Advanced Topics in Software Engineering (02265)

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science

Slides 82-151 provide the formalization of the concepts, Which however is not presented in the lecture in detail.

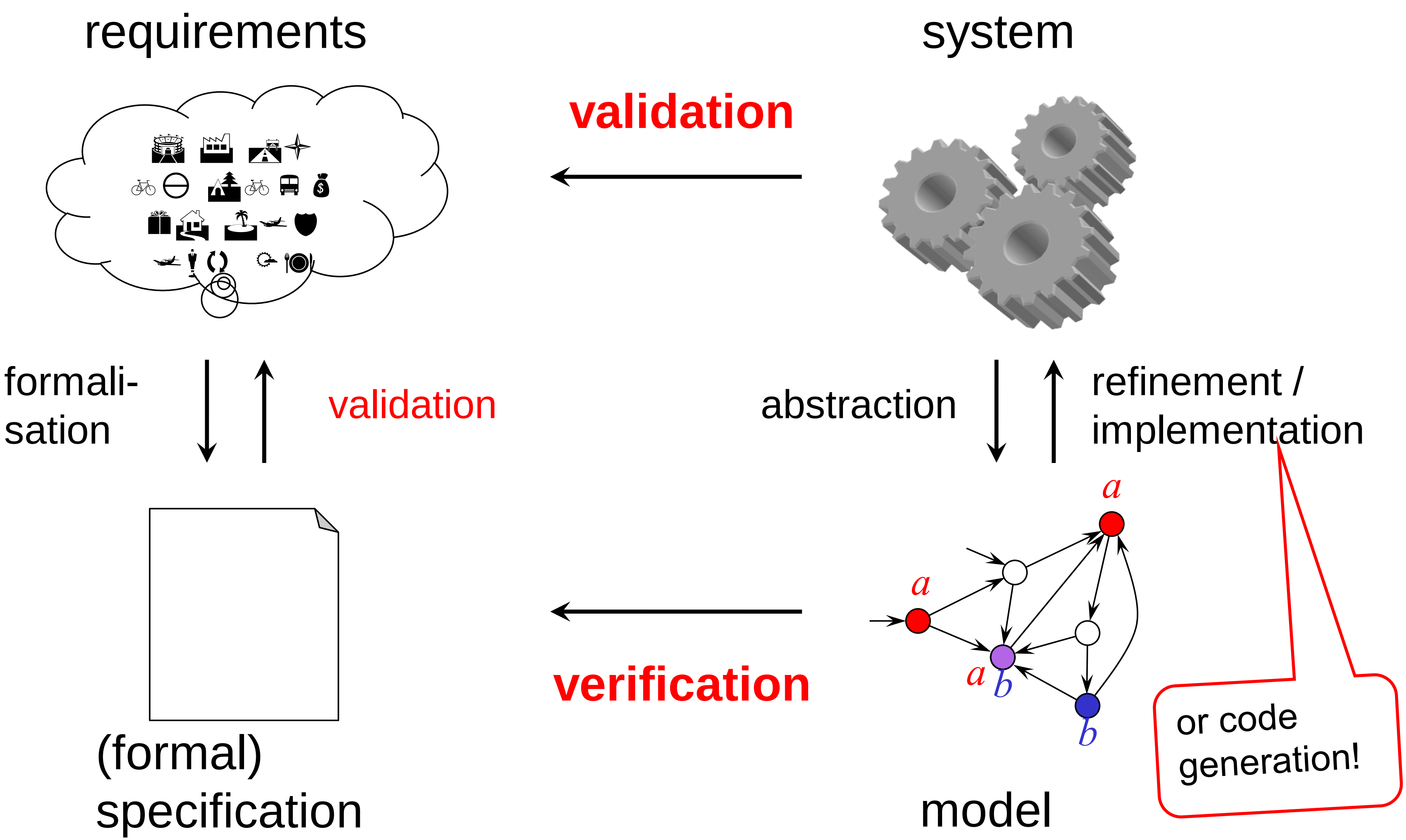
V. Formalisation and Analysis

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Model checking is a **technology** for the fully automatic **verification** of **reactive systems** with a finite state space.

Validation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

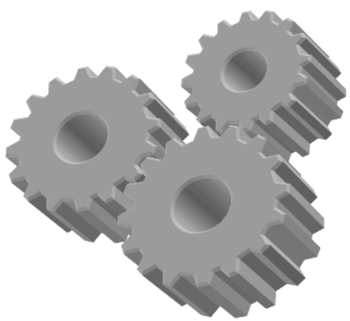


5.2. Main Concepts and Ideas

- Kripke structures (defining the system/model)
- CTL (specifying the properties)
- algorithms (only basic idea)
- complexity

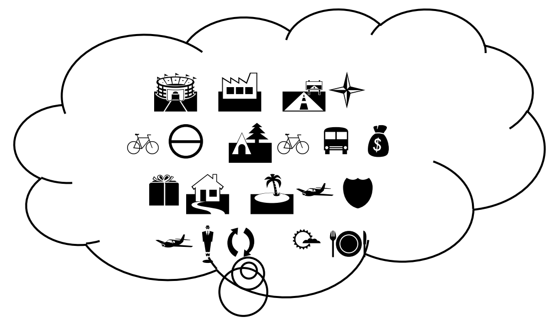
Systems and Requirements

system



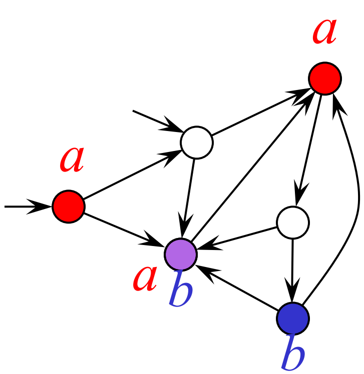
meets

requirements



DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

model M specification A

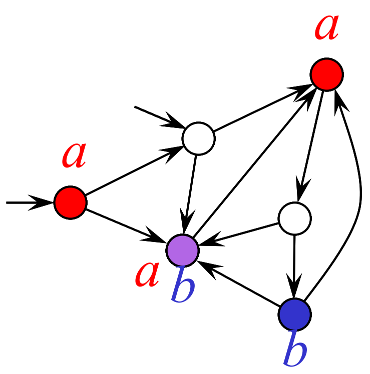


$$\models \quad \mathbf{AG}\,(\, a \Rightarrow \mathbf{AF}\, b\,)$$

Kripke structure **C**omputation **T**ree **L**ogic (**CTL**)

Kripke Structure

A **Kripke structure** consists of

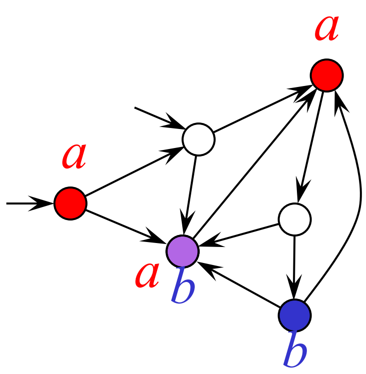


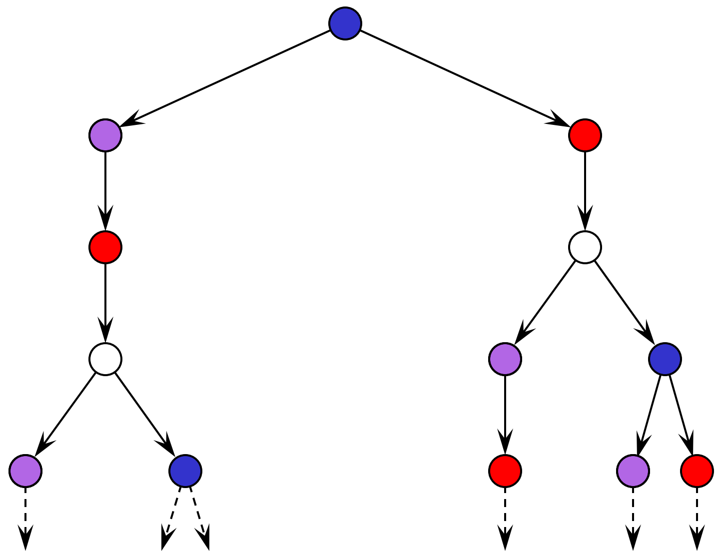
- a set of **states**,
- with distinguished **initial states**,
- a total **transition relation** and
- a **labelling** of states with a set of **atomic propositions**.

Total means that each state has a transition to somewhere!

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

The **behaviour** at a state can be represented as a **computation tree**:





Note that all paths are infinite!

That is a consequence of the totality of the transition relation.

CTL-Formulas

a, b, ...

$. \land .$, $. \lor .$, $\neg .$, ...

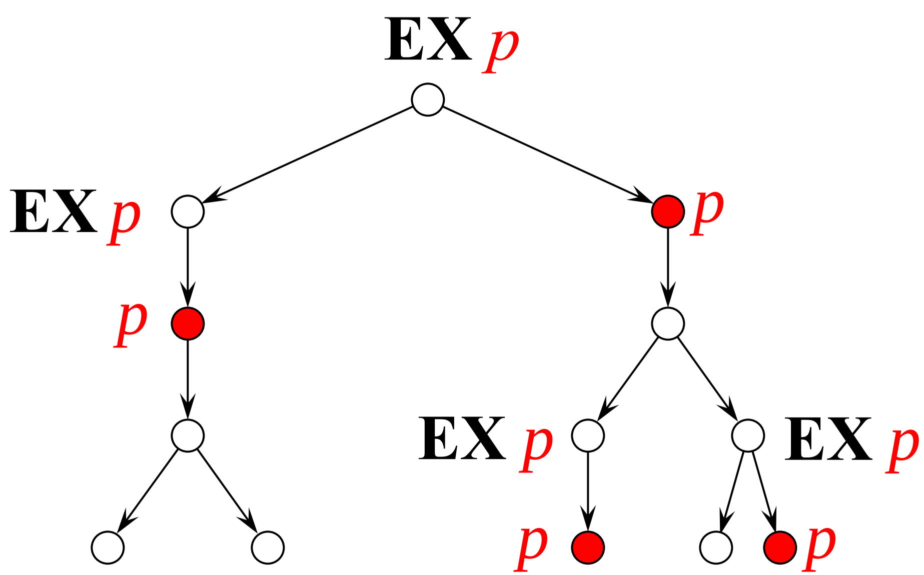
$\mathbf{EX}\ .$, $\mathbf{EG}\ .$, $\mathbf{E}[\ .\ \mathbf{U}\ .\ ]$, ...

CTL-formulas are inductively defined:

- atomic propositions are CTL-formulas
- CTL-formulas combined with a Boolean operator are CTL-formulas
- CTL-formulas combined with temporal operators are CTL-formulas

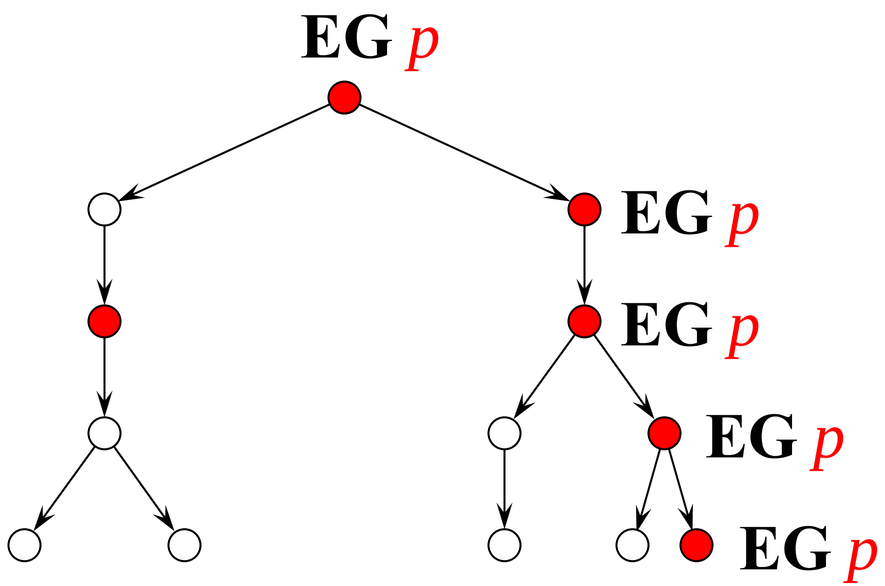
Exists neXt: $\mathbf{EX}\ p$

there exists an (immediate) successor in which p holds true:



Exists **G**lobally: **EG** *p*

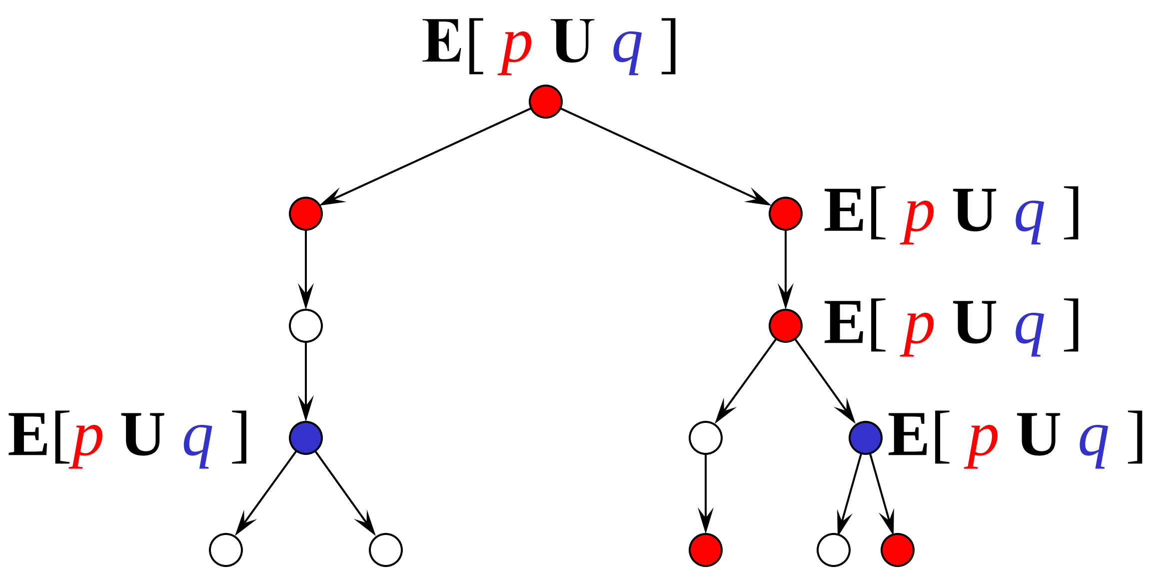
there exists an infinite path on which *p* holds in each state:



Exists Until: $\mathbf{E}[\ p\ \mathbf{U}\ q\ ]$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

there exists a reachable state in which b holds true, and up to this state p holds true:



Abbreviations

$\mathbf{AX}\ p \equiv \neg\ \mathbf{EX} \neg\ p$

for all immediate successors, p holds true

$\mathbf{EF}\ p \equiv \mathbf{E}\ [\ \mathit{true}\ \mathbf{U}\ p\ ]$

in some reachable state, p holds true

$\mathbf{AG}\ p \equiv \neg\ \mathbf{EF} \neg\ p$

in all reachable states, p holds true

$\mathbf{AF}\ p \equiv \neg\ \mathbf{EG} \neg\ p$

on each path, there exists a state in which p holds true

Dualities

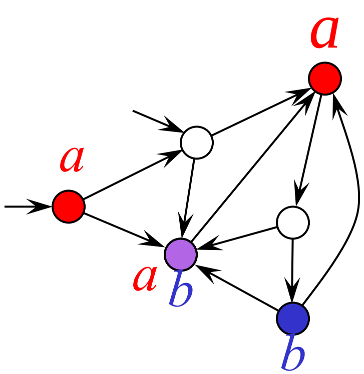
System “meets” formula

A CTL-formula **holds** for a Kripke structure if the formula holds in each initial state.

Example

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

model M $\qquad\qquad$ specification p



$$\models \quad \mathbf{AG}\,(\,a \Rightarrow \mathbf{AF}\; b\,)$$

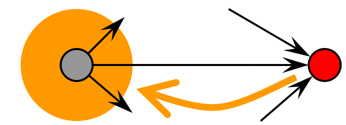
How do we prove it?

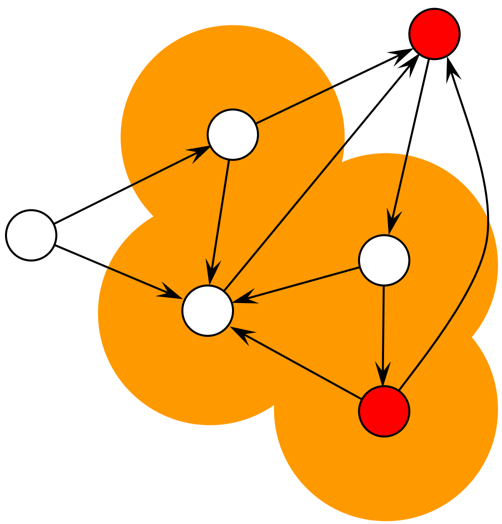
Algorithms

For each sub-formula, we inductively calculate the **set of states**, in which this sub-formula is true:

- atomic propositions ☑
- Boolean operators ☑
- temporal operators

„Algorithm“ for $\mathbf{EX}\ p$





Given:
The set of states in which p holds: S_p

Wanted:
The set of states in which $\mathbf{EX}\ p$ holds: $S_{\mathbf{EX}\ p}$

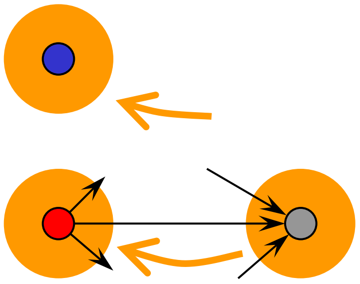
We also write $\mathbf{EX}(S_p)$ for $S_{\mathbf{EX}\ p}$

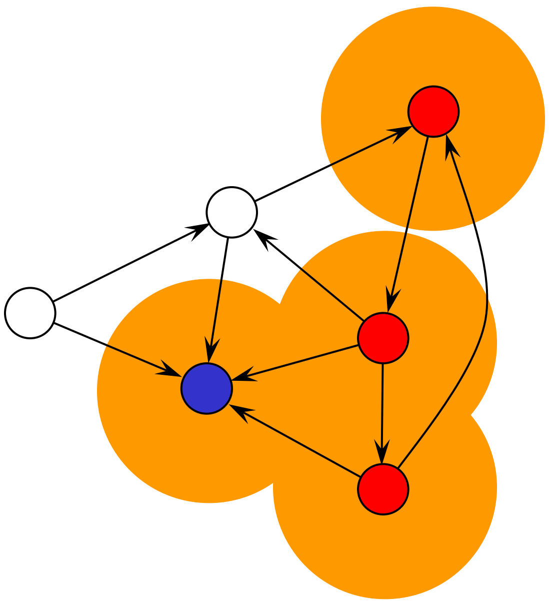
Algorithm for $\mathbf{E}[\, p \;\; \mathbf{U} \; q \,]$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

As stated here, this algorithm is quite inefficient. There are more efficient ways to do this.

But, even this inefficient algorithm turns out to be quite efficient when used with the right data structure (ROBDDs, see 5.4).





Given: S_p und S_q
Wanted: $S_{\mathbf{E}[\, p \;\mathbf{U}\; q \,]}$

$$S_0 = \varnothing$$

$$S_1 = S_q \cup (S_p \cap \mathbf{EX}(S_0))$$

$$S_2 = S_q \cup (S_p \cap \mathbf{EX}(S_1))$$

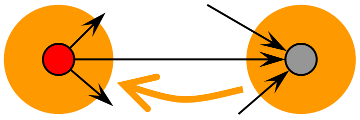
...

$$S_{i+1} = S_q \cup (S_p \cap \mathbf{EX}(S_i))$$

until $S_{i+1} = S_i = S_{\mathbf{E}[\, p \;\mathbf{U}\; q \,]}$

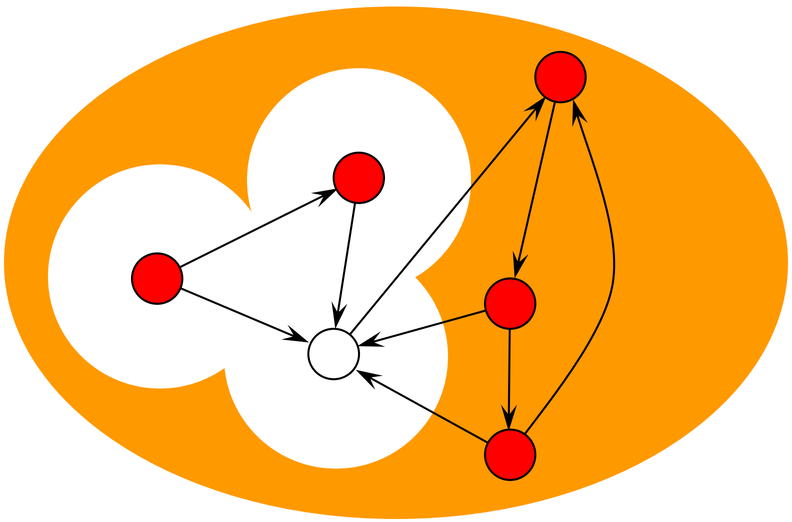
Algorithm for $\mathbf{EG}\ p$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Ditto

Given: S_p
Wanted: $S_{\mathbf{EG}\ p}$



$S_0 = S$

$S_1 = S_p \cap \mathrm{EX}(S_0)$

$S_2 = S_p \cap \mathrm{EX}(S_1)$

...

$S_{i+1} = S_p \cap \mathrm{EX}(S_i)$

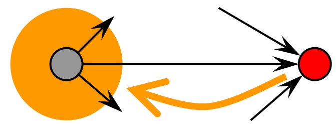
until $S_{i+1} = S_i = S_{\mathbf{EG}\ \mathrm{p}}$

Algorithms Summary

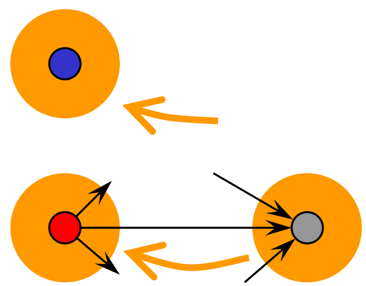
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

CTL model checking ~ marking algorithm + iteration

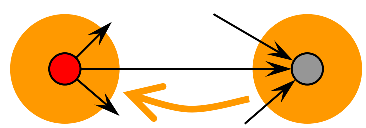
- **EX** p



- **E**[p **U** q]



- **EG** p



Complexity

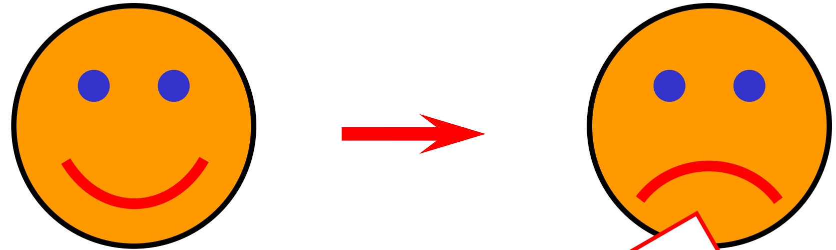
When implemented in an efficient way, the marking algorithm for each operator is linear in the number of states of the system:

$$O(\,|\,M\,|\cdot|\,p\,|\,)$$

size of the model

size of the formula

Complexity



When implemented in an efficient way, th... ...ng algorithm for each operator is linear ... ...mber of states of the system:

$$O(|M| \ldots$$

state space explosion

State Space Explosion

- The number of states of a system is exponential in the number of its variables

- Therefore, naive model checking algorithms are doomed to fail in practice:
  - → more efficient data structures
  - → improved algorithms
  - → partial investigation of state space
  - → …

The main issue in model checking is:

How to avoid or at least to restrict the negative effect of the state space explosion?

5.3. System model

- Kripke Structures
- Syntactic Representation
- Examples

Kripke Structures

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Motivation
- Definition
- Computation paths
- Transition systems

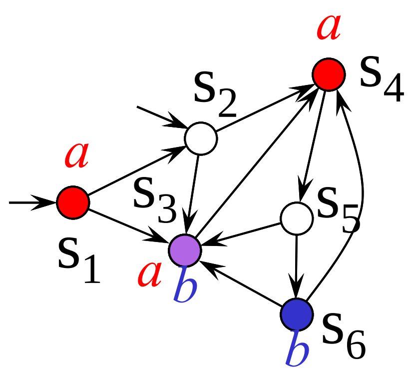
Motivation

There are many different notations for reactive systems; the choice depends on the application area and the purpose of the model.

Most model checking techniques are independent from the particular notation. Therefore, we do not fix a notation.

Rather we define **Kripke structures** as a common underlying **semantic model**.

A **Kripke structure** M consists of



- a finite **set of states**: S,
- a set of **initial states**: $S_0 \subseteq S$,
- a total **transition relation**: $R \subseteq S \times S$
- a **labelling** of the states with a set of **atomic propositions** AP: $L: S \to 2^{AP}$

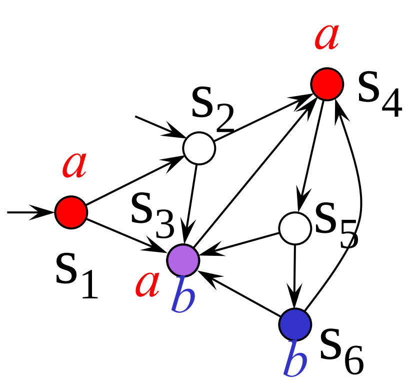
Set of all subsets of AP.

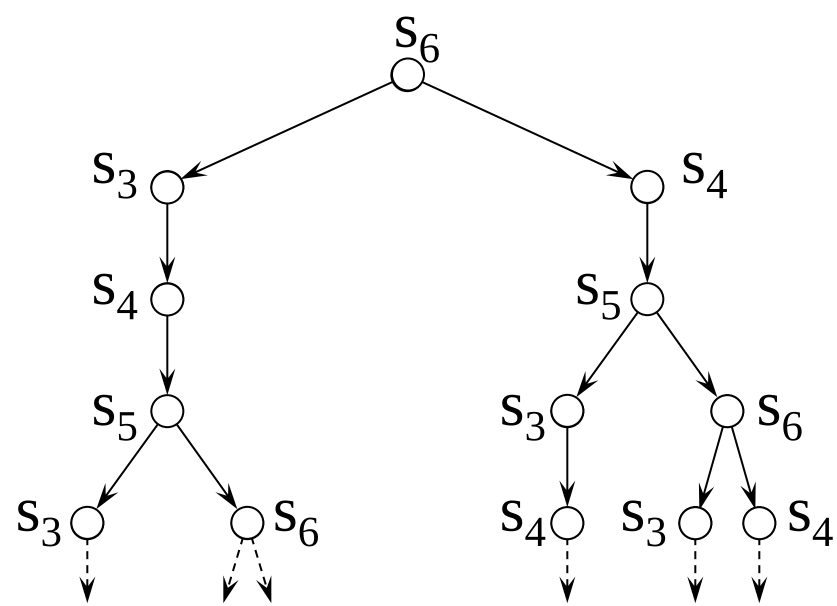
Paths and Computation Trees

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

The set of all paths of M in a state s can be represented as an infinite tree, the **computation tree** of M in s :

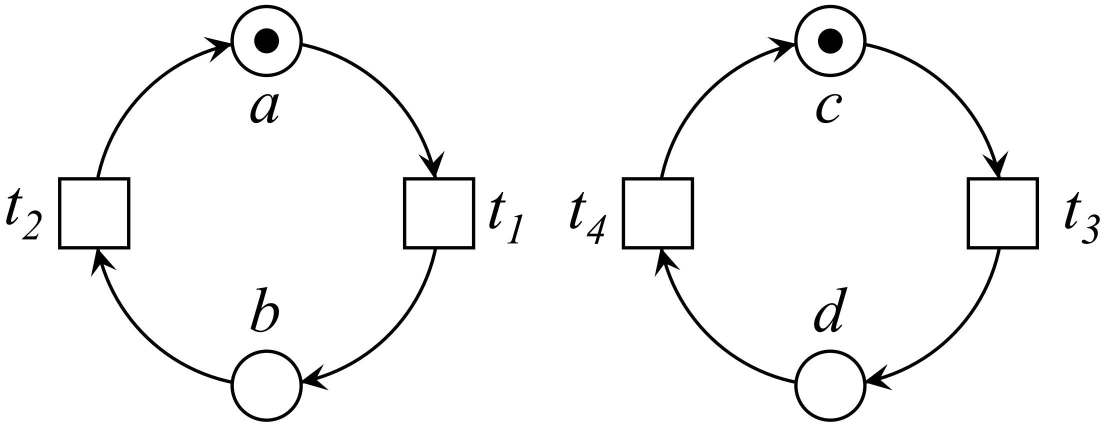
Example:



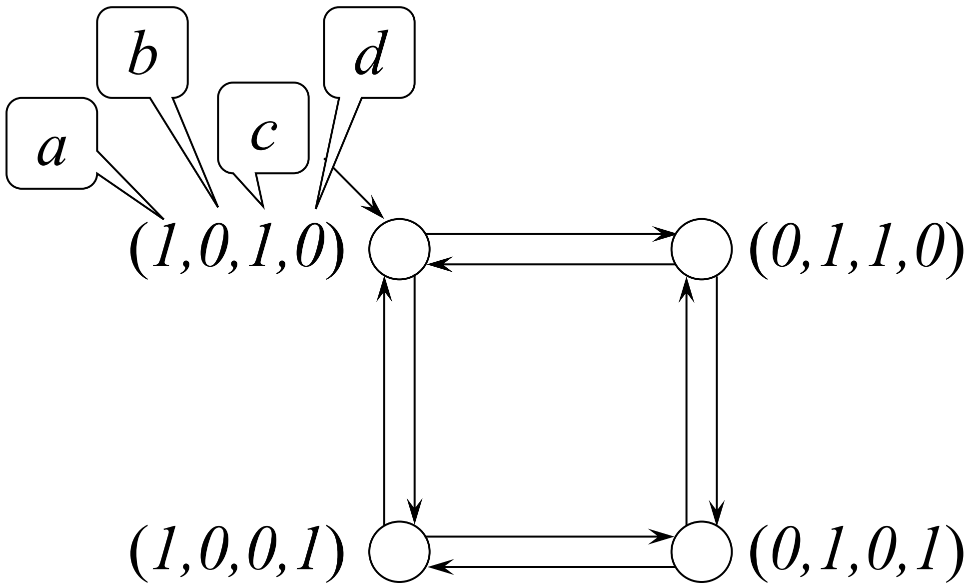


Since the transition relation R is total, all paths (branches) of the tree are infinite!

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

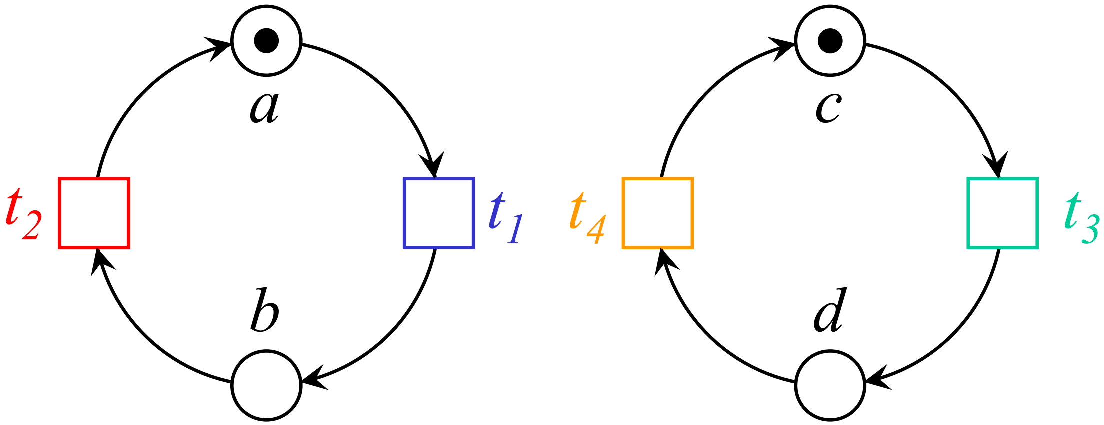


A Petri net



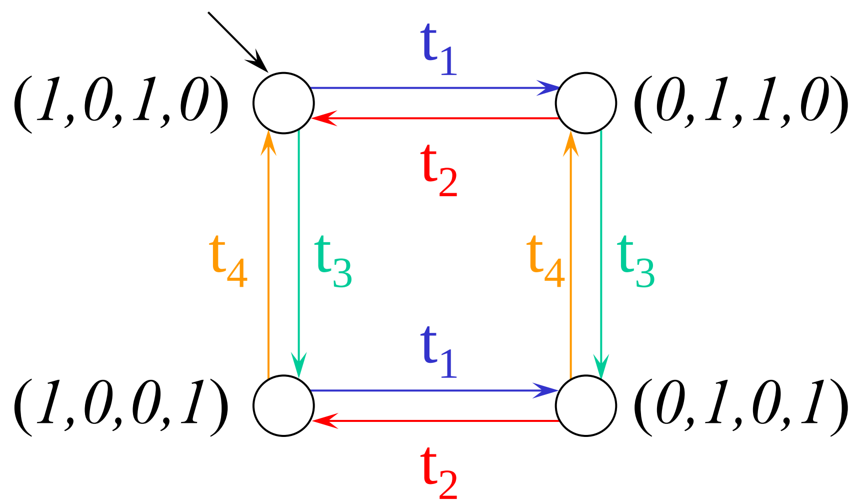
The corresponding Kripke structure

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



A Petri net

The information on related transitions is lost in the Kripke structure!



Extensions

- Labelling of transitions: **Transition systems**
- Instead of a single transition relation, there are many transition relations (in our example for every Petri net transition).

This is also important for efficiency reasons!

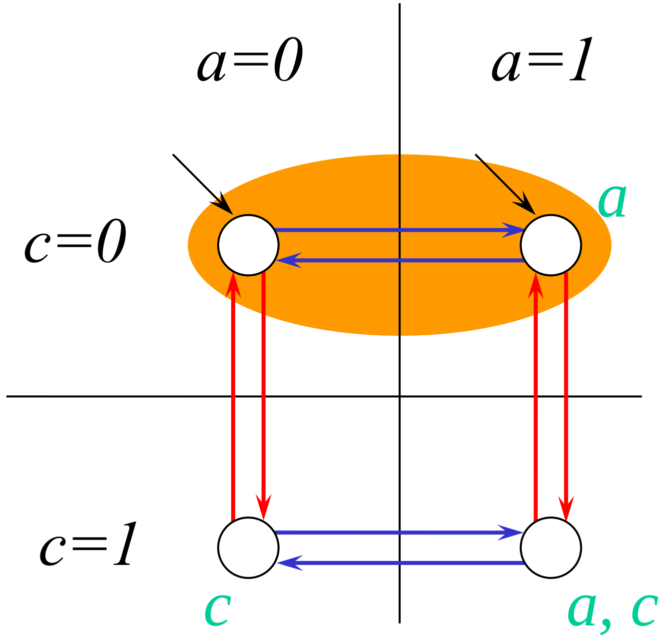
Syntactic Representation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Motivation & Example
- States
- Initial states
- Transitions
- Labels

Formula representation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



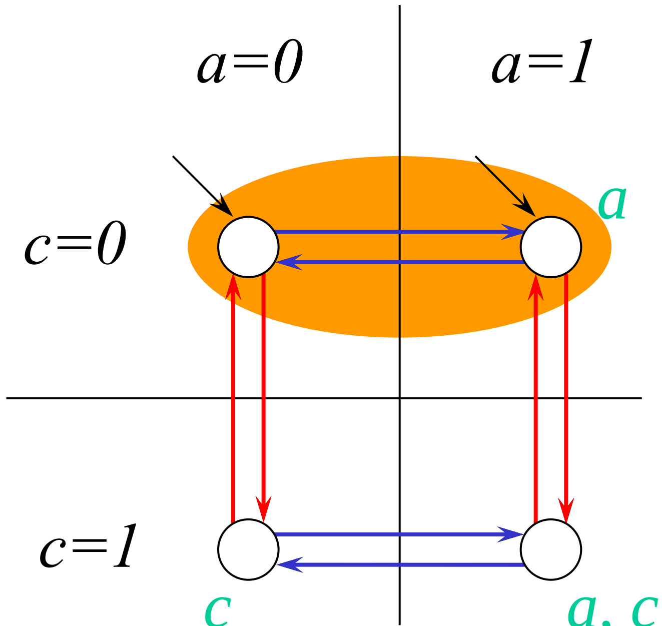
$S = \{ (0,0), (0,1), (1,0), (1,1) \}$

$S_0 = \{ (0,0) , (1,0) \}$

$R = \{ ((0,0),(1,0)) , ((1,0),(0,0)) ,$
$((0,1),(1,1)) , ((1,1),(0,1)) ,$
$((0,0),(0,1)) , ((0,1),(0,0)) ,$
$((1,0),(1,1)) , ((1,1),(1,0)) \}$

- Boolean variables:
  $V = \{ a, c \}$
- Initial formula:
  $S_0 \equiv \neg c$
- Transition formula:
  $\mathcal{R} \equiv$
  $(a' = \neg a \land c' = c) \lor$
  $(a' = a \land c' = \neg c)$
- Implicit labelling:
  $AP = V$

As a Transition System



- Boolean variables:
  $V = \{ a, c \}$
- Initial formula:
  $S_0 \equiv \neg c$
- Transition formula:
  $\mathcal{T} \equiv$
  $\{ (a' = \neg a \wedge c' = c),$
  $(a' = a \wedge c' = \neg c) \}$

  Implicit labelling:
  $AP = V$

This equality is often implicit for variables that do not occur “primed”.

For example in MCiE (important for efficiency).

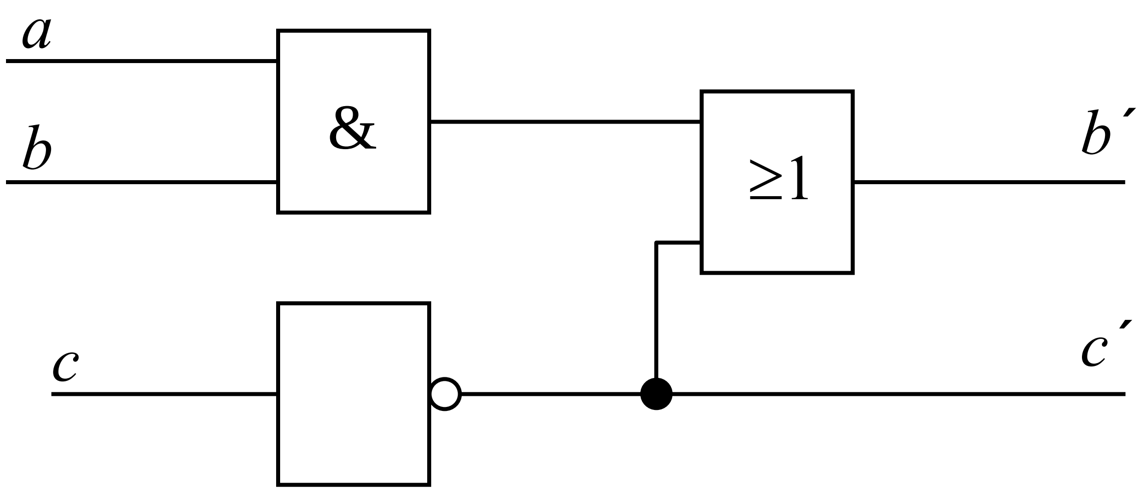
More Examples

In this section, we show by the help of two examples how to represent different kinds of systems as Kripke structures represented by formulas.

- Synchronous circuit (hardware)
- Concurrent processes
- Petri nets

Combinatorial Circuit

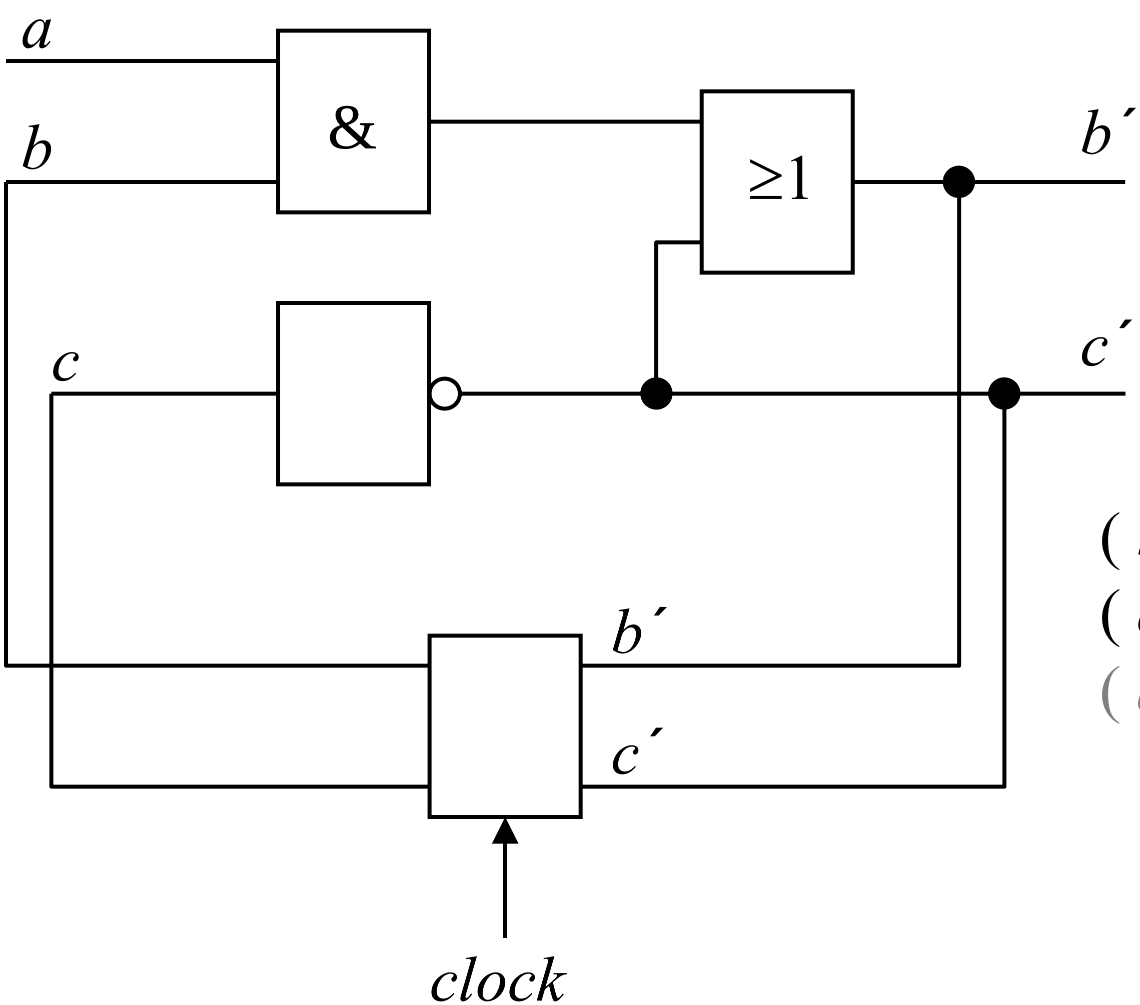
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



$$b' = (a \wedge b \;\vee\; \neg\, c\,)$$
$$c' = \neg\, c$$

Sequential Synchronous Circuit

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



$$(b' = (a \land b \lor \neg c)) \land$$
$$(c' = \neg c) \land$$
$$(a' = 0 \lor a' = 1)$$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
              loop forever          ||          loop forever
pca = 0         x:= 0;              ||  pcb = 0   x:= 1;
pca = 1         y:= 0;              ||  pcb = 1   y:= 1;
```

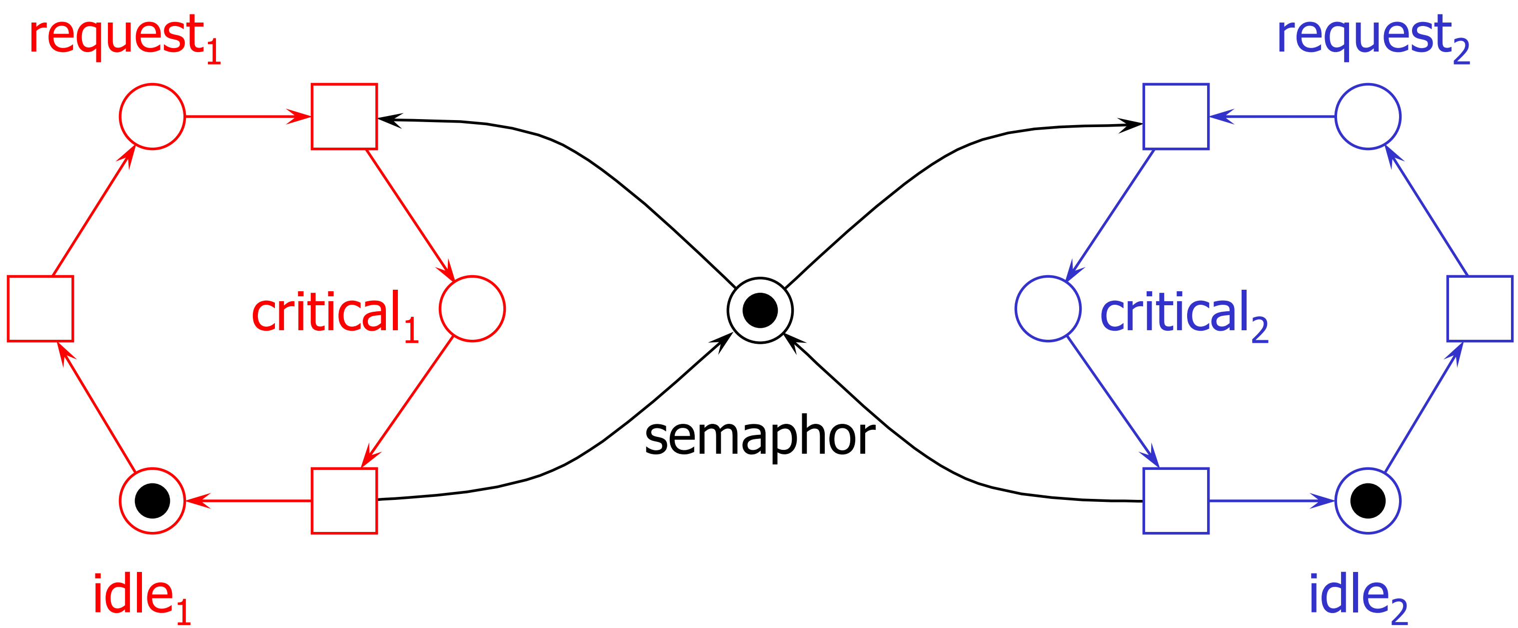
$$(pca = 0 \wedge pca' = 1 \wedge x' = 0 \;\wedge\; y' = y \wedge pcb' = pcb) \vee$$

$$(pca = 1 \wedge pca' = 0 \wedge y' = 0 \;\wedge\; x' = x \wedge pcb' = pcb) \vee$$

$$(pcb = 0 \wedge pcb' = 1 \wedge x' = 1 \;\wedge\; y' = y \wedge pca' = pca) \vee$$

$$(pcb = 1 \wedge pcb' = 0 \wedge y' = 1 \;\wedge\; x' = x \wedge pca' = pca)$$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



How do the formulas look for Petri nets as a transition system?

5.4 ROBDDs

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Reduced Ordered Binary Decision Diagrams; for simplicity often just called Binary Decision Diagrams (BDDs).

- Motivation
- Definition
- Operations on ROBDDs
- Quantified Boolean formulas (QBF)

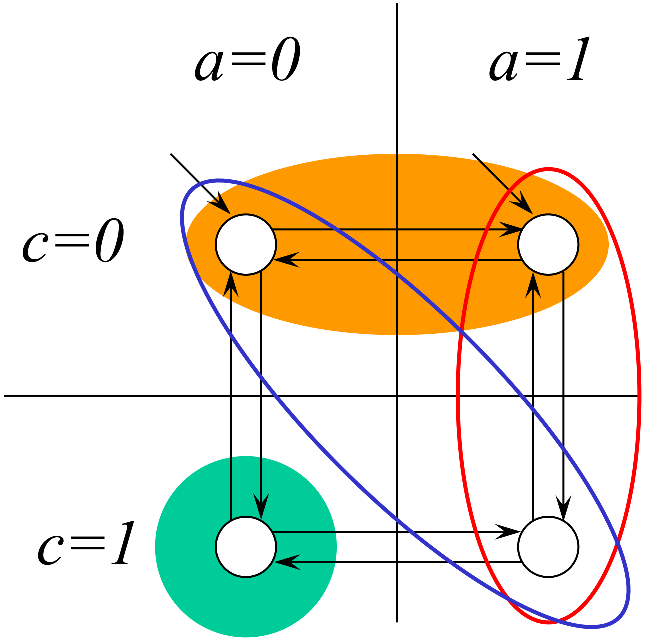
Motivation

- The number of states of realistic systems is gigantic.

$\Rightarrow$ Representing sets of states by enumerating every state explicitly is a bad idea.

- Sets could be represented “symbolically”, e.g. by formulas (see next slide)

Sets as formulas



$\neg$ c

a

a $\Leftrightarrow$ c

$\neg$ a $\wedge$ c

Boolean formulas representing sets of states

Formulas

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Some operations on sets can be efficiently executed for sets that are represented as formulas:
  - union: $p \vee q$
  - disjunction: $p \wedge q$
  - complement: $\neg p$
  - set difference: $p \wedge \neg q$

Problem:

- the same set can have different representations
- it is extremely inefficient to find out whether two formulas represent the same set (NP-complete).
- therefore, formulas are not a good representation for sets of states.

Checking for equality of sets is a very crucial operation in model checking! (BTW: why?) → slide 19/20 (78)

Goal

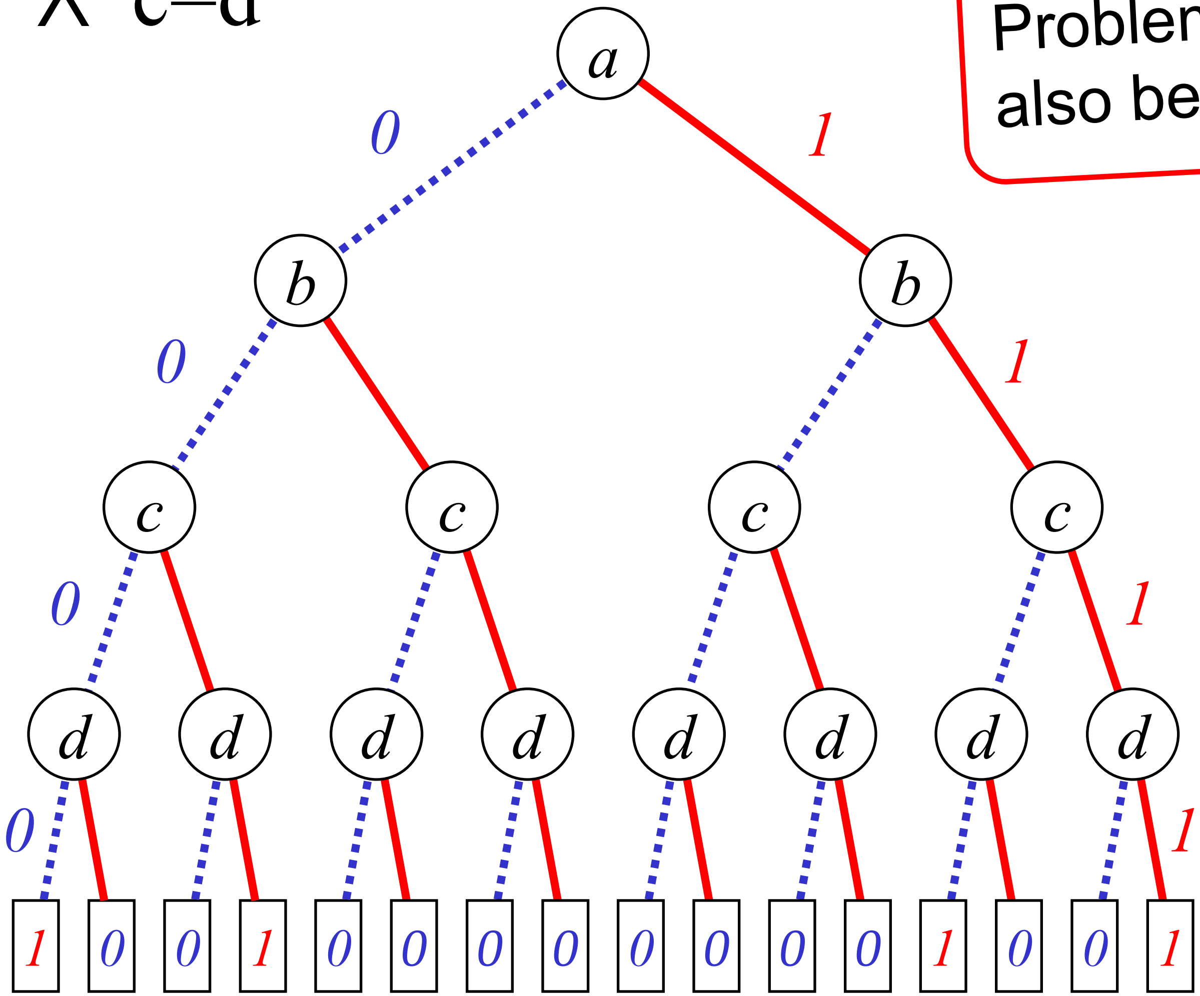
- Representation of sets such that
  - set operations **and**
  - check for equality

  can be computed efficiently

The answer will be Reduced Ordered Binary Decision Diagrams (ROBDDs)!

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

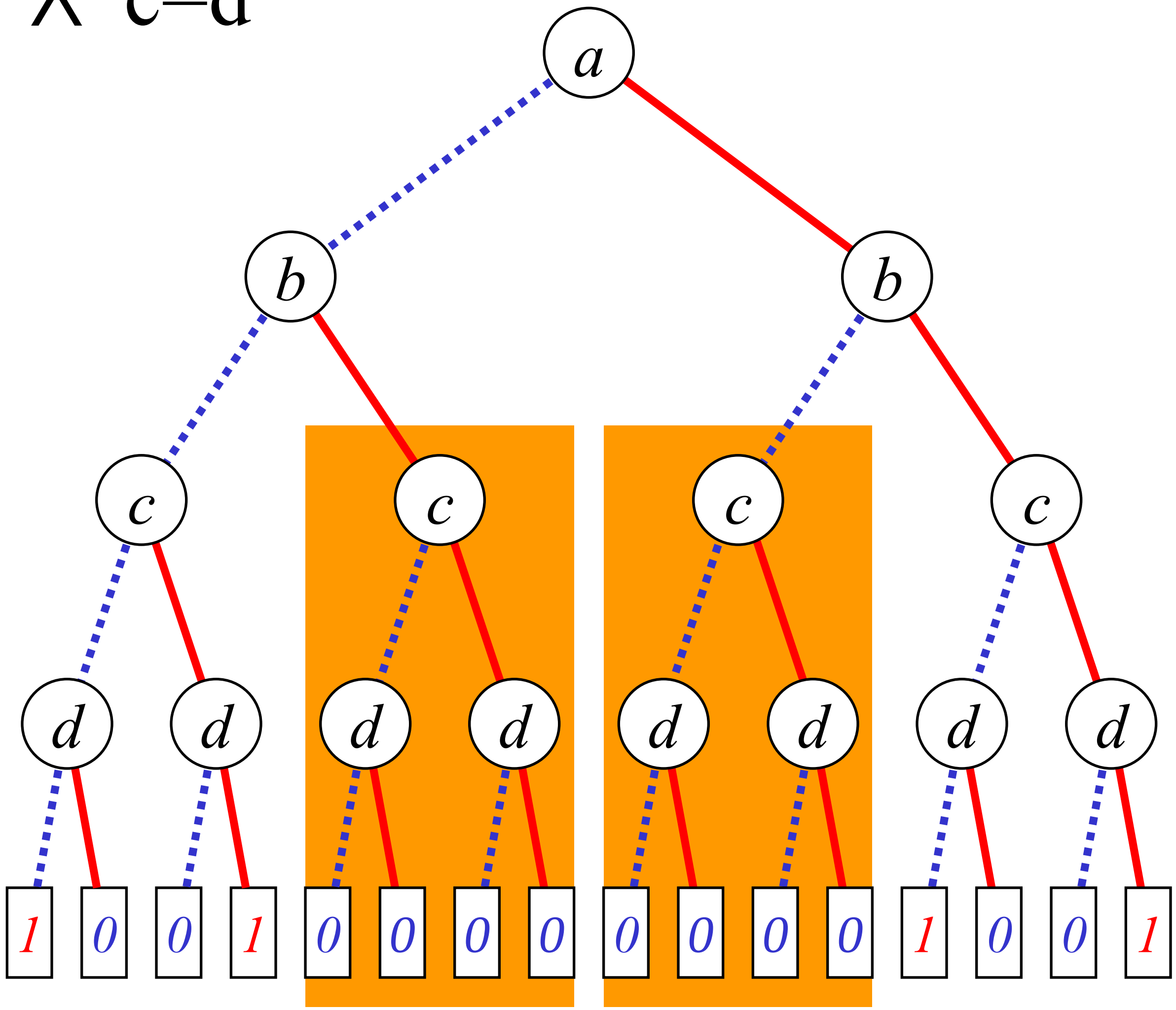
a=b ∧ c=d



Problem: These will also be very big!

Identify same sub-trees

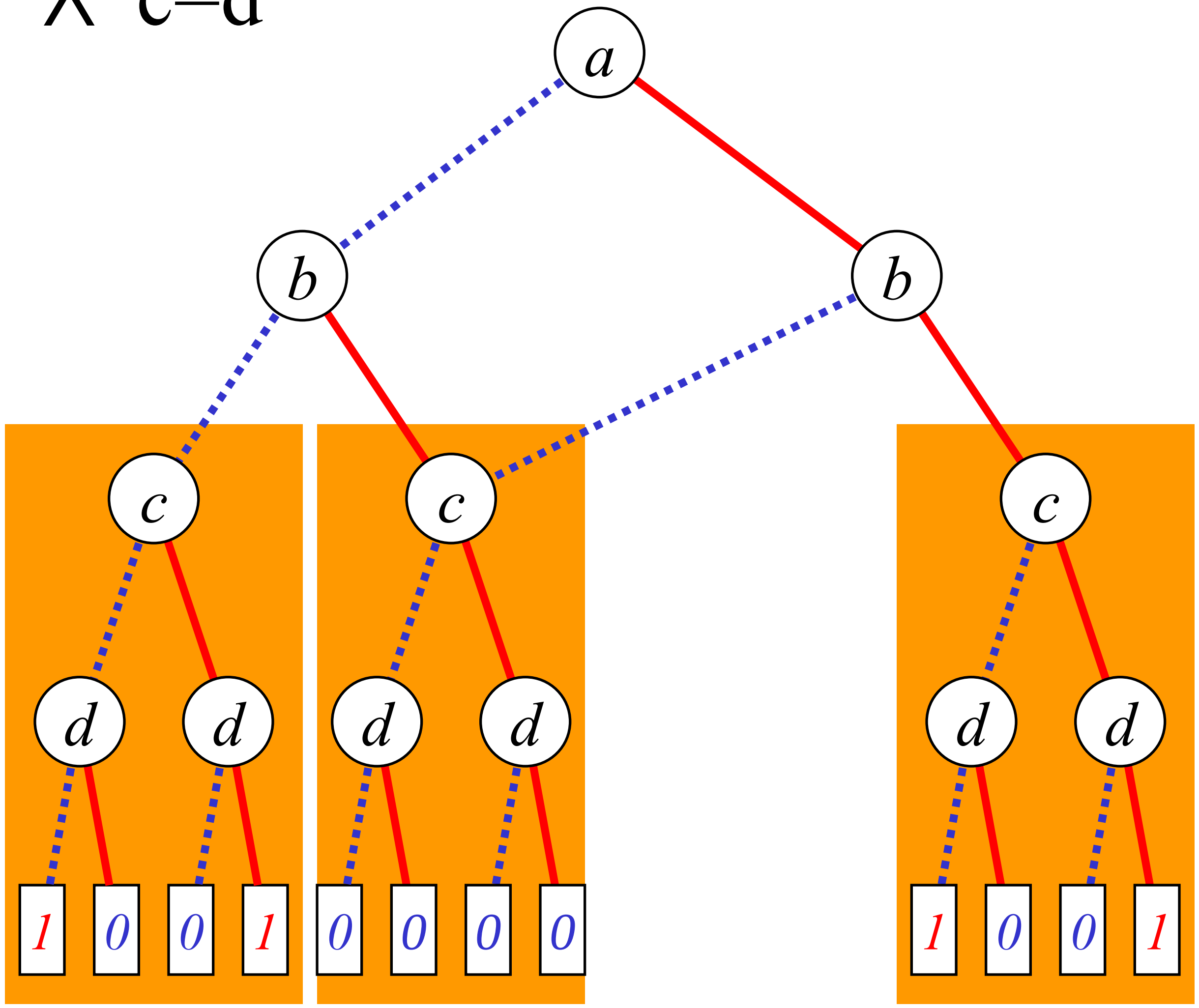
$a = b \;\wedge\; c = d$



Identify same sub-trees

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

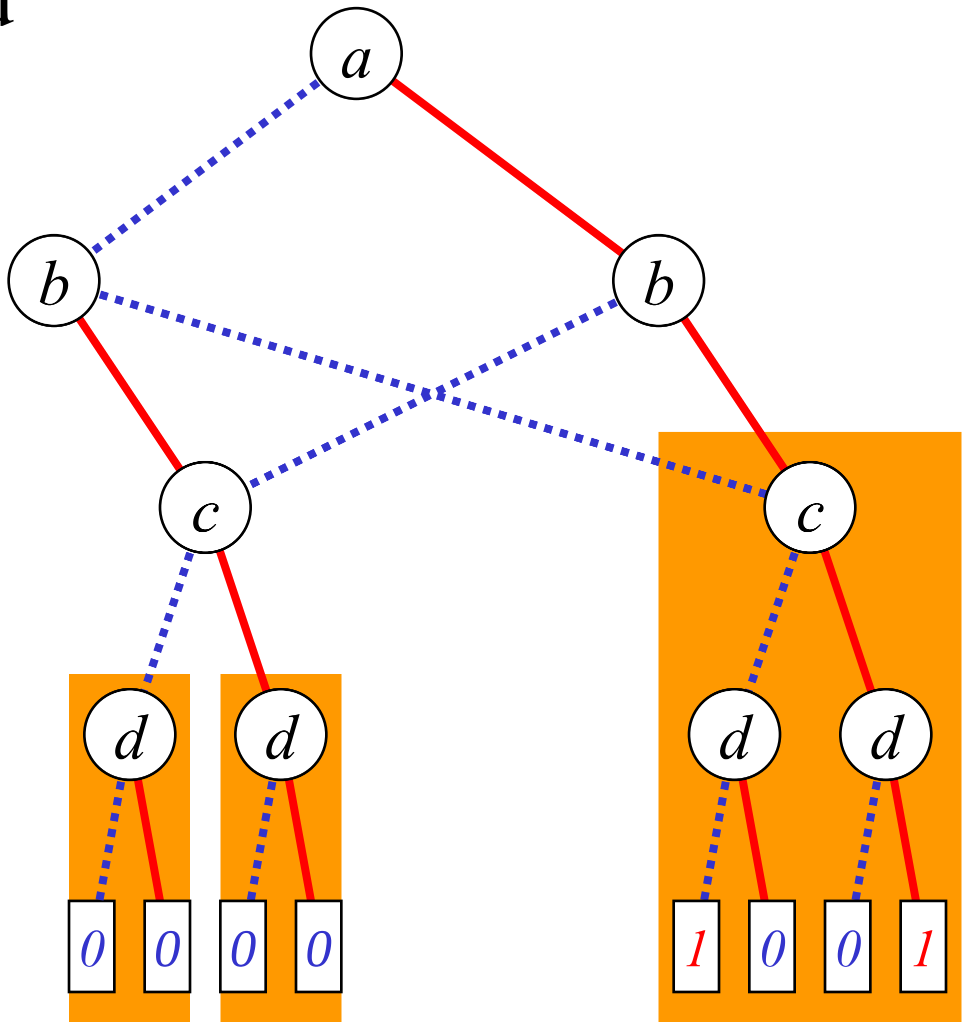
$a{=}b \wedge c{=}d$



Identify same sub-trees

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

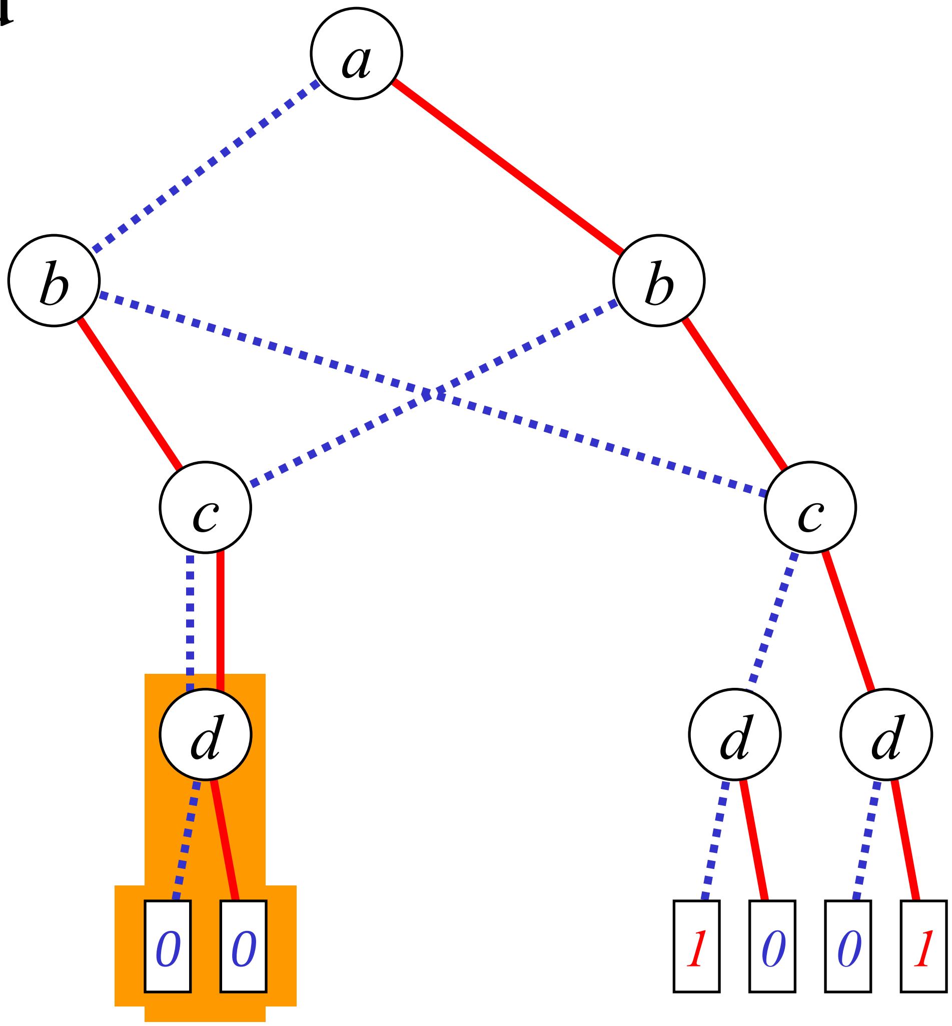
a=b ∧ c=d



Identify same sub-trees

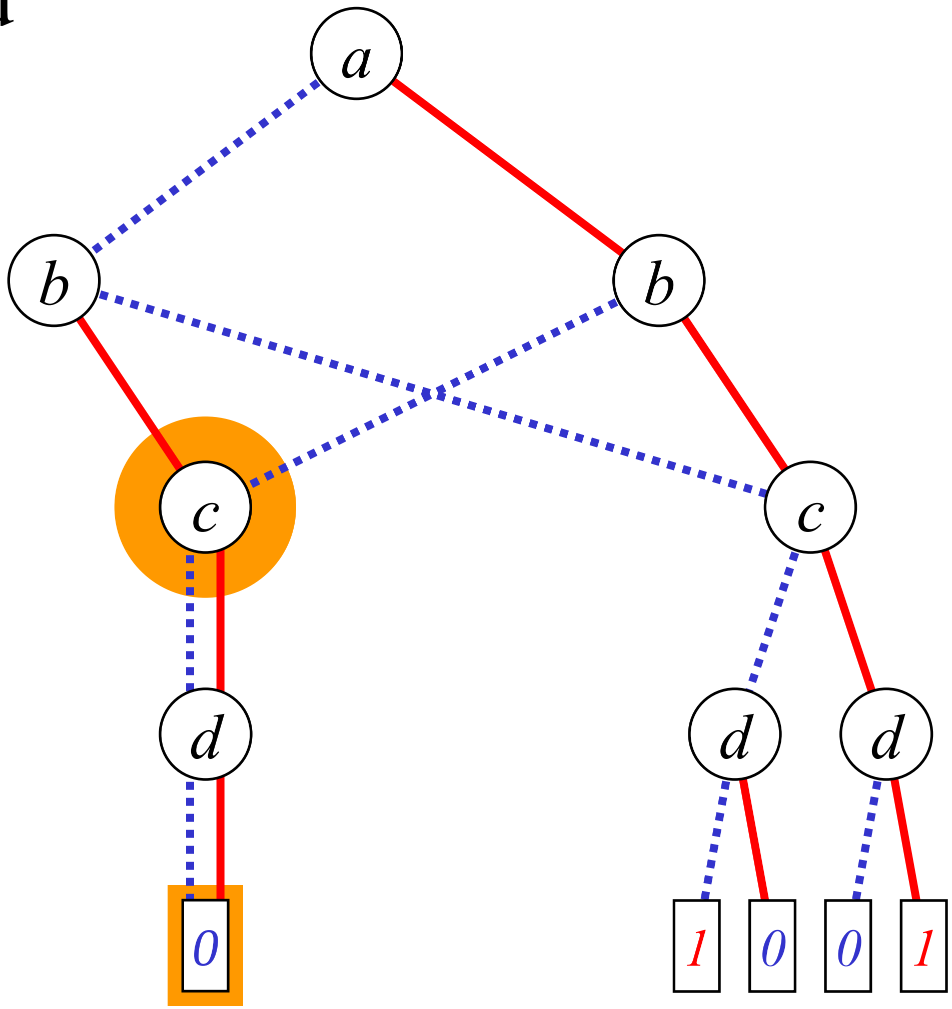
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

a=b ∧ c=d



Delete redundant nodes

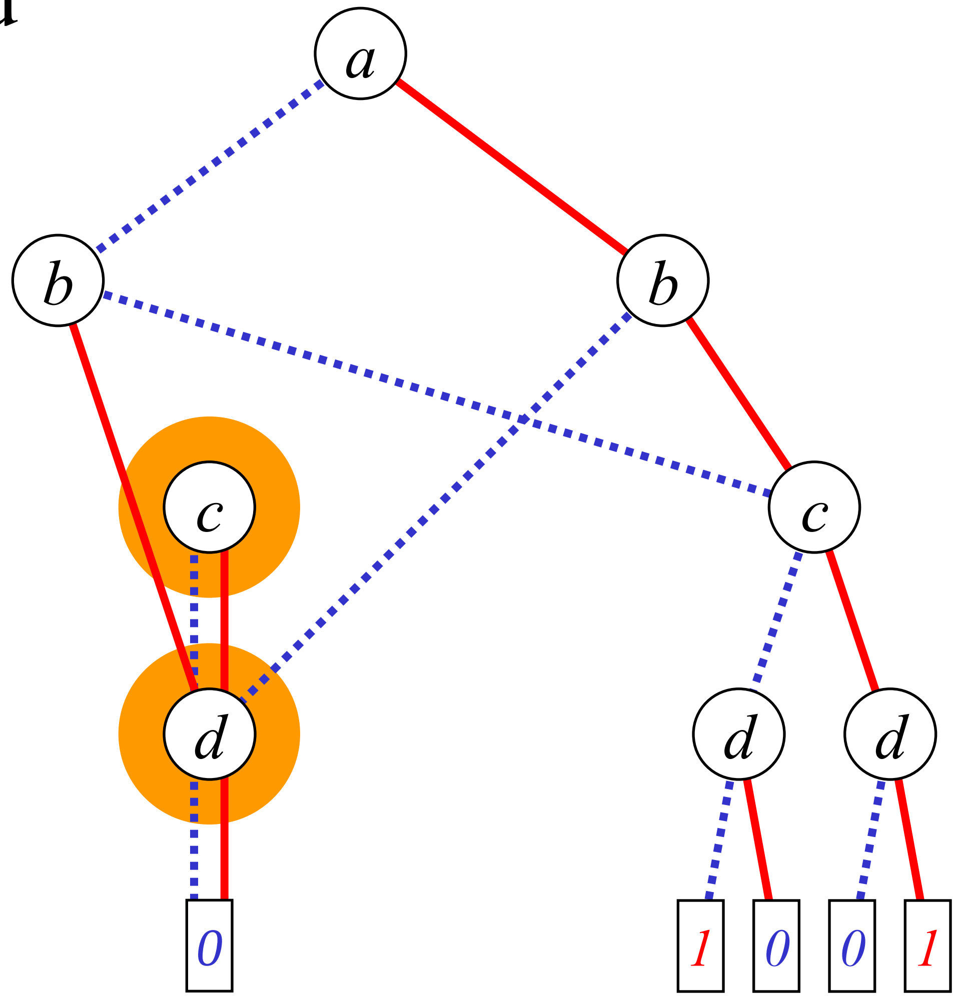
a=b ∧ c=d



Delete redundant nodes

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

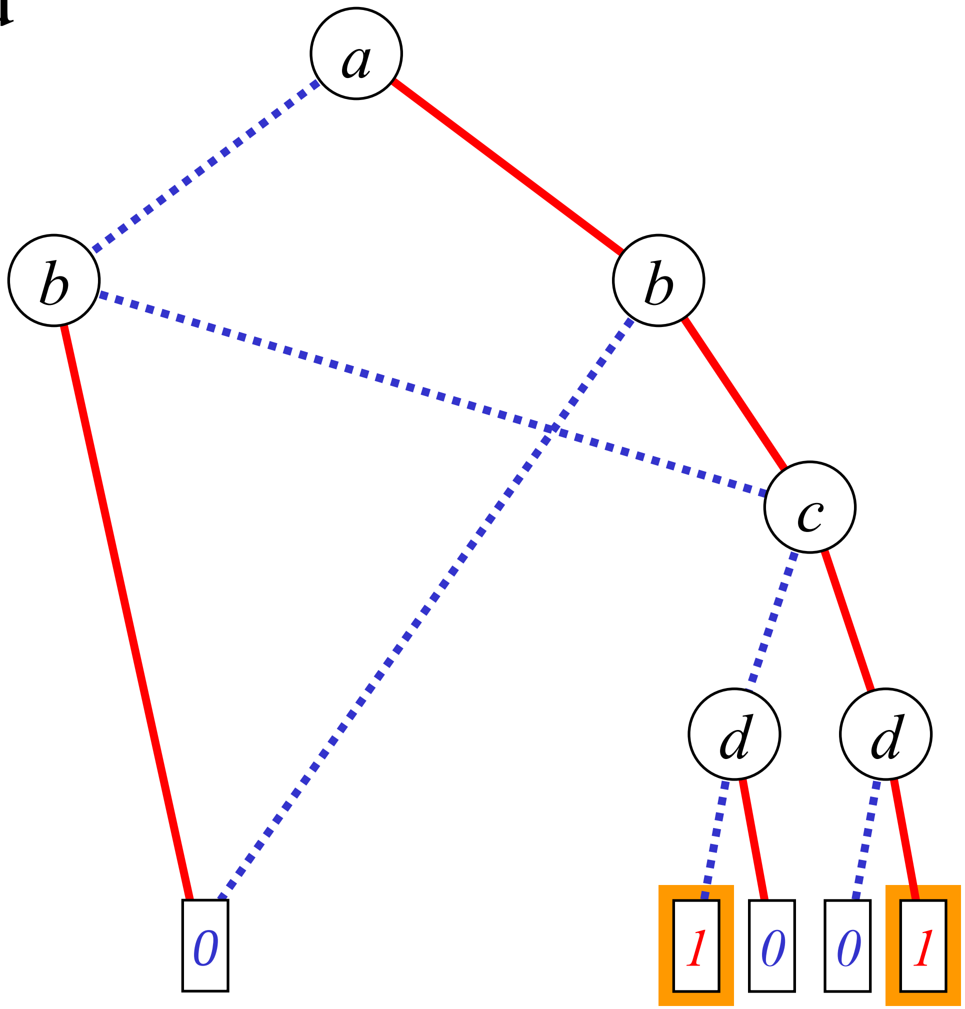
a=b ∧ c=d



Identify same sub-trees

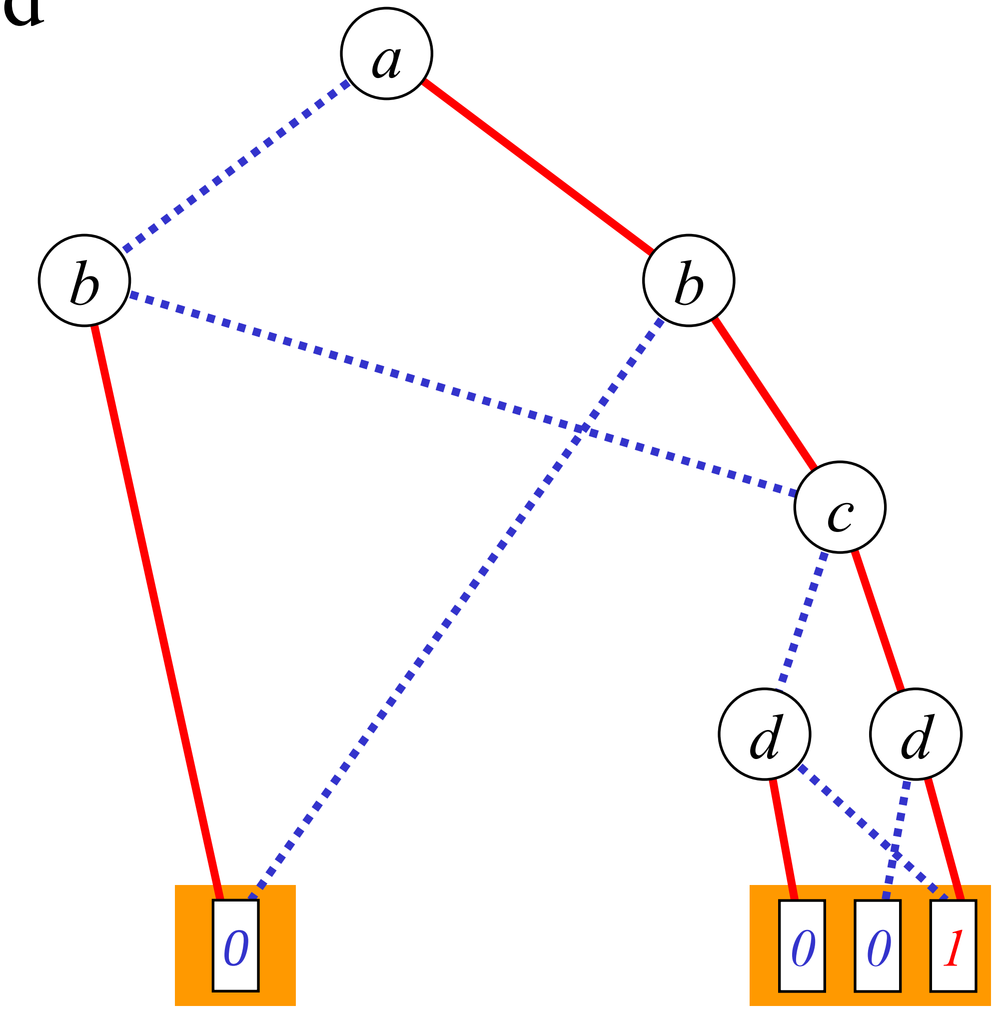
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

$a{=}b \;\wedge\; c{=}d$



DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

$a{=}b \;\wedge\; c{=}d$

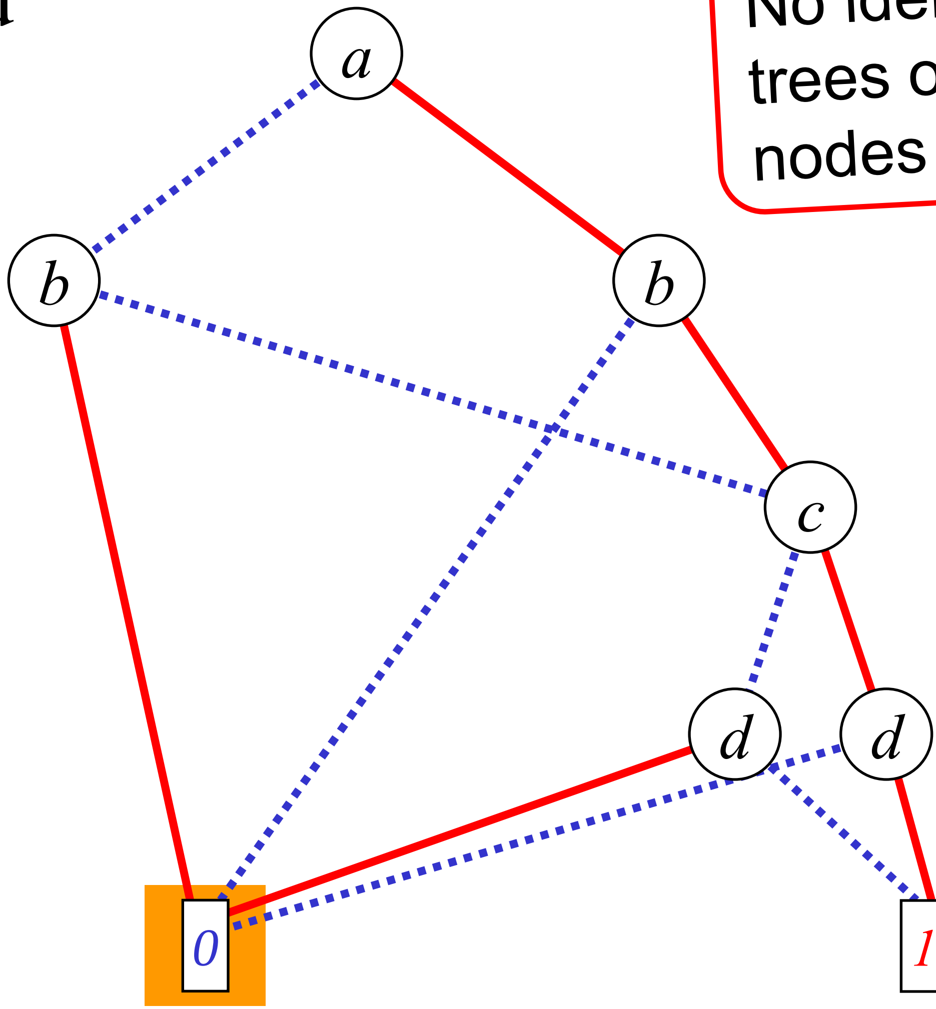


Result

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

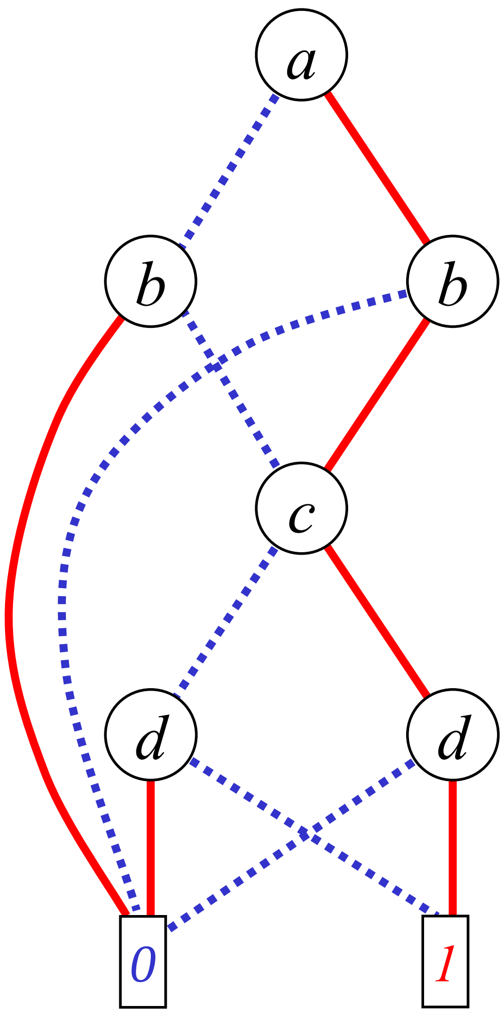
a=b ∧ c=d

No identical sub-trees or redundant nodes anymore!



”Prettified result”: ROBDD

a=b ∧ c=d



ROBDD

- All variables on the paths occur in the same **O**rder (we had that from the start)
- No identical sub-graphs anymore
- No redundant nodes anymore

⇒ **R** educed **O**rdered Binary Decision Diagram

Observations

- For every set (and a fixed variable order) there exists exactly one ROBDD representing it!

- For many practically relevant sets, the ROBBDs representing them are small.

- The size of the ROBDDs depends on the chosen variable order (on the paths):

  For example, the ROBDD for the set characterized by $a{=}b \;\wedge\; c{=}d$ is small with variable order $a < b < c < d$; it is bigger with variable order $a < c < d < b$.

Observations

- There are sets for which the ROBDD will be big for any variable order (multiplication)

- Finding good or even optimal variable orders is one of the challenges of symbolic model checking

- There is no efficient way to find an optimal variable order in general (results from complexity theory)
- But, there are heuristics:
  - Variables that are „somehow related“ should be close to each other
  - Local optimisations by switching two variables

Question

- How do we generate an ROBDD?

- Answer: Start with full tree and reduce it!

Question

- How do we generate an ROBDD?

- Answer: Start with full tree and reduce it!

This is a very bad idea!

Rather, we build them bottom up from formulas with operations on ROBDDs.

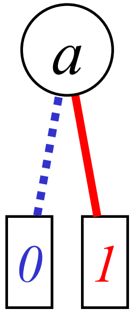
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Boolean variable
- Negation
- Restriction and Shannon expansion
- Binary operations
- ROBDDs and Kripke structures

Boolean variable

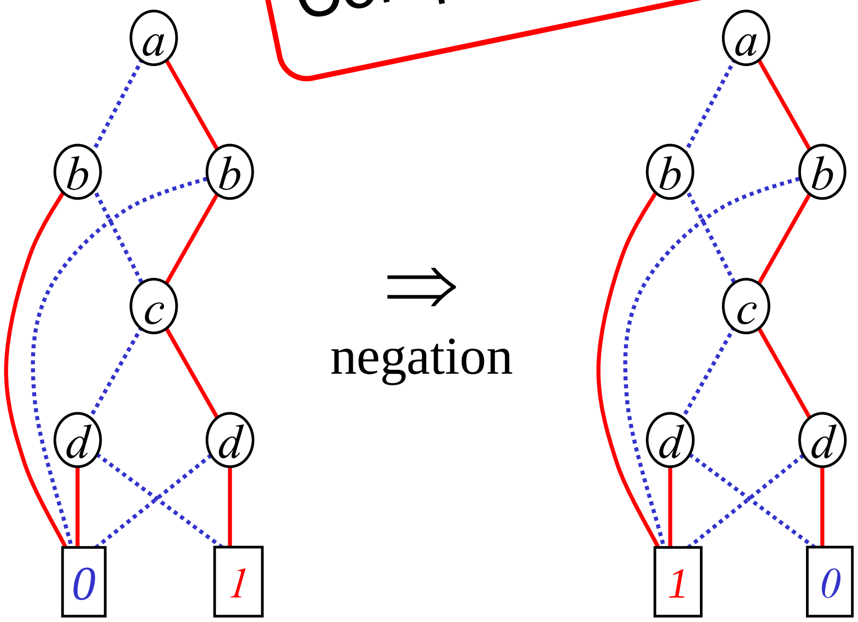
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

The set represented by variable ***a*** is represented by the ROBBD:



Negation

Complexity: $O(1)$

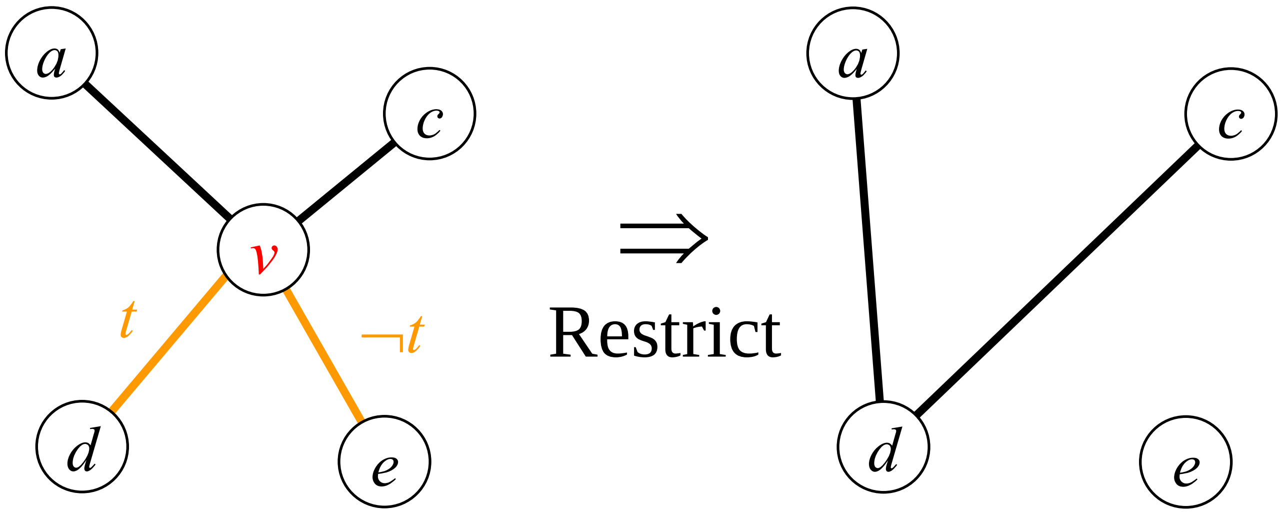


In practice, this is done a bit differently.

Existing ROBDDs are never changed!

Restriktion in ROBDDs

- For a ROBDD representing a Boolean function p, the ROBDD for the $p|_{v \leftarrow t}$ can be obtained as follows:



Complexity:
$O(|p|)$

Size of the ROBDDs for p

- Subsequently: systematic reduction of the resulting ROBDD.

Remember:
Existing ROBDDs are never changed!

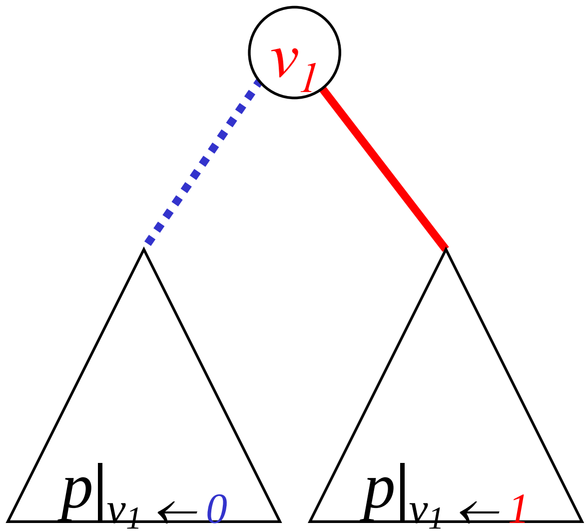
In practice, this is done a bit different.

Complexity:
$O(|p| \cdot log(|p|))$

Restriction: Special case

- An important special case is the restriction to the first variable v_1 of the ROBDD:

$p|_{v_1 \leftarrow 0}$ bzw. $p|_{v_1 \leftarrow 1}$



In practice, this special case is exploited.

Compexity:
$O(1)$

Boolean operators

- The binary Boolean operations can be formulated recursively by the help of the Shannon expansion:

Recursion

- $p \wedge q = \quad (\neg\, v \wedge (p|_{v \leftarrow 0} \wedge q|_{v \leftarrow 0}))\ \vee$
  $\quad (\ \ v \wedge (p|_{v \leftarrow 1} \wedge q|_{v \leftarrow 1}))$
- $p \vee q = \quad (\neg\, v \wedge (p|_{v \leftarrow 0} \vee q|_{v \leftarrow 0}))\ \vee$
  $\quad (\ \ v \wedge (p|_{v \leftarrow 1} \vee q|_{v \leftarrow 1}))$
- $p \clubsuit q = \quad (\neg\, v \wedge (p|_{v \leftarrow 0} \clubsuit q|_{v \leftarrow 0}))\ \vee$
  $\quad (\ \ v \wedge (p|_{v \leftarrow 1} \clubsuit q|_{v \leftarrow 1}))$

works for arbitrary binary operators!

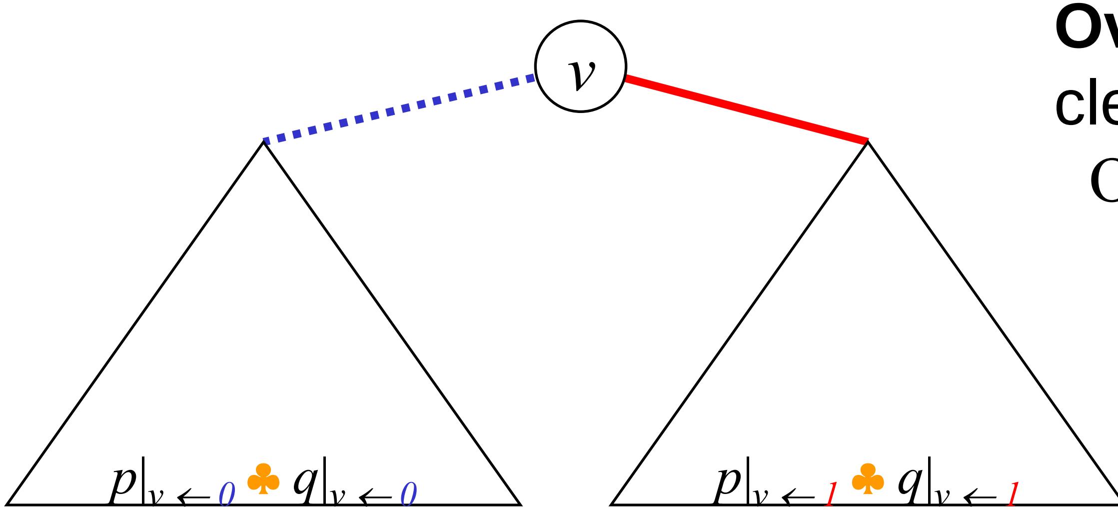
Shannon expansion

Binary Boolean operations

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

ROBDD for $p \clubsuit q$ from ROBDDs for p and q:

- Generate ROBDDs for $p|_{v \leftarrow 0}$, $q|_{v \leftarrow 0}$, $p|_{v \leftarrow 1}$, and $q|_{v \leftarrow 1}$
- Construct recursively $p|_{v \leftarrow 0} \clubsuit q|_{v \leftarrow 0}$ and $p|_{v \leftarrow 1} \clubsuit q|_{v \leftarrow 1}$
- The OBDD for $p \clubsuit q$ is:



Overall complexity (if cleverly implemented):
$O(\,|p| \cdot |q|)$

- Reduce the OBDD systematically to an ROBDD.

ROBDDs: Summary

- As long as all involved ROBDDs remain small, all operations on ROBDDs are efficient

- There are many libraries implementing ROBDDs and the operations on them (often with clever algorithms for optimizing the variable order). MCiE is a very simple implementation.

- In practice, all ROBDDs in the same context are maintained in a single data structure (as a „forest“ of ROBDDs and hash tables for avoiding duplicate nodes). Then, equality of ROBDDs can be decided in constant time (same pointer).

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- For model checking, we need **Boolean formulas** with **quantification** of Boolean variables v (**QBF**):
  $\exists\, v \,.\, p$
- $\exists\, v \,.\, p$ is just an abbreviation for $p|_{v \leftarrow 0} \vee p|_{v \leftarrow 1}$
- $\exists\, \underline{v} \,.\, p$ is an abbreviation for
  $\exists\, v_1 \,.\, (\, \exists\, v_2 \,.\, (\, \ldots\, (\, \exists\, v_n \,.\, p\,)\, \ldots)\,)$

- Respectively, $\forall\, v \,.\, p$ stands for $p|_{v \leftarrow 0} \wedge p|_{v \leftarrow 1}$
- And $\forall\, \underline{v} \,.\, p$ stands for
  $\forall\, v_1 \,.\, (\forall\, v_2 \,.\, (\, \ldots\, (\, \forall\, v_n \,.\, p\,)\, \ldots)\,)$

Relation product

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- For a formula, $p(\underline{u},\underline{v})$ over variables U and V and a formula $q(\underline{v}, \underline{w})$ over variables V and W, we call

$$\exists\, \underline{v}\,.\; p(\underline{u},\underline{v}) \;\wedge\; q(\underline{v}, \underline{w})$$

  the **relation product** of $p(\underline{u},\underline{v})$ and $q(\underline{v}, \underline{w})$.

- The ROBDD for the relation product can be realized with the above abbreviations by the Boolean operations. That, however, is a bit inefficient.

- In practice, the relation product is implemented directly. The worst case complexity is exponential; but, it works reasonably well in many practical setting.

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Represent everything, i.e. initial condition, transition relation as well as the result, as ROBDDs:

Given:

- $\mathcal{S}_0$ and $\mathcal{R}$ as ROBDDs over V resp. $V \cup V'$
- a CTL-Formula p.

Wanted:

- The ROBDD for the set of states S_p
  (the set of states in which p is true).

Algorithms for CTL

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- We assume that we have calculated the ROBDDs for the sets S_p and S_q already

- Next we give the algorithms for calculating the ROBDDs for the sets
  - $S_{p \vee q}$, $S_{p \wedge q}$ and $S_{\neg p}$,
  - $S_{\mathbf{EX}\, p}$,
  - $S_{\mathbf{EG}\, p}$ and
  - $S_{\mathbf{E}[\, p\, \mathbf{U}\, q\,]}$

These are the Boolean operations.

Algorithms on the following slides!

Algorithm for $\mathbf{EX}\ p$

Observation:

- $\mathbf{EX}\ p \;\equiv\; \exists\, \underline{v}'.\ \mathcal{R}(\underline{v}, \underline{v}') \land p(\underline{v}')$

Given ans ROBDD

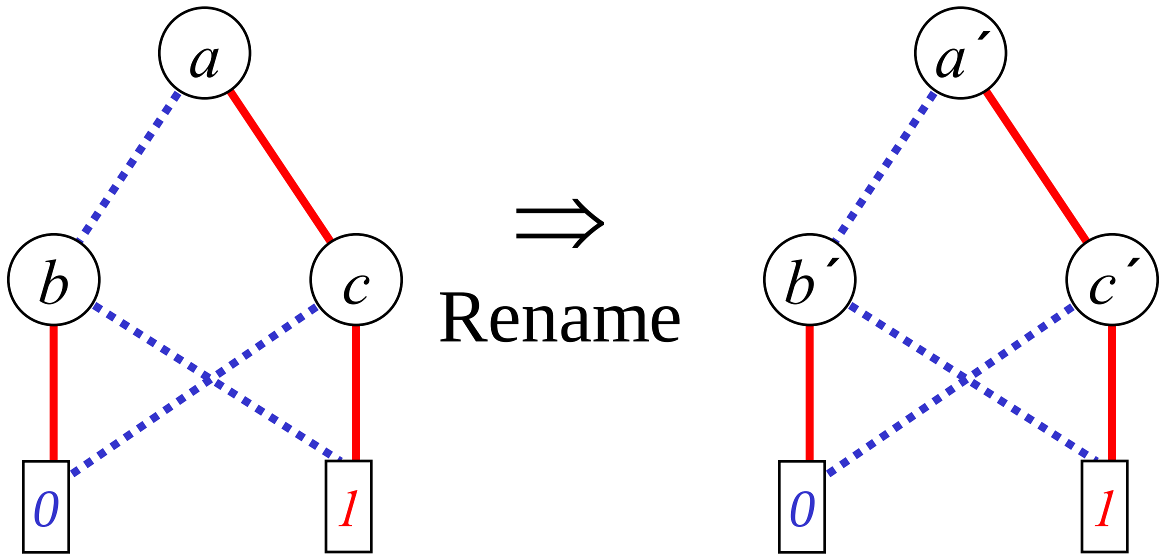
$p(\underline{v})$ given as ROBDD

Relation product on ROBDDs

Algorithm for **EX** p

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

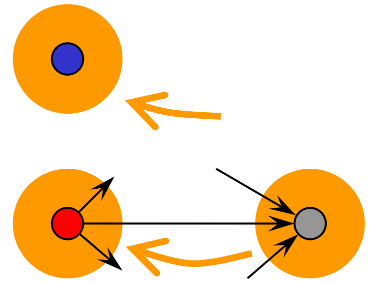
- The only thing left to do is to produce an ROBDD for $p(\underline{v}')$ from an ROBDD for $p(\underline{v})$:

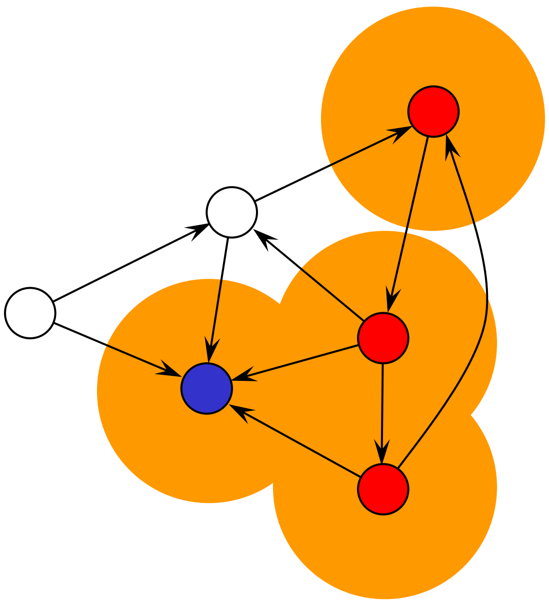


Complexity:
O(|p|)

- In practice, this renaming is done on the fly (and only temporarily) when the relation product is calculated

Reminder: $\mathbf{E}[\, p \; \mathbf{U} \; q \,]$





Given: S_p and S_q

Wanted: $S_{\mathbf{E}[\, p \, \mathbf{U} \, q \,]}$

$$S_0 = S_q$$
$$S_1 = S_q \cup (\, S_p \cap \mathbf{EX}(S_0))$$
$$S_2 = S_q \cup (\, S_p \cap \mathbf{EX}(S_1))$$
$$\ldots$$
$$S_{i+1} = S_q \cup (\, S_p \cap \mathbf{EX}(S_i))$$

until $S_{i+1} = S_i = S_{\mathbf{E}[\, p \, \mathbf{U} \, q \,]}$

Algorithm for $\mathbf{E}[\ p\ \mathrm{U}\ q\ ]$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- In this algorithm, the following operations on sets (ROBDDs) occur:
  - test for equality
  - union
  - intersection
  - $\mathbf{EX}(S)$
- For all these operations, we have algorithms already (more or less efficient)
- If the iteration does not change anything (check for equality), this is the ROBDD for $S_{\mathbf{E}[\ p\ \mathrm{U}\ q\ ]}$.

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

`S:=` S_p `; // represented as ROBDD`

`repeat`

`S´ := S;`

`S:=` $S_q \vee (S_p \wedge$ `checkEX(S));`

`until S = S´;`

`return S;`

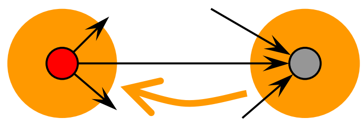
ROBDD operations

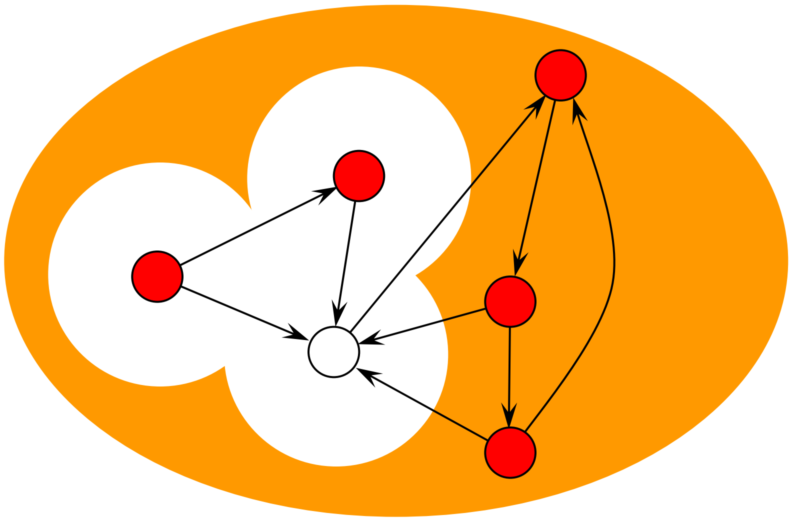
procedure for $\mathbf{EX}(S)$

Check for equality!
(answers question on
→ slide 45)

Reminder $\mathbf{EG}\ p$

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler





Given: S_p

Wanted: $S_{\mathbf{EG}\,p}$

$$S_0 = S_p$$
$$S_1 = S_p \cap \mathbf{EX}(S_0)$$
$$S_2 = S_p \cap \mathbf{EX}(S_1)$$
$$\ldots$$
$$S_{i+1} = S_p \cap \mathbf{EX}(S_i)$$

until $S_{i+1} = S_i = S_{\mathbf{EG}\,p}$

This is the inefficient algorithm from the introduction.

With the help of ROBDDs it becomes reasonably efficient.

Procedure `checkEX`(S_p)

```
S:= Sp; // represented as ROBDD

repeat
   S´:= S;
   S:= Sp ∧ checkEX(S);
until S = S´;

return S;
```

ROBDD operation

procedure for **EX**(S)

Check for equality

Symbolic model checking

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- The use of ROBDDs for the representation of sets of states is called **symbolic model checking** (as in contrast to explicit model checking).
- Symbolic model checking contributed to the initial success of model checking (SMV and today NuSMV)!
- Though it uses more inefficient algorithms as one would use with explicit sets, symbolic model checking is sometimes more efficient (but that depends!).
- It does not work always (for bigger examples).
- There are many other techniques for model checking!
- To date, applying model checking for realistic systems requires much experience.

The following slides are covering the mathematical formalisation and some additional details;

The are not shown in the lecture, but are included

For completness sake.

5.3. System model (details)

- Kripke Structures
- Syntactic Representation
- Examples

Rather, we build them bottom up from formulas with operations on ROBDDs.

Kripke Structures

- Motivation
- Definition
- Computation paths
- Transition systems

Motivation

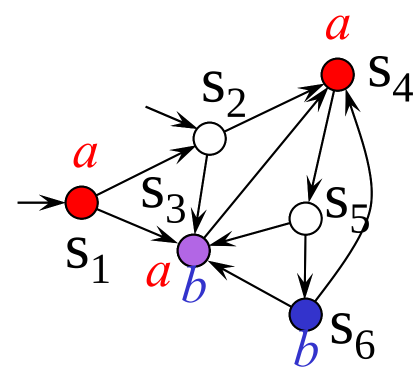
There are many different notations for reactive systems; the choice depends on the application area and the purpose of the model.

Most model checking techniques are independent from the particular notation. Therefore, we do not fix a notation.

Rather we define **Kripke structures** as a common underlying **semantic model**.

Kripke Structures

A **Kripke structure** M consists of



- a finite **set of states**: S,
- a set of **initial states**: $S_0 \subseteq S$,
- a total **transition relation**: $R \subseteq S \times S$
- a **labelling** of the states with a set of **atomic propositions** AP: $L: S \to 2^{AP}$

Set of all subsets of AP.

Kripke Structures

We call $M = (S, S_0, R, L)$ a **Kripke structure** over the atomic propositions AP.

We say that

- proposition $a \in AP$ **is valid in a state** $s \in S$, if $a \in L(s)$, i.e. if a is one of the labels of s.
- state $s' \in S$ is **successor state** of state $s \in S$, if $(s, s') \in R$.

Remarks:

- For technical reasons, we require that the transition relation R is total; i.e. for each state $s \in S$ there exists a successor state.

- In principle, we could avoid this restriction.

Paths

For a Kripke structure $M = (S, S_0, R, L)$ we call an infinite sequence over S

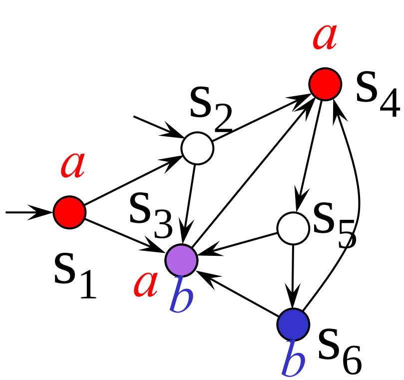
$$\pi = s_0 \, s_1 \, s_2 \, s_3 \ldots$$

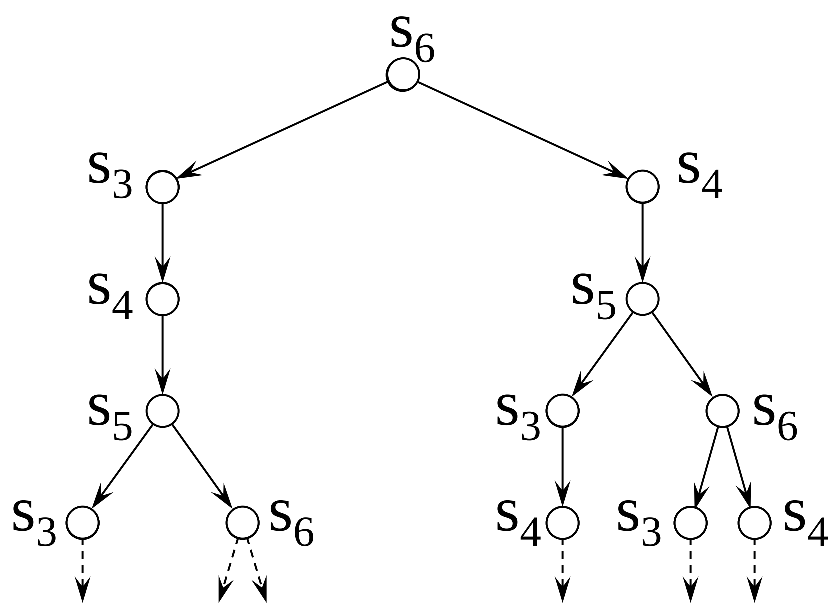
a **path** of M in s_0, if for each $i \in \mathbf{N}$ state is a successor of s_i; i.e. if $(s_i \, , s_{i+1}) \in R$

A path starting in an initial state of M is called a **run** of M.

The set of all paths of M in a state s can be represented as an infinite tree, the **computation tree** of M in s :

Example:



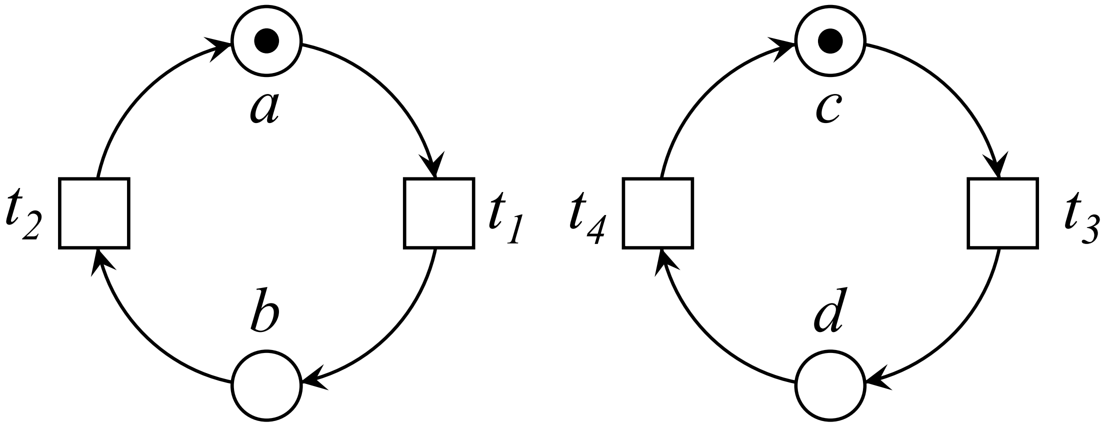


Since the transition relation R is total, all paths (branches) of the tree are infinite!

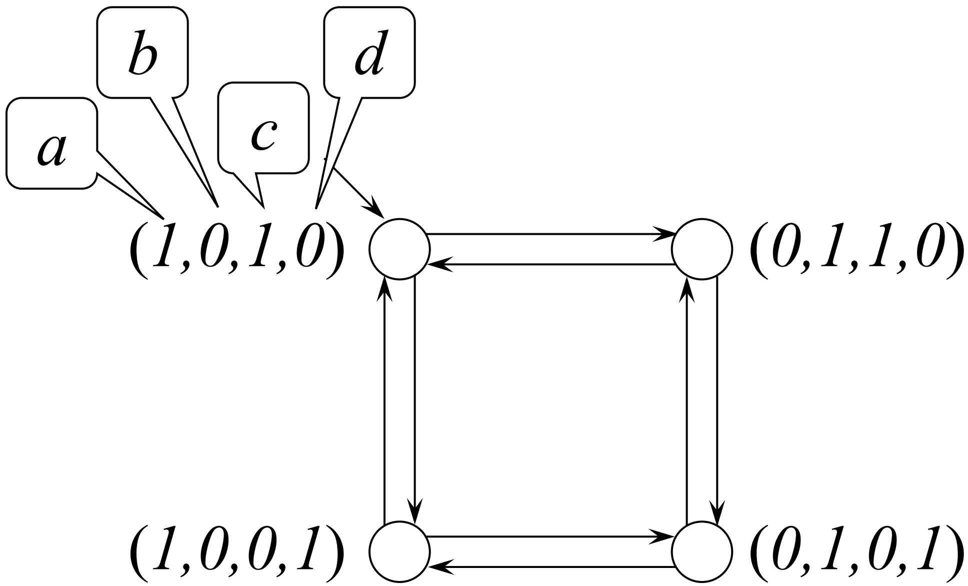
Systems and Kripke Structures

- A system resp. a model of a system in another notation can be easily mapped to a Kripke structure (provided that the model is finite).
- Sometimes some information of the model will be lost.

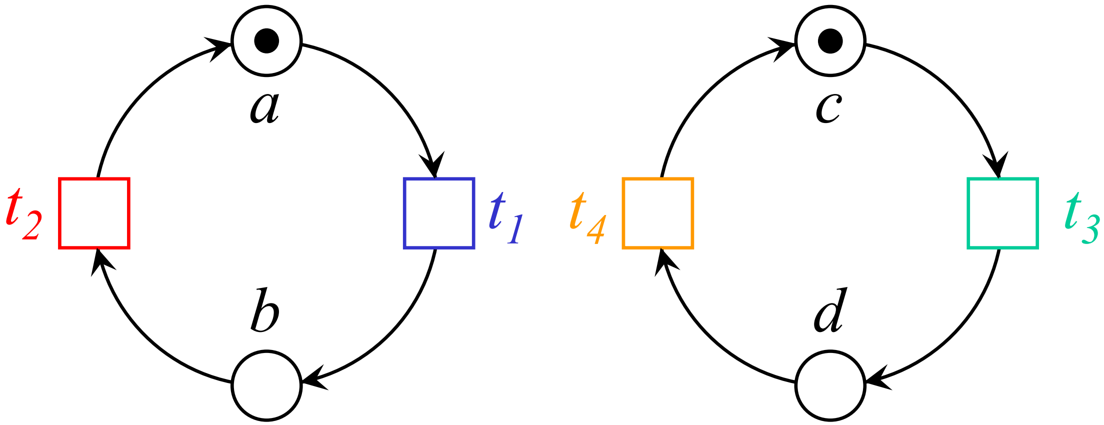
→ Example on next slide



A Petri net

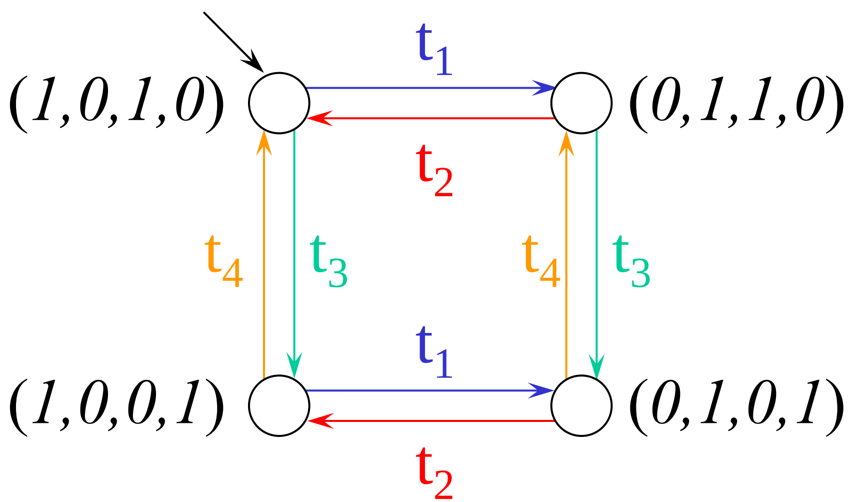


The corresponding Kripke structure



A Petri net

The information on related transitions is lost in the Kripke structure!



Extensions

- Labelling of transitions: **Transition systems**

- Instead of a single transition relation, there are many transition relations (in our example for every Petri net transition).

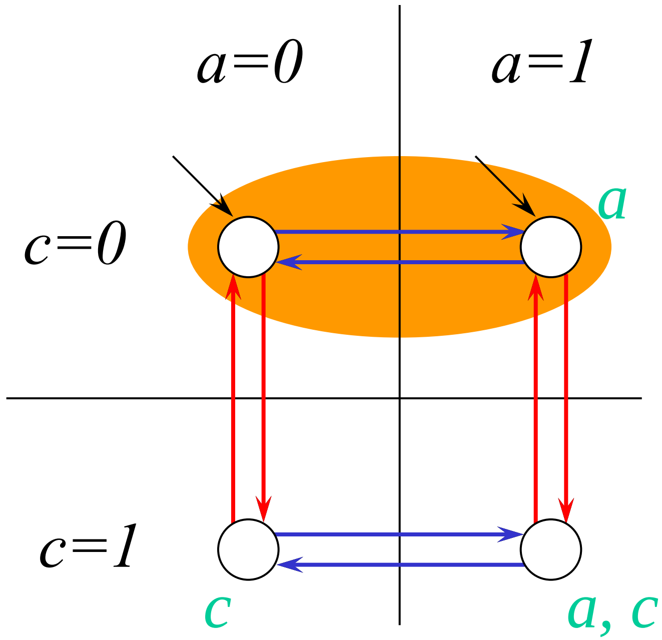
This is also important for efficiency reasons!

- Motivation & Example
- States
- Initial states
- Transitions
- Labels

Motivation

- Kripke structures are a semantic model for reactive systems (a mathematical structure).
- For real (and large) systems, an explicit enumeration of all states and all transitions is tedious (→ state space explosion).
- Therefore, we use a notation from logic, for representing Kripke structures and transition systems in a more compact way.

Example



0 = *false*

1 = *true*

- Boolean variables:
  $V = \{ a, c \}$
- Initial formula:
  $S_0 \equiv \neg c$
- Transition formula:
  $\mathcal{R} \equiv$
  $(a' = \neg a \wedge c' = c) \vee$
  $(a' = a \wedge c' = \neg c)$
- Implicit labelling:
  $AP = V$

States

- Let $V = \{ v_1, \ldots, v_n \}$ be a set of Boolean **variables**.
- We call a mapping $\sigma\colon V \to \mathbf{B}$ an assignment for variables V.

  $\mathbf{B} = \{ 0, 1 \}$ denotes the set of Booleans or truth values (with $0 = \mathit{false}$ and $1 = \mathit{true}$).

- Each assignment can be considered as a state.
- This way, the set V implicitly defines a set of states $S = \{ \sigma \mid \sigma : V \to \mathbf{B} \}$.

Initial States

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- The (propositional) **formulas** over variables V are defined as usual.

- Likewise, the **validity** of a formula p under some assignment σ is defined as usual;
  we write $\sigma \models p$, if p is valid at σ.

- A formula S_0 over V, the **initial formula**, defines the set of initial states:
  $S_0 = \{ \sigma \mid \sigma \models S_0 \}$.

- For a set of variables we define the set of primed variables.

$V = \{ v_1 , \ldots , v_n \}$,
$V' = \{ v'_1, \ldots , v'_n \}$

Idea:

- Assignment for V : source state of the transition
- Assignment for V' : target state of the transition

Transition relation

- An assignment for variables $V \cup V'$ can be represented as a pair of assignments (σ, σ') for V:
  - $\sigma(v)$ defines the value for v
  - $\sigma'(v)$ defines the value for v'

- The validity of formula p over $V \cup V'$ for a pair of assignments (σ, σ') can be defined as usual : We write $(\sigma, \sigma') \models p$, if p is valid for (σ, σ')

Transition relation

- A formula $\mathcal{R}$ over $V \cup V'$, the **transition formula**, defines the transition relation of a Kripke structure in the following way:

$$R = \{ (\sigma, \sigma') \mid (\sigma, \sigma') \models \mathcal{R} \}$$

Labelling

- The labelling of the states (assignment) can be directly derived from the assignment:

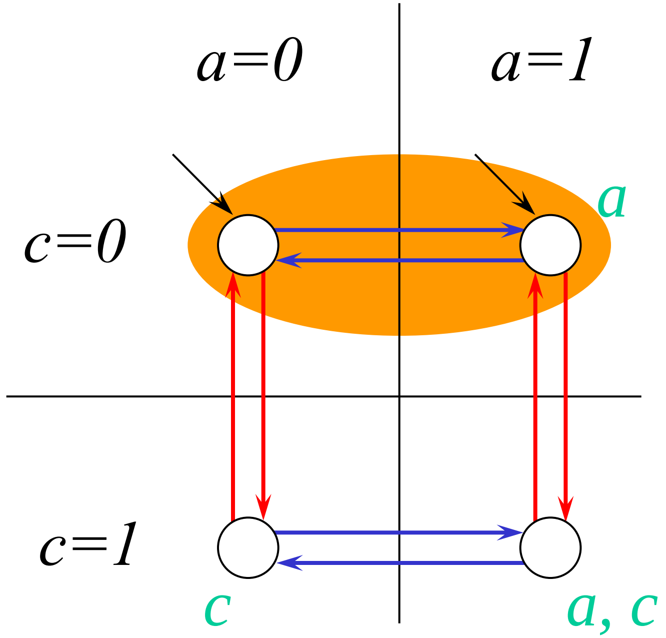
$$AP = V$$

$$L(\sigma) = \{ v \in V \mid \sigma(v) = 1 \} = \{ v \in V \mid \sigma \models v \}$$

i.e. each state (assignment) is labelled with those variables that are true in this assignment

Summary

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



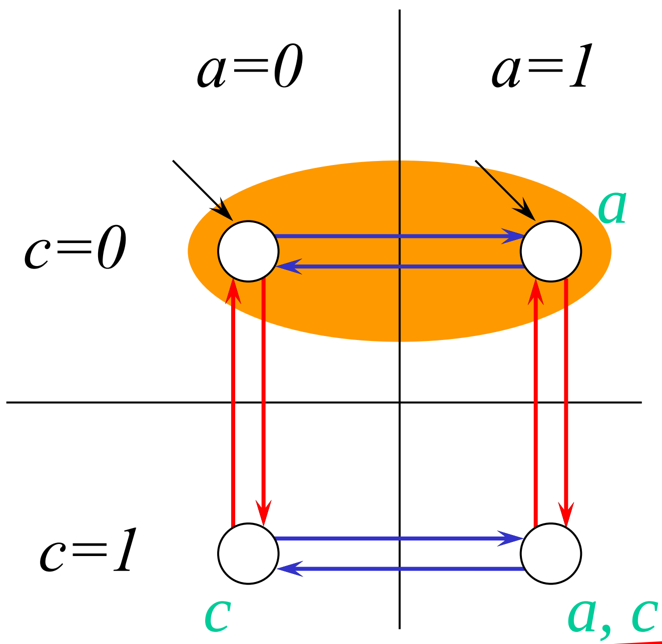
$S = \{\ (0,0), (0,1), (1,0), (1,1)\ \}$

$S_0 = \{\ (0,0)\ ,\ (1,0)\ \}$

$R = \{\ ((0,0),(1,0))\ ,\ ((1,0),(0,0))\ ,$
$((0,1),(1,1))\ ,\ ((1,1),(0,1))\ ,$
$((0,0),(0,1))\ ,\ ((0,1),(0,0))\ ,$
$((1,0),(1,1))\ ,\ ((1,1),(1,0))\ \}$

- Boolean variables:
  $V = \{\ a, c\ \}$
- Initial formula:
  $S_0 \equiv \neg\, c$
- Transition formula:
  $\mathcal{R} \equiv$
  $(a' = \neg a \wedge c' = c) \vee$
  $(a' = a \wedge c' = \neg c)$
- Implicit labelling:
  $AP = V$

As a Transition System



- Boolean variables:
  $V = \{ a, c \}$
- Initial formula:
  $S_0 \equiv \neg c$
- Transition formula:
  $\mathcal{T} \equiv$
  $\{ (a' = \neg a \wedge c' = c),$
  $(a' = a \wedge c' = \neg c) \}$

  Implicit labelling:
  $AP = V$

This equality is often implicit for variables that do not occur primed.

For example in MCiE (important for efficiency).

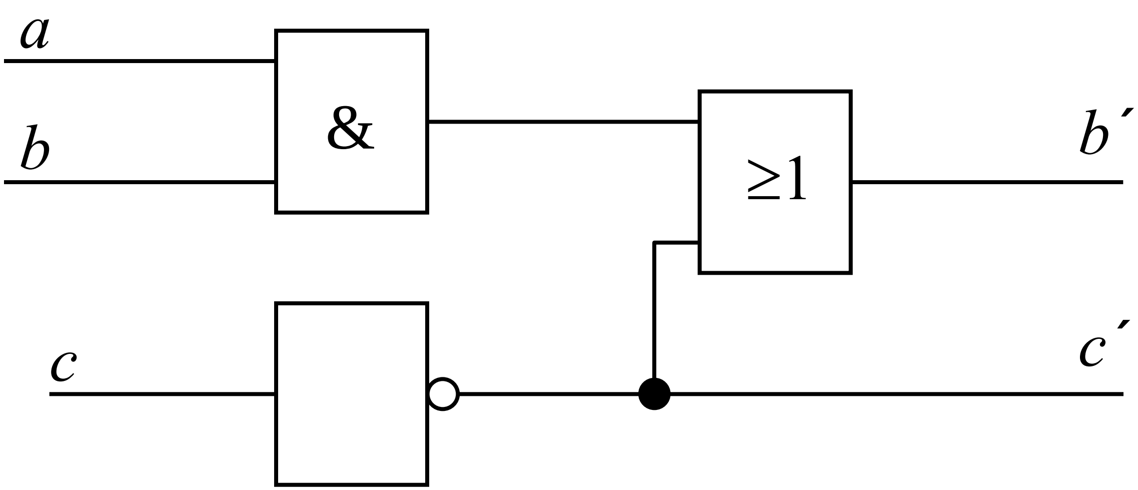
More Examples

In this section, we show by the help of two examples how to represent different kinds of systems as Kripke structures represented by formulas.

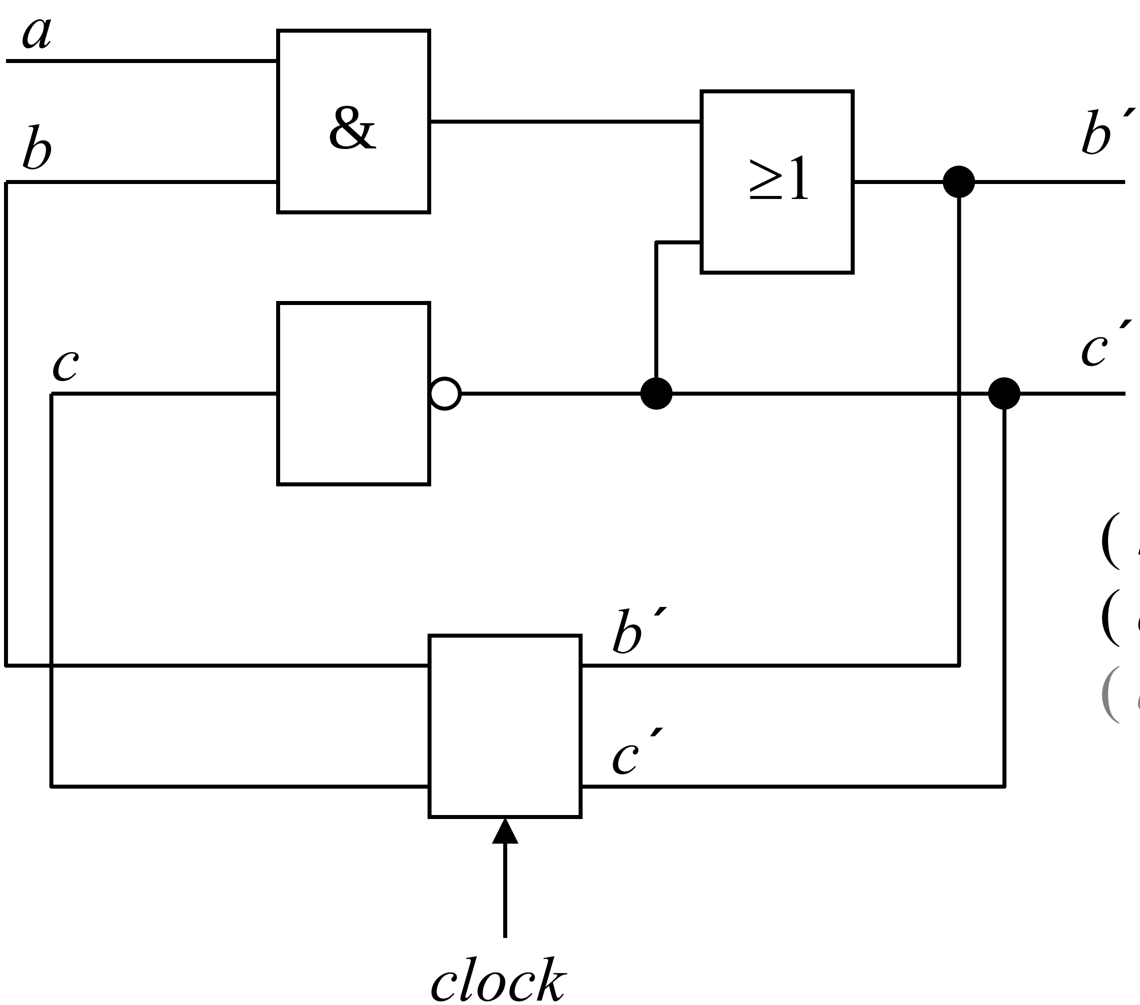
- Synchronous circuit (hardware)
- Concurrent processes
- Petri nets

Combinatorial Circuit

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



$$b' = (a \wedge b \vee \neg c)$$
$$c' = \neg c$$



$$(b' = (a \wedge b \vee \neg c)) \wedge$$
$$(c' = \neg c) \wedge$$
$$(a' = 0 \vee a' = 1)$$

Concurrent Processes

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
         loop forever        ||         loop forever
pca = 0    x:= 0;            ||  pcb = 0   x:= 1;
pca = 1    y:= 0;            ||  pcb = 1   y:= 1;
```

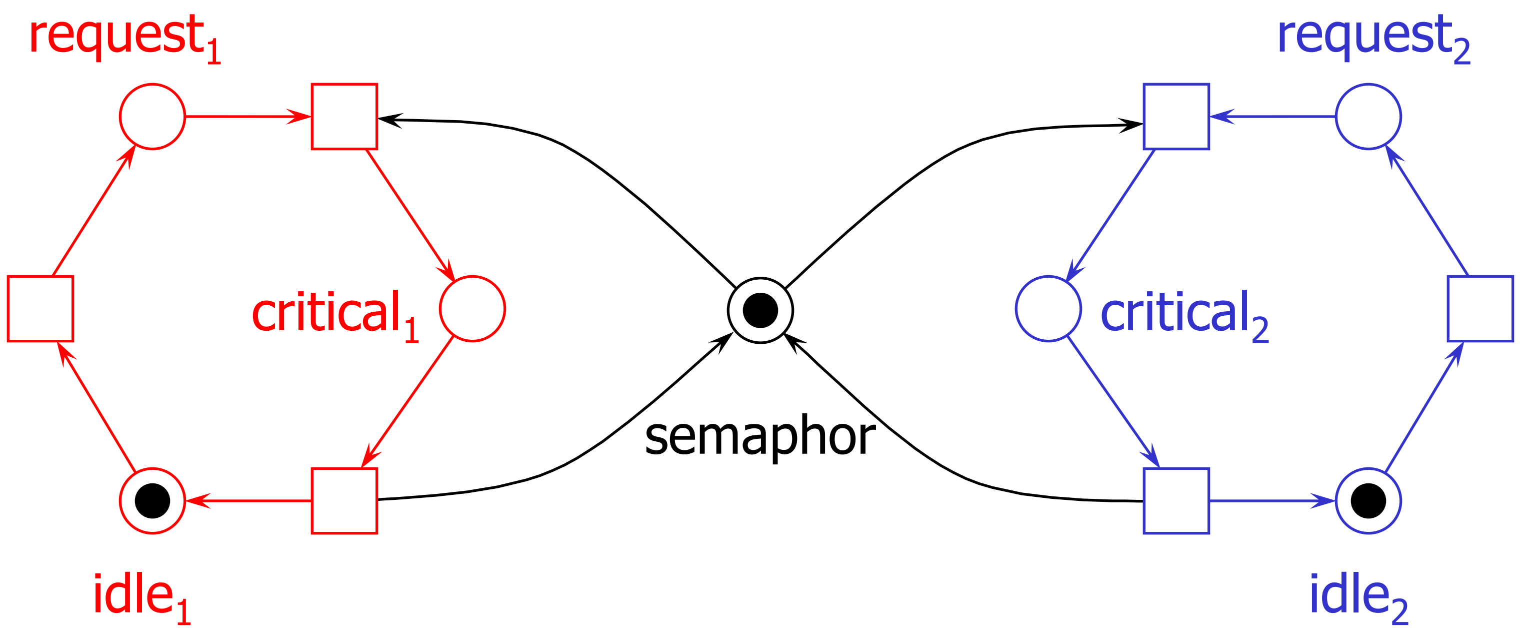
$$(pca = 0 \land pca' = 1 \land x' = 0 \land y' = y \land pcb' = pcb) \lor$$

$$(pca = 1 \land pca' = 0 \land y' = 0 \land x' = x \land pcb' = pcb) \lor$$

$$(pcb = 0 \land pcb' = 1 \land x' = 1 \land y' = y \land pca' = pca) \lor$$

$$(pcb = 1 \land pcb' = 0 \land y' = 1 \land x' = x \land pca' = pca)$$

Petri nets



How do the formulas look for Petri nets as a transition system?

5.4 ROBDDs (details)

Reduced Ordered Binary Decision Diagrams; for simplicity often just called Binary Decision Diagrams (BDDs).

- Motivation
- Definition
- Operations on ROBDDs
- Quantified Boolean formulas (QBF)

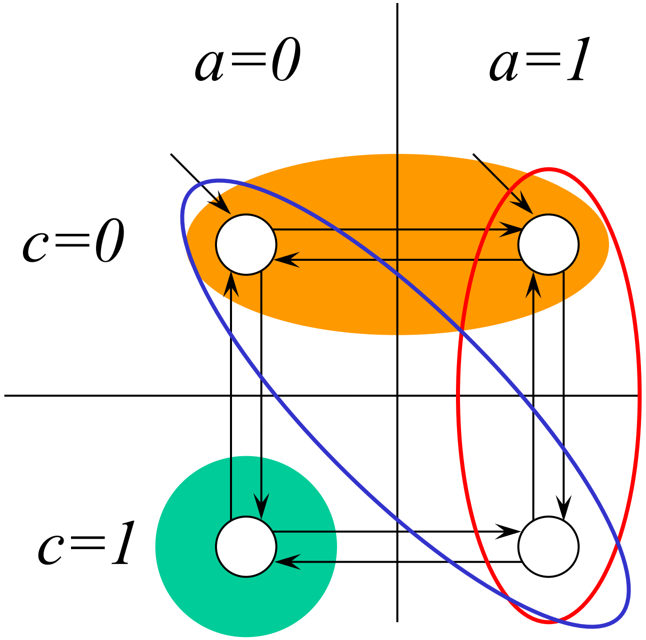
Motivation

- The number of states of realistic systems is gigantic.

$\Rightarrow$ Representing sets of states by enumerating every state explicitly is a bad idea.

- Sets could be represented “symbolically”, e.g. by formulas (see next slide)

Sets as formulas



$\neg c$

a

$a \Leftrightarrow c$

$\neg a \wedge c$

Boolean formulas representing sets of states

Formulas

- Some operations on sets can be efficiently executed for sets that are represented as formulas:
  - union: $p \vee q$
  - disjunction: $p \wedge q$
  - complement: $\neg p$
  - set difference: $p \wedge \neg q$

Problem:

- the same set can have different representations
- it is extremely inefficient to find out whether two formulas represent the same set (NP-complete).
- therefore, formulas are not a good representation for sets of states.

Checking for equality of sets is a very crucial operation in model checking! (BTW: why?) → slide 19/77

Goal

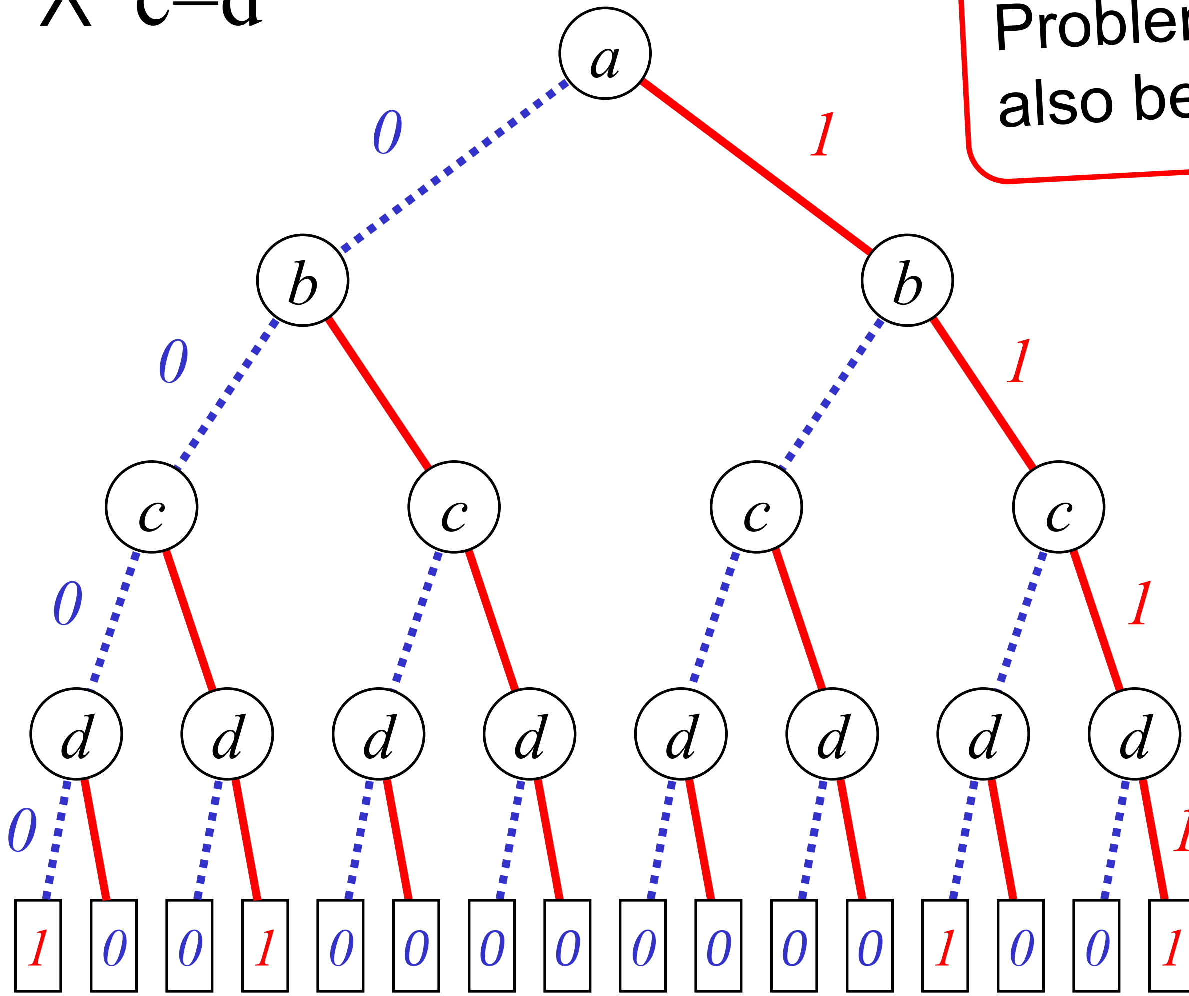
- Representation of sets such that
  - set operations **and**
  - check for equality

  can be computed efficiently

The answer will be Reduced Ordered Binary Decision Diagrams (ROBDDs)!

Binary Decision Trees

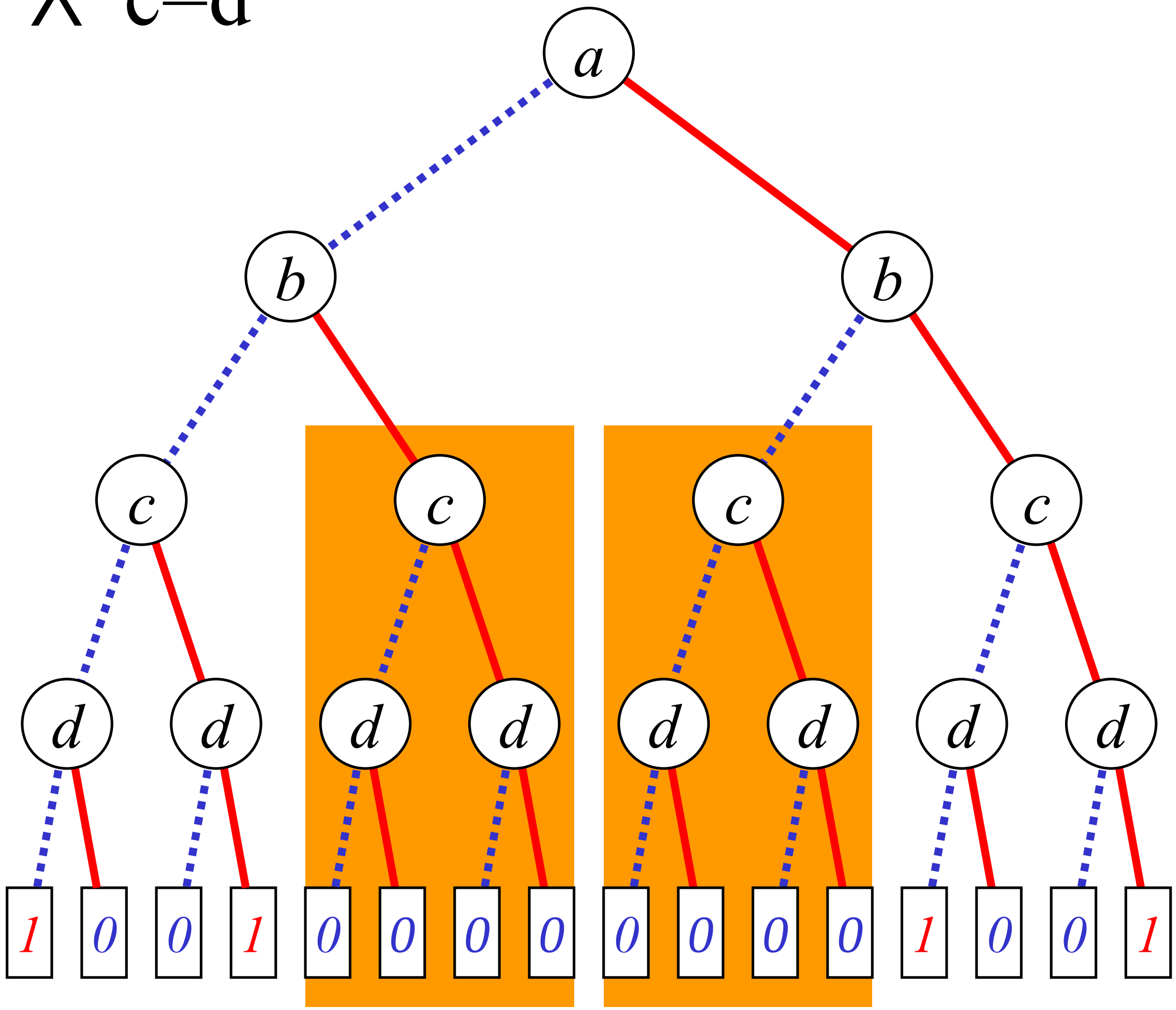
$a{=}b \;\wedge\; c{=}d$



Problem: These will also be very big!

Identify same sub-trees

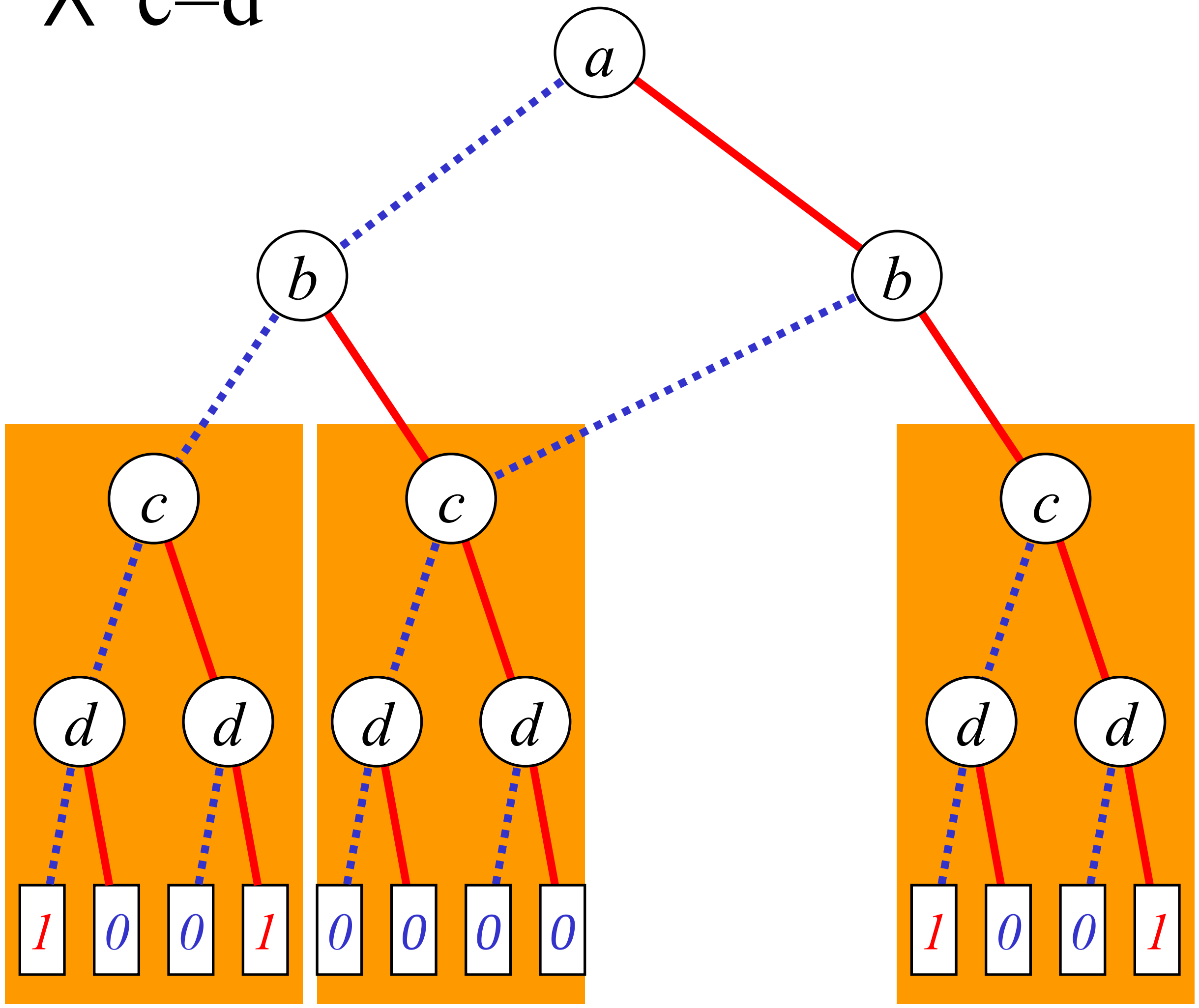
$a{=}b \;\wedge\; c{=}d$



Identify same sub-trees

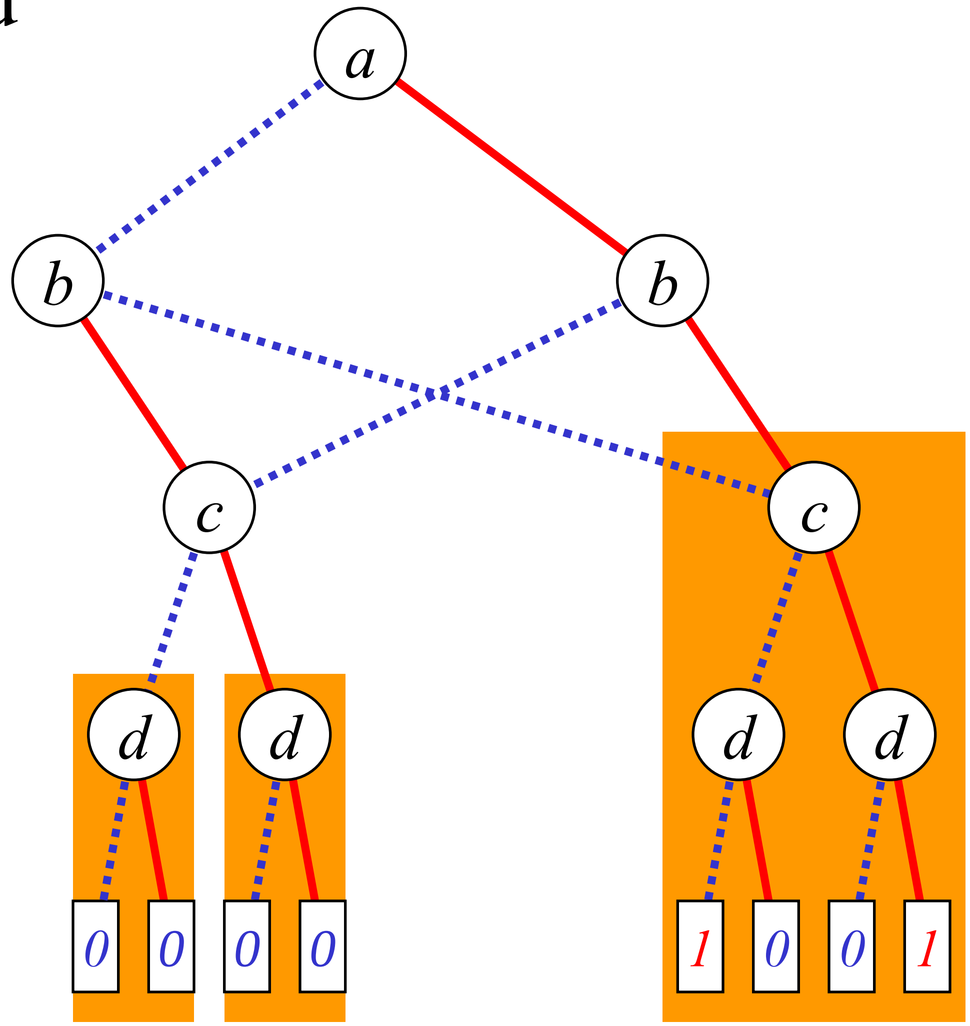
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

$a = b \land c = d$



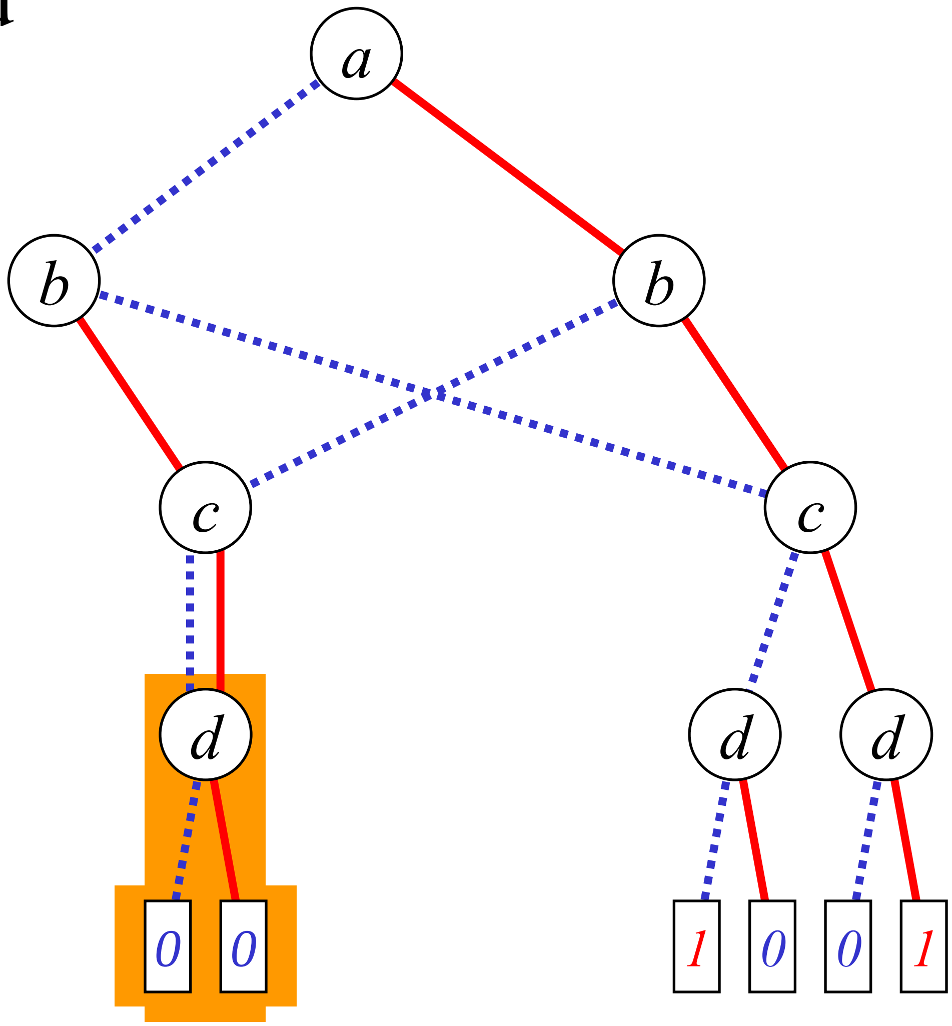
Identify same sub-trees

a=b ∧ c=d



Identify same sub-trees

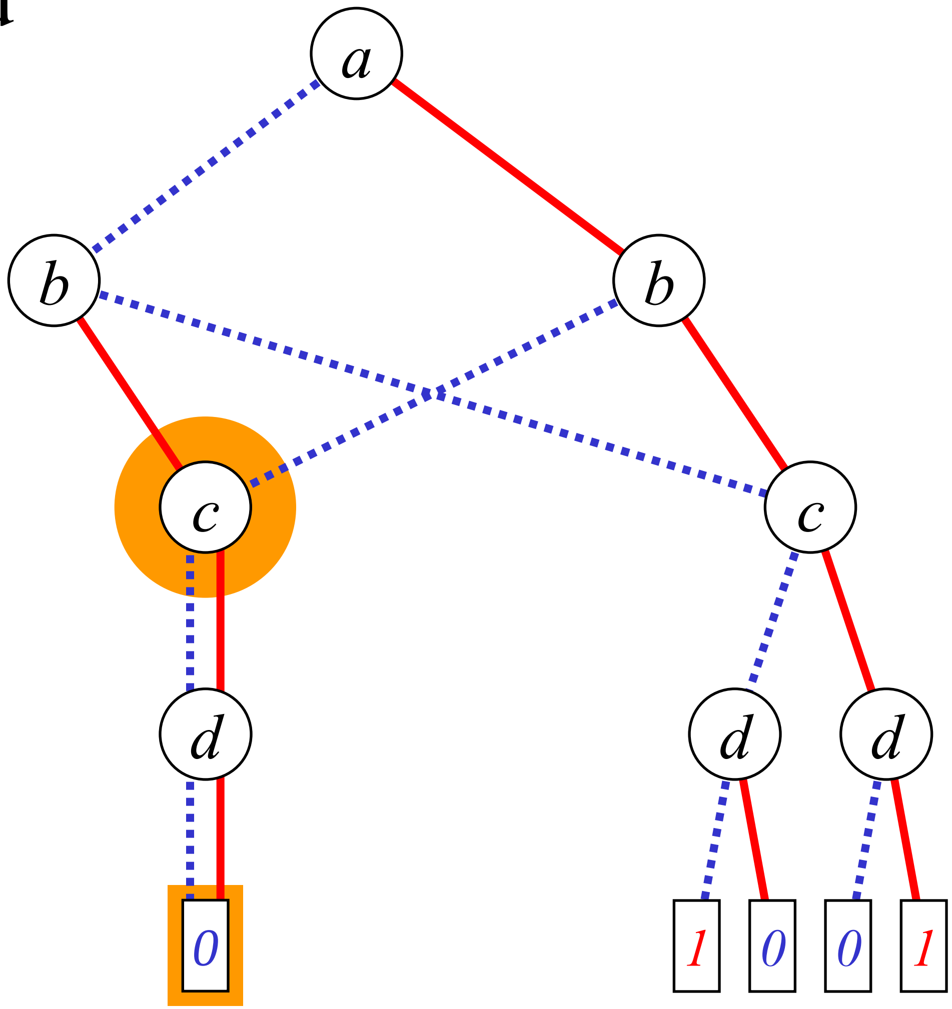
$a=b \ \wedge \ c=d$



Delete redundant nodes

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

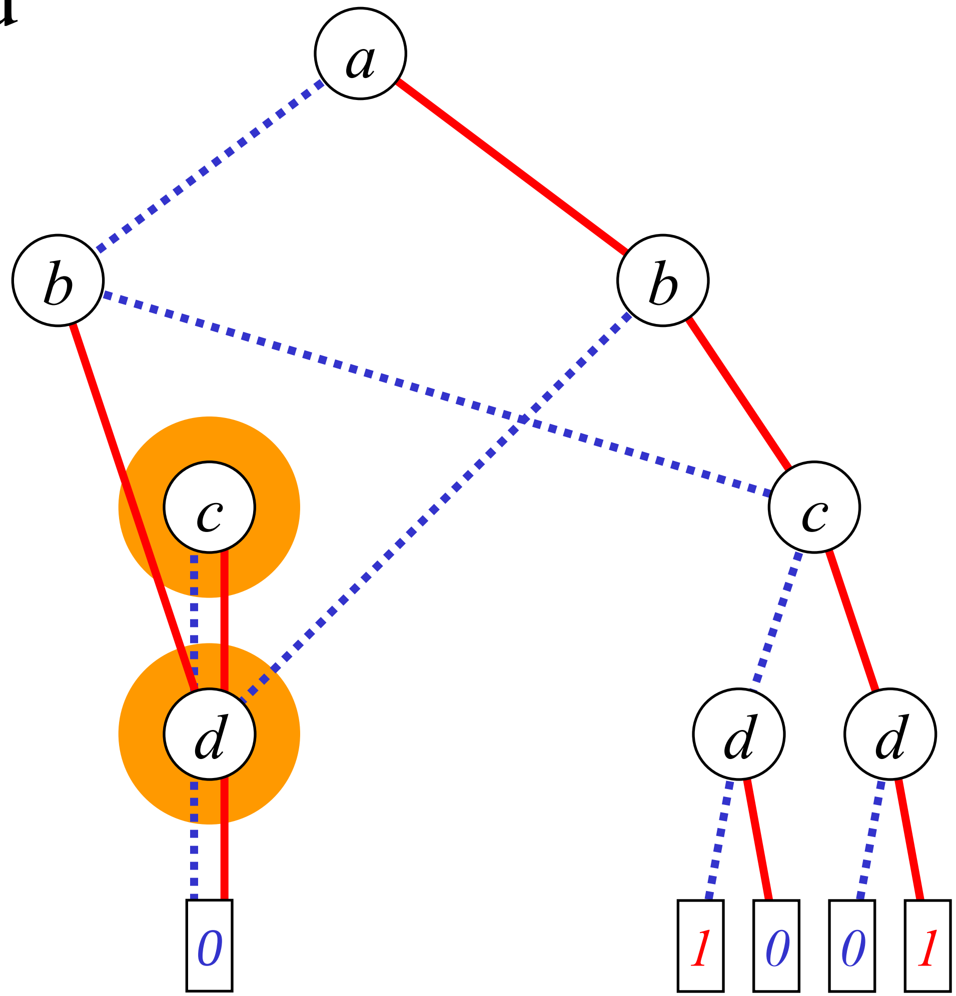
$a{=}b \wedge c{=}d$



Delete redundant nodes

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

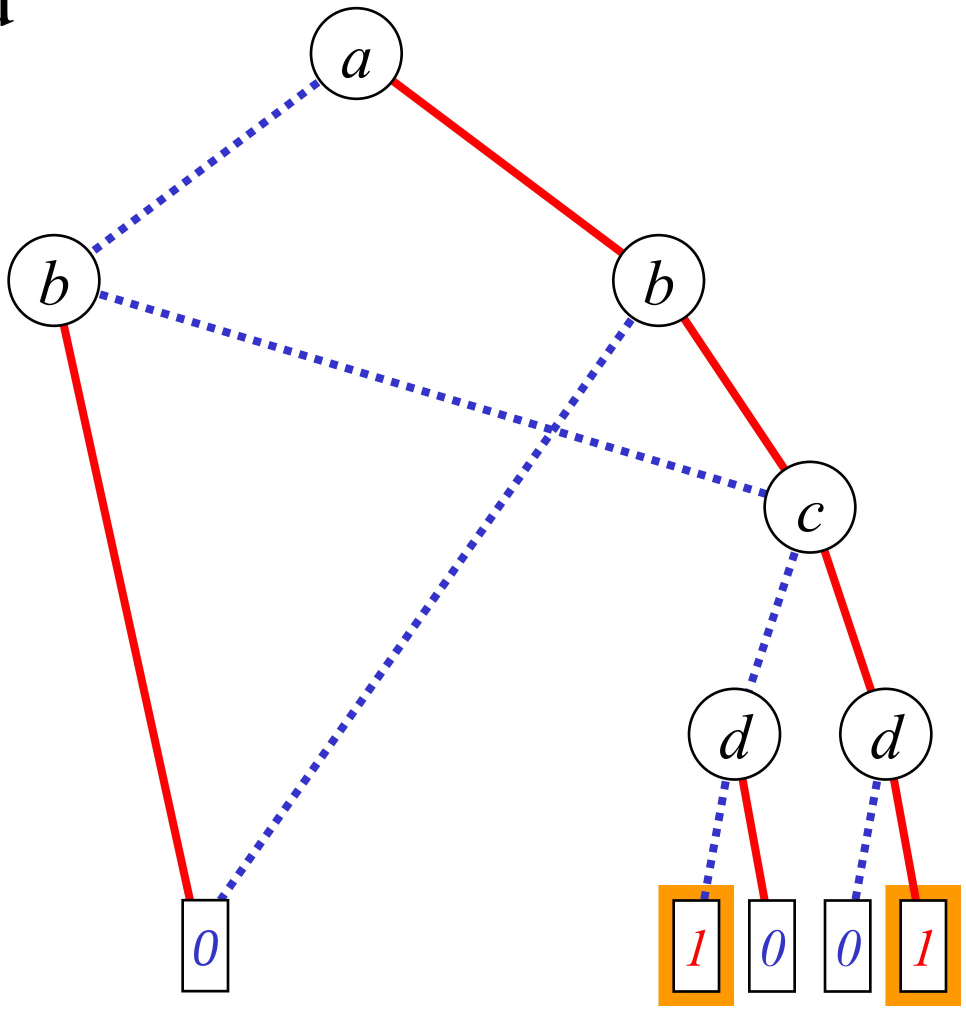
a=b ∧ c=d



Identify same sub-trees

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

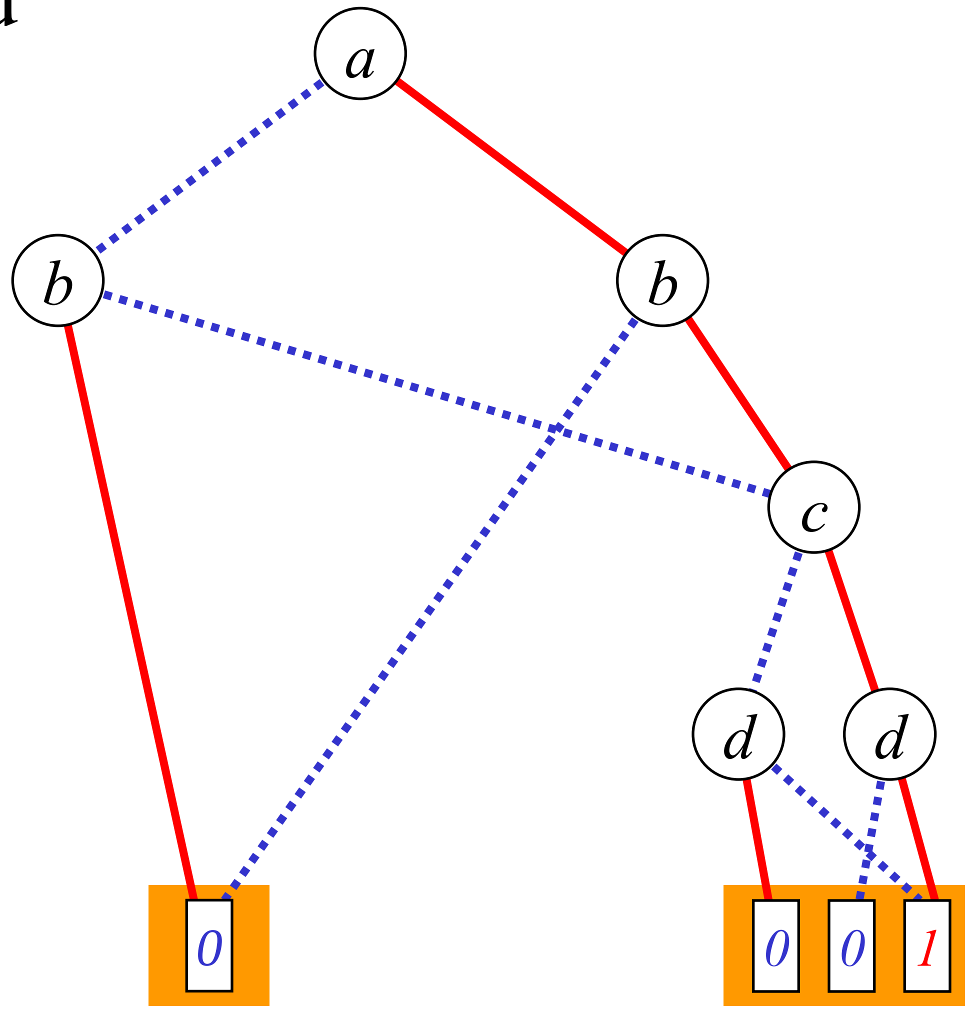
a=b ∧ c=d



Identify same sub-trees

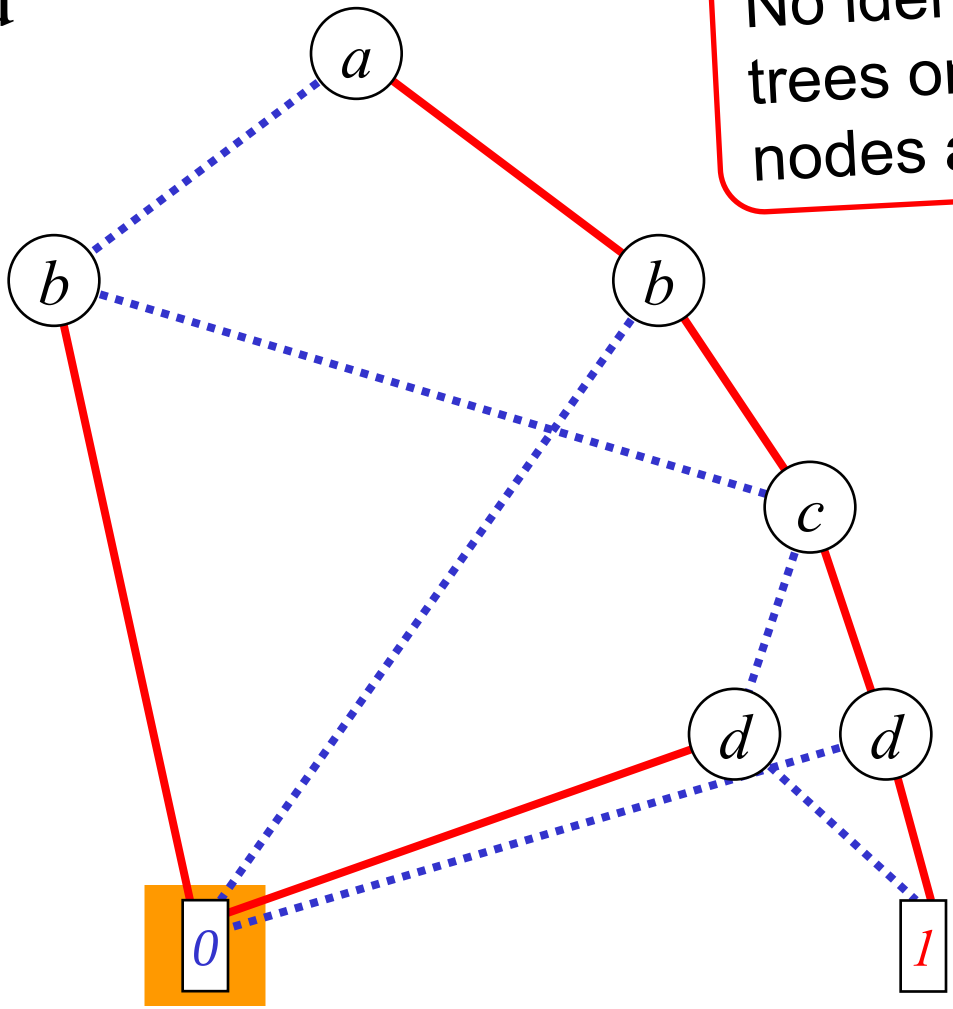
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

a=b $\wedge$ c=d



Result

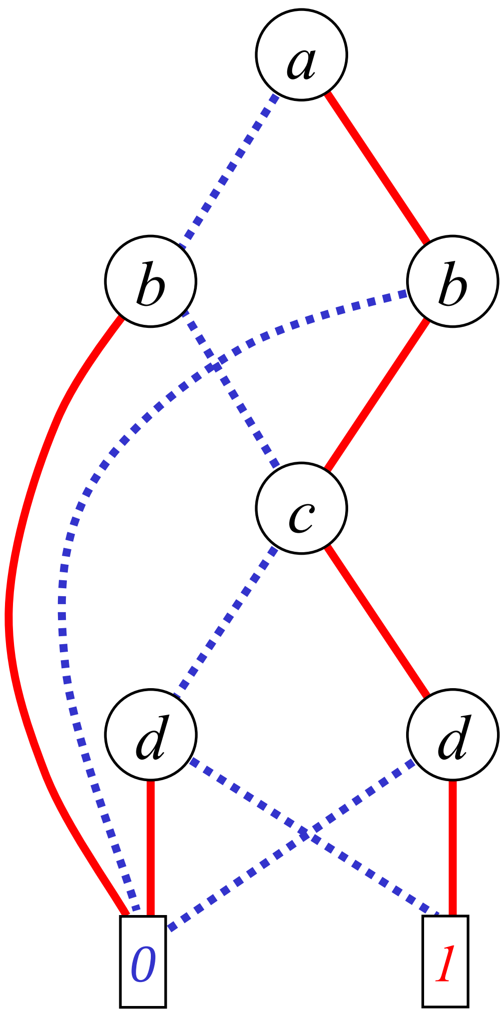
$a{=}b \;\wedge\; c{=}d$



No identical sub-trees or redundant nodes anymore!

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

a=b ∧ c=d



ROBDD

- All variables on the paths occur in the same **O**rder (we had that from the start)
- No identical sub-graphs anymore
- No redundant nodes anymore

⇒ **R** educed **O**rdered
Binary Decision Diagram

Observations

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- For every set (and a fixed variable order) there exists exactly one ROBDD representing it!

- For many practically relevant sets, the ROBBDs representing them are small.

- The size of the ROBDDs depends on the chosen variable order (on the paths):

  For example, the ROBDD for the set characterized by $a{=}b \;\wedge\; c{=}d$ is small with variable order $a < b < c < d$; it is bigger with variable order $a < c < d < b$.

Observations

- There are sets for which the ROBDD will be big for any variable order (multiplication)

- Finding good or even optimal variable orders is one of the challenges of symbolic model checking

- There is no efficient way to find an optimal variable order in general (results from complexity theory)
- But, there are heuristics:
  - Variables that are „somehow related" should be close to each other
  - Local optimisations by switching two variables

Question

- How do we generate an ROBDD?

- Answer: Start with full tree and reduce it!

Question

- How do we generate an ROBDD?

- Answer: Start with full tree and reduce it!

This is a very bad idea!

Rather, we build them bottom up from formulas with operations on ROBDDs.

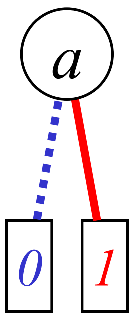
Operationen in ROBBDs

- Boolean variable
- Negation
- Restriction and Shannon expansion
- Binary operations
- ROBDDs and Kripke structures

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

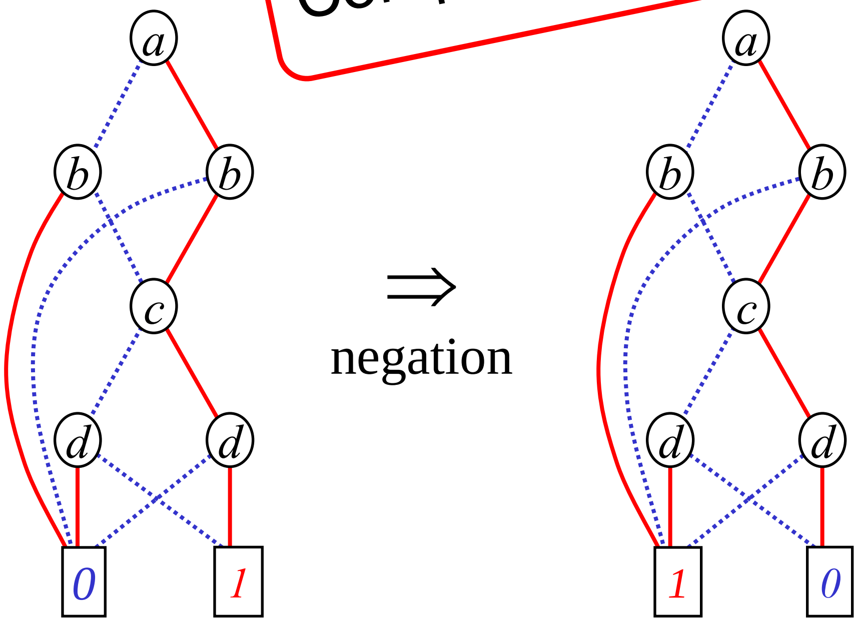
Boolean variable

The set represented by variable $\boldsymbol{a}$ is represented by the ROBBD:



Negation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Complexity: $O(1)$

In practice, this is done a bit differently.

Existing ROBDDs are never changed!

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- For a set (resp. Boolean function) p over variables $v_1, \dots, v_n$ and a Boolean value $t \in \mathbf{B}$, we define the Boolean function $p|_{v_i \leftarrow t}$ by

$$p|_{v_i \leftarrow t}(v_1, \dots, v_n) = p(v_1, \dots v_{i-1}, t, v_{i+1}, \dots, v_n)$$

- $p|_{v_i \leftarrow t}$ is called **restriction** of p.

- It holds (**Shannon expansion** of p):

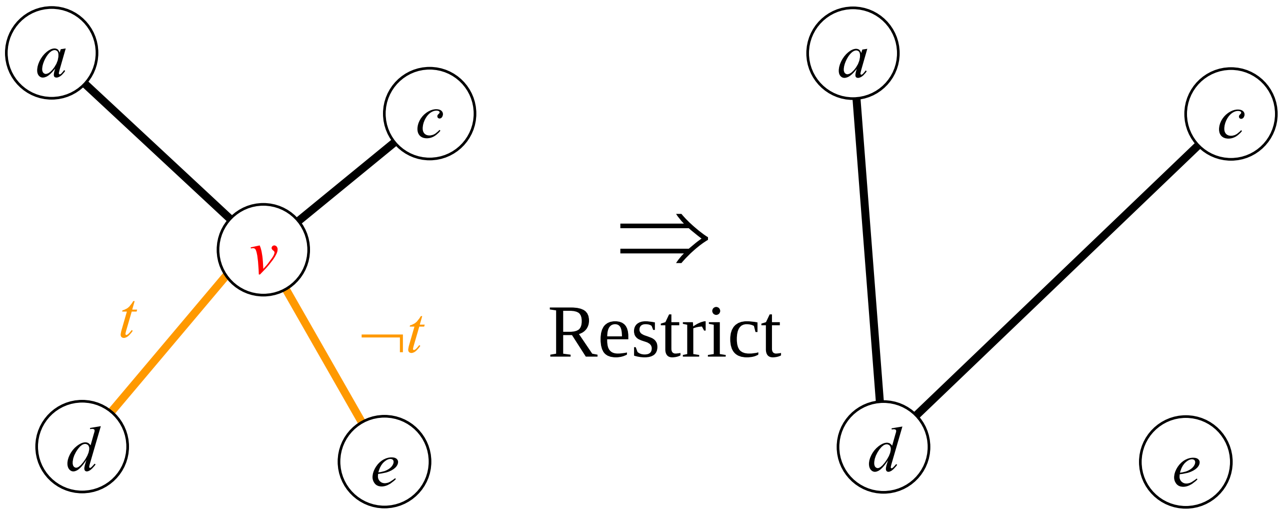
$$p = (\neg v \wedge p|_{v \leftarrow 0}) \vee (v \wedge p|_{v \leftarrow 1})$$

This is like an "if-then-else" in logics.

Restriktion in ROBDDs

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- For a ROBDD representing a Boolean function p, the ROBDD for the $p|_{v \leftarrow t}$ can be obtained as follows:



Complexity: $O(|p|)$

Size of the ROBDDs for p

- Subsequently: systematic reduction of the resulting ROBDD.

Remember:
Existing ROBDDs are never changed!

In practice, this is done a bit different.

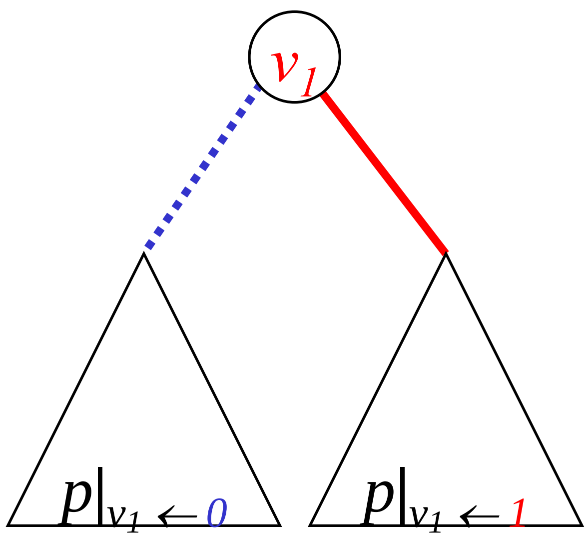
Complexity: $O(|p| \cdot log(|p|))$

Restriction: Special case

- An important special case is the restriction to the first variable v_1 of the ROBDD:

$p|_{v_1 \leftarrow 0}$ bzw. $p|_{v_1 \leftarrow 1}$

In practice, this special case is exploited.



Compexity:
$O(1)$

Boolean operators

- The binary Boolean operations can be formulated recursively by the help of the Shannon expansion:

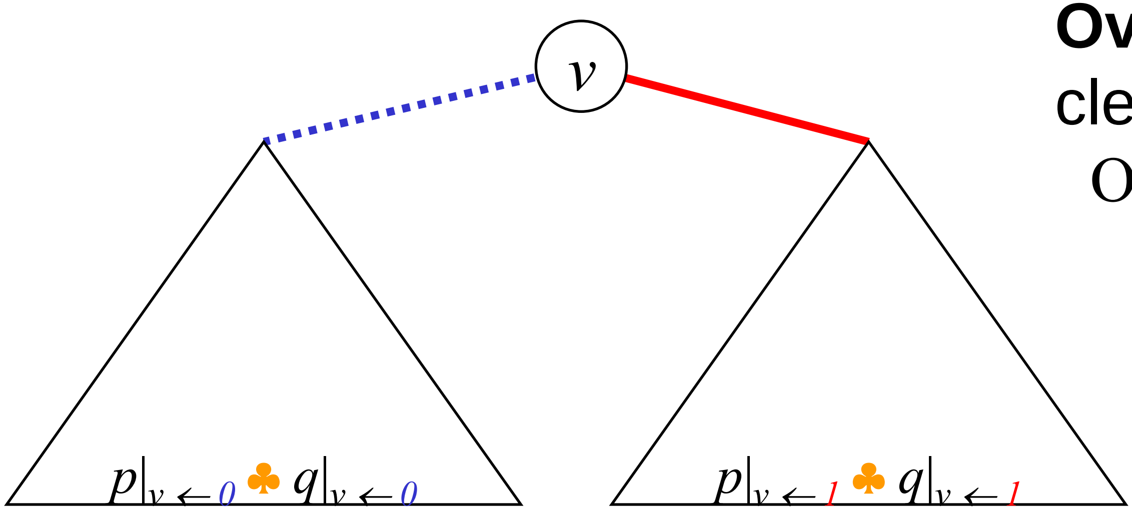
Recursion

- $p \wedge q = \quad (\neg v \wedge (p|_{v\leftarrow 0} \wedge q|_{v\leftarrow 0})) \vee$
  $\quad\quad\quad\quad (v \wedge (p|_{v\leftarrow 1} \wedge q|_{v\leftarrow 1}))$
- $p \vee q = \quad (\neg v \wedge (p|_{v\leftarrow 0} \vee q|_{v\leftarrow 0})) \vee$
  $\quad\quad\quad\quad (v \wedge (p|_{v\leftarrow 1} \vee q|_{v\leftarrow 1}))$
- $p \clubsuit q = \quad (\neg v \wedge (p|_{v\leftarrow 0} \clubsuit q|_{v\leftarrow 0})) \vee$
  $\quad\quad\quad\quad (v \wedge (p|_{v\leftarrow 1} \clubsuit q|_{v\leftarrow 1}))$

works for arbitrary binary operators!

ROBDD for $p \clubsuit q$ from ROBDDs for p and q:

- Generate ROBDDs for $p|_{v \leftarrow 0}$, $q|_{v \leftarrow 0}$, $p|_{v \leftarrow 1}$, and $q|_{v \leftarrow 1}$
- Construct recursively $p|_{v \leftarrow 0} \clubsuit q|_{v \leftarrow 0}$ and $p|_{v \leftarrow 1} \clubsuit q|_{v \leftarrow 1}$
- The OBDD for $p \clubsuit q$ is:



Overall complexity (if cleverly implemented): $O(\,|p| \cdot |q|)$

- Reduce the OBDD systematically to an ROBDD.

ROBDDs: Summary

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- As long as all involved ROBDDs remain small, all operations on ROBDDs are efficient
- There are many libraries implementing ROBDDs and the operations on them (often with clever algorithms for optimizing the variable order). MCiE is a very simple implementation.
- In practice, all ROBDDs in the same context are maintained in a single data structure (as a „forest“ of ROBDDs and hash tables for avoiding duplicate nodes). Then, equality of ROBDDs can be decided in constant time (same pointer).

Quantified Boolean formulas

- For model checking, we need **Boolean formulas** with **quantification** of Boolean variables v (**QBF**):
  $\exists\, v \,.\, p$
- $\exists\, v \,.\, p$ is just an abbreviation for $p|_{v \leftarrow 0} \vee p|_{v \leftarrow 1}$
- $\exists\, \underline{v} \,.\, p$ is an abbreviation for
  $\exists\, v_1 \,.\, (\, \exists\, v_2 \,.\, (\, \ldots\, (\, \exists\, v_n \,.\, p\,)\, \ldots)\,)$

- Respectively, $\forall\, v \,.\, p$ stands for $p|_{v \leftarrow 0} \wedge p|_{v \leftarrow 1}$
- And $\forall\, \underline{v} \,.\, p$ stands for
  $\forall\, v_1 \,.\, (\forall\, v_2 \,.\, (\, \ldots\, (\, \forall\, v_n \,.\, p\,)\, \ldots)\,)$

Relation product

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- For a formula, $p(\underline{u},\underline{v})$ over variables U and V and a formula $q(\underline{v}, \underline{w})$ over variables V and W, we call

$$\exists\, \underline{v}\,.\, p(\underline{u},\underline{v}) \wedge q(\underline{v}, \underline{w})$$

  the **relation product** of $p(\underline{u},\underline{v})$ and $q(\underline{v}, \underline{w})$.

- The ROBDD for the relation product can be realized with the above abbreviations by the Boolean operations. That, however, is a bit inefficient.

- In practice, the relation product is implemented directly. The worst case complexity is exponential; but, it works reasonably well in many practical setting.

5.5 Symbolic model checking

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Represent everything, i.e. initial condition, transition relation as well as the result, as ROBDDs:

Given:

- $\mathcal{S}_0$ and $\mathcal{R}$ as ROBDDs over V resp. $V \cup V'$
- a CTL-Formula p.

Wanted:

- The ROBDD for the set of states S_p (the set of states in which p is true).

Algorithms for CTL

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- We assume that we have calculated the ROBDDs for the sets S_p and S_q already

- Next we give the algorithms for calculating the ROBDDs for the sets
  - $S_{p \vee q}$, $S_{p \wedge q}$ and $S_{\neg p}$,
  - $S_{\mathbf{EX}\ p}$,
  - $S_{\mathbf{EG}\ p}$ and
  - $S_{\mathbf{E}[\ p\ \mathbf{U}\ q\ ]}$

These are the Boolean operations.

Algorithms on the following slides!

Algorithm for $\mathbf{EX}\ p$

Observation:

- $\mathbf{EX}\ p \equiv \exists\, \underline{v}'.\ \mathcal{R}(\underline{v}, \underline{v}') \wedge p(\underline{v}')$

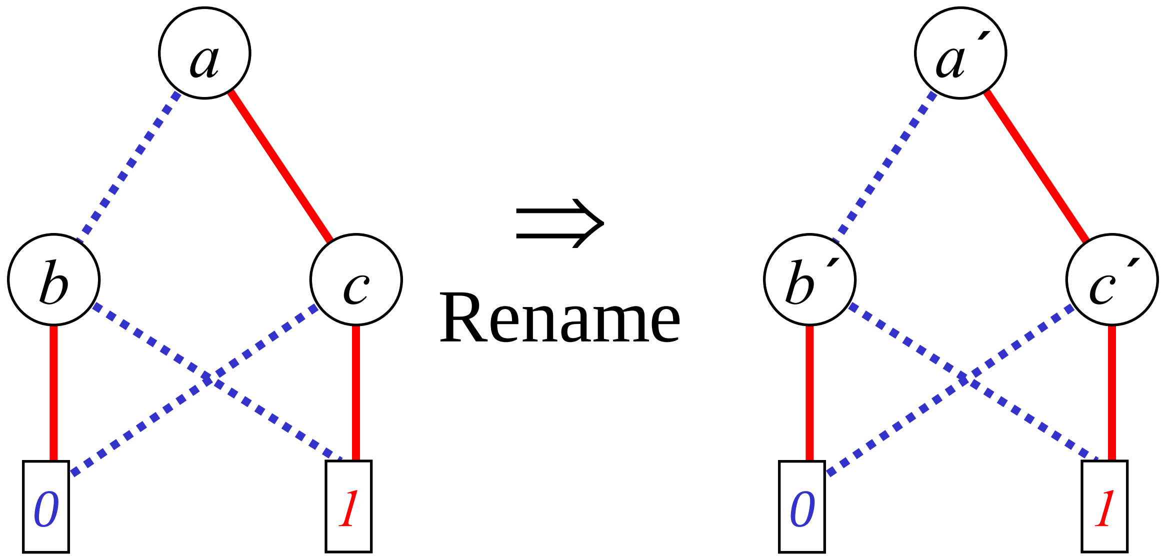
Given ans ROBDD

$p(\underline{v})$ given as ROBDD

Relation product on ROBDDs

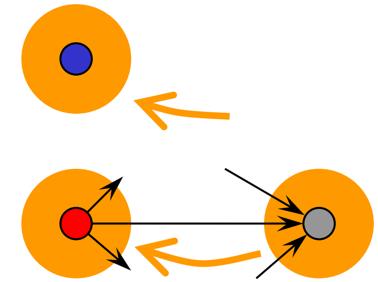
Algorithm for **EX** p

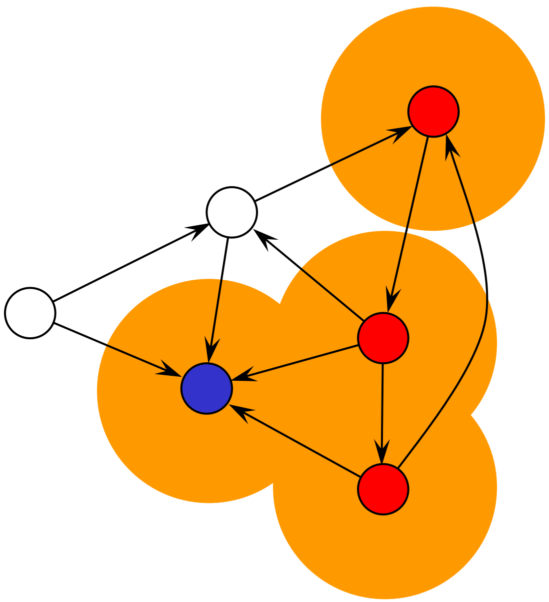
- The only thing left to do is to produce an ROBDD for $p(\underline{v}')$ from an ROBDD for $p(\underline{v})$:



Complexity:
$O(|p|)$

- In practice, this renaming is done on the fly (and only temporarily) when the relation product is calculated





Given: S_p and S_q

Wanted: $S_{\mathbf{E}[\ p\ \mathbf{U}\ q\ ]}$

$$S_0 = S_q$$

$$S_1 = S_q \cup (\ S_p \cap \mathbf{EX}(S_0))$$

$$S_2 = S_q \cup (\ S_p \cap \mathbf{EX}(S_1))$$

...

$$S_{i+1} = S_q \cup (\ S_p \cap \mathbf{EX}(S_i))$$

until $S_{i+1} = S_i = S_{\mathbf{E}[\ p\ \mathbf{U}\ q\ ]}$

Algorithm for $\mathbf{E}[\ p\ \mathrm{U}\ q\ ]$

- In this algorithm, the following operations on sets (ROBDDs) occur:
  - test for equality
  - union
  - intersection
  - $\mathbf{EX}(S)$
- For all these operations, we have algorithms already (more or less efficient)
- If the iteration does not change anything (check for equality), this is the ROBDD for $S_{\mathbf{E}[\ p\ \mathrm{U}\ q\ ]}$.

Procedure `checkEU(`S_p`,`S_q`)`

$S := S_p$; // represented as ROBDD

repeat
  $S' := S$;
  $S := S_q \vee (S_p \wedge$ checkEX(S));
until $S = S'$;

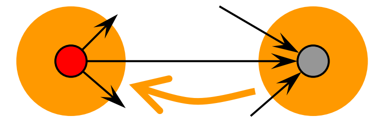
return S;

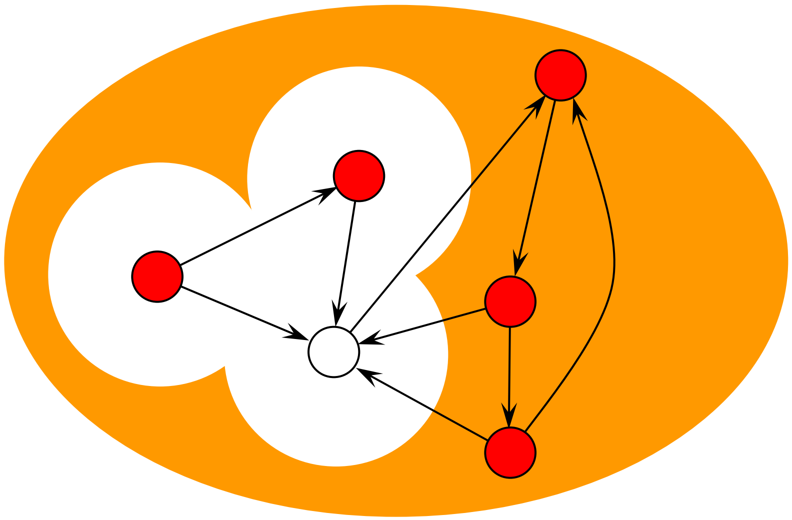
ROBDD operations

procedure for **EX**(S)

Check for equality!
(answers question on
→ slide 10)

Reminder $\mathbf{EG}\,p$





Given: S_p

Wanted: $S_{\mathbf{EG}\,p}$

$$S_0 = S_p$$
$$S_1 = S_p \cap \mathbf{EX}(S_0)$$
$$S_2 = S_p \cap \mathbf{EX}(S_1)$$
$$\ldots$$
$$S_{i+1} = S_p \cap \mathbf{EX}(S_i)$$

until $S_{i+1} = S_i = S_{\mathbf{EG}\,p}$

This is the inefficient algorithm from the introduction.

With the help of ROBDDs it becomes reasonably efficient.

Procedure `checkEX(`S_p`)`

```
S:= Sp; // represented as ROBDD

repeat
   S´:= S;
   S:= Sp ∧ checkEX(S);
until S = S´;

return S;
```

ROBDD operation

procedure for **EX**(S)

Check for equality

Symbolic model checking

- The use of ROBDDs for the representation of sets of states is called **symbolic model checking** (as in contrast to explicit model checking).
- Symbolic model checking contributed to the initial success of model checking (SMV and today NuSMV)!

- Though it uses more inefficient algorithms as one would use with explicit sets, symbolic model checking is sometimes more efficient (but that depends!).

- It does not work always (for bigger examples).
- There are many other techniques for model checking!
- To date, applying model checking for realistic systems requires much experience.