TECHNISCHE
UNIVERSITÄT
DRESDEN

Fakultät Informatik, Institut für Software- und Multimediatechnik, Lehrstuhl für Softwaretechnologie

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

Wintersemester 14/15, 10.01.2015

Lecturer: Dr. Sebastian Götz

1. Decision Analysis
2. Ordered BDDs
3. ECA-based Design

Obligatory Reading

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

- Red Hat. JBoss Enterprise BRMS Platform 5: JBoss Rules 5 Reference Guide. (lots of examples for ECA Drools)

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, OBDD)
- **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

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

A House-Selling Expert System

- *Ok, I do not like bungalows, but my wife does not like that the car stands in free space in winter. Hmm... I rather would like to have the half double house... But we need anyway 2 floors, because I need this space for my hobbies. My wife also would like a garden….*

- How does the system analyze the customers requirements and derive appropriate proposals?

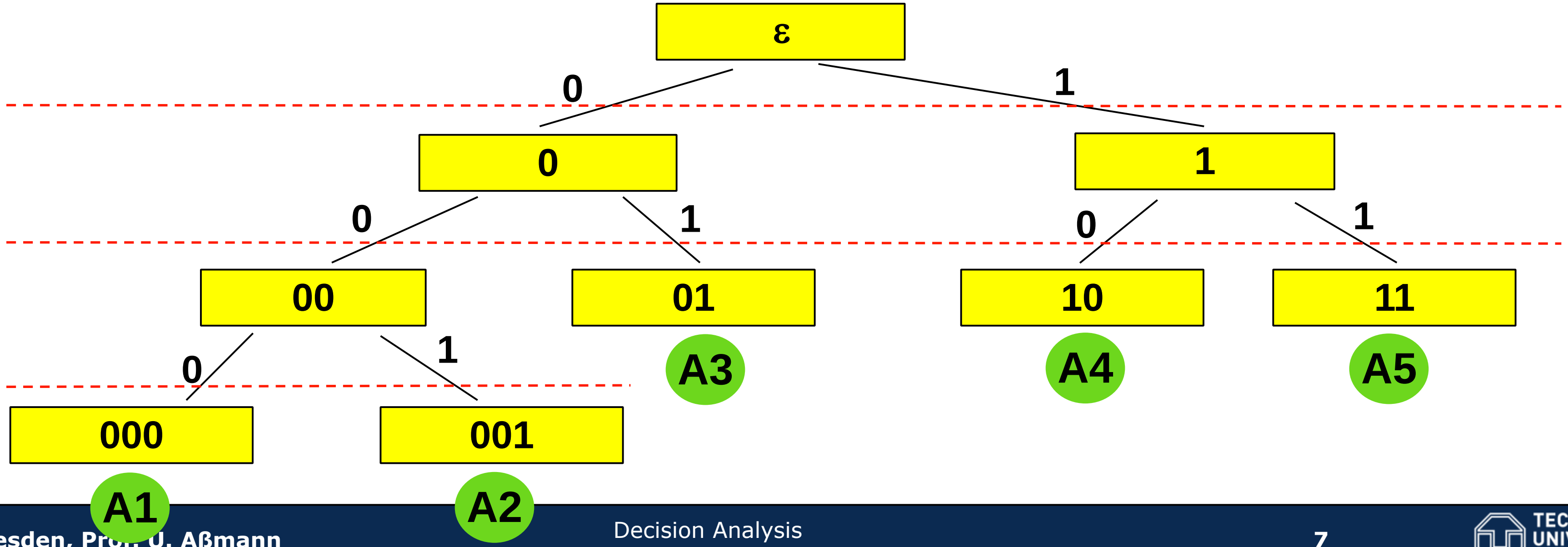
Decision Analysis (Condition-Action Analysis)

- **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, OBDD
    - Decision tables
  - 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

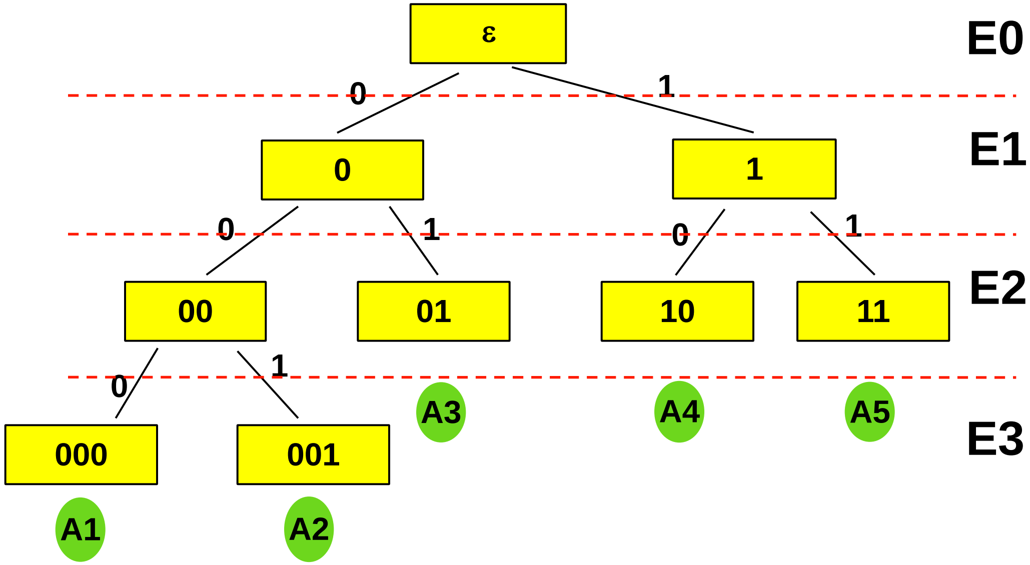
- Decisions can be analyzed with a ***decision tree***, a simple form of a decision algebra
- A trie (Präfixbaum) is a tree which has an edge marking
  - Every path in the trie assembles a word from a language of the marking
- A trie on IB = {0,1} is called ***decision tree***
  - Paths denote sequences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



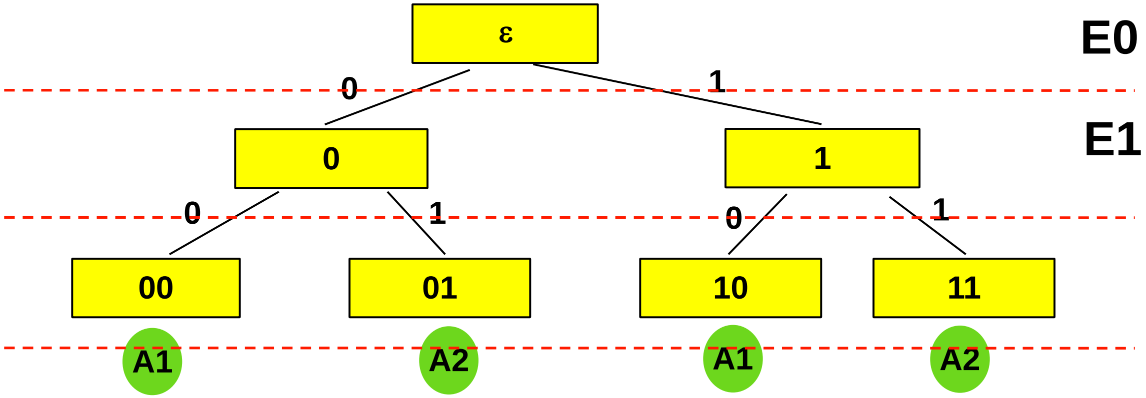
Decision Tables

- 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 NORMALIZED BDD

Truth Tables

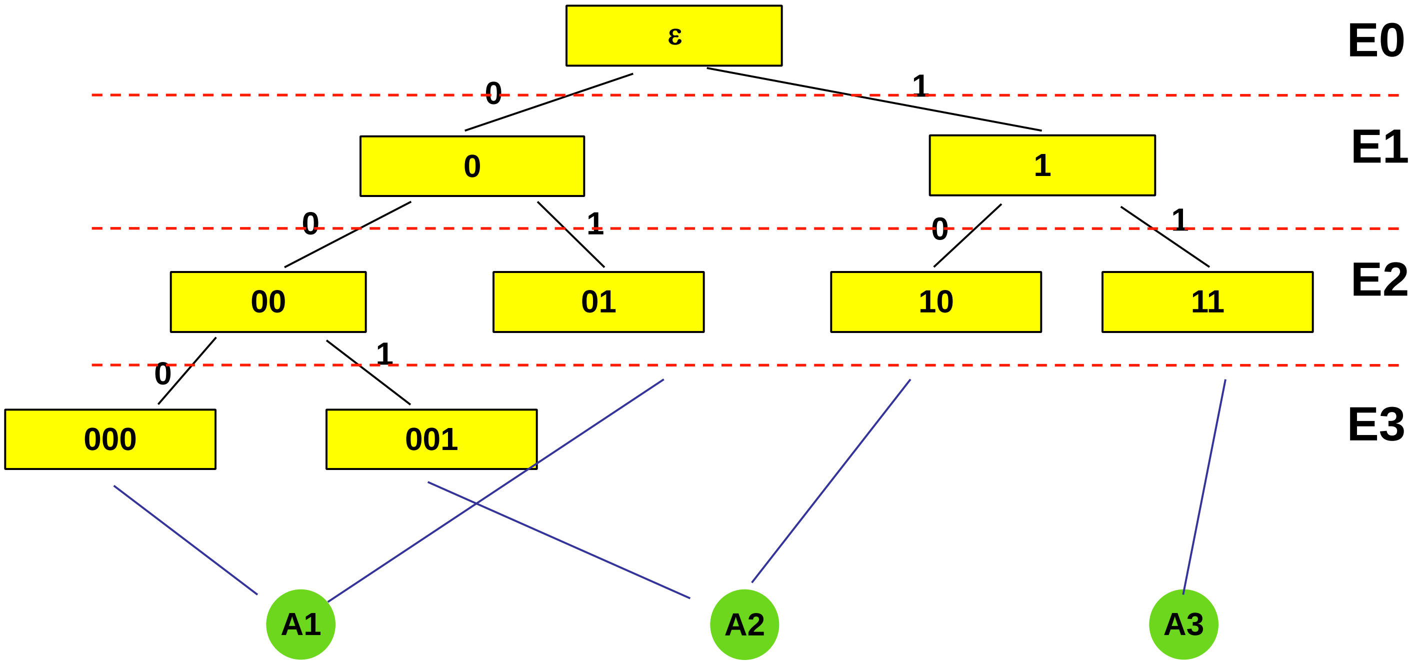
- With action = {true, false}, boolean decision tables are truth tables

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

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

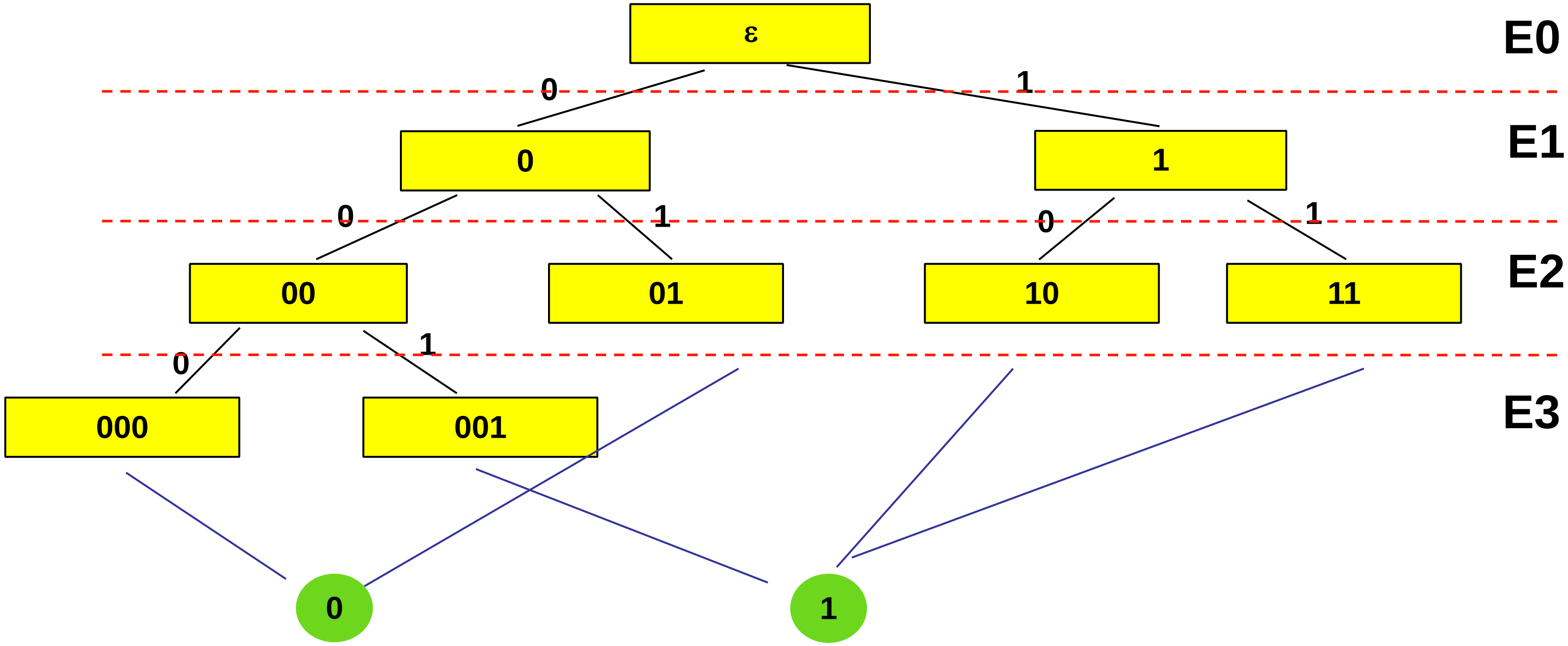
BDDs (Binary Decision Diagrams)

- BDD are dags that result by merging the same subtrees of a decision tree into one (common subtree elimination)



BDDs (Binary Decision Diagrams)

- If the action is just a boolean value boolean functions f: $\mathbb{B}^n \rightarrow \mathbb{B}$ can be represented
- The decisions E[i] are regarded as boolean variables

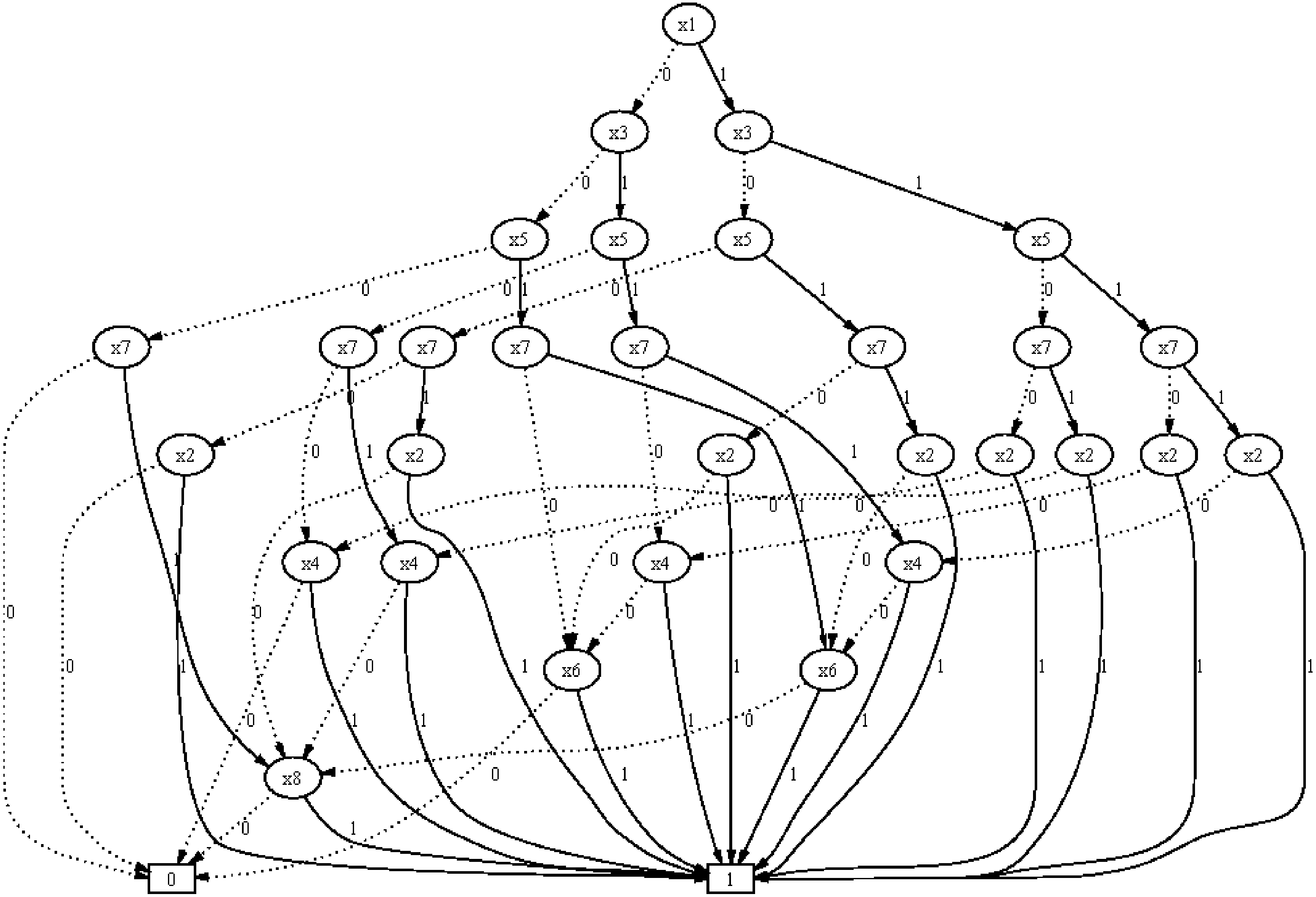


OBDDs (Ordered Binary Decision Diagrams)

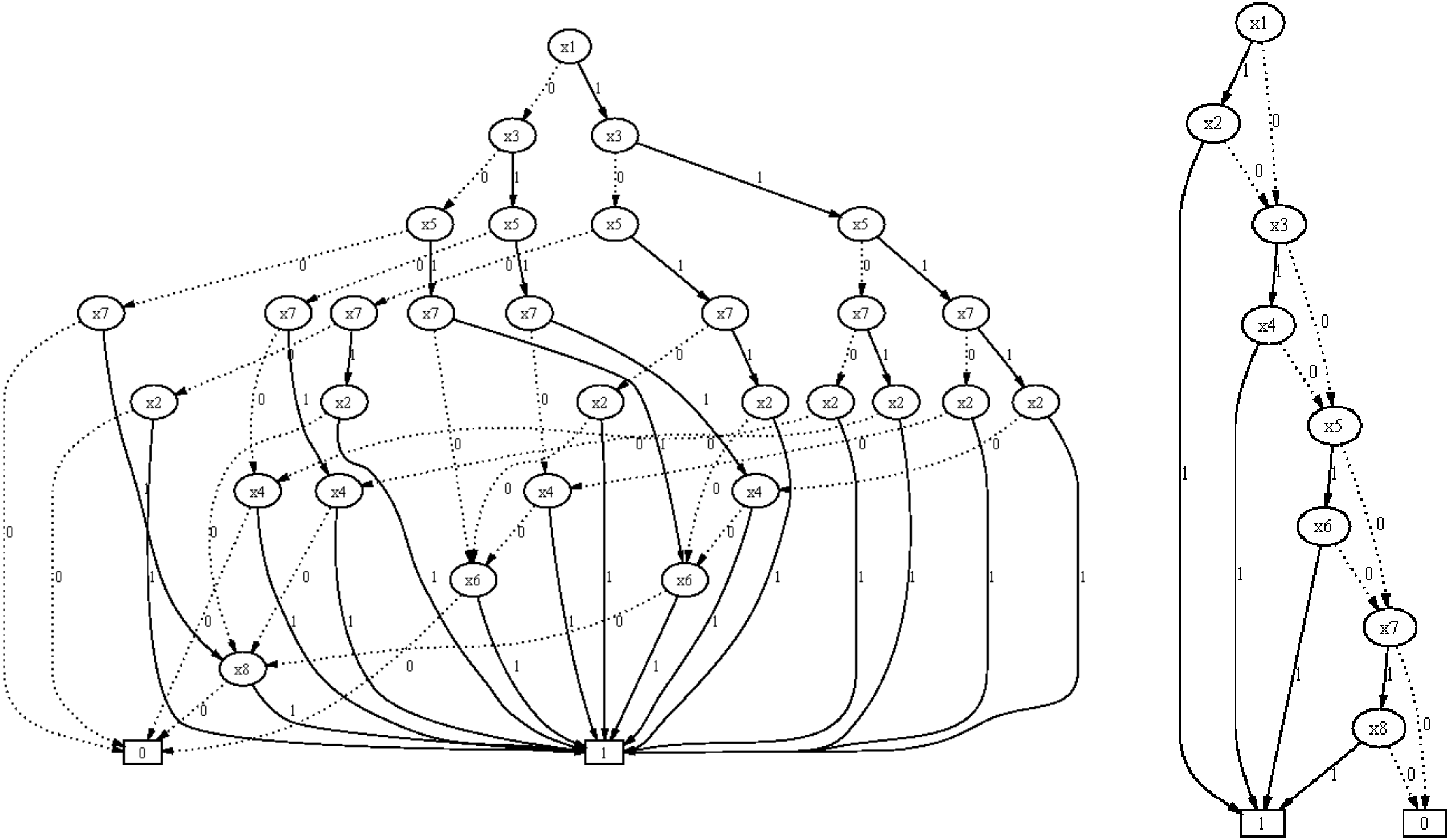
- Problem: for one boolean function there are many BDD
  - Idea: introduce a standardized order for the variables
  - Result: orderd binary decision diagrams
- In all OBDD holds
  - for all children u of parents v ord(u) > ord(v).
- For one order of variables there is one normal form OBDD (canonical OBDD)
- Leads to an efficient **BDD-based comparison algorithm of boolean functions**:

```
compareBooleanFunction() = {
    Fix variable order for two BDD
    Transform both BDD into OBDD
    Compare both OBDD syntactically
}
```

Complex BDD

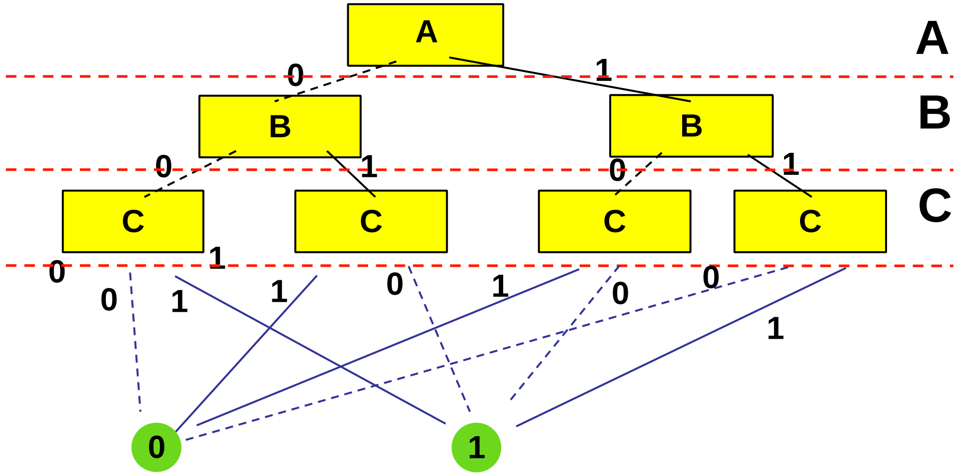

x1
x3
x3
x5
x5
x5
x5
x7
x7
x7
x7
x7
x7
x7
x7
x2
x2
x2
x2
x2
x2
x2
x2
x4
x4
x4
x4
x6
x6
x8
0
1

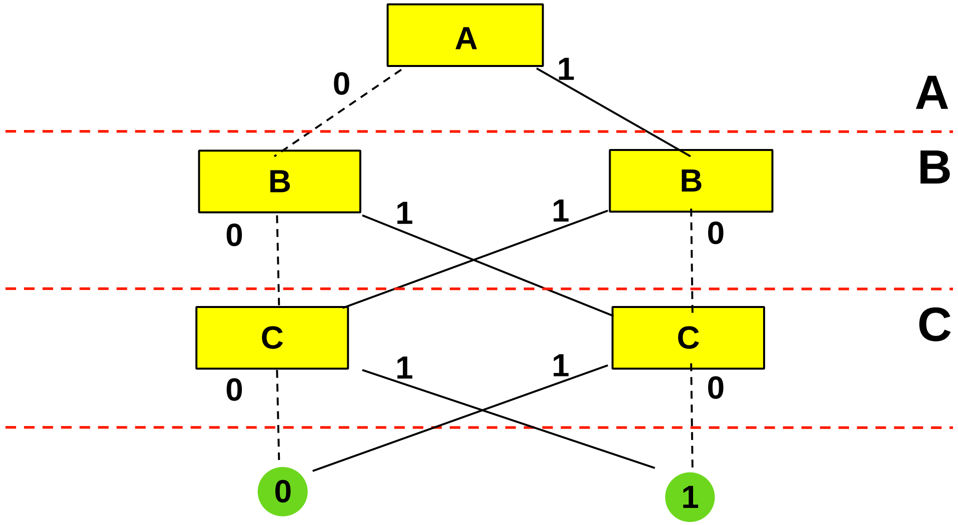
The Influence of Variable Ordering



If-cascades, BDD and OBDD

```
if A then
  if B then
      if C then true else false
  else
      if C then false else true
else
  if B then
      if C then false else true
  else
      if C then true else false
```





Variable order is [A,B,C]

Normalizing Wild Procedures: Normalized If-Structures with OBDD

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

Acyclic control flow can be represented canonically by an OBDD

Applications

- 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 EVENT-CONDITION-ACTION BASED DESIGN (ECA)

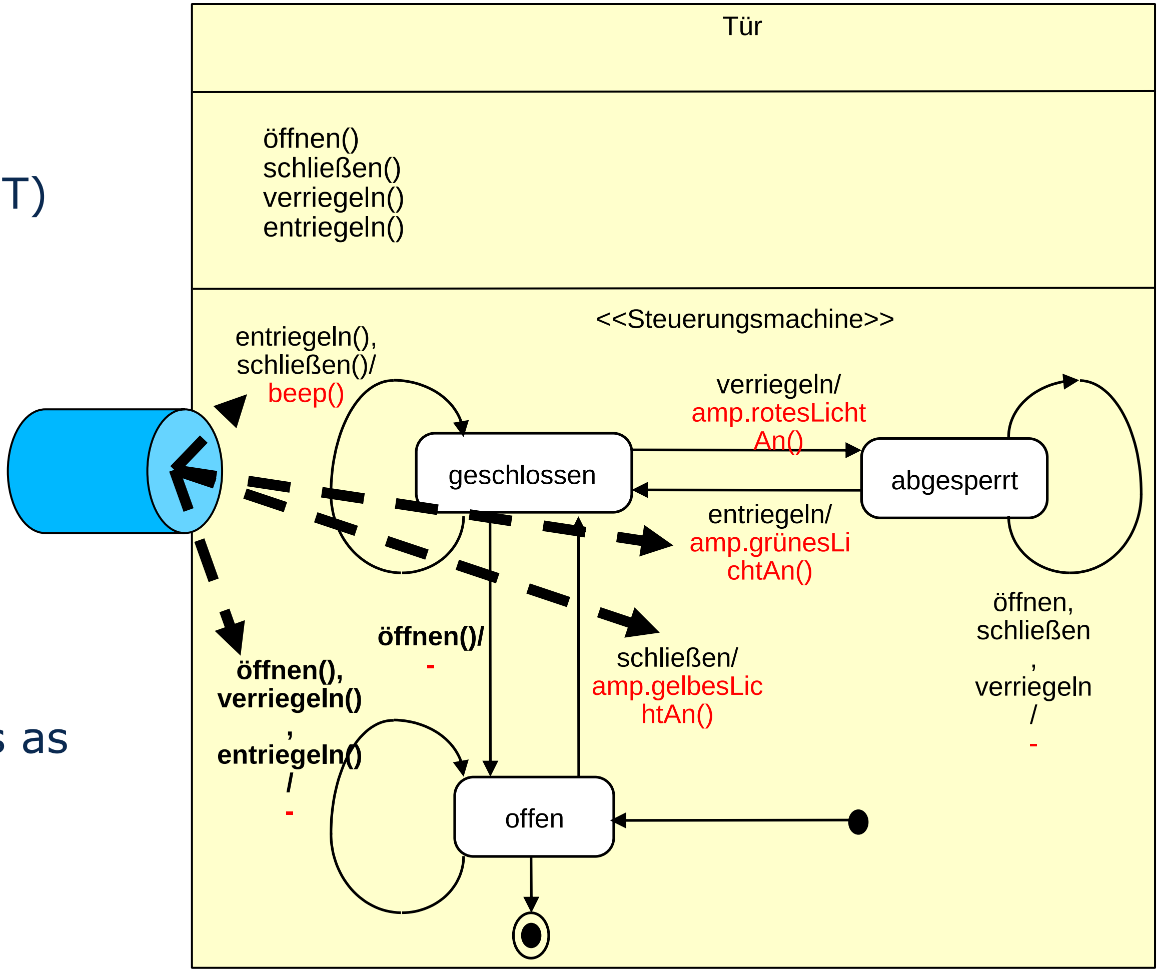
Event-Condition-Action Design

- 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

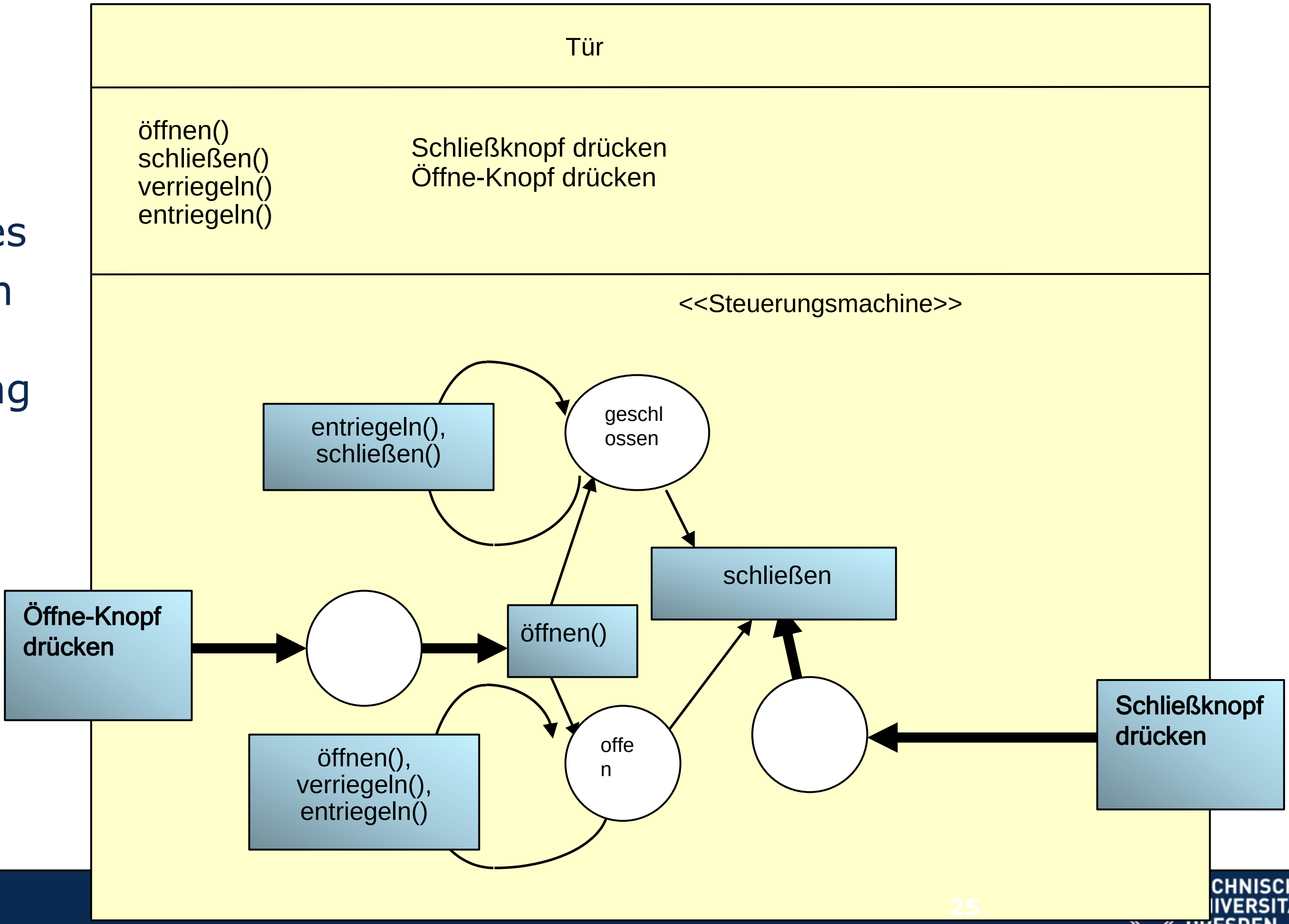
- An event-condition-action (ECA) system listens on channel(s) for events, analyses a condition, and executes an action
    - Statecharts (see course ST)
    - Petri Nets (see corr. Chapter)
    - ECA rules (Drools)
    - Condition analysis can be done by BDD
- Process:
    - Collect all ECA rules
 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



ECA with Petri Nets

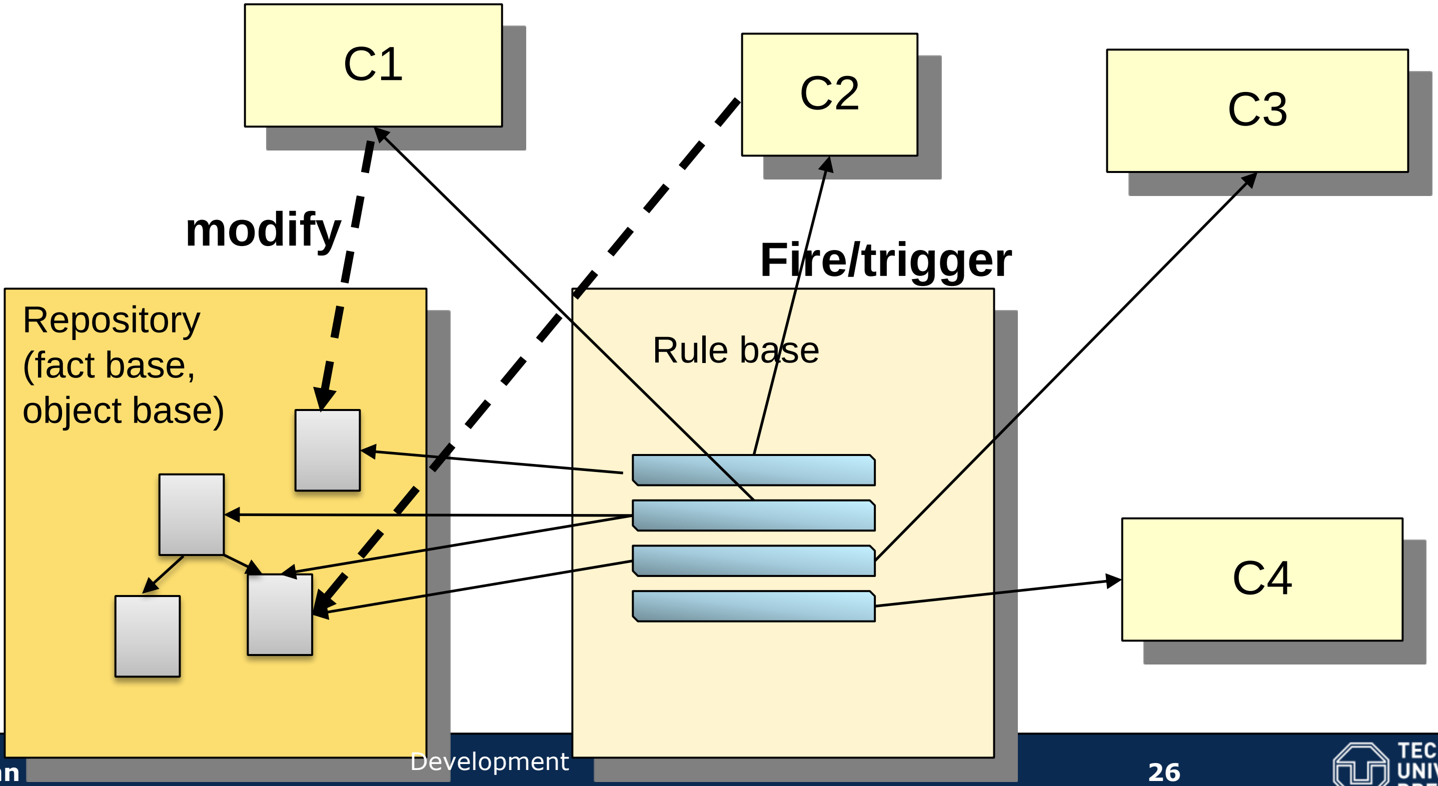
- In a Petri Net, an **event-generating channel** is a transition with fan-in=0
- Listening to the events, the Petri Net can do condition-action analysis

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



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



JBOSS ECA Rule Engine

- Drools (.drl-files) is an active repository with ECA rule processing
- Ex. Fire Alarm Rules [JRules]:

```
rule "Status output when things are ok"
when
  not Alarm()
  not Sprinkler( on == true )
then
  System.out.println( "Everything is ok" );
end

rule "Raise the alarm when we have one or more fires"
when
  exists Fire() // tests whether a Fire object exists
then
  insert( new Alarm() );
  System.out.println( "Raise the alarm" );
end
```

Ex. Fire Alarm

- Create a blackboard and fill the object base

```
// make a new blackboard
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
// add a .drl-file to the rule base
kbuilder.add( ResourceFactory.newClassPathResource( "fireAlarm.drl",
     getClass() ), ResourceType.DRL );
if ( kbuilder.hasErrors() )
  System.err.println( kbuilder.getErrors().toString() );
// open a session with the blackboard
StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();

// allocate objects in the object/fact base
String[] names = new String[]{"kitchen","bedroom","office","livingroom"};
Map<String,Room> name2room = new HashMap<String,Room>();
for( String name: names ) {
  Room room = new Room( name ); name2room.put( name, room );
  ksession.insert( room );
  Sprinkler sprinkler = new Sprinkler( room ); ksession.insert( sprinkler );
}
ksession.fireAllRules();
```

```
// output>> ”Everything is ok”
```

- Raise fire by inserting a Fire object into the object base

```
Fire kitchenFire = new Fire( name2room.get( "kitchen" ) );
Fire officeFire = new Fire( name2room.get( "office" ) );

// insert into the session
FactHandle kitchenFireHandle = ksession.insert( kitchenFire );
FactHandle officeFireHandle = ksession.insert( officeFire );

// investigate:
ksession.fireAllRules();
```

```
// output>> “Raise the alarm"
```

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

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 is hard to prove

The End: 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 OBDD
  - The control-flow of a procedure can be normalized with a BDD and OBDD
  - Conditions in large state spaces can be encoded in OBDD and efficiently checked
- ECA-based design reacts on events and conditions with actions