CTL

Chapter 6 – Part 2

Overview

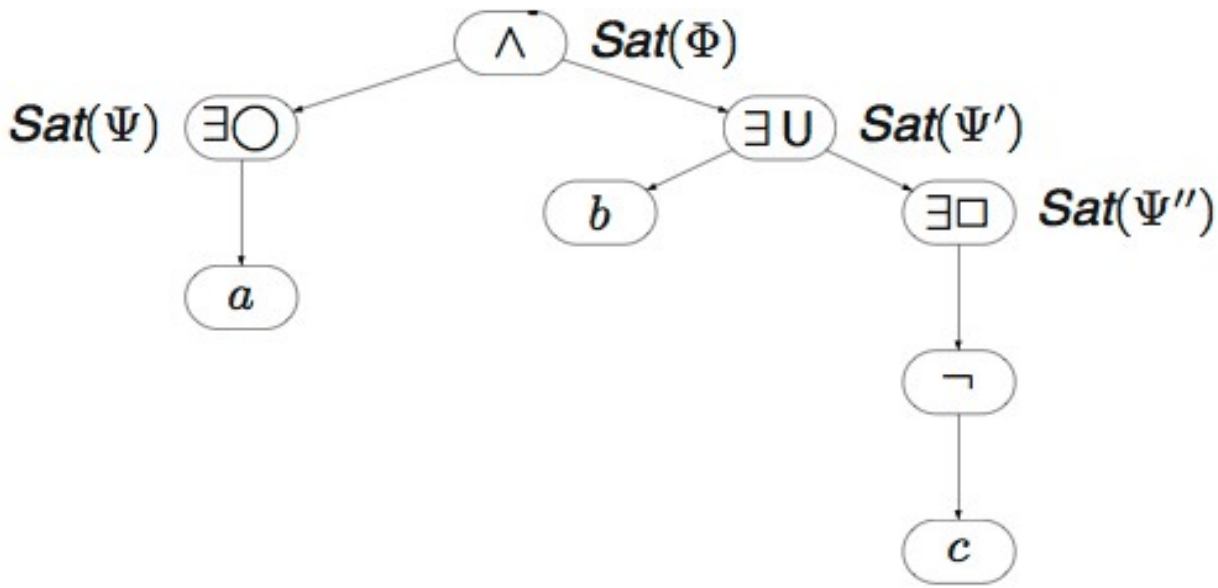
- Review CTL Model Checking
- CTL model Checking algorithms for
  - $\exists(\phi \mathrm{U} \Psi)$
  - $\exists\Box\phi$
- Counter Examples and witnesses
- Symbolic Model Checking (Thursday)
  - Binary Decision Trees
  - Encoding Transition Systems by Switching functions
  - Symbolic Computation
  - Ordered Binary Decision Diagrams

Model checking CTL

- Convert the formula Φ' into the equivalent Φ in ENF
- How to check whether state *TS* satisfies Φ?
  - compute *recursively* the set *Sat*(Φ) of states that satisfy Φ
  - check whether all initial states belong to *Sat*(Φ)
- Recursive bottom-up computation:
  - consider the parse-tree of Φ
  - start to compute *Sat*(a), for all leafs in the tree
  - then go one level up in the tree and check the formula of these nodes
  - then go one level up and check the formula of these nodes
  - and so on..... until the root of the tree (i.e., Φ) is checked

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: katoen@cs.rwth-aachen.de

Example parse tree for CTL



$$\Phi = \underbrace{\exists\bigcirc a}_{\Psi} \wedge \underbrace{\exists(b \,\mathsf{U}\, \underbrace{\exists\Box\neg c}_{\Psi''})}_{\Psi'} \quad .$$

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

January 25, 2006

Basic idea of CTL model checking

Input: finite transition system *TS* and CTL formula Φ (both over *AP*)
Output: $TS \models \Phi$

(* compute the sets $Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$ *)

for all $i \leqslant |\Phi|$ **do**
  for all $\Psi \in Sub(\Phi)$ with $|\Psi| = i$ **do**
    compute $Sat(\Psi)$ from $Sat(\Psi')$ (* for maximal genuine $\Psi' \in Sub(\Psi)$ *)
  od
od
return $I \subseteq Sat(\Phi)$

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: katoen@cs.rwth-aachen.de

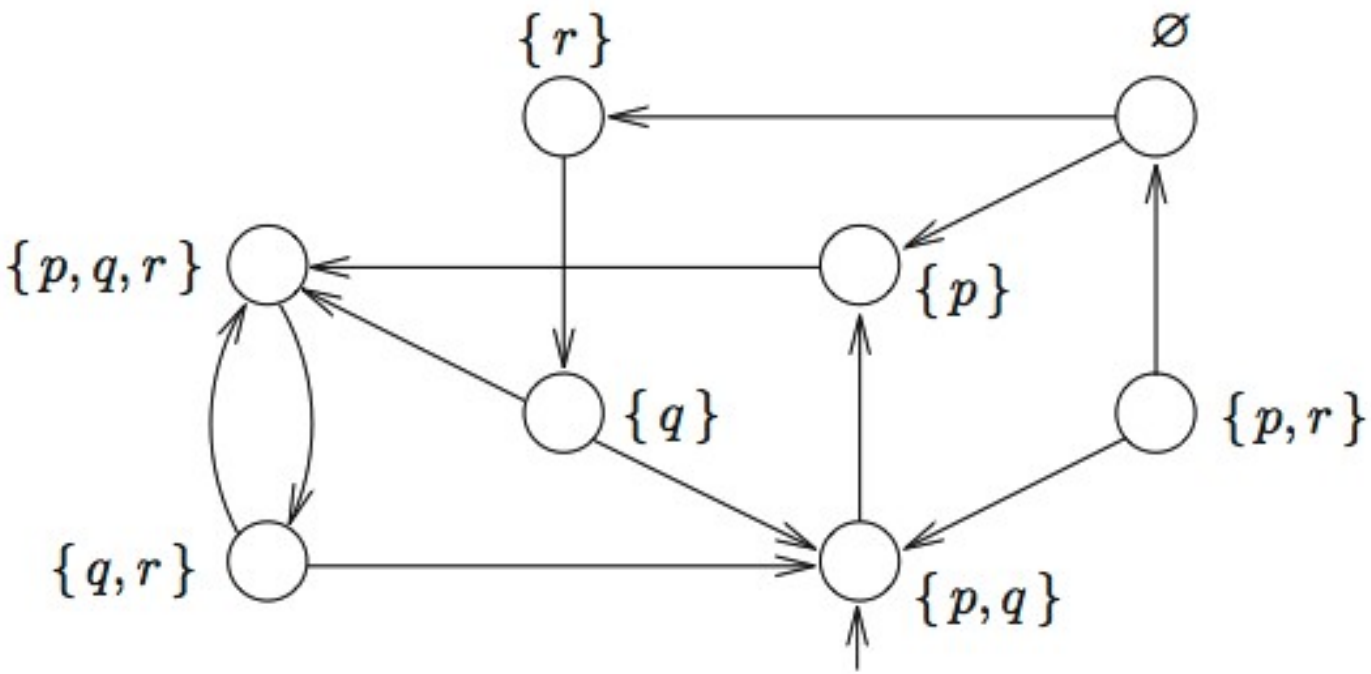
Computing $Sat(\exists(\Phi \mathsf{U} \Psi))$

Input: finite transition system *TS* with state-set S and CTL-formula $\exists(\Phi \mathsf{U} \Psi)$
Output: $Sat(\exists(\Phi \mathsf{U} \Psi)) = \{ s \in S \mid s \models \exists(\Phi \mathsf{U} \Psi) \}$

$E := Sat(\Psi)$; (* E administers the states s with $s \models \exists(\Phi \mathsf{U} \Psi)$ *)
$T := E$; (* T contains the already visited states s with $s \models \exists(\Phi \mathsf{U} \Psi)$ *)
while $E \neq \emptyset$ **do**
  let $s' \in E$;
  $E := E \setminus \{ s' \}$;
  for all $s \in Pre(s')$ **do**
    if $s \in Sat(\Phi) \setminus T$ **then** $E := E \cup \{ s \}; T := T \cup \{ s \}$; **endif**
  od
od
return T

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: `katoen@cs.rwth-aachen.de`

Example



let's check the CTL-formula $\exists\Diamond((p = r) \wedge (p \neq q))$

$\exists\Diamond((p=r)\wedge(p \neq q)) = \exists(\text{true U } ((p=r) \wedge (p \neq q)))$

Trace Algorithm on Whiteboard...

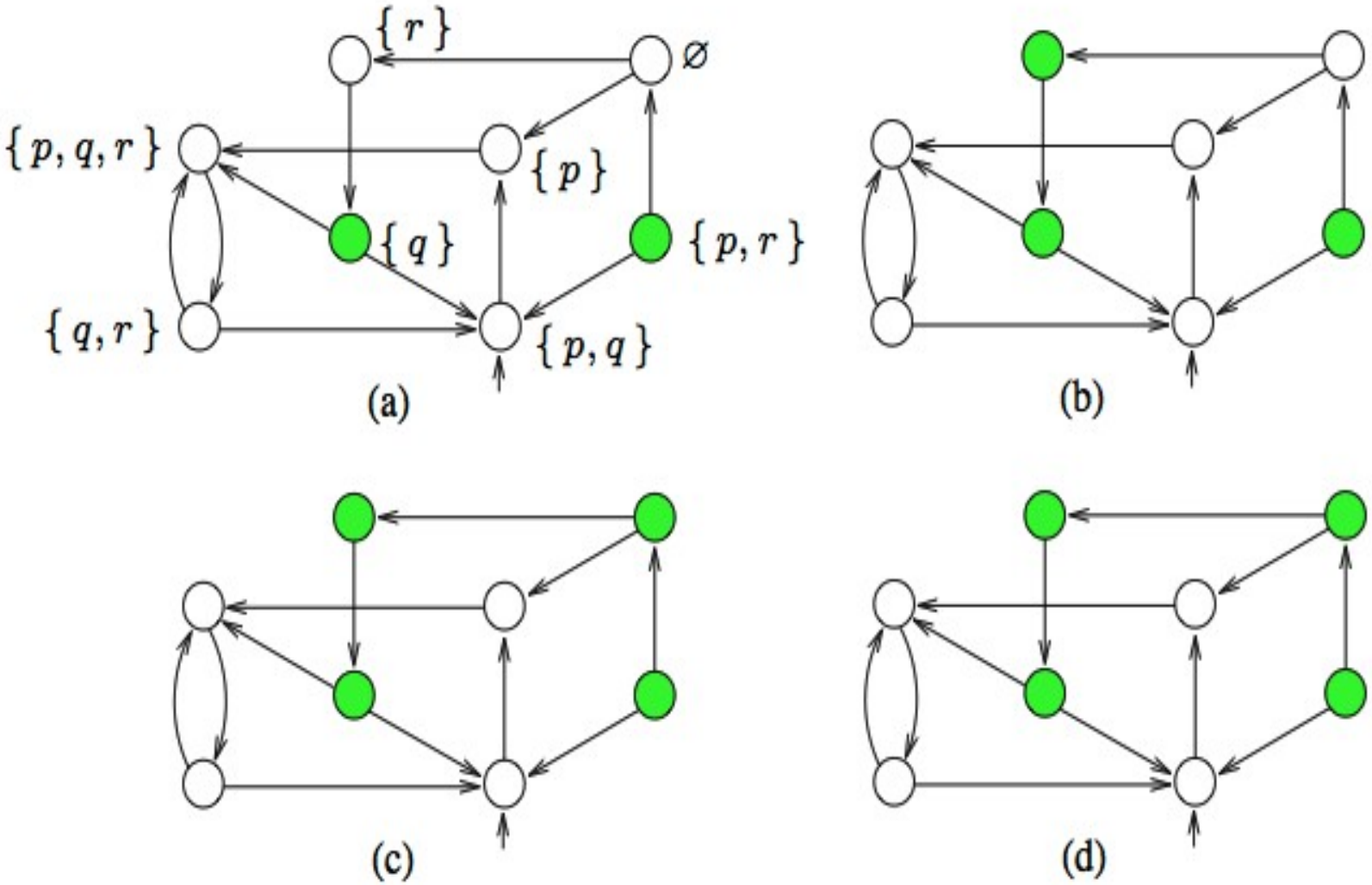
Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: katoen@cs.rwth-aachen.de

January 25, 2006

The computation in snapshots



Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

January 25, 2006

Computing $Sat(\exists\Box\Phi)$

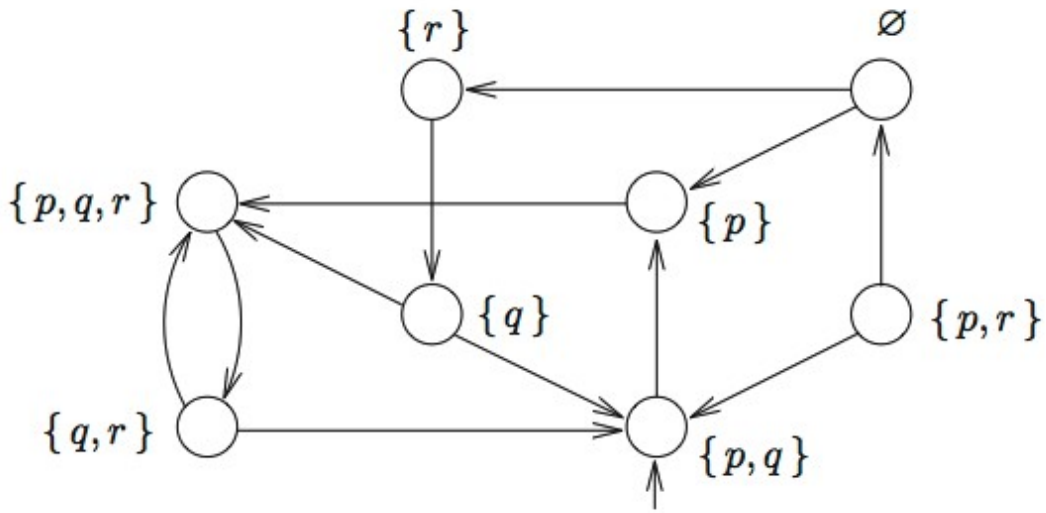
```
E := S \ Sat(Φ);                                   (* E contains any not visited s' with s' ⊭ ∃□Φ *)

T := Sat(Φ);                                       (* T contains any s for which s ⊨ ∃□Φ has not yet been disproven *)

for all s ∈ Sat(Φ) do c[s] := | Post(s) |; od      (* initialize array c *)

while E ≠ ∅ do
                                                   (* loop invariant: c[s] = | Post(s) ∩ (T ∪ E) | *)
  let s' ∈ E;                                      (* s' ⊭ Φ *)
  E := E \ { s' };                                 (* s' has been considered *)
  for all s ∈ Pre(s') do
    if s ∈ T then
      c[s] := c[s] − 1;                            (* update counter c[s] for predecessor s of s' *)
      if c[s] = 0 then
        T := T \ { s }; E := E ∪ { s };            (* s does not have any successor in T *)
      fi
    fi
  od
od
return T
```

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

Compute ∃□q



Trace Algorithm on Whiteboard...

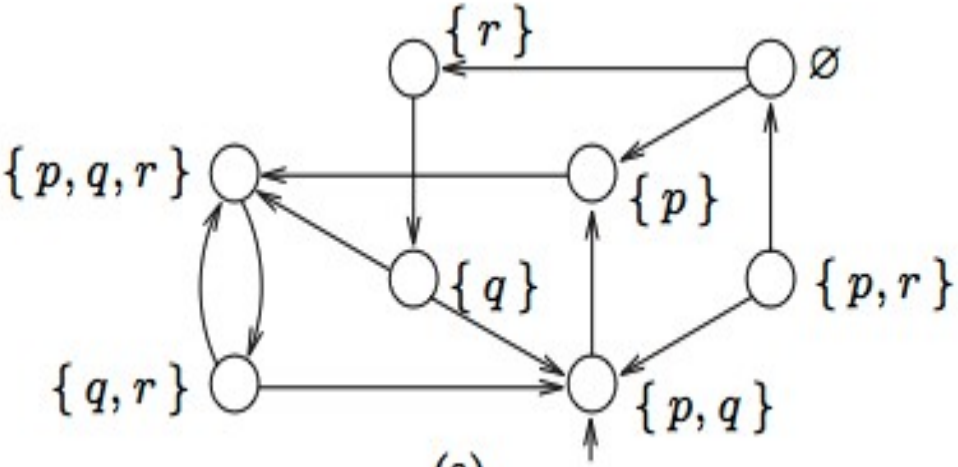
Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

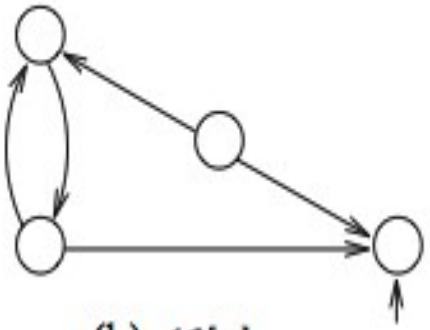
E-mail: katoen@cs.rwth-aachen.de

January 25, 2006

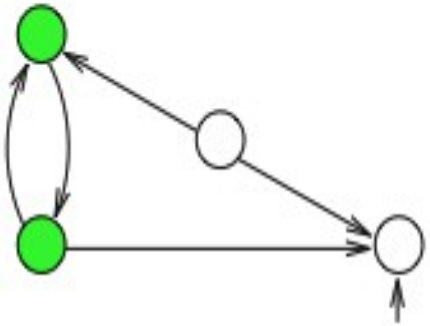
Example



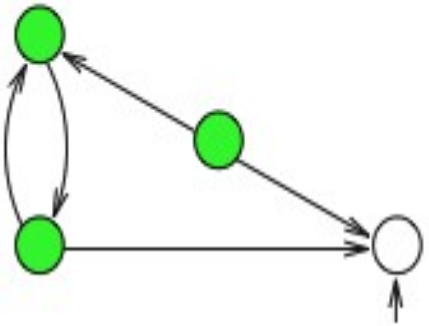
(a)



(b) $\mathcal{K}[q]$



(c) SCC



(d)

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

Alternative algorithm

1. Consider only state s if $s \models \Phi$, otherwise *eliminate* s
   - change *TS* into $TS[\Phi] = (S', Act, \rightarrow', I', AP, L')$ with $S' = Sat(\Phi)$,
   - $\rightarrow' = \rightarrow \cap (S' \times Act \times S')$, $I' = I \cap S'$, and $L'(s) = L(s)$ for $s \in S'$

   $\Rightarrow$ all removed states will not satisfy $\exists\Box\Phi$, and thus can be safely removed

2. Determine all *non-trivial strongly connected components* in $TS[\Phi]$
   - non-trivial SCC = maximal, connected subgraph with at least one transition

   $\Rightarrow$ any state in such SCC satisfies $\exists\Box\Phi$

3. $s \models \exists\Box\Phi$ is equivalent to "some *SCC is reachable* from s"
   - this search can be done in a backward manner

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: katoen@cs.rwth-aachen.de

Counter Examples and Witnesses

- Counter Examples
  - Indicate the refutation of universally quantified path formulae
- Witness
  - Indicate the satisfaction of existentially quantified path formulae

for $\forall\varphi$, a prefix of π with $\pi \not\models \varphi$ acts as counterexample

for $\exists\varphi$, a prefix of π with $\pi \models \varphi$ acts as witness

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: katoen@cs.rwth-aachen.de

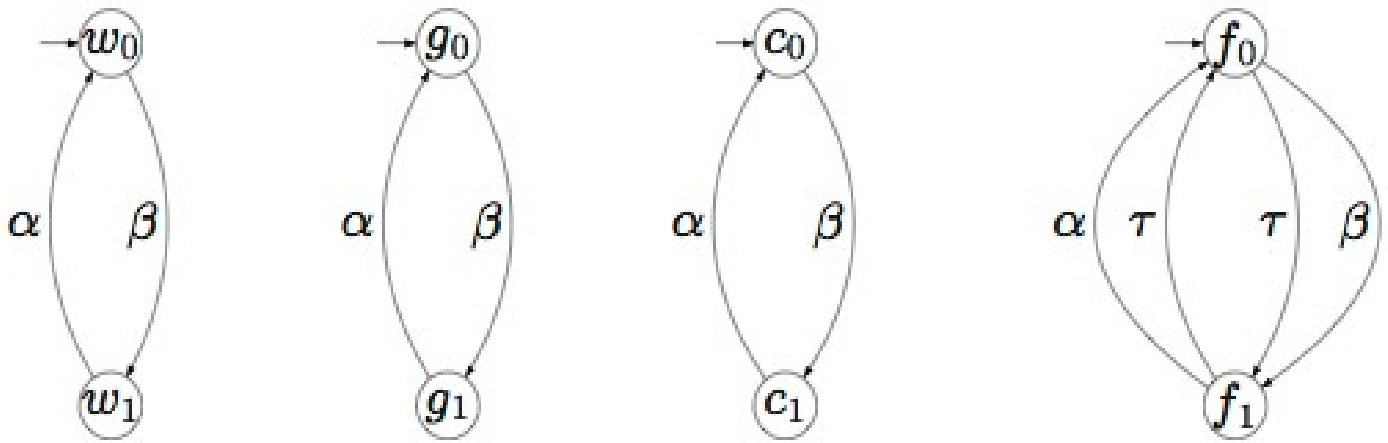
The wolf-goat-cabbage problem

- A goat (g), a cabbage (c) and a wolf (w) and two riverbanks (0 and 1)
  - A boat with ferryman (f) that can carry at most two occupants
  - Only the ferryman can steer the boat
  - Goat and cabbage, goat and wolf should neither travel nor left together
- Is there a schedule such that brings c, g, and w to the other side?
- ... Model this as a CTL model-checking problem
  - transition system $TS = (wolf \mid\mid\mid goat \mid\mid\mid cabbage) \parallel ferryman$
  - check whether $TS \models \exists\varphi$ with

$$\varphi = \left(\bigwedge_{i=0,1} (w_i \wedge g_i \rightarrow f_i) \ \wedge \ (c_i \wedge g_i \rightarrow f_i) \right) \ \mathsf{U} \ (c_1 \wedge f_1 \wedge g_1 \wedge w_1)$$

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: katoen@cs.rwth-aachen.de

The wolf-goat-cabbage problem



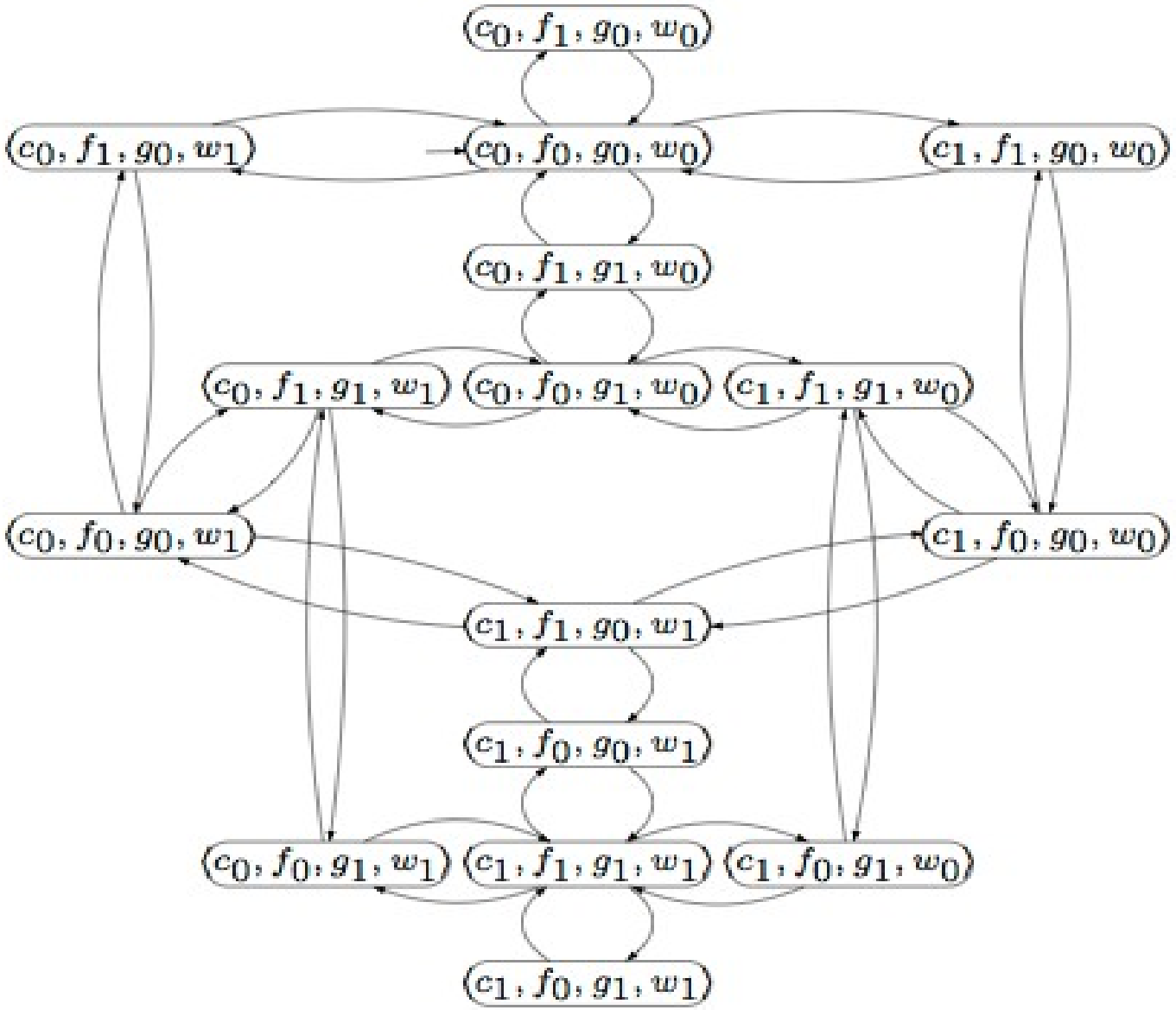
$$TS = (wolf \,|||\, goat \,|||\, cabbage) \parallel ferryman$$

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

January 14, 2009



Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

Wolf-goat-cabbage problem

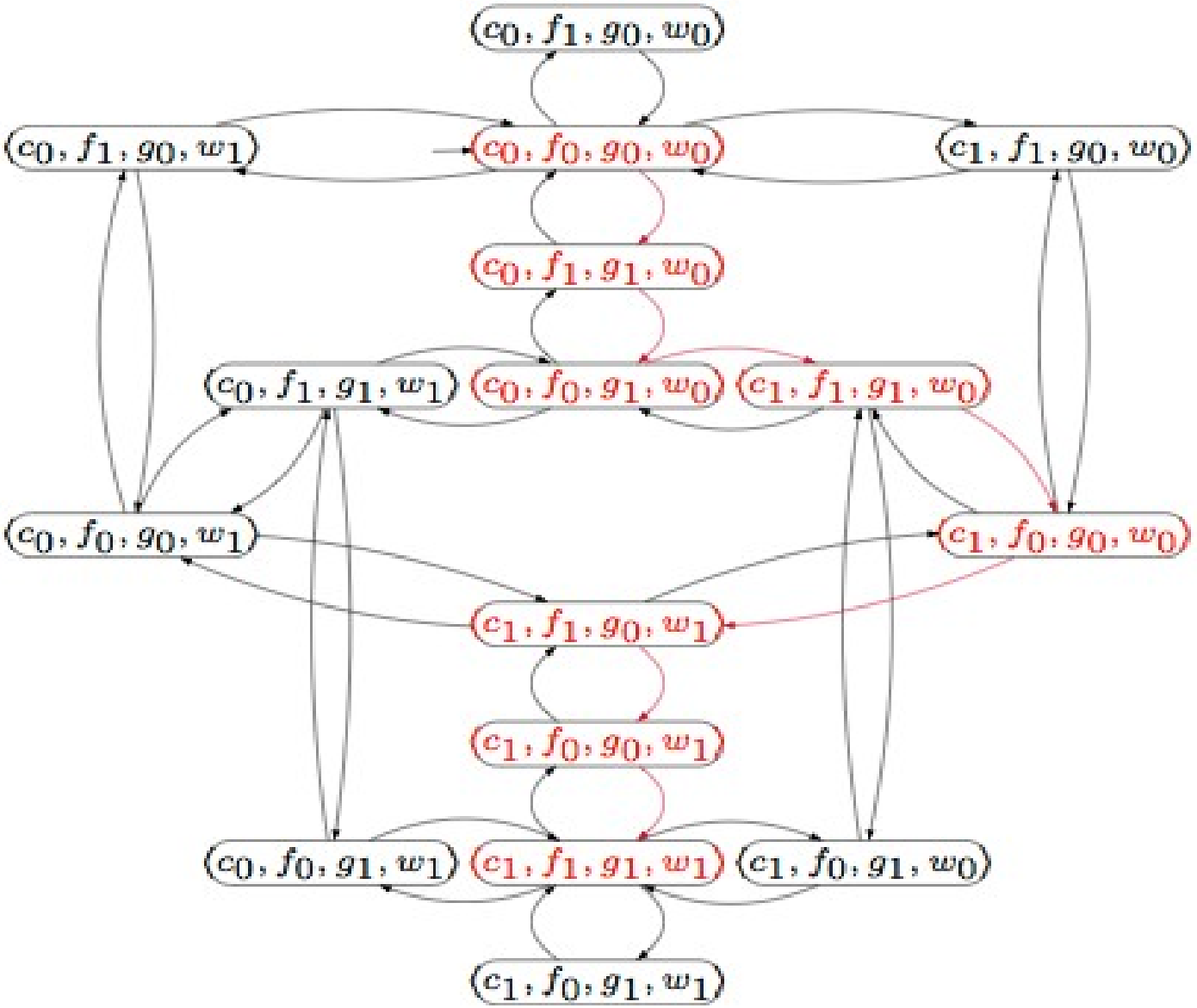
A witness of $\exists\varphi$ with:

$$\varphi \;=\; \left(\bigwedge_{i=0,1} (w_i \wedge g_i \rightarrow f_i) \;\wedge\; (c_i \wedge g_i \rightarrow f_i) \right) \;\mathsf{U}\; (c_1 \wedge f_1 \wedge g_1 \wedge w_1)$$

is a path fragment from initial state $\langle c_0, f_0, g_0, w_0 \rangle$ to target state $\langle c_1, f_1, g_1, w_1 \rangle$ such that g, c and g, w are not left on a single riverbank. Such as:

$\langle c_0, f_0, g_0, w_0 \rangle$	goat to riverbank 1
$\langle c_0, f_1, g_1, w_0 \rangle$	ferryman comes back to riverbank 0
$\langle c_0, f_0, g_1, w_0 \rangle$	cabbage to riverbank 1
$\langle c_1, f_1, g_1, w_0 \rangle$	goat back to riverbank 0
$\langle c_1, f_0, g_0, w_0 \rangle$	wolf to riverbank 1
$\langle c_1, f_1, g_0, w_1 \rangle$	ferryman comes back to riverbank 0
$\langle c_1, f_0, g_0, w_1 \rangle$	goat to riverbank 1
$\langle c_1, f_1, g_1, w_1 \rangle$	

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: `katoen@cs.rwth-aachen.de`



Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

Counterexamples for $\bigcirc\Phi$

- A counterexample of $\bigcirc\Phi$ is a path fragment $s\,s'$ with
  - $s \in I$ and $s' \in \mathit{Post}(s)$ with $s' \not\models \Phi$
- A witness of $\bigcirc\Phi$ is a is a path fragment $s\,s'$ with
  - $s \in I$ and $s' \in \mathit{Post}(s)$ with $s' \models \Phi$
- Algorithm: inspection of direct successors of initial states

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: katoen@cs.rwth-aachen.de

Counterexamples for $\Phi \mathsf{U} \Psi$

- A witness is an initial path fragment $s_0\, s_1 \ldots s_n$ with
  - $s_n \models \Psi$ and $s_i \models \Phi$ for $0 \leqslant i < n$
- Algorithm: backward search starting in the set of Ψ-states
- A counterexample is an initial path fragment that indicates a path π:
  - for which either $\pi \models \Box(\Phi \wedge \neg\Psi)$ or $\pi \models (\Phi \wedge \neg\Psi)\, \mathsf{U}\, (\neg\Phi \wedge \neg\Psi)$
- Counterexample is initial path fragment of either form:
  - $\underbrace{s_0 \ldots s_{n-1}\ \underbrace{s_n\, s_1' \ldots s_r'}_{\text{cycle}}}_{\text{satisfy } \Phi \wedge \neg\Psi}$ with $s_n = s_r'$ or $\underbrace{s_0 \ldots s_{n-1}}_{\text{satisfy } \Phi \wedge \neg\Psi}\ s_n$ with $s_n \models \neg\Phi \wedge \neg\Psi$

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: `katoen@cs.rwth-aachen.de`

Counterexample generation

Determine the SCCs by of the digraph $G = (S, E)$ where

$$E \;=\; \{\, (s, s') \in S \times S \mid s' \in \mathit{Post}(s) \;\wedge\; s \models \Phi \wedge \neg\Psi \,\}$$

Each path in G that starts in an initial state $s_0 \in S$ and leads to a non-trivial SCC C in G provides a counterexample of the form:

$$s_0\, s_1 \ldots s_n \underbrace{s'_1 \ldots s'_r}_{\in C} \quad \text{with} \quad s_n = s'_r$$

Each path in G that leads from an initial state s_0 to a trivial terminal SCC

$$C = \{\, s' \,\} \quad \text{with} \quad s' \not\models \Psi$$

provides a counterexample of the form $s_0\, s_1 \ldots s_n$ with $s_n \models \neg\Phi \wedge \neg\Psi$

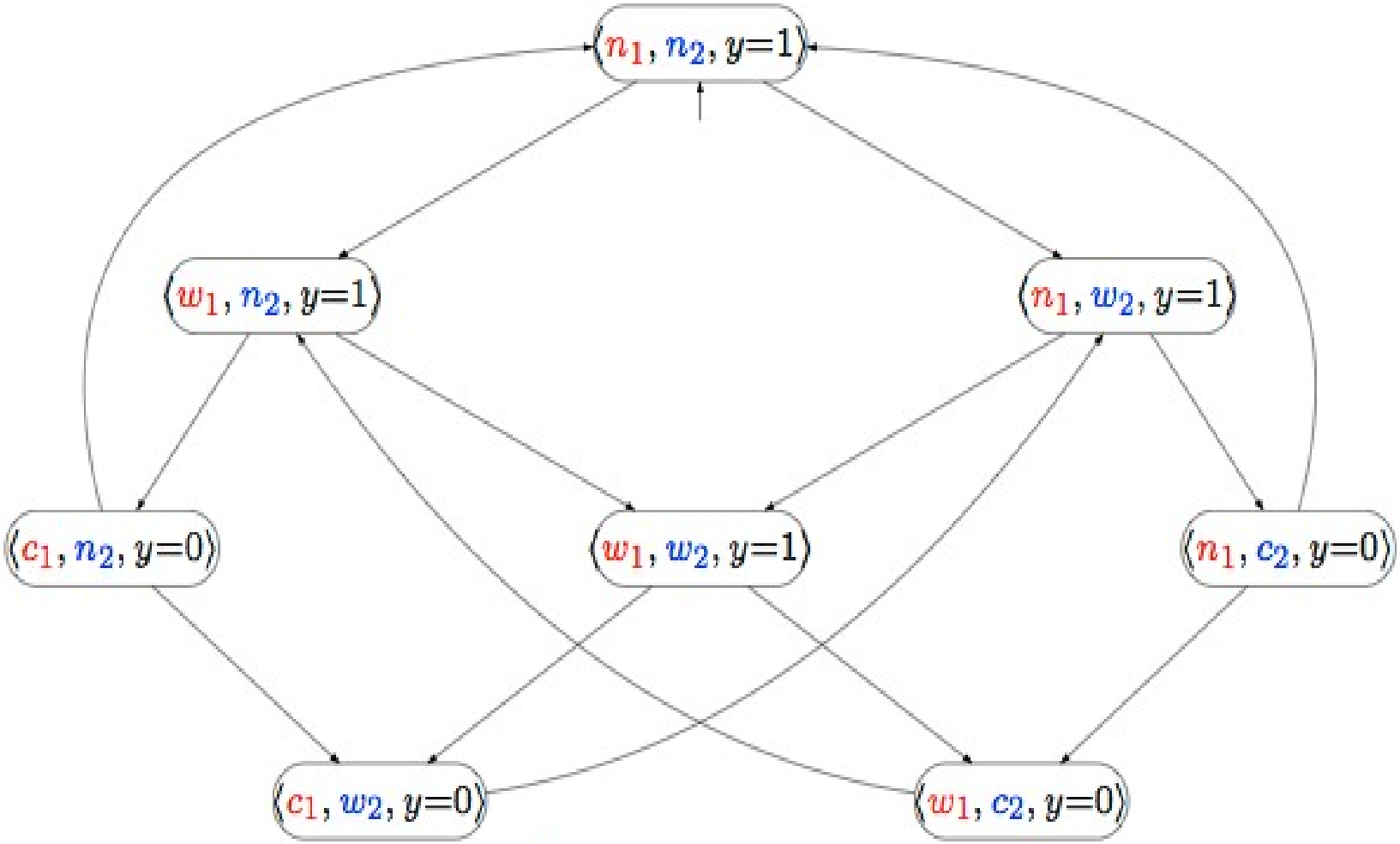
Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: `katoen@cs.rwth-aachen.de`

Counterexamples for $\Box\Phi$

- Counterexample is initial path fragment $s_0\, s_1 \ldots s_n$ such that:
  - $s_0, \ldots, s_{n-1} \models \Phi$ and $s_n \not\models \Phi$
- Algorithm: backward search starting in $\neg\Phi$-states
- A witness of $\varphi = \Box\Phi$ consists of an initial path fragment of the form:
  - $\underbrace{s_0\, s_1 \ldots s_n\, s_1' \ldots s_r'}_{\text{satisfy } \Phi}$ with $s_n = s_r'$
- Algorithm: cycle search in the digraph $G = (S, E)$ where the set of edges E:
  - $E = \{\, (s, s') \mid s' \in \mathit{Post}(s) \;\wedge\; s \models \Phi \,\}$

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: katoen@cs.rwth-aachen.de

Example



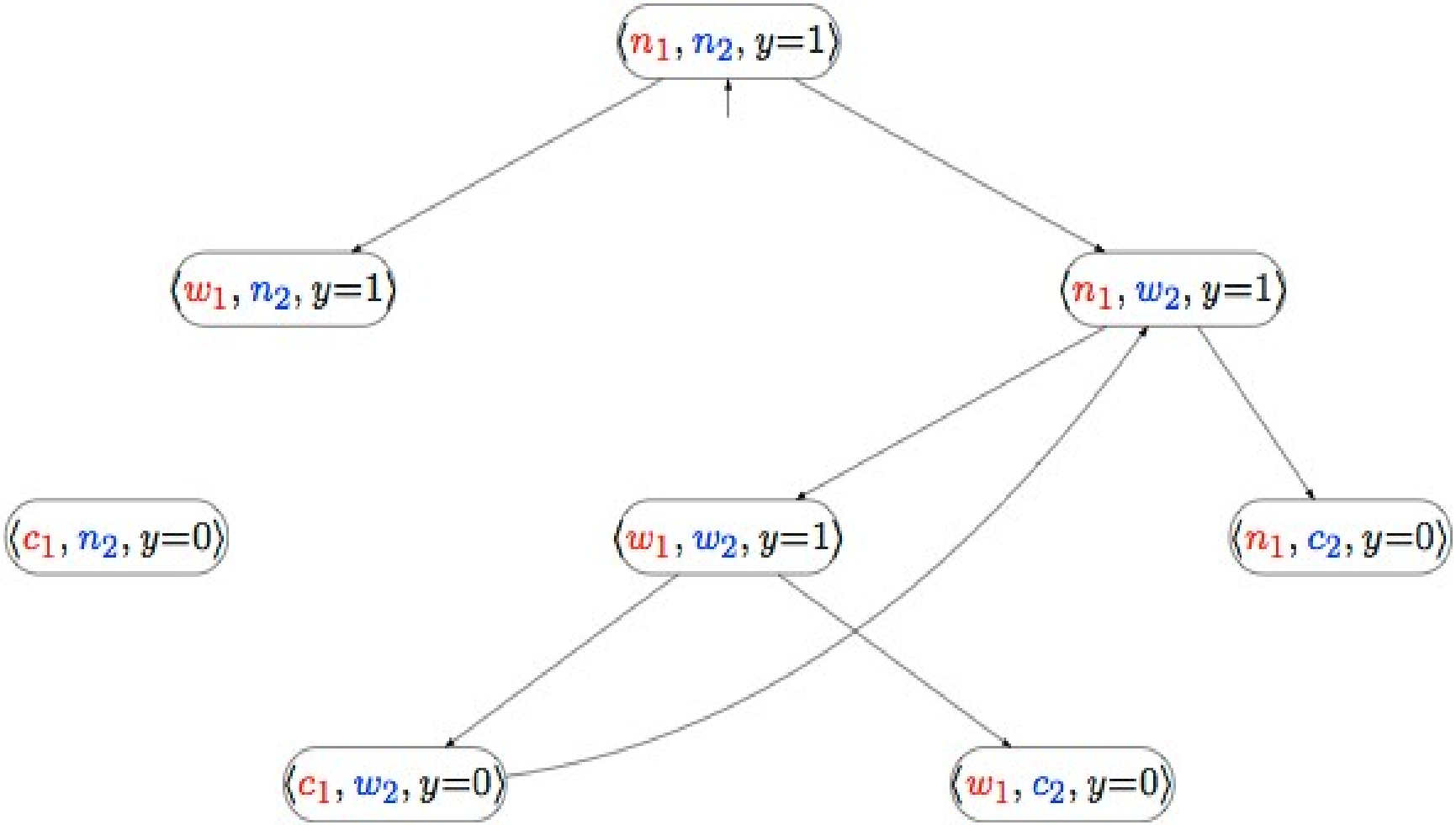
$$\forall\Big(\underbrace{((n_1 \wedge n_2) \ \vee \ w_2)}_{\Phi} \ \mathsf{U} \ \underbrace{c_2}_{\Psi}\Big)$$

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

SCC graph



Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: `katoen@cs.rwth-aachen.de`

Time complexity

Let *TS* be a transition system *TS* with N states and K transitions and φ a CTL- path formula

If *TS* $\not\models \forall\varphi$ then a counterexample for φ in *TS* can be determined in time $\mathcal{O}(N+K)$.

The same holds for a witness for φ, provided that *TS* $\models \exists\varphi$.

Joost-Pieter Katoen
Lehrstuhl 2: Software Modeling & Verification
E-mail: katoen@cs.rwth-aachen.de

Symbolic Model Checking

- Original version of CTL model checking
  - single state and single transition at a time
  - record all the predecessors and successors in each state
  - iterative computation: union and intersection of sets
  - state explosion problem in large transition systems
- Symbolic version
  - sets of states and sets of transitions at a time
  - binary encoding of states
  - one boolean function for each satisfaction set
  - one boolean function for all the transitions
  - iterative computation:conjunction and disjunction of a sequence of bits
  - very efficient

Crystal Chang Din

Precise Modeling and Analysis group (PMA),
University of Oslo

Symbolic Model Checking

Binary encoding (*enc*) of states, as vectors of n bits:

$$\boxed{enc : S \rightarrow \{0,1\}^n}$$

For example:
8 states $(s_0, s_1, \ldots, s_7)$ can be encoded with 3 bits
s_0: 000
s_1: 001
---
s_7: 111

Crystal Chang Din

Precise Modeling and Analysis group (PMA),
University of Oslo

Symbolic Model Checking

- X_T: to encode set of states $T \subseteq S$ (i.e. $T = Sat(\alpha)$):

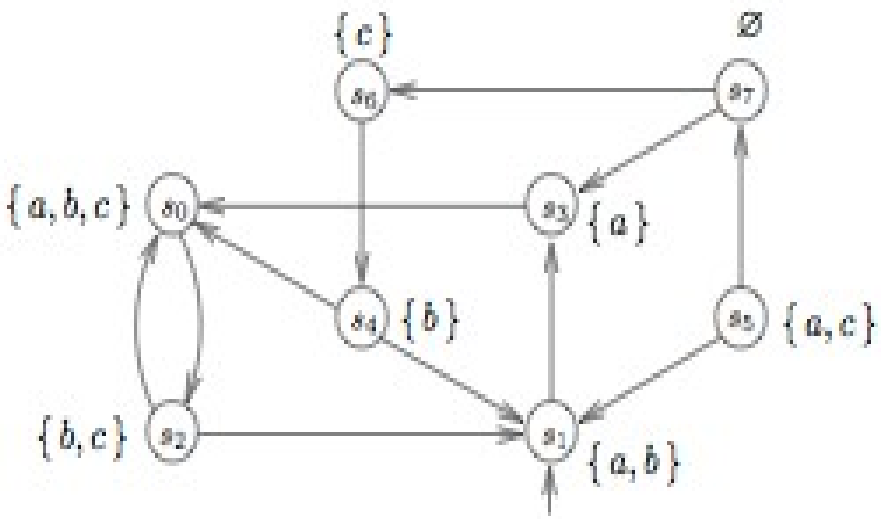
$$\boxed{X_T : \{0,1\}^n \rightarrow \{0,1\} \quad \text{s.t.} \quad X_T(s)=1 \quad \textit{iff} \quad s \in T}$$

- Δ: to encode set of transitions $\rightarrow \subseteq S \times S$:

$$\boxed{\Delta : \{0,1\}^{2n} \rightarrow \{0,1\} \quad \text{s.t.} \quad \Delta(s,s')=1 \quad \textit{iff} \quad s \rightarrow s'}$$

Symbolic Model Checking

- Encoding states and Transitions



Symbolic Model Checking, $\exists(\phi \mathrm{U} \Psi)$

- Algorithm

Algorithm 20 Symbolic computation of $Sat(\exists(C \mathrm{U} B))$

$f_0(\bar{x}) := \chi_B(\bar{x})$;
$j := 0$;
repeat
  $f_{j+1}(\bar{x}) := f_{j+1}(\bar{x}) \vee (\chi_C(\bar{x}) \wedge \exists \bar{x}'.(\Delta(\bar{x}, \bar{x}') \wedge f_j(\bar{x}')))$;
  $j := j + 1$
until $f_j(\bar{x}) = f_{j-1}(\bar{x})$;
return $f_j(\bar{x})$.

- Correction pp. 390, l. Algorithm 20, line 4, $f_{j+1}(\bar{x}) := f_{j+1}(\bar{x}) \vee \ldots \triangleright f_{j+1}(\bar{x}) := f_j(\bar{x}) \vee \ldots$

- Main operation in loop

$$T_{j+1} = T_j \cup \{s \in S \mid \exists s' \in S \text{ s.t. } s' \in Post(s) \wedge s' \in T_j\}$$

Symbolic Model Checking, $\exists(\phi \text{ U } \Psi)$

$\exists\Diamond((a=c)\wedge(a \neq b)) = \exists(\text{true U } ((a=c) \wedge (a \neq b)))$

Trace Algorithm with BDDs on Whiteboard...

We need:

- Sat(a),
- Sat(b),
- Sat(c),
- Transitions (s,s')

Symbolic Model Checking, $\exists\Box\phi$

- Algorithm

Algorithm 21 Symbolic computation of $Sat(\exists\Box B)$

$f_0(\overline{x}) := \chi_B(\overline{x})$;
$j := 0$;
repeat
  $f_{j+1}(\overline{x}) := f_{j+1}(\overline{x}) \wedge \exists\overline{x}'.(\Delta(\overline{x},\overline{x}') \wedge f_j(\overline{x}'))$;
  $j := j+1$
until $f_j(\overline{x}) = f_{j-1}(\overline{x})$;
return $f_j(x)$.

- Correction pp. 391, l. Algorithm 21, line 4, $f_{j+1}(\bar{x}) := f_{j+1}(\bar{x}) \wedge \ldots \triangleright\ f_{j+1}(\bar{x}) := f_j(\bar{x}) \wedge \ldots$

- Main operation in loop

$$T_{j+1} = T_j \cap \{s \in S \mid \exists s' \in S \text{ s.t. } s' \in Post(s) \wedge s' \in T_j\}$$

Symbolic Model Checking, $\exists\Box\phi$

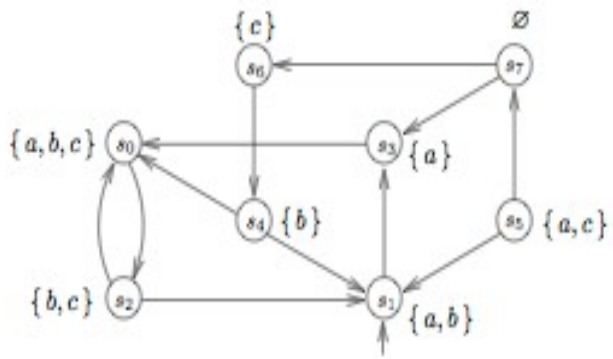
$\exists\Box\phi$

Trace Algorithm with BDDs on Whiteboard...

We need:

- Sat(b),
- Transitions (s,s')

Symbolic Model Checking - ∃□q



8 states $(s_0, s_1, \ldots, s_7)$ can be encoded with 3 bits $(z_1 z_2 z_3)$

$X_T(z_1, z_2, z_3) \rightarrow \{0,1\}, T = Sat(b)$ i.e. $X_T(s_2) = X_T(0,1,0) = 1$

$\Delta(z_1, z_2, z_3, z_1', z_2', z_3') \rightarrow \{0,1\}$ i.e. $\Delta(s_1, s_2) = \Delta(0,0,1,0,1,0) = 0$

BDT Cons...

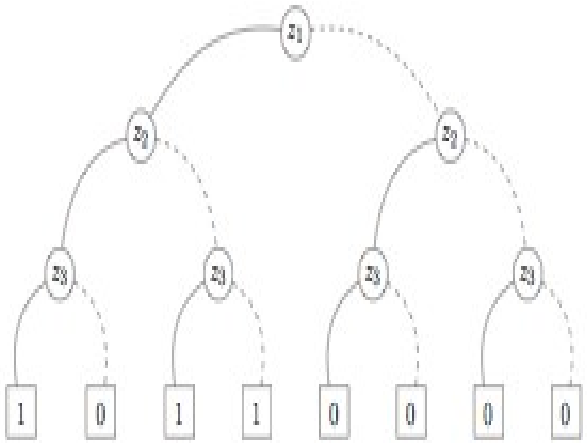
- May not have a data structure which is efficient for all computations
- Data structures which yield compact representations for many computations that appear in practical applications
- OBDDs - data structure that has been proven to be very successful for model checking purposes
  - particularly in the area of hardware verification
  - Besides yielding compact representation for many “realistic” transition systems, they enjoy the property that the Boolean connectives can be realized in time linear in the size of the input OBDDs
  - with appropriate implementation techniques, equivalence checking can even be performed in constant time.

BDT to OBDD

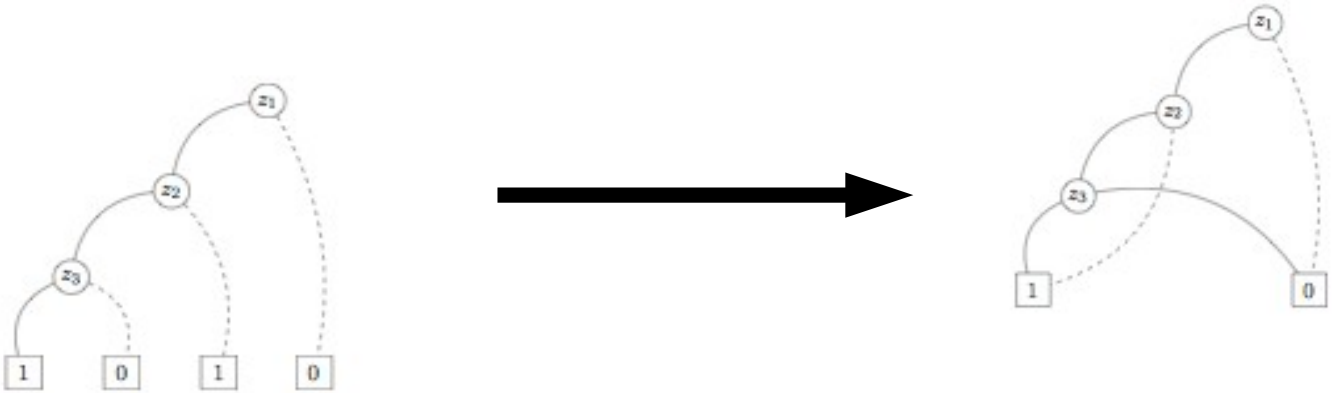
- yield a data structure for computations that relies on a compactification of binary decision trees.
- skip redundant fragments of a binary decision tree.
  - This means collapsing constant subtrees (i.e., subtrees where all terminal nodes have the same value) into a single node
  - identifying nodes with isomorphic subtrees
  - we obtain a directed acyclic graph of outdegree 2
    - the inner nodes are labeled by variables and their outgoing edges stand for the possible evaluations of the corresponding variable.
    - The terminal nodes are labeled by the function value.

BDD to OBDD

- Consider BDD

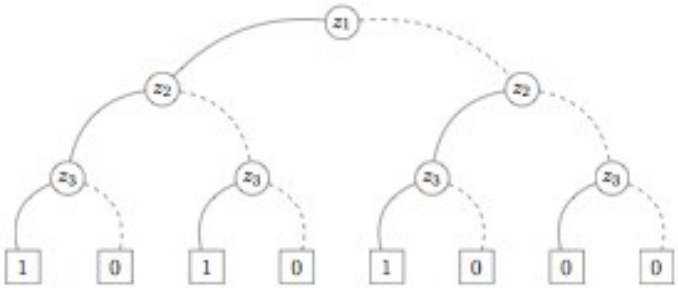


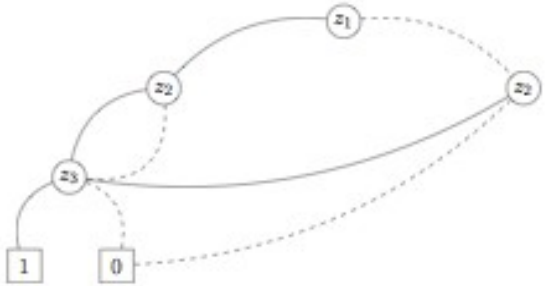
- To ODBBs

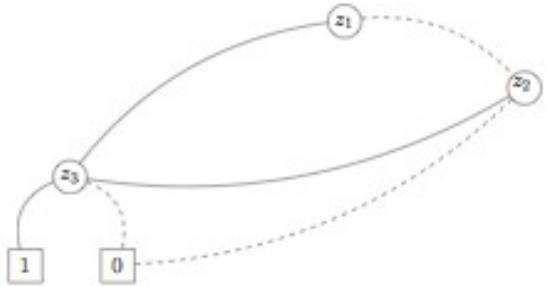


- Correction: line from z3 to 0 should be a dashed line

BDD to OBDD







Variable Ordering

Notation 6.62. Variable Ordering

Let Var be a finite set of variables. A *variable ordering* for Var denotes any tuple $\rho = (z_1, \ldots, z_m)$ such that $Var = \{z_1, \ldots, z_m\}$ and $z_i \neq z_j$ for $1 \leqslant i < j \leqslant m$. We write $<_\rho$ for the induced total order on Var. I.e., for $\rho = (z_1, \ldots, z_m)$ the binary relation $<_\rho$ on Var is given by $z_i <_\rho z_j$ if and only if $i < j$. We write $z_i \leqslant_\rho z_j$ iff either $z_i <_\rho z_j$ or $i = j$. ∎

OBDD formal definition

Definition 6.63. Ordered Binary Decision Diagram (OBDD)

Let $\wp$ be a variable ordering for *Var*. An *$\wp$-ordered binary decision diagram* ($\wp$-OBDD for short) is a tuple

$$\mathfrak{B} = (V, V_I, V_T, succ_0, succ_1, var, val, v_0)$$

consisting of

- a finite set V of nodes, disjointly partitioned into V_I and V_T where the nodes in V_I are called *inner nodes*, while the nodes in V_T are called *terminal nodes* or *drains*;
- successor functions $succ_0$, $succ_1 : V_I \to V$ that assign to each inner node v a 0-successor $succ_0(v) \in V$ and a 1-successor $succ_1(v) \in V$;
- a variable labeling function $var : V_I \to Var$ that assigns to each inner node v a variable $var(v) \in Var$;
- a value function $val : V_T \to \{0, 1\}$ that assigns to each drain a function value 0 or 1;
- a *root (node)* $v_0 \in V$.