24) Condition-Action-Analysis and Event-Condition-Action-Based Design

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
Gruppe Softwaretechnologie
http://st.inf.tu-dresden.de/teaching/swt2
WS17/18 17.01.2018

Lecturer: Dr. Sebastian Götz

1. Structured decisions: decision diagrams and decision tables
2. Binary decision diagrams (BDD) and Reduced Ordered BDD
3. Model Checking ROBDDs
4. Event-Condition Action Design
5. Extensibility of ECA

Obligatory Reading

- Balzert, Kapitel über Entscheidungstabellen
- Ghezzi 6.3 Decision-table based testing
- Pfleeger 4.4, 5.6

Literature on BDDs and ROBDDs

- C.Y. Lee: Representation of Switching Circuits by Binary-Decision Programs, Bell System Technical Journal, Vol. 38, July 1959, pp. 985-999. http://ieeexplore.ieee.org/document/6768525/
- Randal E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers, 1986 http://ieeexplore.ieee.org/document/1676819/

Goal

- **Decision analysis** (Condition analysis) is a very important method to analyze complex decisions
  - Understand that several views on a decision tree exist (tables, BDD, ROBDD)
- **Condition-action analysis** can also be employed for requirements analysis
  - Understand how to describe the control-flow of methods and procedures and their actions on the state of a program
- **Event-condition-action-based design (ECA-based design)** relies on condition-action analysis
- Understand the importance of model checking

24.1 DECISION ANALYSIS WITH DECISION TREES AND TABLES (CONDITION-ACTION ANALYSIS)

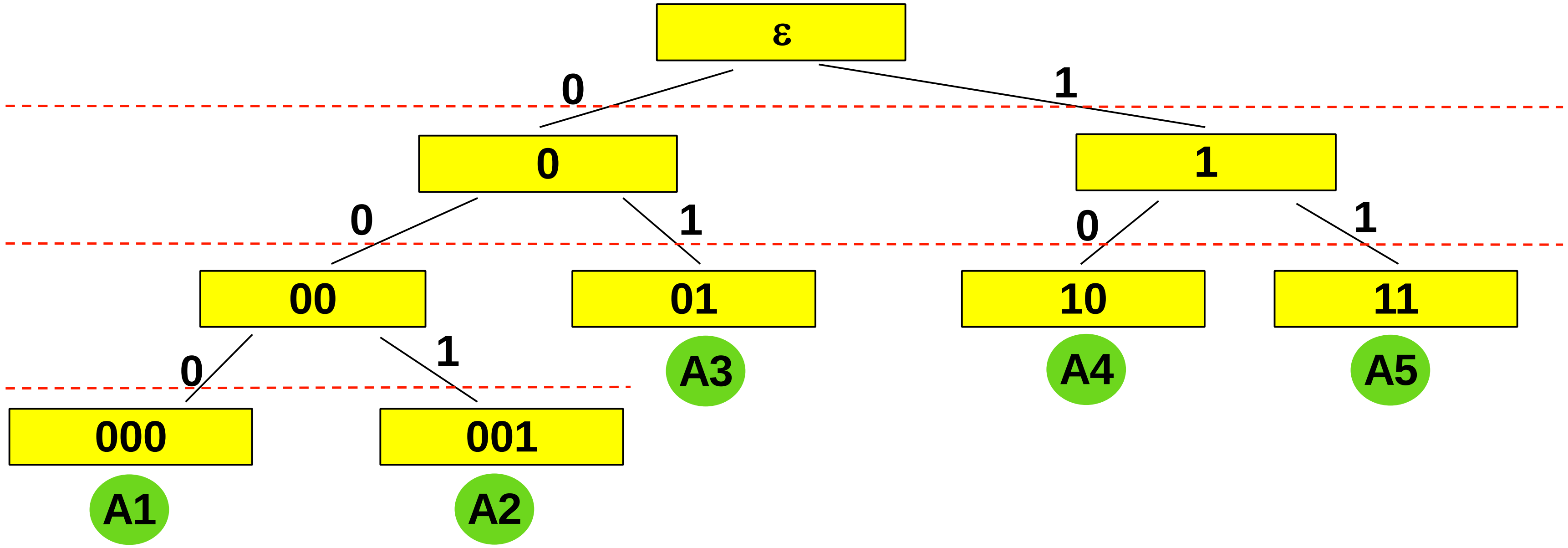
Decision Analysis (Condition-Action Analysis)

- **Decision analysis** is necessary when complex, intertwined decisions should be made
  - In requirements analysis and elicitation
  - In complex business cases, described with business rules
  - In testing, for specification of complex test cases
- Decision analysis can be made in a **decision algebra**
  - Boolean functions and their representations:
    - Truth tables, decision trees, BDD, ROBDD
    - Decision tables
  - Static single assignment form (SSA) (not treated here)
  - Lattice theory, such as formal concept analysis (FCA) (not treated here)
- Decision trees and tables collect actions based on conditions
- Condition action analysis is a decision analysis that results in actions
  - A simple form of event-condition-action (ECA) rules
  - However, without events, only conditions

Which conditions provoke which actions?

Decision Trees

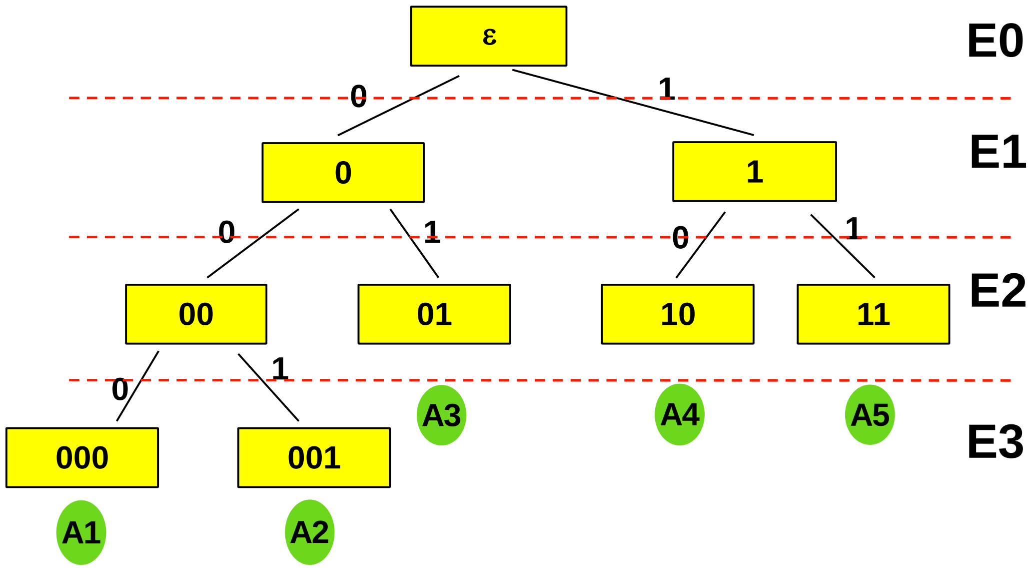
- Decisions can be analyzed with a ***decision tree***, a simple form of a decision algebra
- A trie (Präfixbaum) is a tree which has an edge marking
  - Every path in the trie assembles a word from a language of the marking
- A trie on B = {0,1} is called ***decision tree***
  - Paths denote sequences of decisions (a set of vectors over B). A path corresponds to a vector over B
  - A set of actions, each for one sequence of decisions
  - Sequences of decisions can be represented in a path in the decision tree



Decision Trees with Code Actions

- The action may be code
- The inner nodes of a tree layer correspond to a condition E[i]
- Then, a Trie is isomorphic to an If-then-else cascade

```
if (E0) then // case E0 === true
  if (E1) then
    if[E2) then            A5
    else                   A4
else // case E0 === false
 if (E1) then
    if[E2) then            A3
    else
      if (E3) then         A2
      else                 A1
```



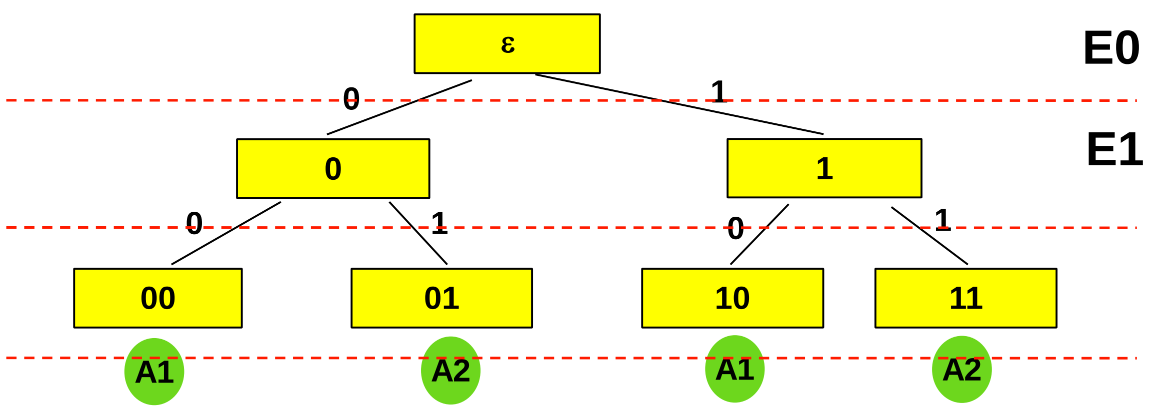
Decision Tables

- An alternative representation of decision trees are **decision tables**
- Conditions and actions can be entered in a table

Condition E0	yes	yes	no	no
Condition E1	yes	no	yes	no
Action A1		X		X
Action A2	X		X	

Boolean cross product

Multiple choice quadrant



Process: How to Construct A Decision Table

1) **Elaborate** decisions
2) **Elaborate** actions
3) Enter into table
4) **Elaborate**: Construct a cross boolean product as upper right quadrant (set of boolean vectors)
5) **Elaborate:** Construct a multiple choice quadrant (lower right) by associating actions to boolean vectors
6) **Consolidate**
   - Coalesce yes/no to "doesn't matter"
   - Introduce Else rule

Applications of Decision Tables and Trees

- **Requirements analysis:**
  - Deciding (decision analysis, case analysis)
  - Complex case distinctions (more than 2 decisions)
- **Design:**
  - Describing the behavior of methods
  - Describing business rules
    - Before programming if-cascades, better make first a nice decision tree or table
  - Formal design methods
  - CASE tools can generate code automatically
- **Configuration management of product families:**
  - Decisions correspond here to configuration variants
  - Processor=i486?
  - System=linux?
  - Same application as #ifdefs in C preprocessor

24.2 NORMALIZING CONTROL FLOW WITH BINARY DECISION DIAGRAMS

Truth Tables

- With action = {true, false}, boolean decision tables are truth tables

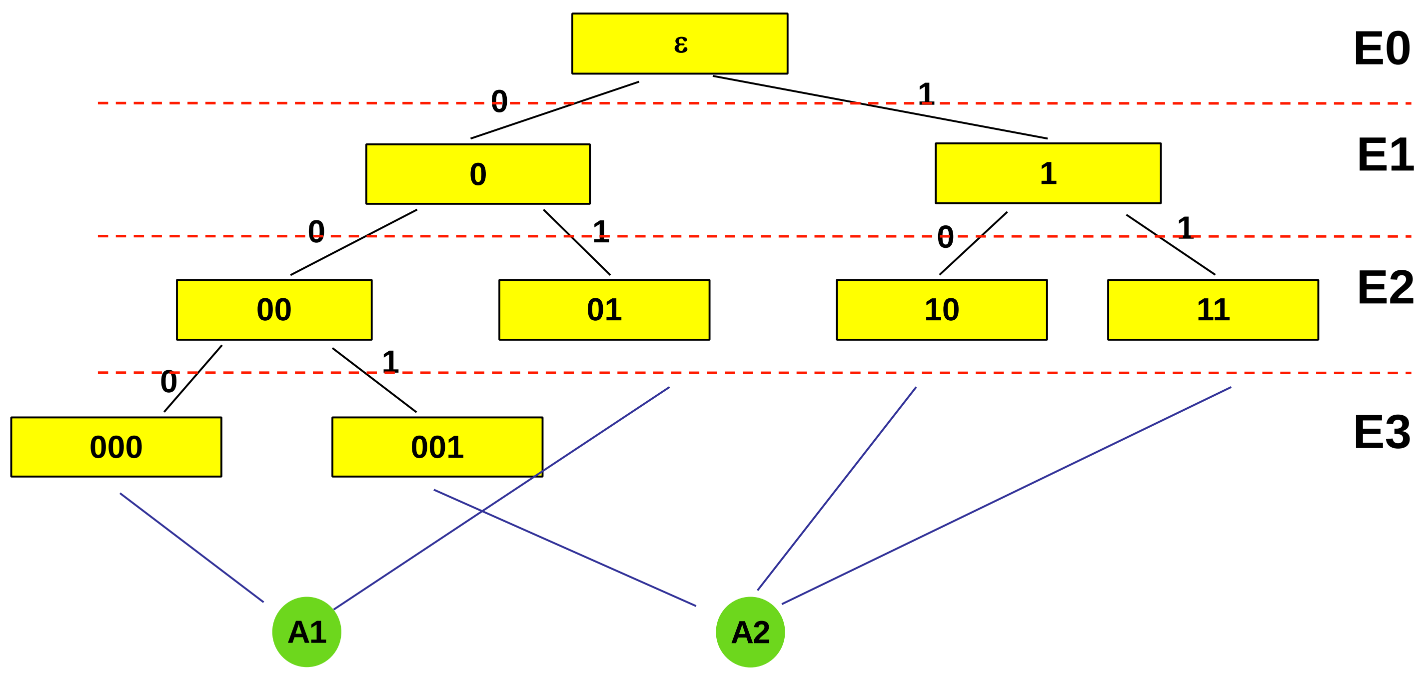
Condition E0	Yes	Yes	No	No
Condition E1	Yes	No	Yes	No
Value of F = 0	X		X	
Value of F = 1		X		X

- Truth table:

E0	**E1**	**F**
Yes	Yes	0
Yes	No	1
No	Yes	0
No	No	1

BDDs (Binary Decision Diagrams) [Lee'59]

- BDD are DAGs that result by merging the same subtrees of a decision tree into one (common subtree elimination)
- If the action is just a boolean value boolean functions f: $B^n \rightarrow B$ can be represented
- The decisions E[i] are regarded as boolean variables



[C.Y. Lee, *Representation of Switching Circuits by Binary-Decision Programs*, Bell System Technical Journal, Vol. 38, July 1959, pp. 985-999.]

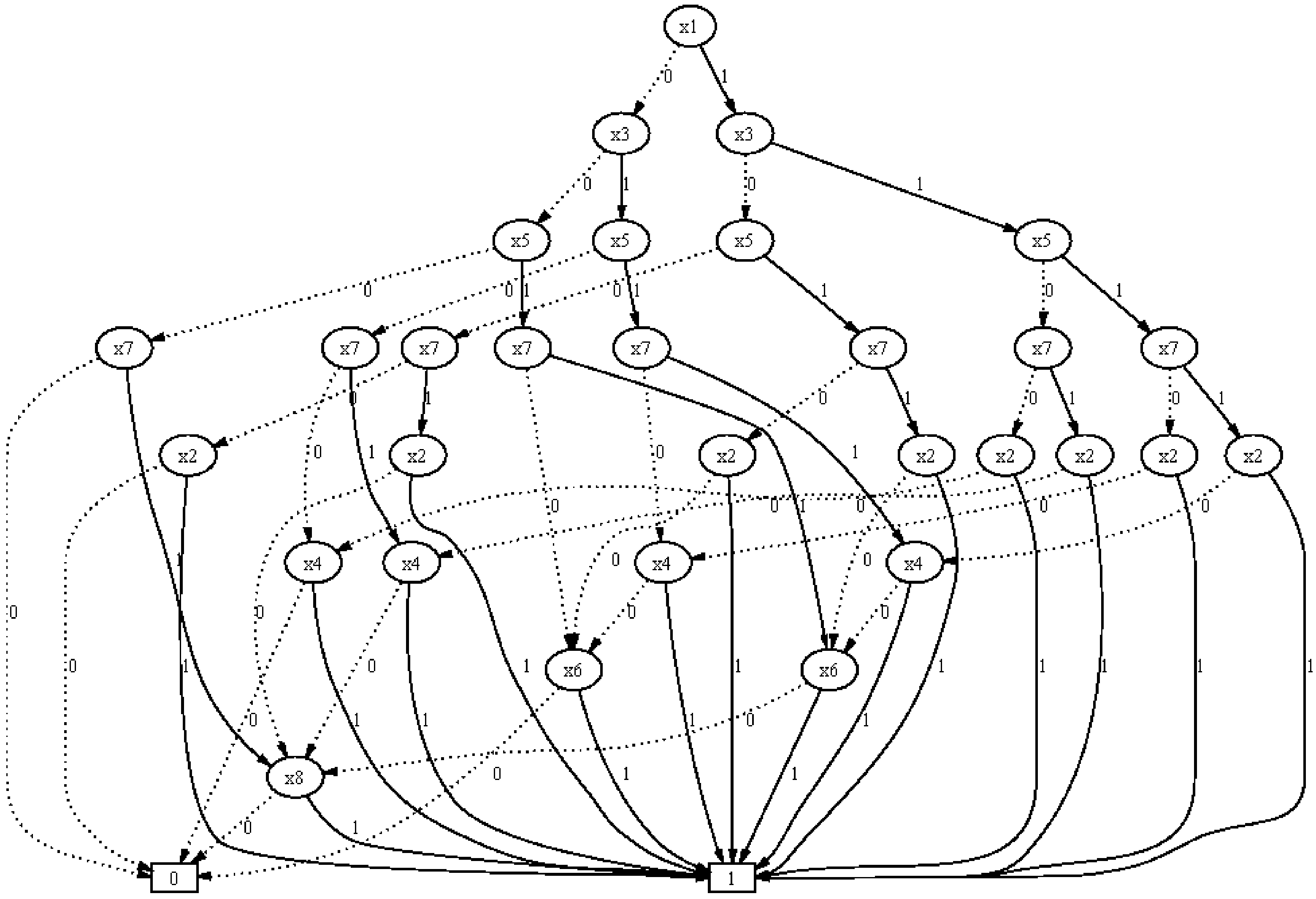
Prof. U. Aßmann

ROBDDs (Reduced Ordered Binary Decision Diagrams) [Bryant'86]

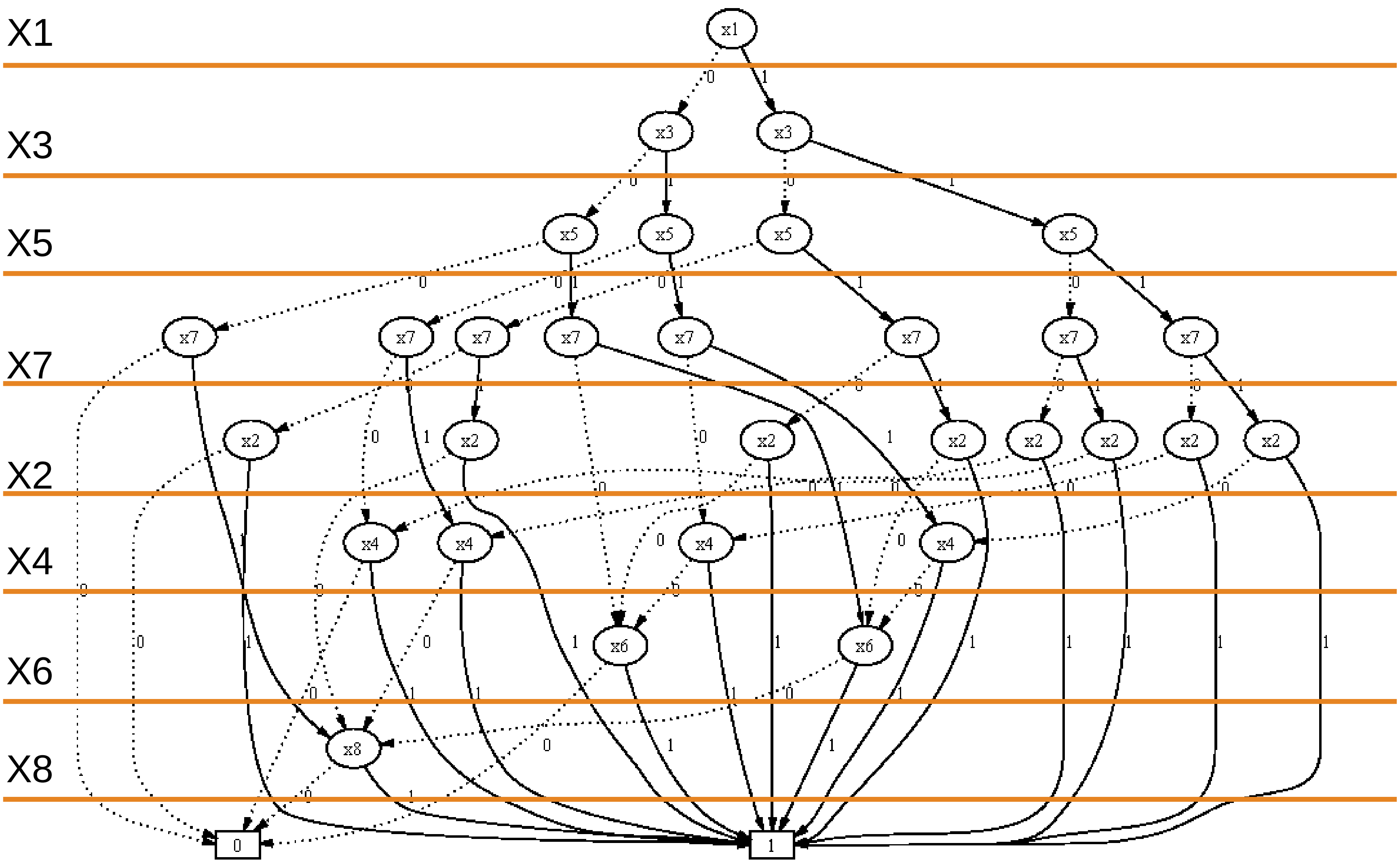
- **Problem**: for one boolean function there are many BDDs, depending on the order of the variables
  - Idea: introduce a standardized order for the variables
  - Result: ordered binary decision diagrams (OBDD)
  - Common subtree elimination (as in BDDs) leads to ROBDD
- In all OBDD holds:
  - for all children u of parents v ord(u) > ord(v).
- For one order of variables there is one ROBDD for all BDDs representing the same boolean function
- Using this canonical form the answer to the question whether two BDDs represent the same boolean function becomes trivial!

[Randal E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Transactions on Computers, 1986]

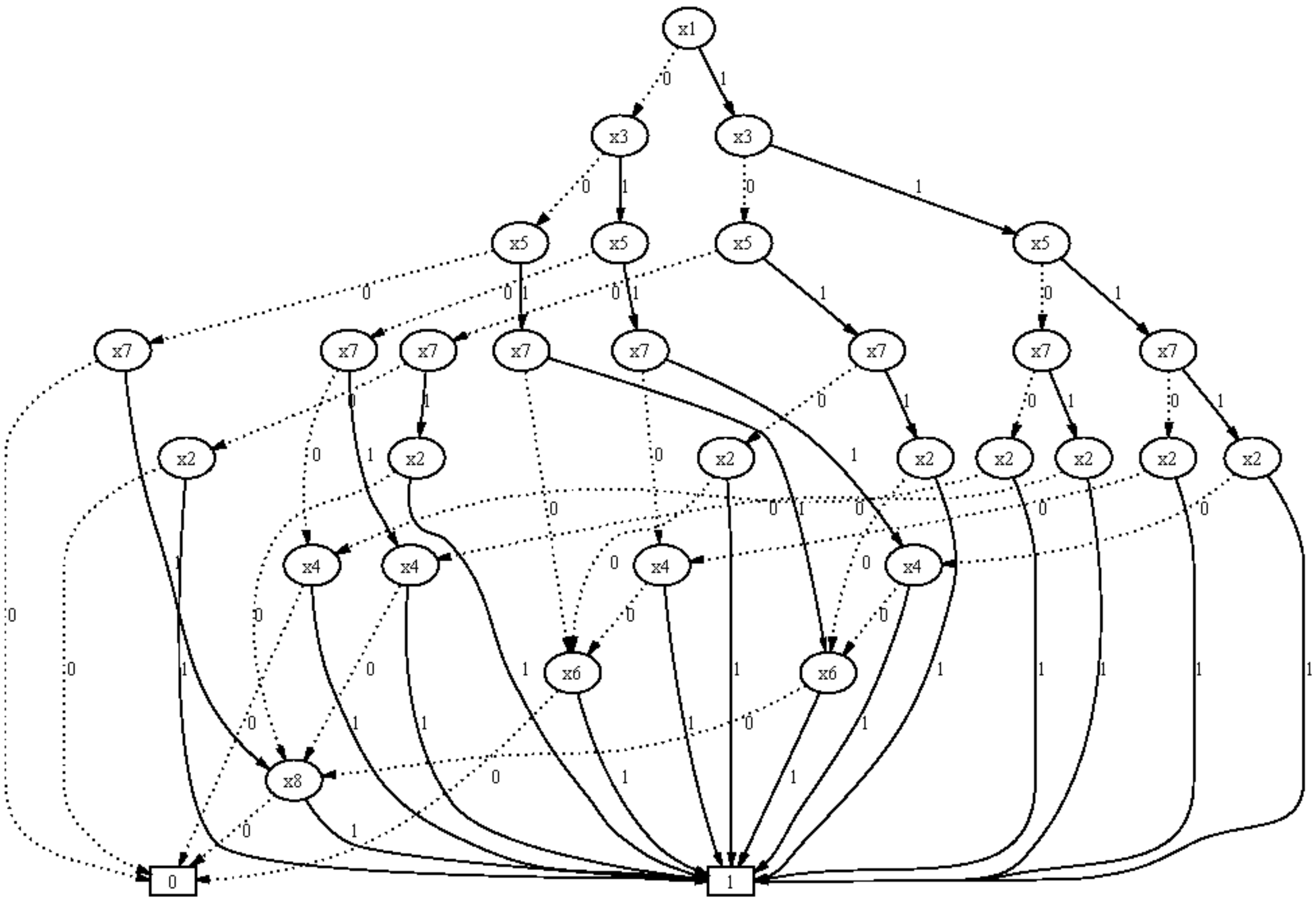
Complex BDD

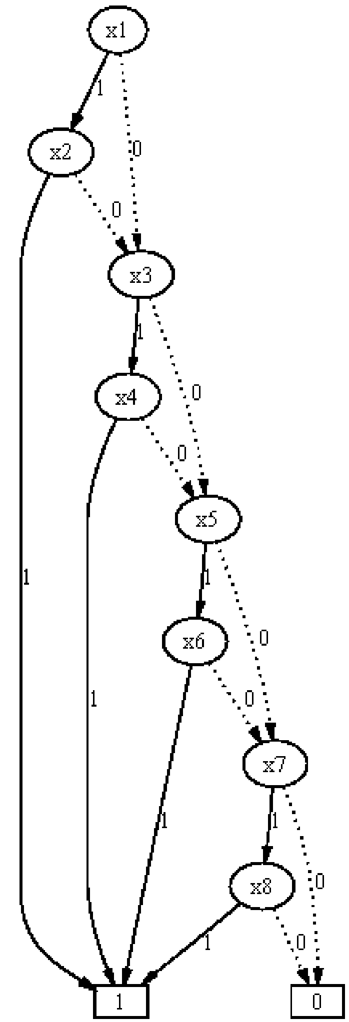

x1
x3
x3
x5
x5
x5
x5
x7
x7
x7
x7
x7
x7
x7
x7
x2
x2
x2
x2
x2
x2
x2
x2
x2
x4
x4
x4
x4
x6
x6
x8
0
1

Complex BDD



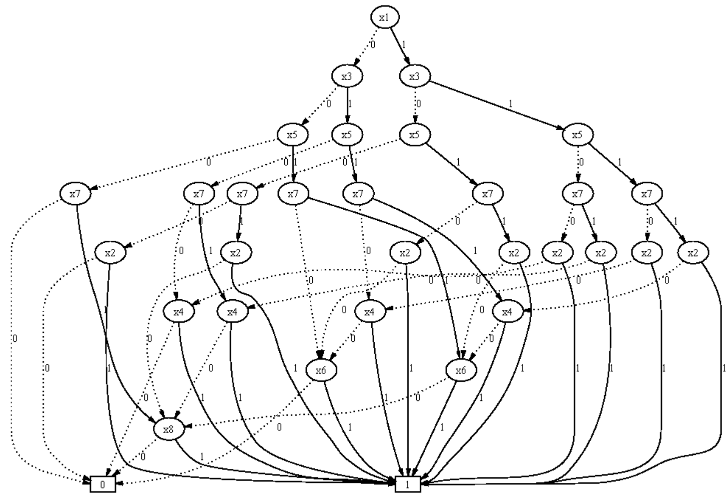
Reduced Ordered BDD

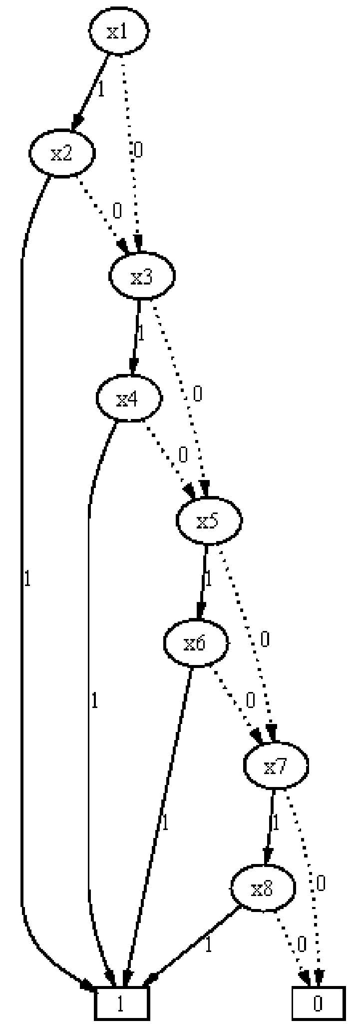




If-cascade, BDD, ROBDD, factorized if-cascade

```
if x1 then
  if x3 then
    if x5 then
      …
else
  if x3 then
      …
```






```
if x1 then
  if x2 then return true
if x3 then
  if x4 then return true
if x5 then
  if x6 then return true
if x7 then
  if x8 return true
return false;
```

E.g., Implementation in Python:
https://pyeda.readthedocs.io/en/latest/bdd.html

Normalizing Wild Procedures: Normalized If-Structures with ROBDD

- There is only one canonical ROBDD for one order
- Develop *normalized and factorized* if-structures with it:
  1. Elaborate arbitrary decision tree
  2. Choose a variable order
  3. Transform to ROBDD
  4. Transform to If structure
  5. Factor out common subtrees by subprograms

Acyclic control flow can be represented canonically by a ROBDD

Applications

- Requirements analysis
- Design
  - Normalized control-flow structures
  - Complex case analyses
- Reengineering
  - Structuring of legacy procedures: read in control-flow; construct control-flow graph
  - Produce a canonical OBDD for all acyclic parts of control-flow graph
  - Pretty-print again
  - Or: produce a statechart
- Configuration management
  - Development of canonical versions of C preprocessor nestings
- Help to master large systems

24.3 MODEL CHECKING LARGE STATE SPACES

Model Checking on BDD

- ROBDD are a very compact representation for state machines, boolean functions, predicate logic, and modal logic
- They build a basis for checking state transition systems with modal logic (model checking)
  - System is modeled as a state transition system and encoded as ROBDD
  - Features of the system (predicates, logic formulas) are encoded as ROBDD, too
  - Important: System **and** predicates to be checked are both encoded as ROBDD
- Model checking:
  - Then, a model checker compares the ROBDDs and checks whether a feature holds in a state
  - Effectively, the model checker only compares normalized representations of boolean functions, the ROBDD

The Use of Model Checking

- State spaces up to 2**120 can be handled
- Model checking checks whether features hold in states of large state spaces
  - Used in hardware verification
    - Proving circuits correct
  - Software verification
    - Safety-critical systems
    - Minimization of boolean circuits
- Very important technique for verification of safety-critical hard- and software

24.4 EVENT-CONDITION-ACTION BASED DESIGN (ECA)

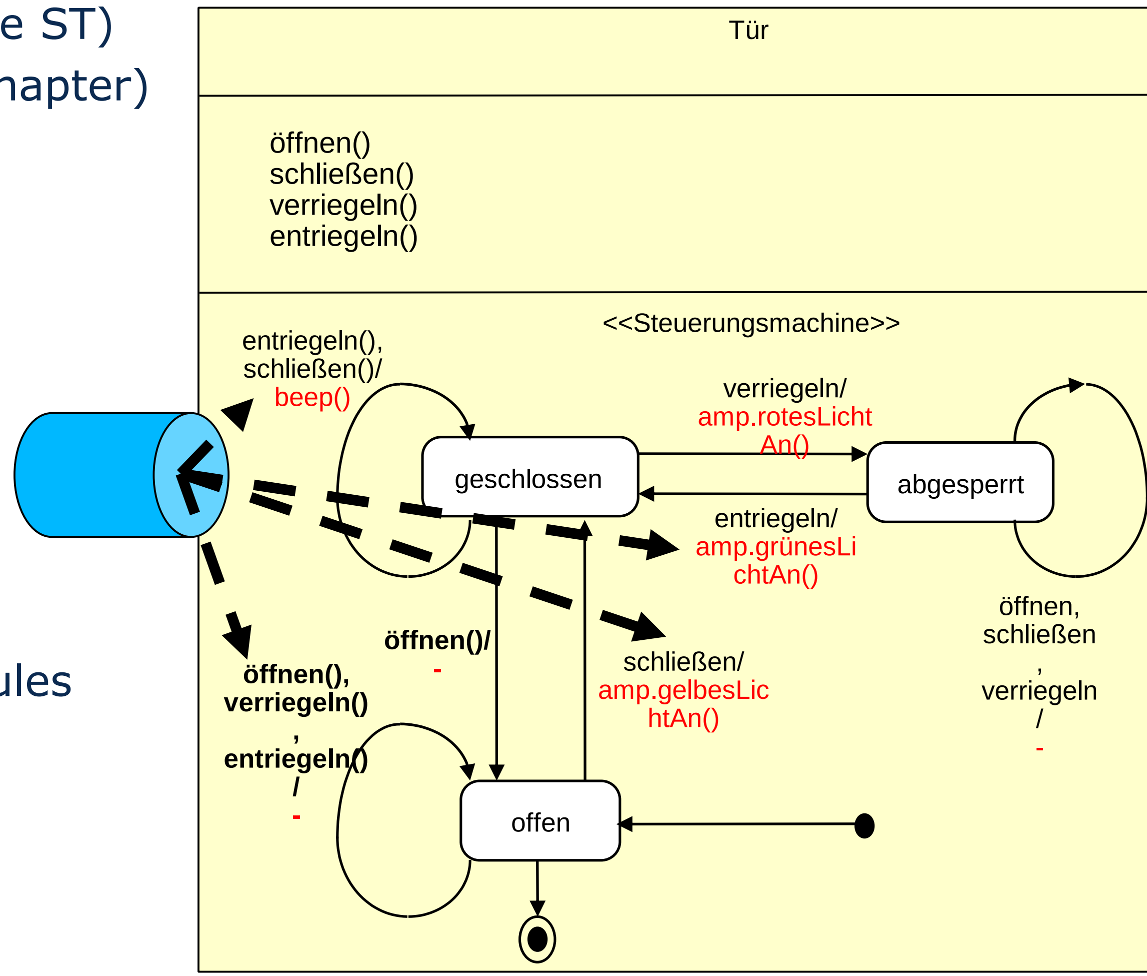
Event-Condition-Action Design

- Decision analysis is invoked when events occur
- Event-condition-action (ECA) based design uses
  - ECA rules with condition-action analysis
  - Complex event processing (CEP) for recognition of complex events

Given some (complex) events, which conditions provoke which actions?

ECA with State-Based Specifications

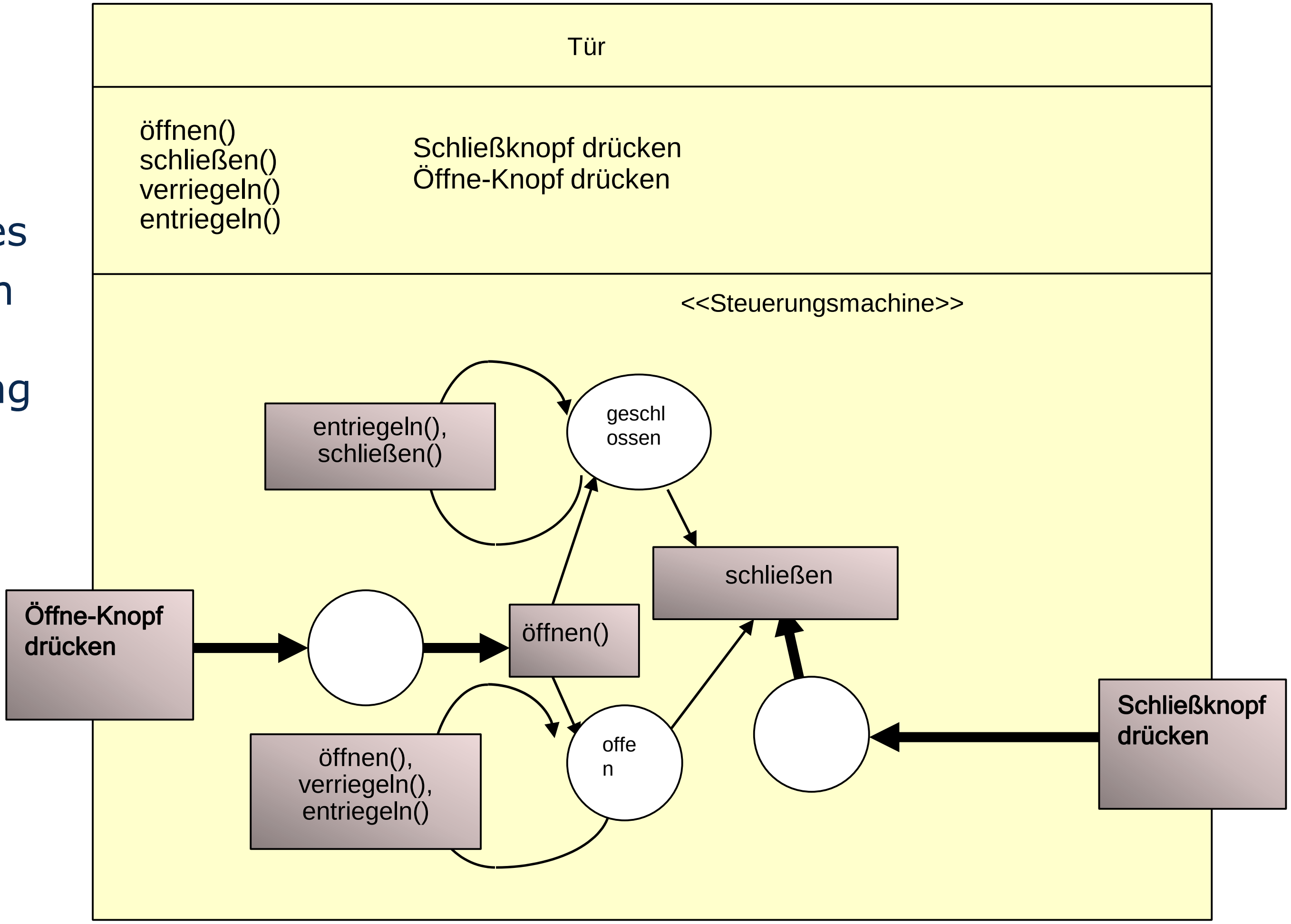
- An event-condition-action (ECA) system listens on channel(s) for events, analyses a condition, and executes an action
  - Statecharts (see course ST)
  - Petri Nets (see corr. Chapter)
  - ECA rules
  - Condition analysis can be done by BDD
  - Verification by model checking
- Process:
  - Collect all ECA rules
  - Collect all states
  - Link states with ECA rules as transitions



ECA with Petri Nets

- In a Petri Net, an **event-generating channel** is a transition with fan-in=0
- Listening to the events, the Petri Net can do condition-action analysis

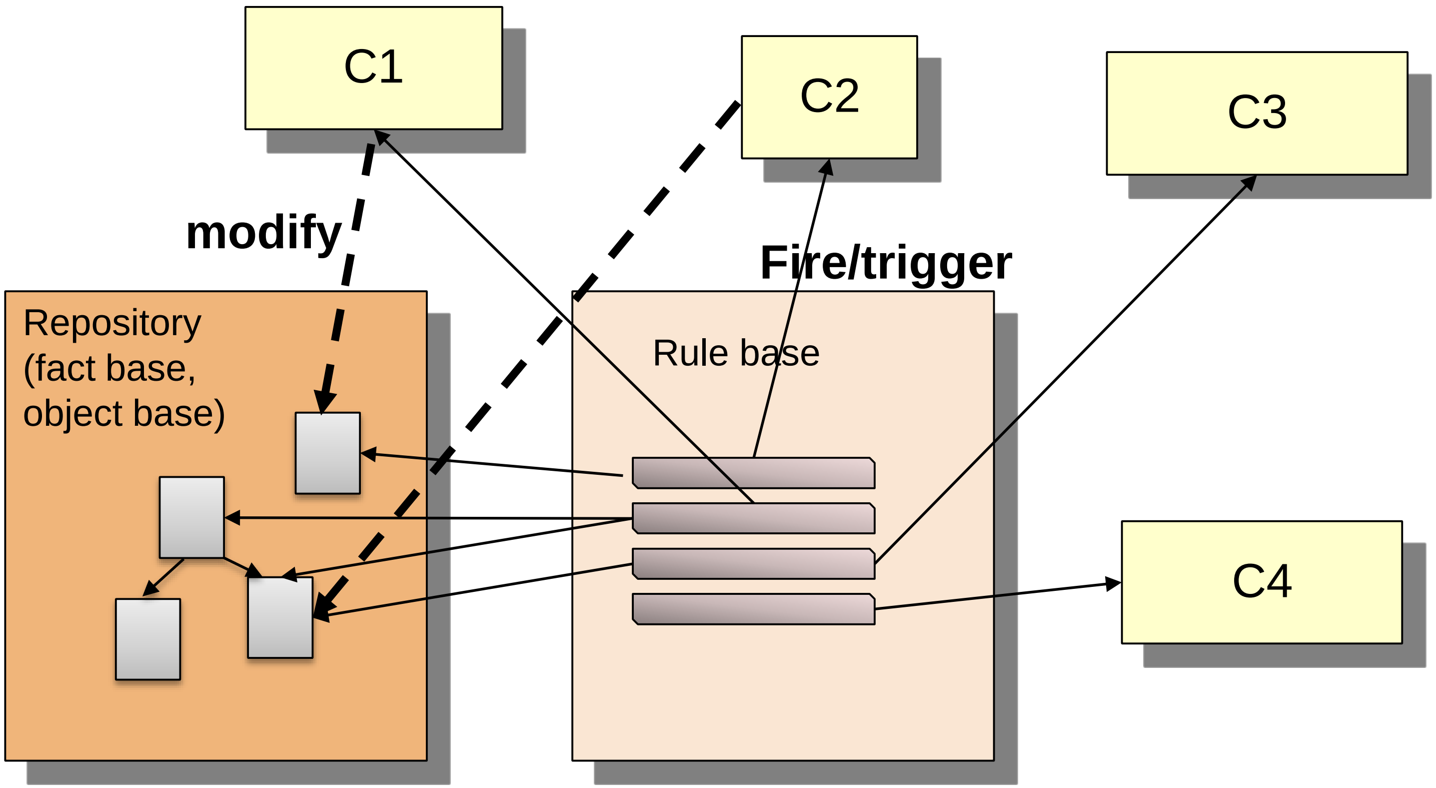
- Process:
  - Collect all ECA rules
  - Collect all states
  - Link states with ECA rules as subnets reacting on event-generating channels



Prof. U. Aßmann

ECA-based Blackboard Style

- The ECA-blackboard has two repositories: a fact/object base and a rule base
- The **rule base** is an active repository (i.e., an active component) that coordinates all other components
  - It investigates the state of the repository. If an event has occured by entering something in the repository (modify), components are fired/triggered to work on or modify the repository



Other Application Areas

- Event-based Web systems (AJAX systems)
  - Scripts in Javascript react on user-triggered events on the client side
  - Server actions are called
- Interactive Systems
  - Event-reaction tables record event-condition-action rules
- Complex event processing in clouds and embedded systems

Prof. U. Aßmann

24.5 EXTENSIBILITY OF ECA RULES

Extensibility of ECA Rule Systems

- Extensibility means to add more ECA rules
- Rules are open constructs
- Problem: new rules should be conflict-free with the old rules
- Harmless extension is usually not provable
- In general, contracts of the old system cannot be retained

ECA-Systems are extensible, but harmlessness of extensions are hard to prove

What Have We Learned

- Decision analysis (Condition-Action analysis) is an important analysis
  - to describe requirements,
  - to describe complex behavior of a procedure
- Decision analysis must be encoded in a decision algebra
  - Boolean functions, decision trees, relations, graphs, automata can be encoded in ROBDD
  - The control-flow of a procedure can be normalized with a ROBDD
  - Conditions in large state spaces can be encoded in ROBDD and efficiently checked
- ECA-based design reacts on events and conditions with actions

The End

- Explain the difference of decision trees, tables, BDD and ROBDD.
- Why is a BDD an „optimized" decision tree?
- Explain how to encode a subset of a finite set with a BDD
- Explain how to encode a relation over two finite sets with a BDD
- How would you reengineer a program with a wild, spaghetti-like control flow structure?