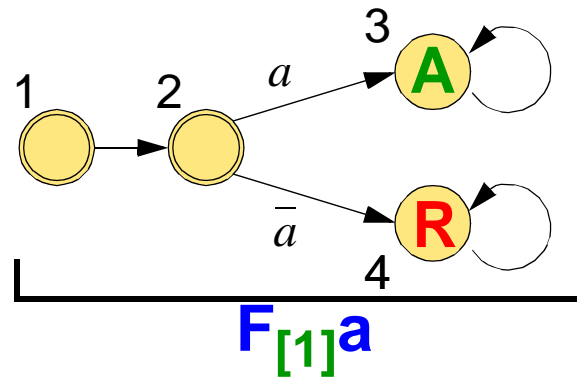


Example: $F_{[1]}a$



Problem:
construction may result in nondet. AR-automata

Removing Nondeterminisms



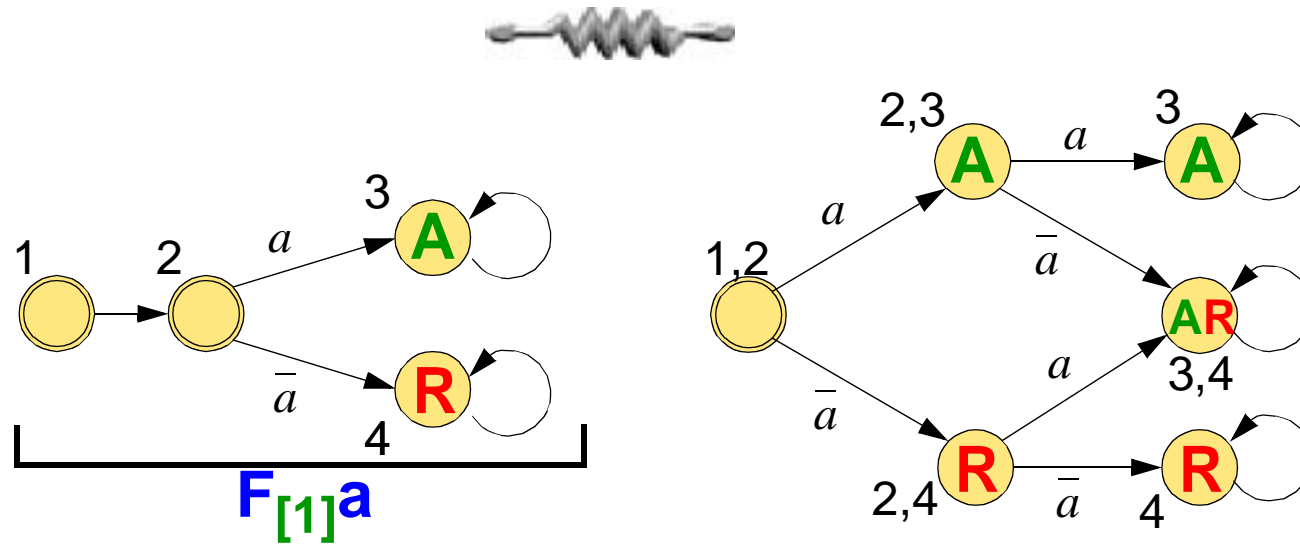
Problem:

construction may result in nondet. AR-automata

Solution:

Using standard FSM-operations for removing non-determinisms (building set states)

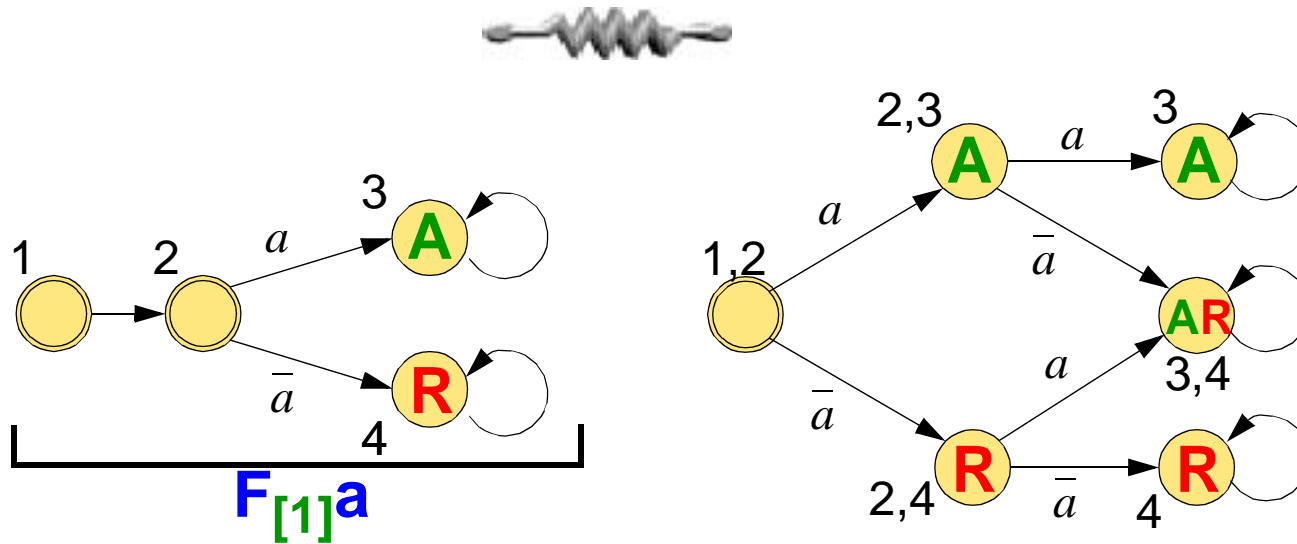
Removing Nondeterminisms



Problem:
 construction may result in nondet. AR-automata

Solution:
 Using standard FSM-operations for removing nondeterminisms (building set states)

Removing Nondeterminisms



Problem:

construction may result in nondet. AR-automata

Solution:

Using standard FSM-operations for removing nondeterminisms (building set states)

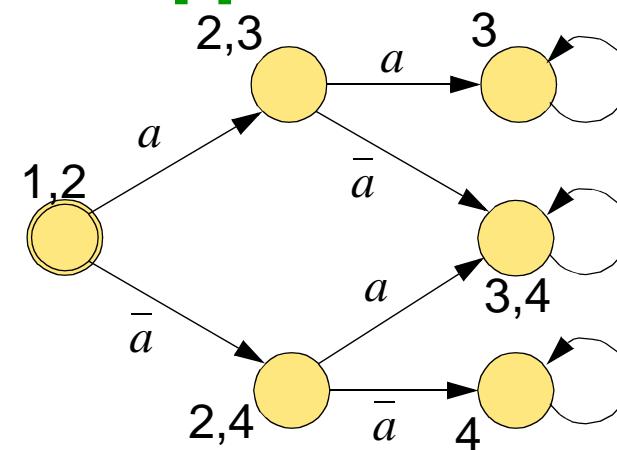
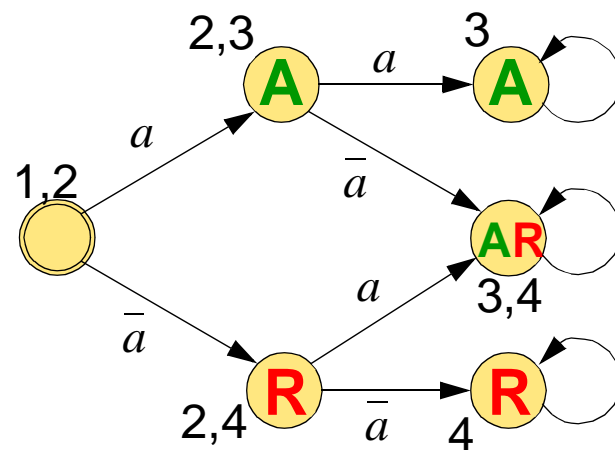
How to handle the labels in the new automaton?

Label Inheritance Patterns



strong	weak
a state will be labelled with "A" if all sub-states are labelled with "A"	a state will be labelled with "A" if one sub-state is labelled with "A"
a state will be labelled with "R" if one sub-state is labelled with "R"	a state will be labelled with "R" if all sub-states are labelled with "R"

Weak inheritance lead to the $F_{[1]}$ -operator

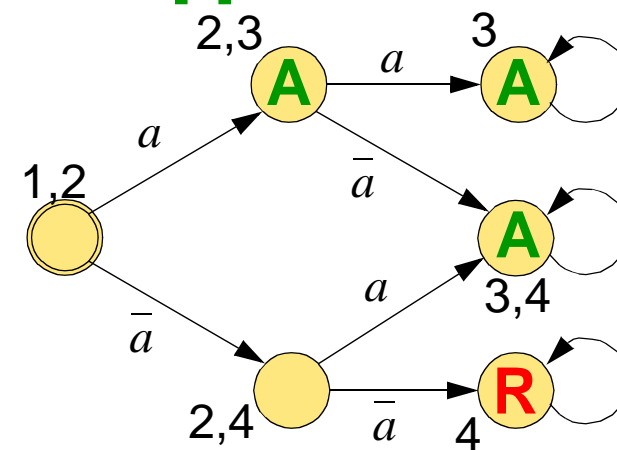
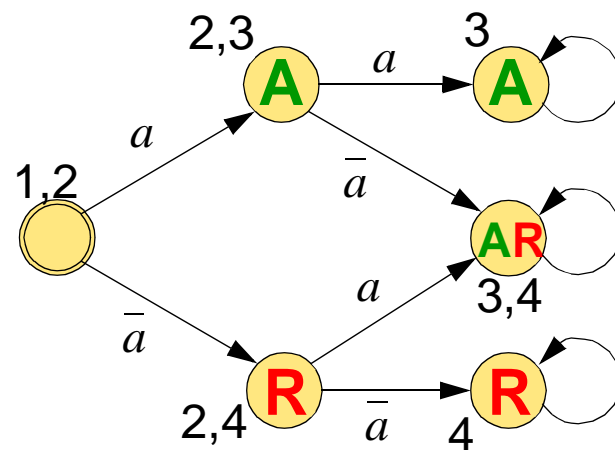


Label Inheritance Patterns



strong	weak
a state will be labelled with "A" if all sub-states are labelled with "A"	a state will be labelled with "A" if one sub-state is labelled with "A"
a state will be labelled with "R" if one sub-state is labelled with "R"	a state will be labelled with "R" if all sub-states are labelled with "R"

Weak inheritance lead to the $F_{[1]}$ -operator



Removing Nondeterminisms



- ▶ The "nondeterminism remove" operation has to be applied after **each** construction step
- ▶ removing nondeterminisms blows up the state space of the AR-automata

using reduction methods for decreasing the number of states

State Space Reduction



Many states are bisimilar
apply bisimulation reduction [Miln92]

- + decreased number of states
- time intensive computation

State Space Reduction



Many states are bisimilar

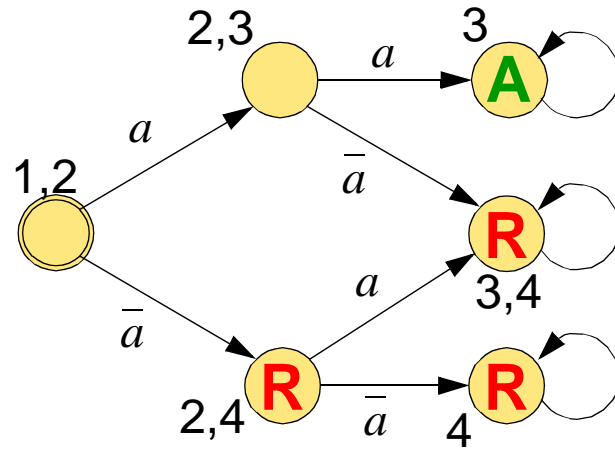
apply bisimulation reduction [Miln92]

- + decreased number of states
- time intensive computation

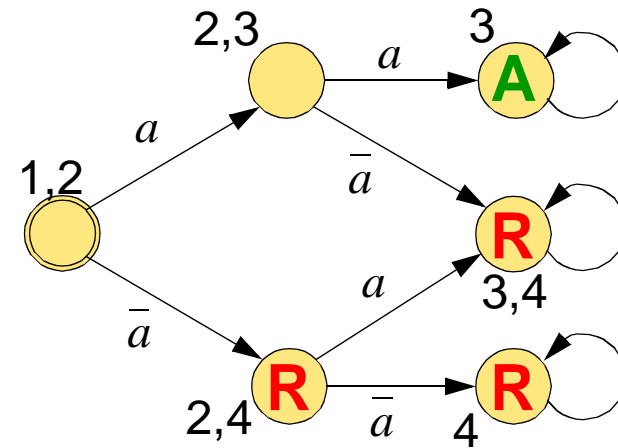
Pruning:

- ▶ introduce one new accepting state and one new rejecting state
- ▶ prune all successors of these states
- + less time intensive
- less reduction in the number of states

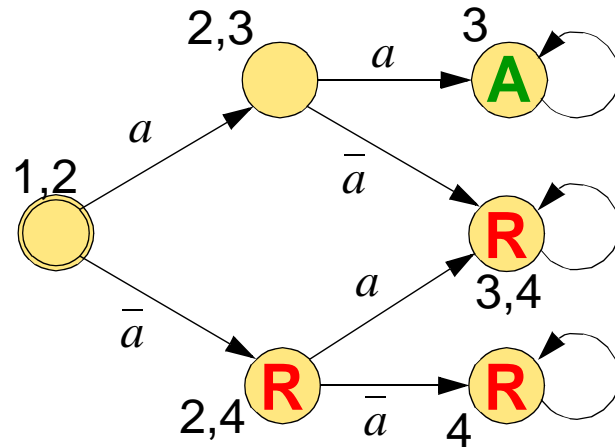
Pruning



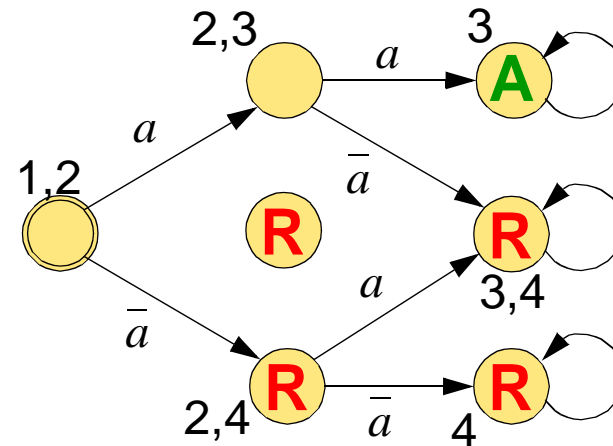
original automaton



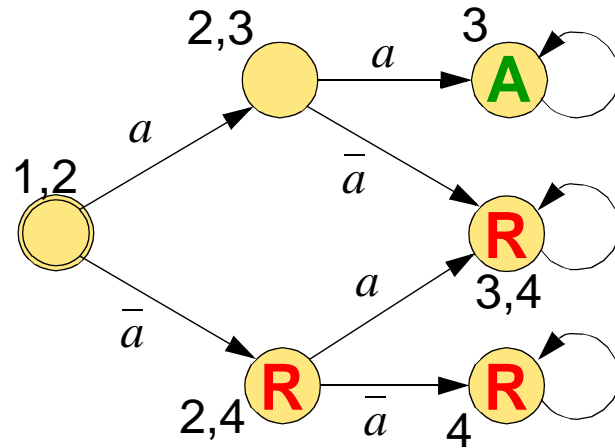
Pruning



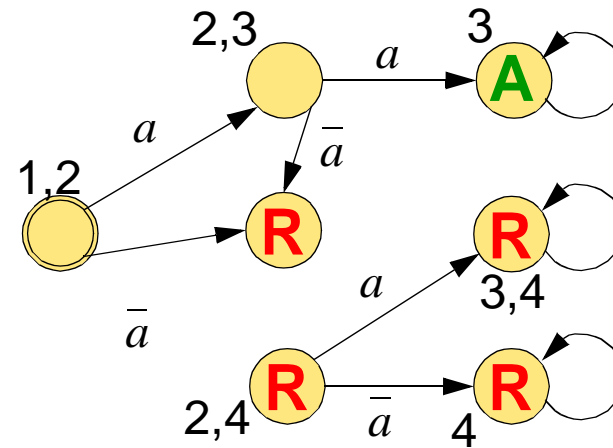
introducing a new rejection state



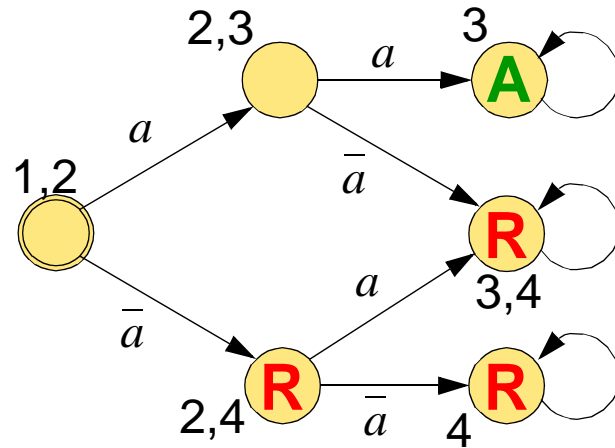
Pruning



redirect the transitions to the new state



Pruning



prune the old rejecting states and all successors

