

24) Event-Condition-Action Design and Conditions Analysis

Prof. Dr. U. Aßmann
Technische Universität Dresden
Institut für Software- und
Multimediatechnik
Gruppe Softwaretechnologie
<http://st.inf.tu-dresden.de>
[Version 12-1.o, 15.12.12](#)

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

- ▶ Balzert, Kapitel über Entscheidungstabellen
- ▶ Ghezzi 6.3 Decision-table based testing
- ▶ Pfleeger 4.4, 5.6
- ▶ Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers, C-35:677-691, 1986.
- Red Hat. JBoss Enterprise BRMS Platform 5: JBoss Rules 5 Reference Guide. (lots of examples for ECA Drools)
 - http://docs.redhat.com/docs/en-US/JBoss_Enterprise_BRMS_Platform/5/pdf/JBoss_Rules_5_Reference_Guide/JBoss_Enterprise_BRMS_Platform-5-JBoss_Rules_5_Reference_Guide-en-US.pdf

- ▶ Decision algebra:
 - ▶ Danylenko, Antonina, Lundberg, Jonas, Löwe, Welf. Decisions: Algebra and Implementation. In Machine Learning and Data Mining in Pattern Recognition. Perner, Petra(ed.) Lecture Notes in Computer Science, 6871, Springer 2011. http://dx.doi.org/10.1007/978-3-642-23199-5_3
- ▶ ECA state of the art
 - ▶ [REVERSE-DEL-2004-I5-D1](#)
 José Júlio Alferes, James Bailey, Mikael Berndtsson, François Bry, Jens Dietrich, Alex Kozlenkov, Wolfgang May, Paula-Lavinia Pătrânjan, Alexandre Miguel Pinto, Michael Schroeder, and Gerd Wagner:
 Wolfgang May (editor): *State-of-the-art on evolution and reactivity*.
<http://reverse.net/publications/reverse-publications.html#REVERSE-DEL-2004-I5-D1>
 - ▶ http://en.wikipedia.org/wiki/Complex_event_processing
- ▶ ECA Engines
 - ▶ Websphere Jrules engine
<http://www-01.ibm.com/software/integration/business-rule-management/decision-server/>
 - ▶ JBOSS Rules <http://www.jboss.org/drools>
 - ▶ http://docs.redhat.com/docs/de-DE/JBoss_Enterprise_BRMS_Platform/index.html

- **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
- Understand that model checking is a technology with future

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

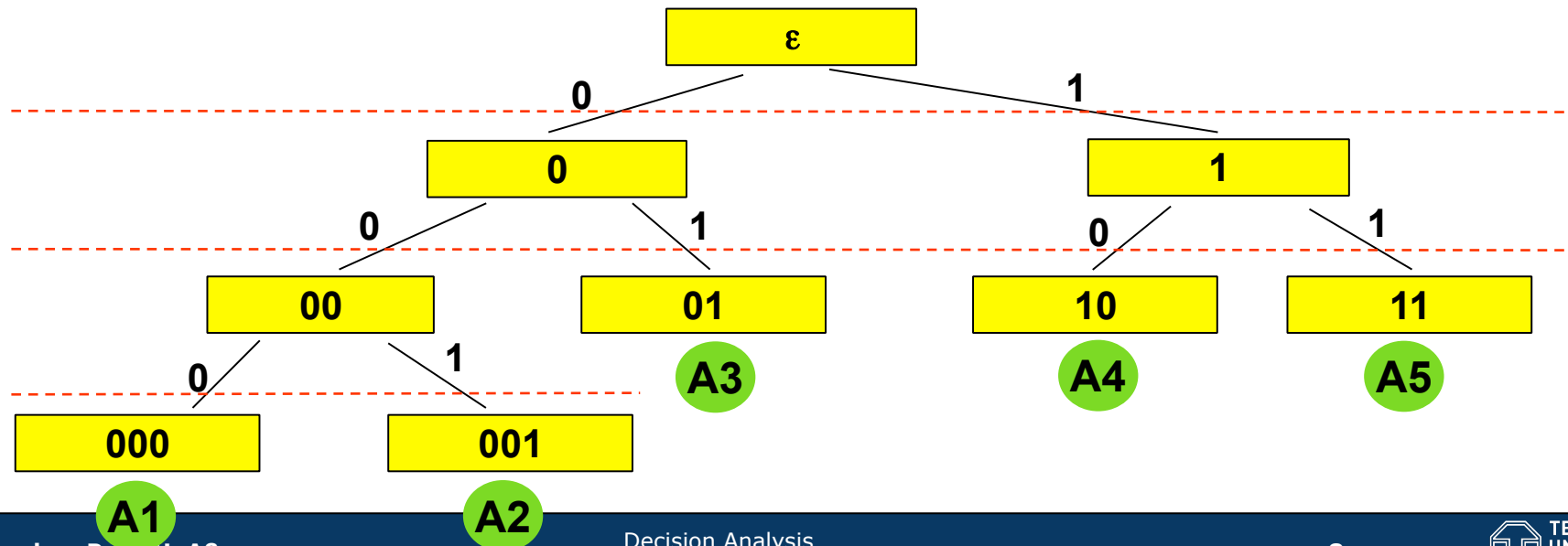
- *Ok, I do not like bungalows, but my wife does not like that the car stands in free space in winter. Hmm... I rather would like to have the half double house... But we need anyway 2 floors, because I need this space for my hobbies. My wife also would like a garden....*

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

- **Decision analysis** 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
 - 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?

- 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



- ▶ 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

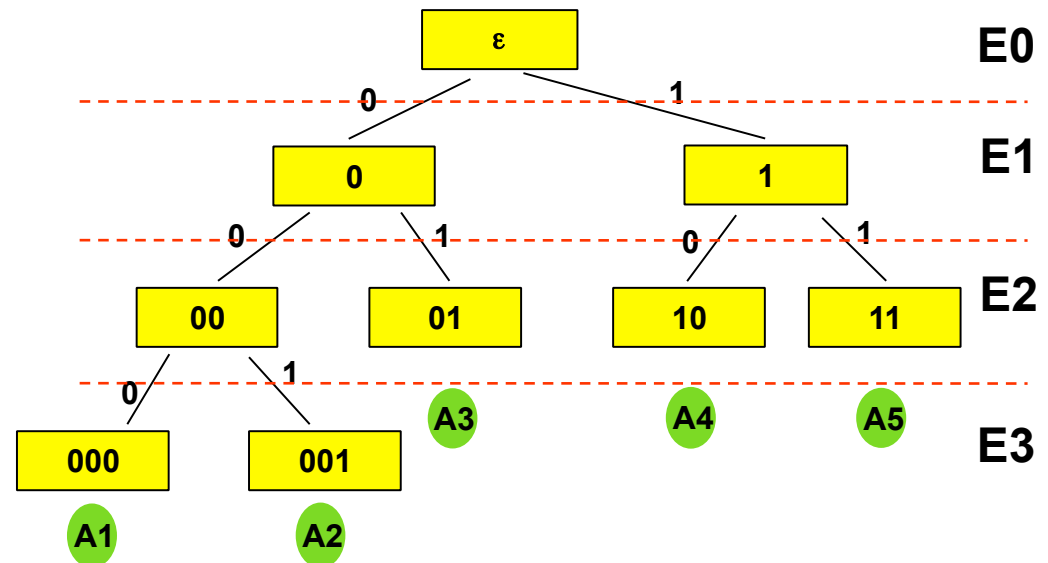
 if (E1) then

 if(E2) then A3

 else

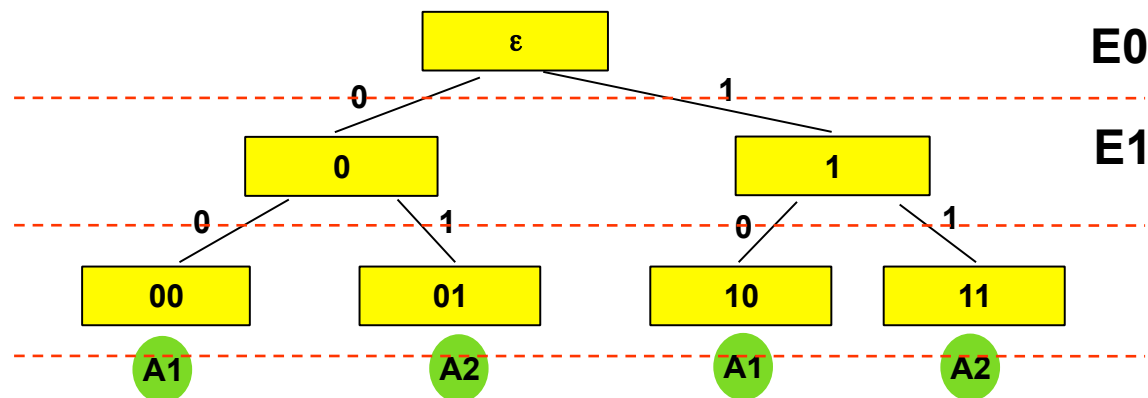
 if (E3) then A2

 else A1



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

Condition E0	yes	yes	no	no	Boolean cross product
Condition E1	yes	no	yes	no	
Action A1		X		X	Multiple choice quadrant
Action A2	X		X		



- 1) Elaborate decisions
- 2) Elaborate actions
- 3) Enter into table
- 4) Construct a cross boolean product as upper right quadrant (set of boolean vectors)
- 5) 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



What Students Should Do to Professors After Exams

Points ≤ 30	yes	yes	yes	yes	No	No	No	no
$50 < \text{Points}$	yes	yes	no	no	yes	yes	no	no
St. Francophil?	yes	no	yes	no	yes	no	yes	No
Student pays a Bordeaux			X				X	
Professor pays a Bordeaux					X			
Professor pays a beer						X		
Student pays a beer				X				
Professor drinks a beer	X	X						X



Common Columns Can Be Folded

Points \leq 30	yes	-	yes	no	no	no
50 < Points	yes	no	no	yes	yes	no
St. Francophil?	-	yes	no	yes	no	no
<hr/>						
Student pays a Bordeaux		X				
Professor pays a Bordeaux				X		
Professor pays a beer					X	
Student pays a beer			X			
Professor drinks a beer	X					X



Or Abbreviated to Else Action (Consolidated Decision Table)

Points ≤ 30	-	yes	No	No	Else
50 < Points	no	no	yes	yes	
Francophil?	yes	no	yes	no	
Student pays a Bordeaux	X				
Professor pays a Bordeaux			X		
Professor pays a beer				X	
Student pays a beer		X			
Professor drinks a beer					X

- 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



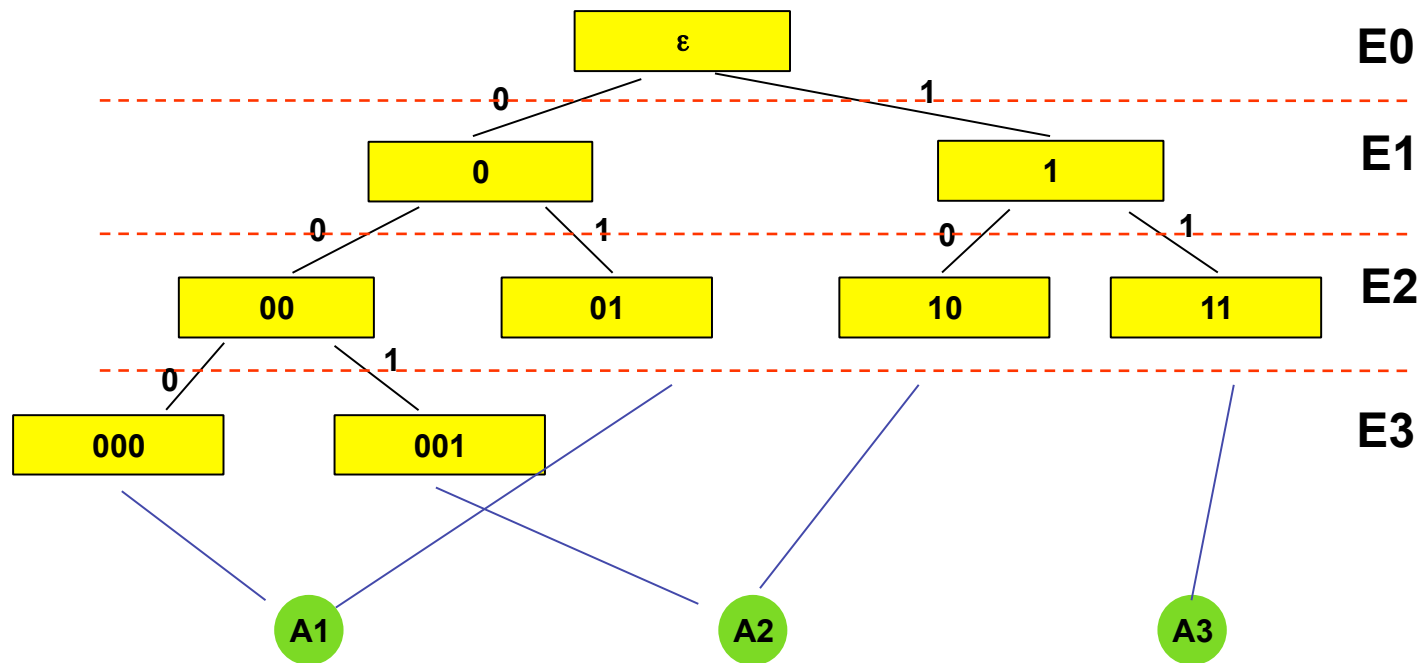
24.2 NORMALIZING CONTROL FLOW WITH NORMALIZED BDD

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

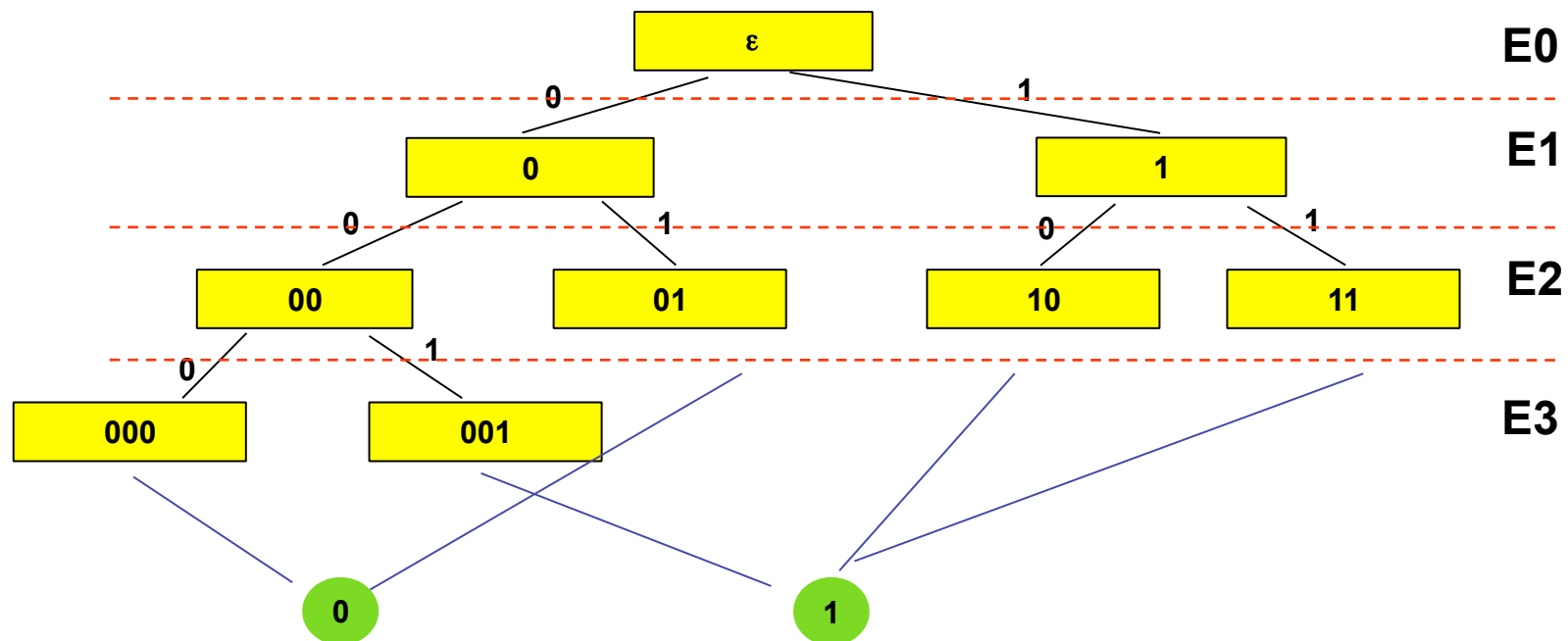
Condition E0	Yes	Yes	No	No
Condition E1	Yes	No	Yes	No
Value of F = 0	X		X	
Value of F = 1		X		X

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

- ▶ 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 $f: IB^n \rightarrow IB$ can be represented
- ▶ The decisions $E[i]$ are regarded as boolean variables

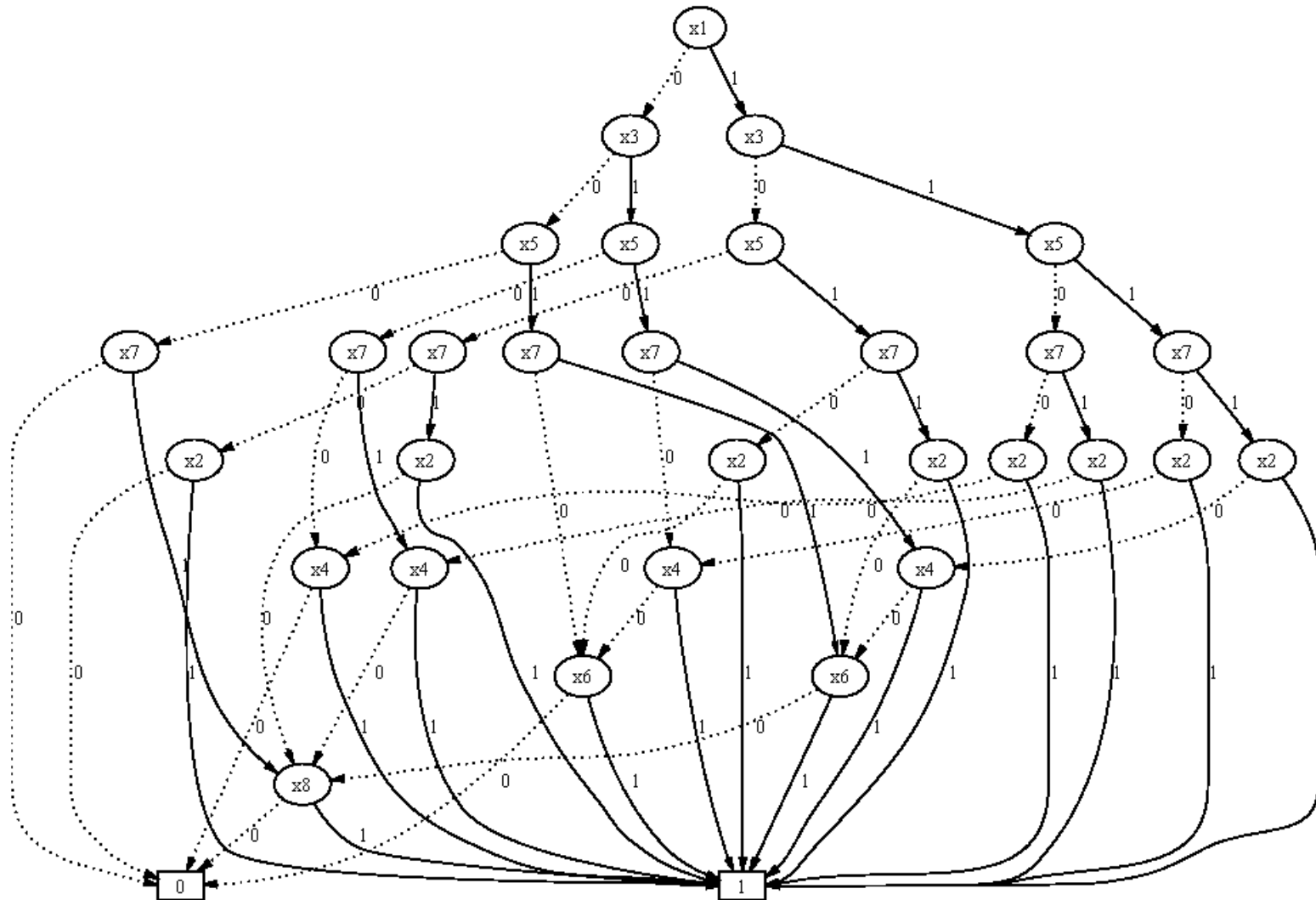


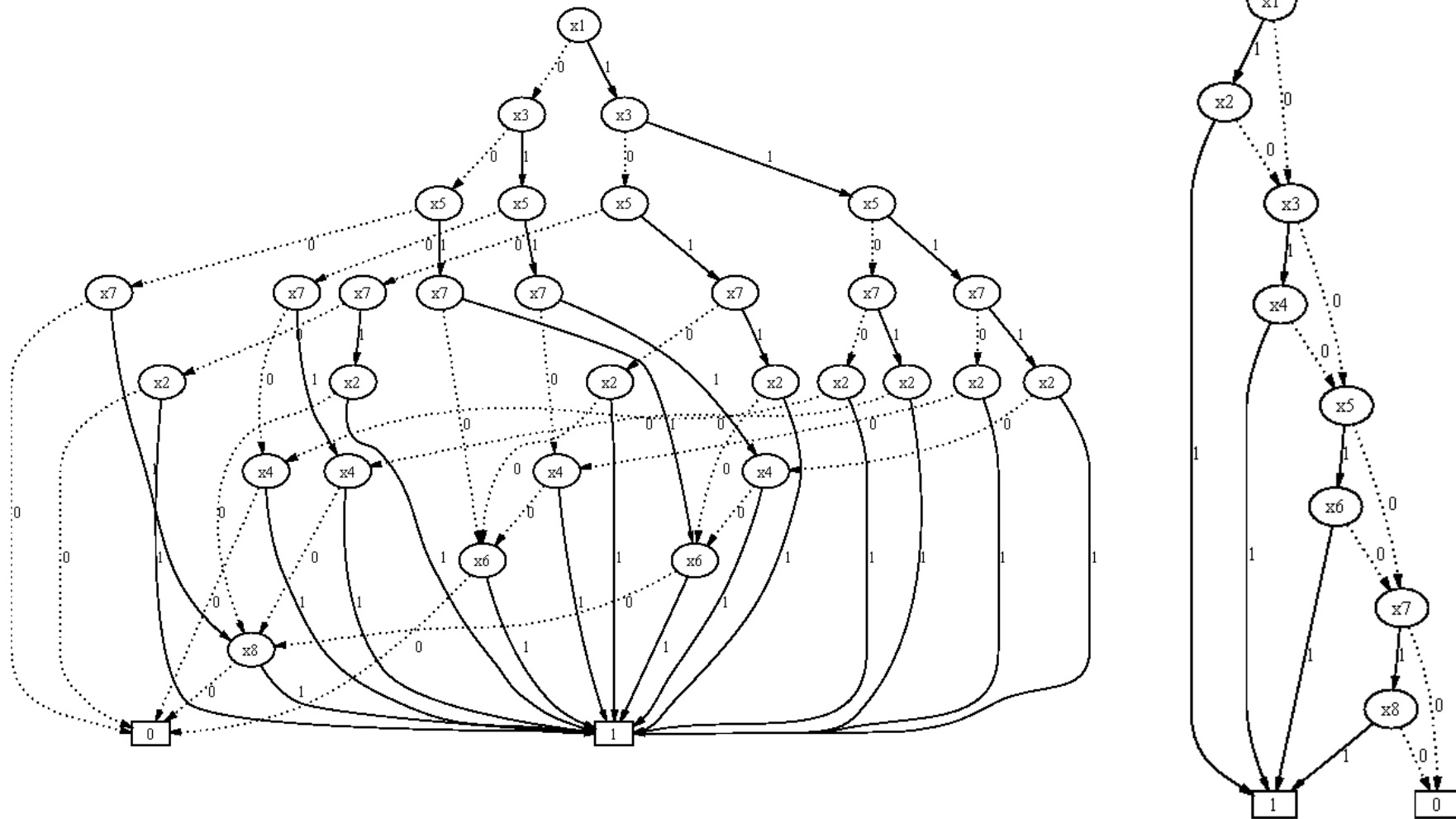


OBDDs (Ordered Binary Decision Diagrams)

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

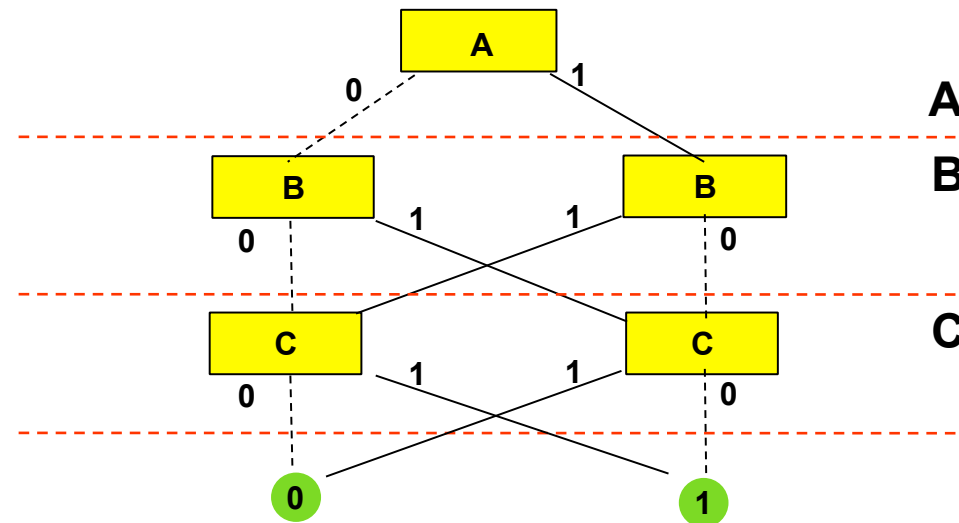
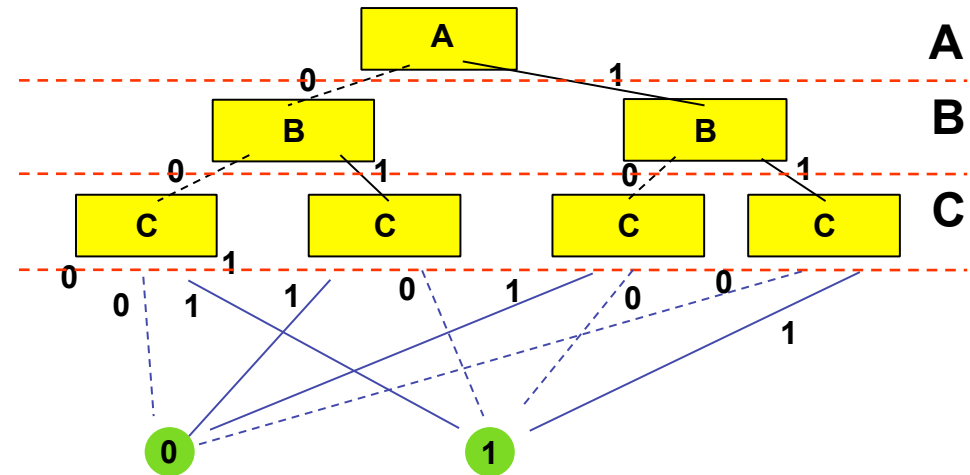
```
compareBooleanFunction() = {  
    Fix variable order for two BDD  
    Transform both BDD into OBDD  
    Compare both OBDD syntactically  
}
```





```

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]

- 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

- 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



24.2.2 MODEL CHECKING LARGE STATE SPACES



Many mathematical data types can be represented with decision algebras (most efficiently with BDD/OBDD):

- ▶ Functions over finite domains of size n [Bryant86]
 - Associate to every element a vector from \mathbb{B}^k , where $k = \text{Id } n$
 - Code sets with sets of such vectors
 - Map again to boolean algebra
- ▶ Sets, partial orders and lattices (e.g., in Z, VDM, SETL)
 - Represent subsets of a set in the powerset lattice of the set
 - Map the powerset lattice to a boolean algebra (theorem of Stone)
 - Use a BDD to encode the sets
 - Uniform efficient representation in space and time
- ▶ Relations and graphs
 - ▶ Interpret the elements of the relation (the edges) as sets of ordered k -tuples
 - ▶ Represent as in the case of sets
- ▶ State machines
 - ▶ Data-flow graphs
- ▶ Propositional logic formulas

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

- ▶ 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

24.3 VARIABILITY OF CA RULES

- Variability means that actions are exchanged for boolean combinations

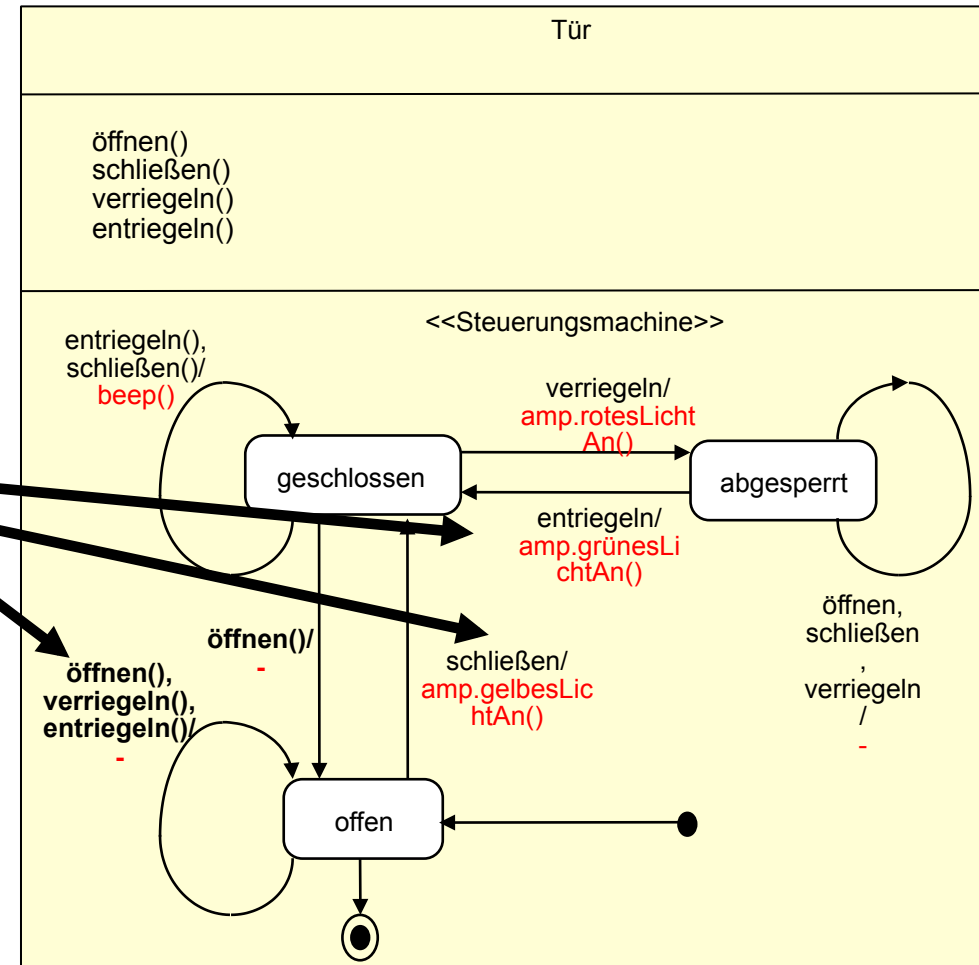
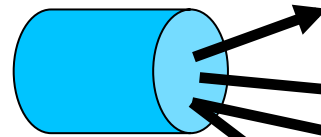
Variability is simple in CA-Systems

24.4 EVENT-CONDITION- ACTION BASED DESIGN (ECA)

- 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?

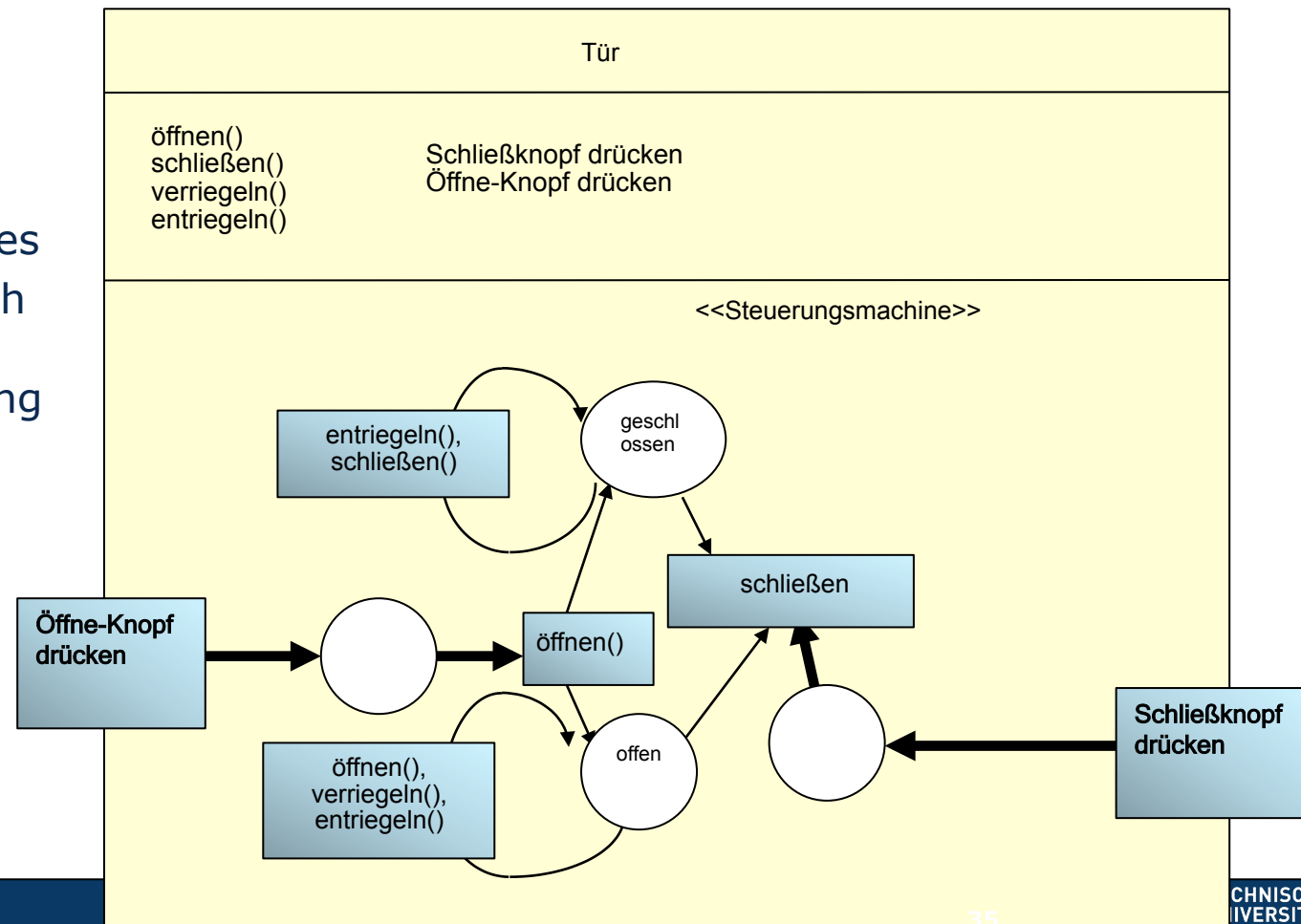
- ▶ An event-condition-action (ECA) system listens on channel(s) for events, analyses a condition, and executes an action
 - Statecharts (see course ST)
 - Petri Nets (see corr. Chapter)
 - ECA rules (Drools)
 - Condition analysis can be done by BDD
 - Verification by model checking
- Process:
 - Collect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



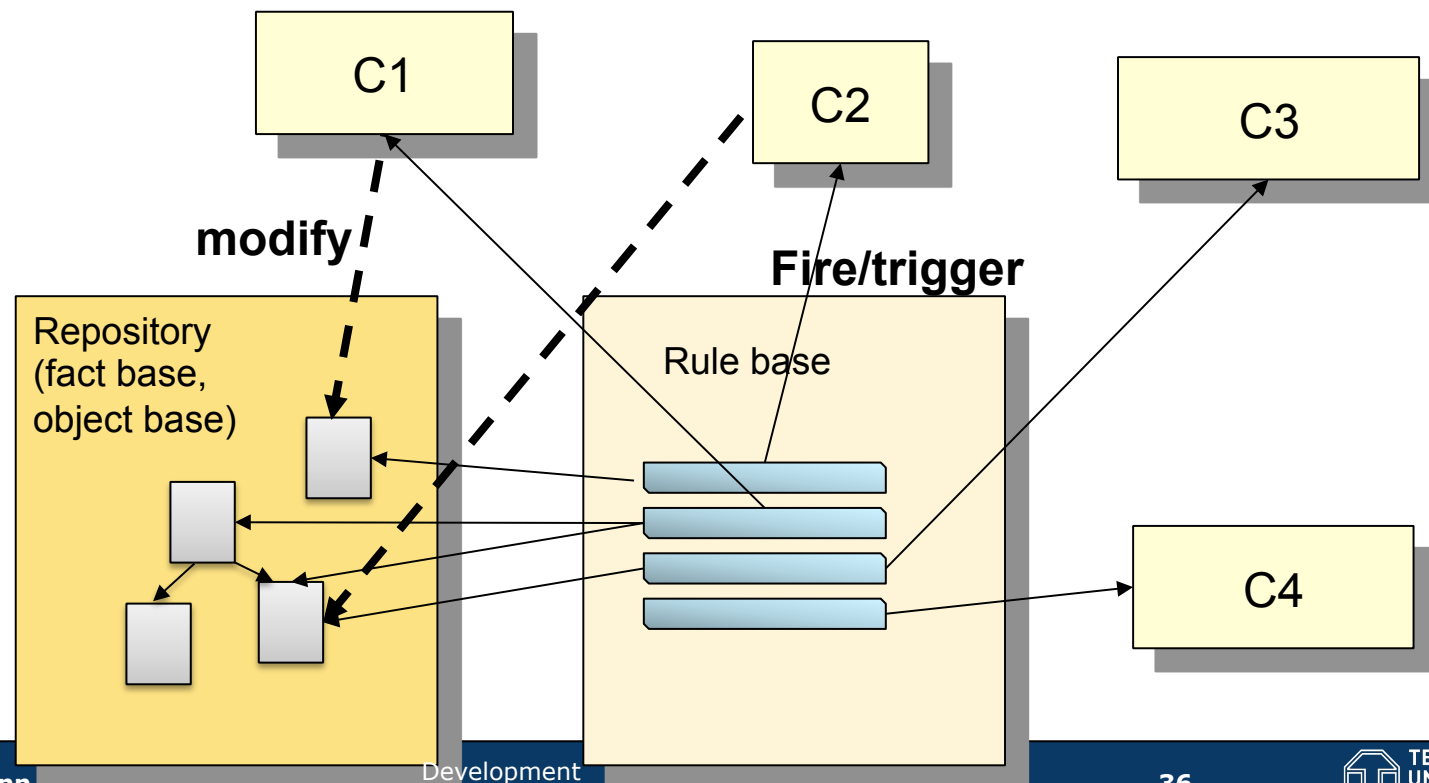
- ▶ In a Petri Net, an **event-generating channel** is a transition with fan-in=0
- ▶ Listening to the events, the Petri Net can do condition-action analysis

■ Process:

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



- 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 occurred by entering something in the repository (modify), components are fired/triggered to work on or modify the repository



- 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
```

- 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"
```

24.5 EXTENSIBILITY OF ECA RULES

- 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

- ▶ 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