NSWI101: System Behaviour Models and Verification

5. OBDD, Lattices and Fixpoints

Jan Kofroň

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Department of
Distributed and
Dependable
Systems

TODAY

- Ordered Binary Decision Diagrams (OBDDs)
- Lattices
- Fixpoints

Explicit vs. symbolic model checking

Explicit model checking

- each particular state of model is *explicitly* represented in memory
- model is explored state-by-state

Symbolic model checking

- based on manipulation with **Boolean formulae**
- operates on entire sets of states rather than individual states
- usually substantial reduction of time and memory consumption

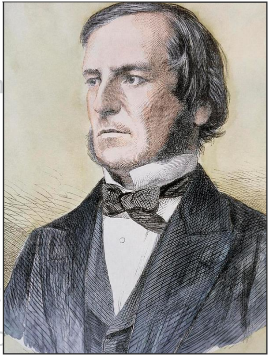
Explicit vs. symbolic model checking

Explicit model checking

- each particular state of model is *explicitly* represented in m
- model is explored state-by-state

Symbolic model checking

- based on manipulation with **Boolean formulae**
- operates on entire sets of states rather than individual stat
- usually substantial reduction of time and memory consump



George Boole (1815 – 1864)
English matematician,
philosopher and logician

Ordered Binary Decision Diagrams (OBDD)

Canonical representation for Boolean formulae

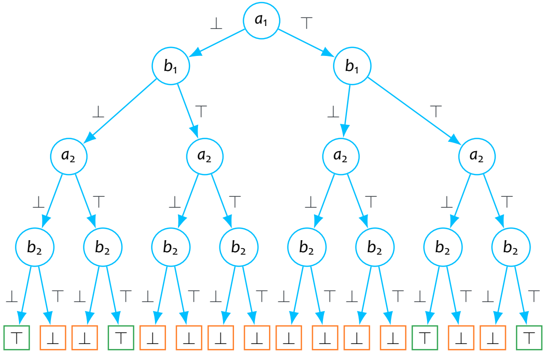
- often substantially more compact than traditional normal forms (CNF, DNF)
- variety of applications:
  - symbolic simulation
  - verification of combinational logic
  - verification of finite-state concurrent systems

Based on binary decision trees

BINARY DECISION TREE

Binary tree with edges directed from root to leaves

- each node level associated with one particular variable
  - the same variable ordering on each path from root to leaf
- one edge from each node represent ⊤ while the other represent ⊥
- terminal nodes (leaves) correspond to final decision – ⊤ or ⊥



Binary decision tree

- Every Boolean formula can be represented by binary decision tree
- Every binary decision tree represents a Boolean formula
- To decide upon value of formula upon given variable assignment, proceed from BDT root to leaf and follow edges according to values assigned to particular variables
- BDTs are not very concise representation of Boolean formulae – essentially same as truth tables, i.e., exponential in number of variables
- Lots of redundancy present in BDT usually

BINARY DECISION DIAGRAM

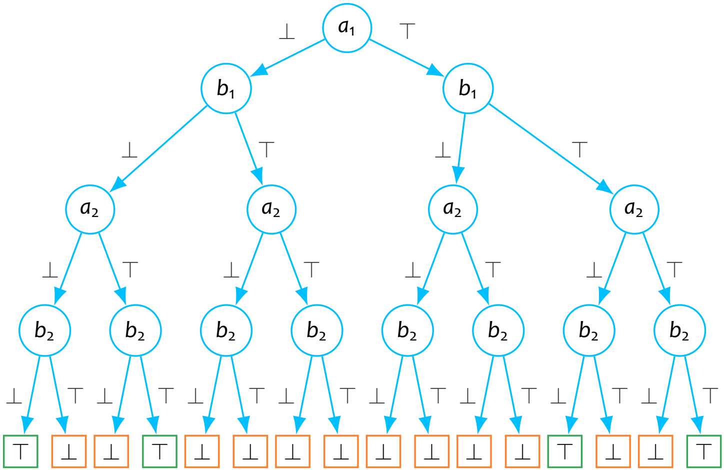
Redundancies in BDT:

- Many terminal symbols with just two different values – $\perp$ and $\top$
- Usually several sets of isomorphic sub-trees that can be merged
- Two sub-trees are isomorphic if:
  - their roots represent the same variable
  - edges originating in them lead to the target states representing the same variables
  - the edges are pair-wise labelled with the same values

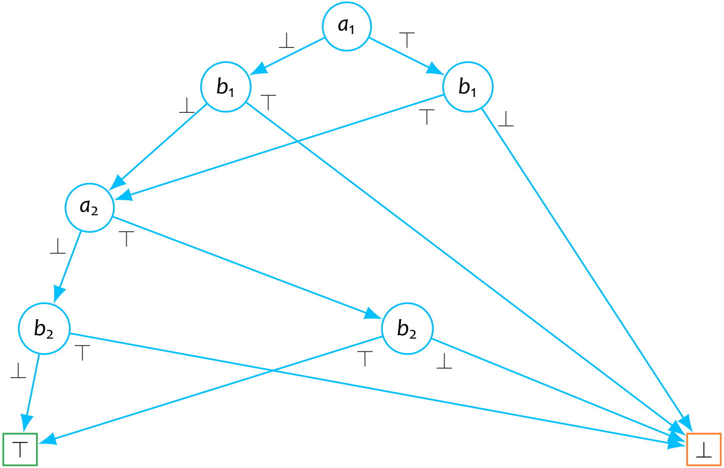
- After removal and merge of nodes from two points above, redundant tests – both edges from node lead to the same target node – can appear and can be removed

Result is not tree anymore, but *directed acyclic graph* (DAG)

Reduction of BDT into OBDD



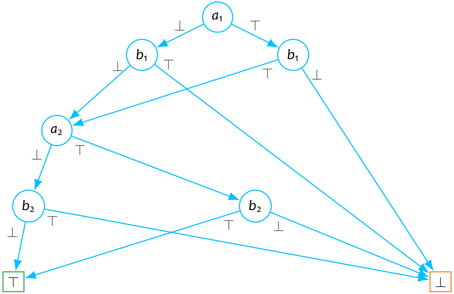
Reduction of BDT into OBDD



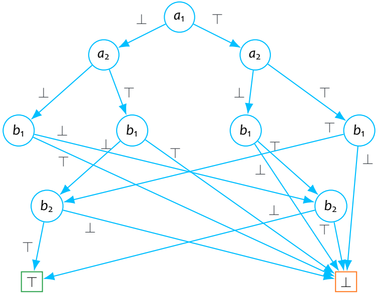
VARIABLE ORDERING

Variable ordering – the order variables are checked on each path from root to leaf – influences size of OBDD substantially:

$a_1 < b_1 < a_2 < b_2$



$a_1 < a_2 < b_1 < b_2$



Variable ordering

- For our n-bit comparator, OBDD size ranges from linear ($3n + 2$) in optimal case to exponential ($3 * 2^n - 1$) in worst case
- In general finding optimal (w.r.t. OBDD size) ordering is not feasible – even checking that particular ordering is optimal is NP-complete
- There are many functions for which every ordering results exponentially large OBDD

- Fortunately there are heuristics that help
- Using OBDD for representation of Boolean functions (and set of states, in turn) is usually highly efficient:
  - related variables "close together"
  - depth-first traversal
  - dynamic reordering

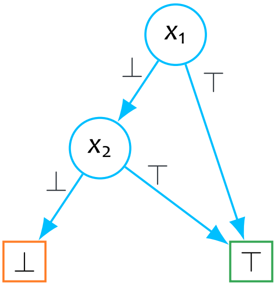
Logical operations upon OBDD

- For practical use (to exploit efficiency) we need to perform logical operations just upon OBDDs, *not* using their "textual" form
- Required operations: restriction, negation, conjunction, and disjunction
  - other operations (e.g., quantification) can be re-written using just these

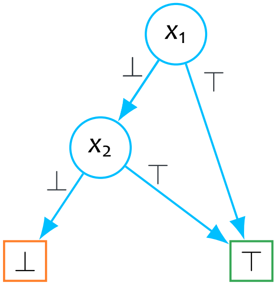
LOGICAL OPERATIONS – RESTRICTION

Restriction refers to fixing variable to particular value ($\top$ or $\perp$)

$f_1 : x_1 \vee x_2$



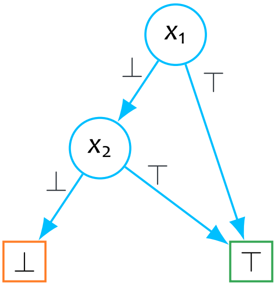
$f_1|_{x_1=\perp}$



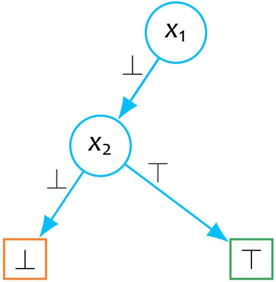
LOGICAL OPERATIONS – RESTRICTION

Restriction refers to fixing variable to particular value ($\top$ or $\bot$)

$f_1 : x_1 \vee x_2$



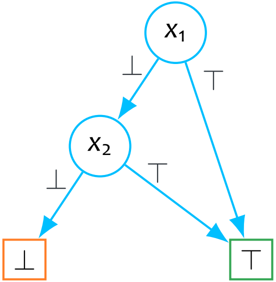
$f_1|_{x_1=\bot}$



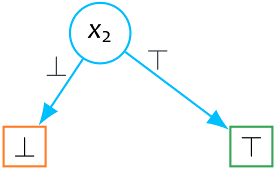
LOGICAL OPERATIONS – RESTRICTION

Restriction refers to fixing variable to particular value ($\top$ or $\perp$)

$f_1 : x_1 \vee x_2$



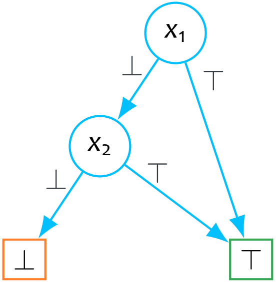
$f_1|_{x_1=\perp}$



LOGICAL OPERATIONS – NEGATION

Performing *negation* is straightforward by swapping terminals

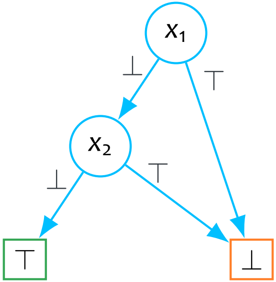
$$f_1 : x_1 \vee x_2$$



LOGICAL OPERATIONS – NEGATION

Performing *negation* is straightforward by swapping terminals

$$\neg f_1 : \neg(x_1 \vee x_2)$$



Logical operations – general case

Let $*$ be arbitrary binary logical operation, e.g. conjunction

Notation:

- f, f' – Boolean functions to be combined by $*$
- v, v' – roots of OBDDs representing f, f', respectively
  - both OBDDs respect the same variable ordering
- x_v – variable associated with non-terminal vertex v

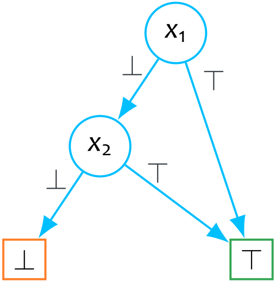
Logical operations – general case

- If v, v' are both terminals: $f * f = \mathit{value}(v) * \mathit{value}(v')$
- If v, v' are both non-terminals and $x_v = x_{v'}$:
  $f * f' = (\neg x_v \wedge (f|_{x_v=\bot} * f'|_{x_v=\bot})) \vee (x_v \wedge (f|_{x_v=\top} * f'|_{x_v=\top}))$
- If v is non-terminal and v' is either non-terminal and $x_v < x'_v$ or v' is terminal:
  $f * f' = (\neg x_v \wedge (f|_{x_v=\bot} * f')) \vee (x_v \wedge (f|_{x_v=\top} * f'))$
- Symmetrically, if v' is non-terminal and v is either non-terminal and $x_v > x'_v$ or v is terminal:
  $f * f' = (\neg x'_v \wedge (f * f'|_{x'_v=\bot})) \vee (x'_v \wedge (f * f'|_{x'_v=\top}))$
- Split into sub-problems and solved by recursion
- To prevent exponential complexity, dynamic programming to be used yielding polynomial algorithm

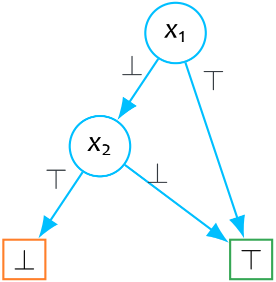
LOGICAL OPERATIONS – CONJUNCTION

Conjunction of two OBDDs: $f_1 \wedge f_2 = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$

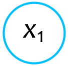
$f_1 : x_1 \vee x_2$



$f_2 : x_1 \vee \neg x_2$



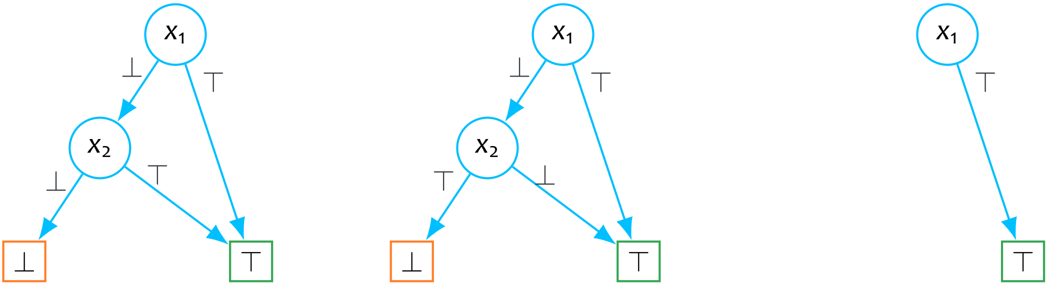
$f_1 \wedge f_2$



Logical operations – conjunction

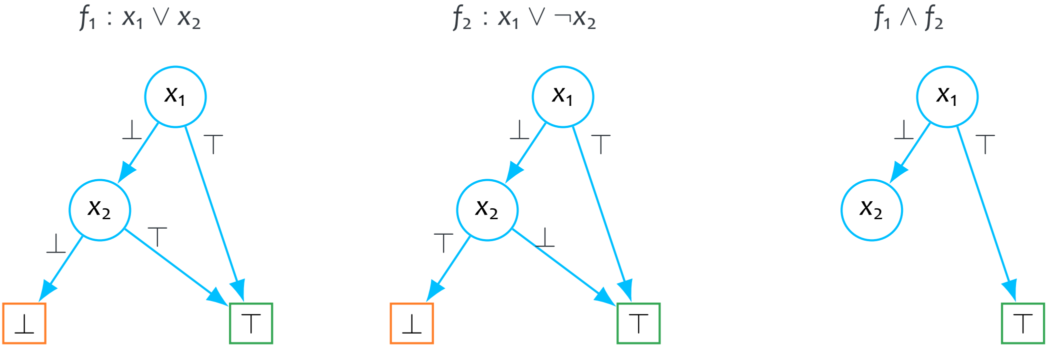
Conjunction of two OBDDs: $f_1 \wedge f_2 = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$

$f_1 : x_1 \vee x_2$ $\qquad$ $f_2 : x_1 \vee \neg x_2$ $\qquad$ $f_1 \wedge f_2$



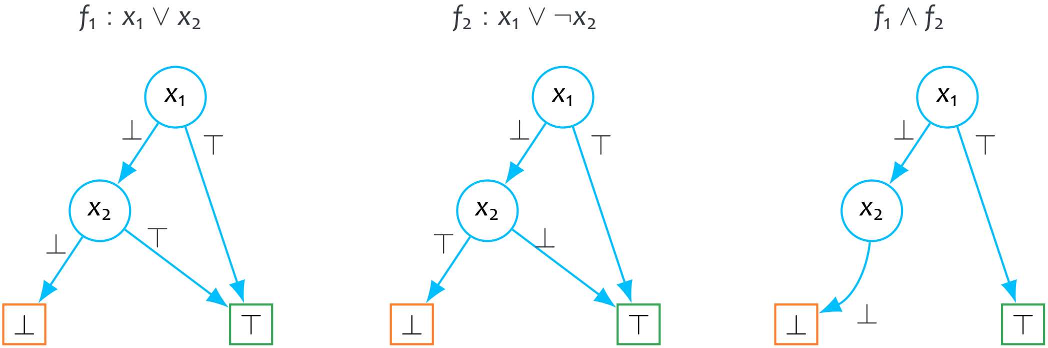
LOGICAL OPERATIONS – CONJUNCTION

Conjunction of two OBDDs: $f_1 \wedge f_2 = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$



Logical operations – conjunction

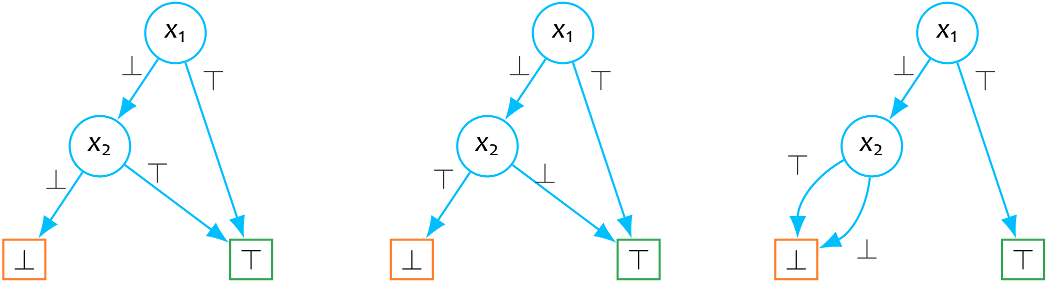
Conjunction of two OBDDs: $f_1 \wedge f_2 = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$



Logical operations – conjunction

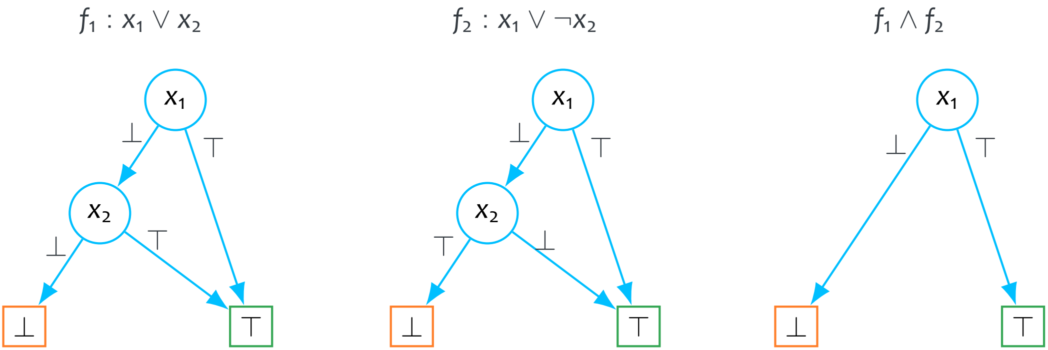
Conjunction of two OBDDs: $f_1 \wedge f_2 = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$

$f_1 : x_1 \vee x_2$ $f_2 : x_1 \vee \neg x_2$ $f_1 \wedge f_2$



Logical operations – conjunction

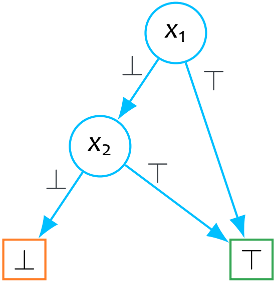
Conjunction of two OBDDs: $f_1 \wedge f_2 = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2)$



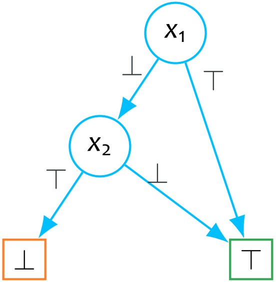
LOGICAL OPERATIONS – DISJUNCTION

Disjunction of two OBDDs: $f_1 \vee f_2 = (x_1 \vee x_2) \vee (x_1 \vee \neg x_2)$

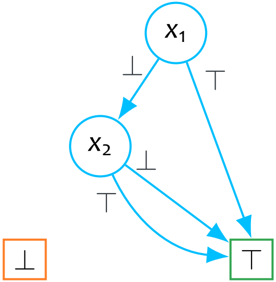
$f_1 : x_1 \vee x_2$



$f_2 : x_1 \vee \neg x_2$



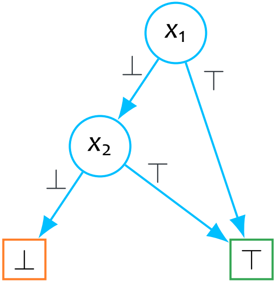
$f_1 \vee f_2$



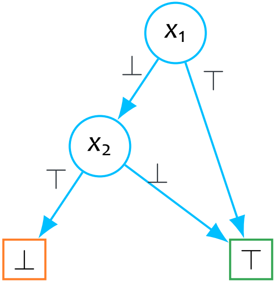
Logical operations – disjunction

Disjunction of two OBDDs: $f_1 \vee f_2 = (x_1 \vee x_2) \vee (x_1 \vee \neg x_2)$

$f_1 : x_1 \vee x_2$



$f_2 : x_1 \vee \neg x_2$



$f_1 \vee f_2$

LOGICAL OPERATIONS – QUANTIFICATION

Quantification of Boolean formula does not introduce greater expressive power:

- $\exists x : f \leftrightarrow f|_{x=\bot} \vee f|_{x=\top}$
- $\forall x : f \leftrightarrow f|_{x=\bot} \wedge f|_{x=\top}$

However, it is convenient in many cases

Relations using OBDDs

Let Q be n-ary relation over $\{0, 1\}$

- Q can be represented by OBDD using its characteristic function: $f_Q(x_1, ..., x_n) = 1 \equiv Q(x_1, ..., x_n)$

Let Q be n-ary relation over finite domain D

- W.l.o.g. assume D has 2^m elements for some $m > 0$
- D can be encoded using bijection: $\phi : \{0, 1\}^m \mapsto D$
- Define relation Q_b of arity $m * n$: $Q_b(\langle x_1 \rangle, ..., \langle x_n \rangle) = Q(\phi(\langle x_1 \rangle), ..., \phi(\langle x_n \rangle))$
  - $\langle x_i \rangle$ is vector of m Boolean variables encoding variable x_i
- Q can be represented as OBDD using characteristic function for Q_b

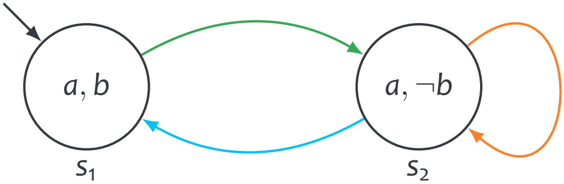
Kripke structure as OBDDs

Let $M = (S, I, R, L)$ be Kripke structure:

- Sets of states S, I: $\phi : \{0, 1\}^m \mapsto S$, assuming 2^m states for some m
- Transition relation R: using characteristic function f_{R_b} of $R_b(\langle x \rangle, \langle x' \rangle)$
- Labelling function L:
  - in contrast to usual direction of mapping states to subset of atomic proposition satisfied in particular states, inverse mapping used here
  - each atomic proposition corresponds to subset of states satisfying it: $L_p = \{s \in S | p \in L(s)\}$
  - OBDDs for each one created using its characteristic function

Kripke structure as OBDDs

	x
s_1	0
s_2	1



$I : \neg x$

$R : (\neg x \wedge x') \vee (x \wedge x') \vee (x \wedge \neg x')$

$L : a \mapsto \{s_1, s_2\}, b \mapsto \{s_1\}$

$L_a = \{0, 1\}, L_b = \{0\}$

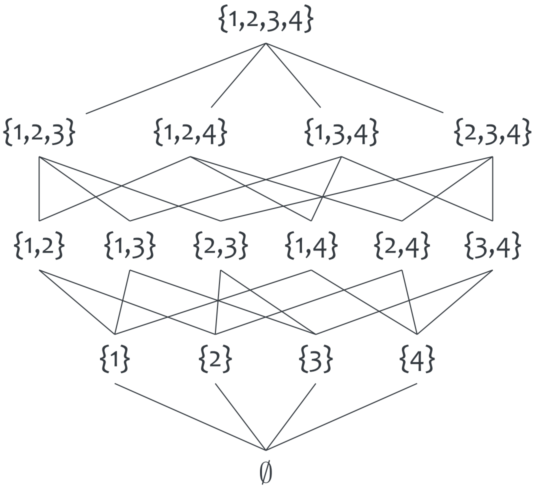
STEP TO SYMBOLIC CTL MODEL CHECKING

- We have Kripke structure represented as OBDDs
  - but we still do not know how to use them for model checking
- We need to define more structures allowing us to model-check
  - lattices
  - fixpoints

Lattice

- *Lattice* L is structure consisting of partially ordered set S of elements where every two elements have
  - unique *supremum* (least upper bound or join) and
  - unique *infimum* (greatest lower bound or meet)
- Set $P(S)$ of all subsets of S forms complete lattice
- Each element $E \in L$ can also be thought as *predicate* on S
- Greatest element of L is S ($\top$, true)
- Least element of L is $\emptyset$ ($\bot$, false)
- $\tau : P(S) \mapsto P(S)$ is called *predicate transformer*

EXAMPLE: SUBSET LATTICE OF {1, 2, 3, 4}



Fixpoints

Let $\tau : P(S) \mapsto P(S)$ be predicate transformer

- τ is *monotonic* $\equiv Q \subseteq R \implies \tau(Q) \subseteq \tau(R)$
- Q is *fixpoint* of $\tau \equiv \tau(Q) = Q$

Fixpoints

Theorem (Knaster-Tarski): A monotonic predicate transformer τ on $P(S)$ always has the least fixpoint $\mu Z.\tau(Z)$, and the greatest fixpoint $\nu Z.\tau(Z)$.

- $\mu Z.\tau(Z) = \cap\{Z | \tau(Z) \subseteq Z\}$
- $\nu Z.\tau(Z) = \cup\{Z | \tau(Z) \supseteq Z\}$

We write $\tau^i(Z)$ to denote *i* applications of τ to Z:

- $\tau^0(Z) = Z$
- $\tau^{i+1}(Z) = \tau\left(\tau^i(Z)\right)$

Fixpoints

Lemma: If τ is monotonic, then for each *i*:

- $\tau^i(\mathit{false}) \subseteq \tau^{i+1}(\mathit{false})$
- $\tau^i(\mathit{true}) \supseteq \tau^{i+1}(\mathit{true})$

Lemma: If τ is monotonic and S is finite, then:

- $\exists i_0 \geq 0 : \forall i \geq i_0 : \tau^i(\mathit{false}) = \tau^{i_0}(\mathit{false})$
- $\exists j_0 \geq 0 : \forall j \geq j_0 : \tau^j(\mathit{true}) = \tau^{j_0}(\mathit{true})$

Lemma: If τ is monotonic and S is finite, then:

- $\exists i_0 : \mu Z.\tau(Z) = \tau^{i_0}(\mathit{false})$
- $\exists j_0 : \nu Z.\tau(Z) = \tau^{j_0}(\mathit{true})$

Knaster-Tarski theorem for finite lattices directly follows from these lemmas

Fixpoints

Kripke structures are finite-state $\Rightarrow$ only finite versions of the theorem needed.

The least and greatest fixpoints of a monotonic predicate transformer can be computed easily (next lecture)