Formal Methods in Software Development

OBDDs, FixPoints, and Symbolic Model Checking

Ivano Salvo

Computer Science Department

SAPIENZA UNIVERSITÀ DI ROMA

Lesson **6**, November 9th, 2019

Symbolic Model Checking

Basic Idea: represent Kripke structures by using **boolean functions**:

1. sets of states (as well as relations) are represented by their **characteristic function**: $x \in S \Leftrightarrow c_S(x)$=True
2. Model Checking problems (such as reachability) solved by working **on set of states**, manipulating their charactheristic functions
3. Set of states satisfying a given temporal logic formula are characterized as the **fixpoint** of some monotone operator.

All this works (sometimes!) thanks to an efficient tool to manipulate boolean functions: **Ordered Binary Decision Diagrams** (OBDDs).

Lesson 7a:

Ordered Binary Decision Diagrams (OBDDs)

Binary Decision Trees

A **binary decision tree** is a **rooted, directed binary tree** that contains two types of vertices:

- ❖ **non-terminal** vertices v labeled by a variable **var(v)** and successors low(v) (when e v has value 0) and **high(v)** (when v has value 1).

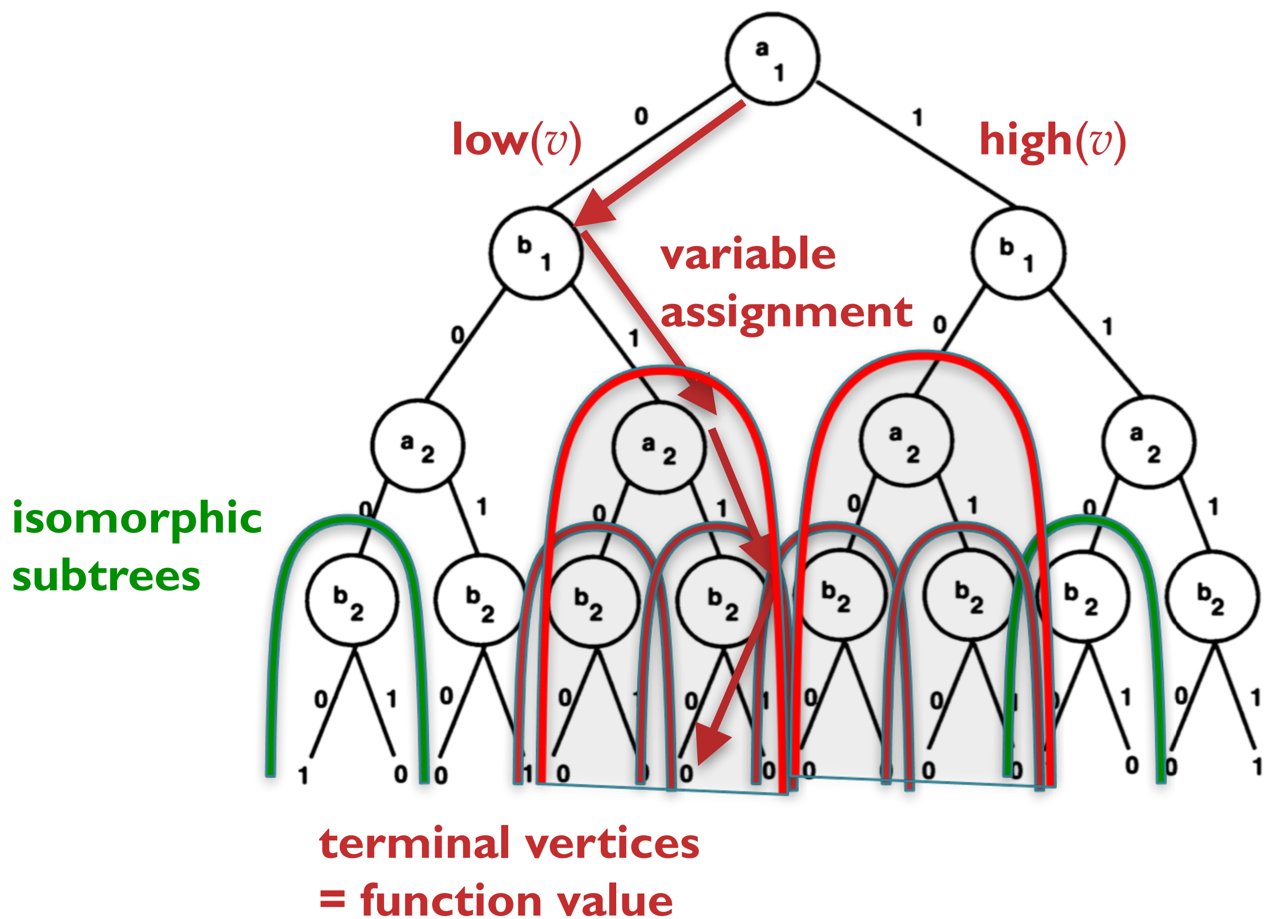
- ❖ a **terminal** vertex v labeled by value(v) that is either 0 or 1.

A binary decision tree represents a **boolean function**.

Each **path** represents an **assignment to boolean variables** and the value of the terminal node represents the value of the function for that assignment.

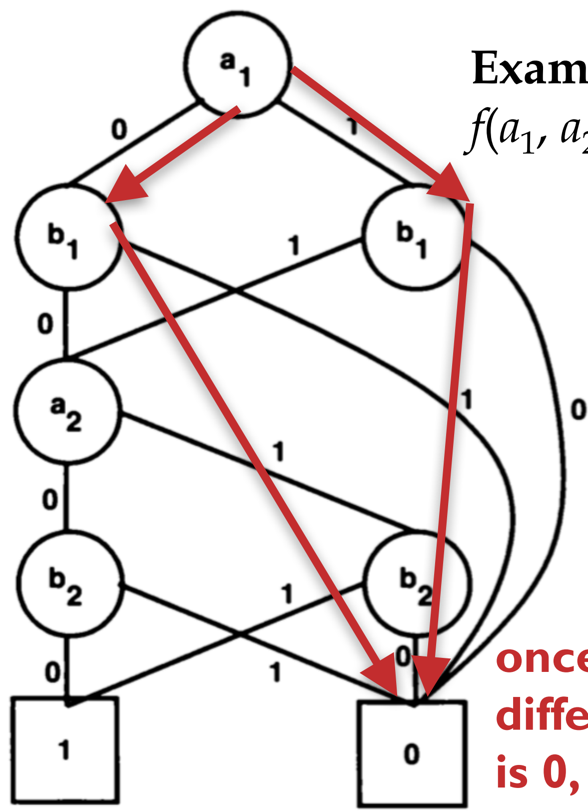
Binary Decision Trees

Example: two bit comparator: $f(a_1, a_2, b_1, b_2) = a_1 \leftrightarrow b_1 \wedge a_2 \leftrightarrow b_2$



Binary Decision Diagrams

Idea: merging isomorphic subtrees (trivially, for example, just two terminal nodes): this leads to **Directed Acyclic Graphs** (DAGs).

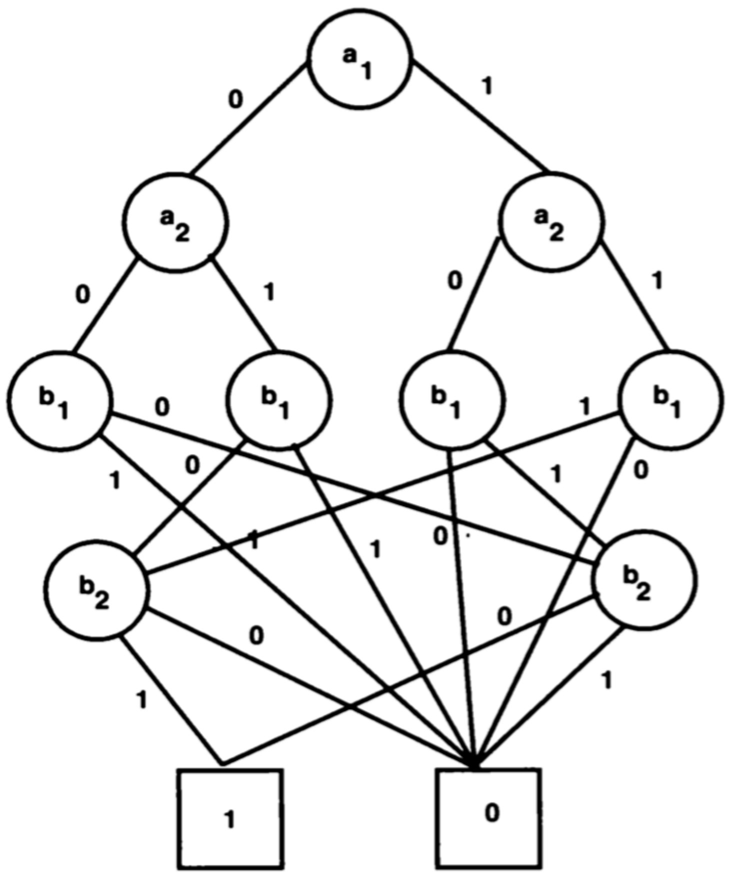


Example: two bit comparator:
$f(a_1, a_2, b_1, b_2) \; = \; a_1 \leftrightarrow b_1 \wedge a_2 \leftrightarrow b_2$

once the first two digits are different, the value of the function is 0, regardless of a_2 and b_2.

Binary Decision Diagrams

The **Binary Decision Diagram** is **highly dependent** from the **order of variables**.



Example: two bit comparator:
$f(a_1, a_2, b_1, b_2) = a_1 \leftrightarrow b_1 \wedge a_2 \leftrightarrow b_2$
with the order $a_1 < a_2 < b_1 < b_2$

In the n bit comparator, using this order, the BDD has $3 \cdot 2^n - 1$
➡ EXPONENTIAL in n

With the order $a_1 < b_1 < \ldots < a_i < b_i < \ldots < a_n < b_n$ it has $3n+2$ nodes
➡ LINEAR in n

Canonical Forms

For several applications it is desirable to have a **canonical form** for BDDs.

Definition: Two BDDs B_1 and B_2 are **isomorphic** if there exists a bijection $h: V(B_1) \to V(B_2)$ that maps terminals to terminals and non-terminals to non-terminals such that: value(v)=value($h(v)$), h(low(v))=low($h(v)$), and h(high(v))=high($h(v)$).

Canonical representations can be obtained by:

1. **Imposing a total ordering on variables**: if u has a successor v, then var(u)<var(v)
2. **Avoid isomorphic sub-trees** or redundant vertices.

Condition **2**. can be obtained by using **a *reduce* function that is linear** in the size of the DAG.

Reduction to a canonical form

Remove duplicate terminals: eliminate **all but one terminal** vertex with a given label and redirect all arcs to eliminated vertices to the remaining one.

Remove duplicate nonterminals: If there exist two nonterminal u and v such that var(u)=var(v), low(u)=low(v), and high(u)=high(v) then **eliminate one of them** and redirect all incoming arcs to the remaining vertex.

Remove redundant tests: if low(u)=high(u) then eliminate the vertex u and redirect incoming arcs to low(u) [=high(u)].

This procedure **can be implemented bottom-up**, linear in the size of the BDD. Some consequences:

- ❖ **Checking equivalence** of boolean functions corresponds to **checking OBDDs isomorphisms**.
- ❖ SAT on OBDDs is just checking **if it is (not) the trivial OBDD**.

Negative Results about OBDDs

- ❖ It is **NP-complete** to find the **variable ordering** for a boolean function $f(x_1, \ldots, x_n)$ that makes the size of the **OBDD** representing f **optimal**.

- ❖ There are boolean functions $f(x_1, \ldots, x_n)$ such that the size of the **OBDD is exponential in n for all variable orders**. For example, the **mid bit of the n bit product**.

However, several **heuristics give good results**: for example, **related variables should be close in the ordering** (as in the n bit comparator example)

OBDD packages usually use **dynamic reordering** when heuristic seems to fail.

Logical operators using OBDDs

Restriction: $f|_{x_i=b}(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i\text{-}1}, b, x_{i+1}, \ldots x_n)$

Just find the node w such that var(w)= $x_{i\text{-}1}$, remove such node and replace all arcs $v \to w$ with $v \to$ low(w) if b=0 and with $v\to$ high(w) if b=1. The resulting OBDD may not be in canonical form, so **reduce must be applied** to it.

Using restriction, one can easily compute **Shannon expansion**:

$$f = (\neg x \wedge f|_{x=0}) \vee (x \wedge f|_{x=1})$$

Then, **binary operations can be recursively computed stemming from Shannon expansion**.

To simplify notations:

* v, v' are the root of OBDDs representing f and f'.
* x, x' are var(v) and var(v')

Logical operators using OBDDs

Let $\ast$ be an arbitrary two-argument boolean connective.

- If v and v' are both terminals, $f \ast f' = \text{value}(v) \ast \text{value}(v')$
- If $x = x'$ then using Shannon expansion, we have:

$$f \ast f' = (\neg x \wedge (f|_{x=0} \ast f'|_{x=0})) \vee (x \wedge (f|_{x=1} \ast f'|_{x=1}))$$

  The root of the OBDD is w such that $\text{var}(w) = x$ and $\text{low}(w) = (f|_{x=0} \ast f'|_{x=0})$ and $\text{high}(w) = (f|_{x=1} \ast f'|_{x=1})$

- If $x < x'$ then $f'|_{x=0} = f'|_{x=1} = f'$ since f' does not depend on x. In this case Shannon expansion simplifies to:

$$f \ast f' = (\neg x \wedge (f|_{x=0} \ast f')) \vee (x \wedge (f|_{x=1} \ast f'))$$

  and the OBDD is computed as before with x as root.

- If $x' < x$: symmetric to the previous case.

Since **each problem generates two subproblems**, to prevent exponential behaviour **dynamic programming** must be used! $\mathcal{O}(|f| \cdot |f'|)$.

Some optimizations

Negation can be computed just flipping terminal nodes.

A single **multi-rooted DAG** can be used to represent **several boolean functions** that share subgraphs. In this case, f and f' are the same if they have the same root!

Adding label to edges to denote boolean negation. In this case f and $\neg f$ can be represented by a single OBDD.

OBDDs can be viewed as a DFA. A n-ary boolean function can be seen as the set of string x in $\{0, 1\}^n$ such that $f(x) = 1$. **The minimal automata that accept this language** is an alternative canonical form for f.

Standard boolean connectives can be seen as operation between languages (for example $\wedge$ is intersection etc.)

Lesson 7b:

Using OBDDs to represent Kripke Structures

Characteristic functions

Let Q be a n-ary relation over {0,1}. Q can be represented by its **characteristic function**: $(x_1, \ldots, x_n) \in Q$ iff $f_Q\,(x_1, \ldots, x_n)=1$.

Let Q be a n-ary relation over a domain D. For simplicity we assume that $|D|=2^m$ for some $m>1$.

Elements of ***D*** can be encoded **using a bijection** $\phi : \{0,1\}^m \rightarrow D$.

Q can be represented by a $m \times n$-ary boolean characteristic function according to

$$f_Q(d_1, \ldots, d_n)=1 \textbf{ iff } Q(\phi(d_1), \ldots, \phi(d_n))$$

where $d_1, \ldots, d_n$ are vectors of length m of boolean variables.

Sets can be viewed simply as **unary relations**.

Kripke structures as OBDDs

Let $\mathcal{M}$=(S, R, L) be a Kripke structure.

An encoding function ϕ (bijection) encodes states of S.

S is the constant function 1 on $\{0,1\}^m$. **Subsets of S** are represented by **their characteristic functions**.

R is represented by a characteristic function $f_R : \{0,1\}^{2m} \rightarrow \{0,1\}$.

The mapping L is represented by **an OBDD f_p for each atomic proposition p**, such that f_p is the characteristic function of the set $\{ s \in S \mid p \in L(s)\}$.

Along the same lines, one can represent the set **of initial states I** or a set of (unconditionally) **fairness constraints** $F=\{P_1, \dots, P_k\}$.

$\mathcal{M}$ **is not explicitely generated** and then converted into its OBDD representation, but rather OBDDs are generated starting from a high level description of $\mathcal{M}$ (for example programs!).

Lesson 7c:

Fixpoints

Classical FixPoint Theorem

Fixpoints have a relevant role in Logic and Theoretical Computer Science. For example, they are used to define semantics of Programming Languages (recursive definitions) or equivalences (bisimulation).

Given an function $T: L \rightarrow L$, $x \in L$ is a **fixpoint** of T if $T(x)=x$. μT (resp. υT) denotes the **minimum** (resp. **maximum**) fixpoint of T.

Definition: A **complete lattice** $(L, \leq)$ is a partially ordered set, such that each subset $A \subseteq L$ has a greatest lower bound $\sqcap A$ (**glb** or **inf** standing for infimum) and a least upper bound $\sqcup A$ (**lub** or **sup** standing for supremum).

$$sup\ A = min\{\ x \mid \forall a \in A . x \geq a\ \}\ \text{ and }\ inf\ A = max\ \{\ x \mid \forall a \in A . x \leq a\ \}$$

Example: Given a set S, $(\mathcal{P}(S), \subseteq)$ is a complete lattice where if $A \subseteq P(S)$ then $inf\ A = \cap_{a \in A}\ a$ and $sup\ A = \cup_{a \in A}\ a$.

Observation: a complete lattice L has always a minimum, that is $\perp = \inf \varnothing$ and a maximum $\top = \sup L$. This implies that $L \neq \varnothing$.

Knarster-Tarski Theorem I

Def: T: $L \rightarrow L$ is **monotonic** if $x \leq y$ implies $T(x) \leq T(y)$.

Theorem: [KNARSTER-TARSKI] If L is a complete lattice and $T : L \rightarrow L$ is *monotonic,* then T has a minimum fixpoint μT and a maximum fixpoint υT. Moreover:

$$\mu T = \inf \{x \mid T(x) \leq x\} \text{ and } \upsilon T = \sup \{x \mid x \leq T(x)\}$$

Proof: Let $G=\{x \mid T(x) \leq x\}$ and $g=\inf G$. **We first show that $g \in G$**. $g \leq x$, $\forall x \in G$. By monocity of T, $T(g) \leq T(x) \leq x$. But, being an inf, g is the maximum lower bound, therefore $T(g) \leq g$. Hence $g \in G$.

From $T(g) \leq g$ we have $T(T(g)) \leq T(g)$. But this implies that $T(g) \in G$. Therefore $g \leq T(g)$. And therefore $g = T(g)$, that is **g is a fixpoint**.

Finally, let $g'=\inf\{x \mid T(x)=x\}$. Since g is a fixpoint $g' \leq g$. But since $\{x \mid T(x)=x\} \subseteq \{x \mid T(x) \leq x\}$, we have also $g \leq g'$. **Hence g is the minimum fixpoint of T**. A dual argument works for υT. □

Knarster-Tarski Theorem II

Definition: Let $T : L \rightarrow L$, we define **transfinite powers** of T as follows: $T^0 = \perp$. $T^{\alpha+1} = T(T^\alpha)$ and if λ is a limit ordinal, $T^\lambda = \sup_{\alpha<\lambda} T^\alpha$. Dually, we define the **transfinite downward powers** as follows: $T_0 = \top$. $T_{\alpha+1} = T(T_\alpha)$ and if λ is a limit ordinal, $T_\lambda = \inf_{\alpha<\lambda} T_\alpha$.

Theorem: If L is a complete lattice and $T : L \rightarrow L$ is *monotonic*, then $T^\alpha \leq \mu T$ and $\upsilon T \leq T_\alpha$. Moreover, there exist two ordinals β_1 and β_2 such that $\mu T = T^\alpha$ for all $\alpha \geq \beta_1$ and $\upsilon T = T_\alpha$ for all $\alpha \geq \beta_2$.

Proof: **(1) $T^\alpha \leq \mu T$**. Trivially, $T^0 = \perp \leq \mu T$. If $T^\alpha \leq \mu T$, by monotonicity of T we have $T(T^\alpha) \leq T(\mu T)$ that means $T^{\alpha+1} \leq \mu T$. $T^\lambda = \sup_{\alpha<\lambda} T^\alpha$ and all $T^\alpha \leq \mu T$ (by transfinite inductive hypothesis) we have the thesis because limits preserve ≤.

(2) $T^\alpha \leq T^{\alpha+1}$. Trivially, $T^0 = \perp \leq T^1$. Assuming $T^\alpha \leq T^{\alpha+1}$, by monotonicity, we have $T(T^\alpha) \leq T(T^{\alpha+1})$, hence $T^{\alpha+1} \leq T^{\alpha+2}$.

(3) If $\alpha \leq \beta$ then $T^\alpha \leq T^\beta$. The property is trivial using (**2**) and observing again that limits preserve ≤ (**for limit ordinals**).

Knarster-Tarskij Theorem II

Proof: (cntnd)

(**4**) If $\alpha \leq \beta$ and $T^\alpha = T^\beta$ then $T^\alpha = \mu T$. If $T^\alpha = T^\beta$ and since by (**2**) the sequence T^α is ascending ordered, all $T^\alpha = T^\gamma = T^\beta$ for all γ such that $\alpha \leq \gamma \leq \beta$. But this implies that $T^\alpha = T(T^\alpha)$, that is T^α is a fixpoint and by (**1**) T^α is μT.

(**5**) There exists α such that $T^\alpha = \mu T$. By contradiction, assume that this is not the case. By (**2**) and (**4**) this implies that the sequence of powers of T is **strictly** ordered and contains **distinct** elements, defining an injection from the set of ordinals into L. Absurd. (for any set L, there exists an ordinal of "bigger" cardinality).

All these reasoning **works dually** for downward powers and for υT (**Exercise** 🙃). □

Set Operators

Given a set S, the powerset $\mathcal{P}(S)$ is a complete lattice, ordered by $\subseteq$ (as expected *sup* is $\cup$ and *inf* is $\cap$).

A function $\tau : \mathcal{P}(S) \rightarrow \mathcal{P}(S)$ is called a **predicate transformer**.

- ❖ τ is **monotonic** if $Q \subseteq Q'$ implies $\tau(Q) \subseteq \tau(Q')$
- ❖ τ is **finitary** if $x \in \tau(Q)$ if and only if $\exists\, Q_0 \subseteq Q$, Q_0 finite, such that $x \in \tau(Q_0)$
- ❖ τ is **$\cup$-continuous** if for a sequence $Q_1 \subseteq Q_2 \subseteq Q_3 \subseteq \ldots$ we have that $\cup_i \tau(Q_i) = \tau(\cup_i Q_i)$
- ❖ τ is **$\cap$-continuous** if for a sequence $Q_1 \supseteq Q_2 \supseteq Q_3 \supseteq \ldots$ we have that $\cap_i \tau(Q_i) = \tau(\cap_i Q_i)$

Finitary Operators

Lemma: If τ is monotone and $\{x_j\}_{j\in J}$ is an ascending chain then $\sup_{j\in J} \tau(x_j) \subseteq \tau(\sup_{j\in J} x_j)$

Proof: for all $i\in J$, we have: $x_i \subseteq \sup_{j\in J} x_j$. By monotonicity of τ we have that $\tau(x_i) \subseteq \tau(\sup_{j\in J} x_j)$, that is, $\tau(\sup_{j\in J} x_j)$ is an upper bound of the chain $\{\tau(x_i)\}_{i\in J}$. Being sup $\{\tau(x_i)\}_{i\in J}$ the minimum of upper bounds, we get the thesis. □

Lemma: If τ is monotone and finitary and $\{x_j\}_{j\in J}$ is an ascending chain then $\sup_{j\in J} \tau(x_j) \supseteq \tau(\sup_{j\in J} x_j)$.

Proof: Let $y \in \tau(\sup_{j\in J} x_j)$. There exists a finite set $z_0 \subseteq \sup_{j\in J} x_j$ such that $y \in \tau(z_0)$. Since $\{x_j\}_{j\in J}$ is a chain, there exists k such that $z_0 \subseteq x_k \subseteq \sup_{j\in J} x_j$. Therefore $y \in \sup_{j\in J} \tau(x_j)$. □

Theorem: $\tau : \mathcal{P}(S) \to \mathcal{P}(S)$ is $\cup$-continuous if and only if it τ is monotonic and finitary.

Kleene Fixpoint Theorem

Theorem: [KLEENE] If τ is $\cup$-continuous then $\mu\tau = \cup_{i\in N} \tau^i(\varnothing) = \tau^\omega$.

Proof:

$\tau^\omega = \sup \{\tau^i \mid i < \omega\}$ (def. of τ^ω)
$= \sup \{\tau(\tau^i) \mid i < \omega\}$ (prestige ☺)
$= \tau(\sup \{\tau^i \mid i < \omega\})$ (continuity)
$= \tau(\tau^\omega)$ (def. of τ^ω) □

Question: Which is the dual notion of finitary? That ensure that for a monotonic operator τ we have $\upsilon\tau = \cap_{i\in N} \tau_i(\varnothing) = \tau_\omega$?

For the purposes of Model Checking… **all this is a bit too much**. In a **finite *S***, the monotonic chain $\tau^i(\varnothing)$ **reaches the least fixpoint $\mu\tau$ after a finite number of steps** and $\tau_i(S)$ reaches the greatest fixpoint $\upsilon\tau$ after a finite number of steps!

On a finite set, **a monotonic operator is necessarily finitary**!

Finite Fixpoint properties

Proofs of the following three lemmas can be easily obtained specializing proofs of the general case to the finite case.

Lemma: If S is finite and τ is monotonic, then τ is also $\cup$-continuous and $\cap$ -continuous.

Lemma: If S is finite and τ is monotonic, then there exist two integers p, q such that $\tau^p(\varnothing) = \tau^i(\varnothing)$ and $\tau_q(S) = \tau_j(S)$ for all $i \geq p$ and for all $j \geq q$.

Lemma: If S is finite and τ is monotonic, then there exist two integers p and q such that $\tau^p(\varnothing) = \mu\tau$ and $\tau_q(S) = \upsilon\tau$.

Computing Finite Fixpoints - Lfp

Using characteristic functions for representing sets, the **constant FALSE is the emptyset**. The invariant of the following program is **INV** $\equiv Q' \subseteq \mu\tau \wedge Q'=\tau(Q)$.

The computed sequence is **strictly increasing** (wrt $\subseteq$) and hence **$|S|$ is an upperbound** to the number of iterations.

It terminates because the guard of the **while** implies $Q \neq Q'$.

```
def leastFixedPoint(Tau: PredicateTransformer):
    Q = FALSE;
    Q' = Tau(Q);
    while (Q ≠ Q') do
        Q = Q';
        Q'=Tau(Q');
    return Q;
```

Computing Finite Fixpoints - Gfp

Using characteristic functions for representing sets, the **constant TRUE is the universe set S**. The invariant of the following program is **INV** $\equiv Q' \supseteq \nu\tau \wedge Q'=\tau(Q)$.

The computed sequence is **strictly decreasing** (wrt $\subseteq$) and hence **$|S|$ is an upperbound** to the number of iterations.

It terminates because the guard of the **while** implies $Q \neq Q'$.

```
def greatestFixedPoint(Tau: PredicateTransformer):
    Q = TRUE;
    Q′ = Tau(Q);
    while (Q ≠ Q′) do
        Q = Q′;
        Q′=Tau(Q′);
    return Q;
```

Lesson 7d:

Symbolic CTL model checking

FixPoints and CTL

Identifying a formula f with the set of states $Sat(f)=\{s \mid \mathcal{M}, s \models f\}$, we can characterize temporal operators as predicate transformers and their semantics as fixpoints of such operators. Intuitively:

- **eventualities** (**F** and **U**) are **least fixpoints** and
- properties that **hold forever** (**R** and **G**) are **greatest fixpoints**.

They use **expansion laws** for **F**, **U**, **R**, and **G** using **X**.

$$\mathbf{AF}\, f = \mu Z.\, f \vee \mathbf{AX}\, Z$$
$$\mathbf{EF}\, f = \mu Z.\, f \vee \mathbf{EX}\, Z$$
$$\mathbf{A}[f\, \mathbf{U}\, g] = \mu Z.\, g \vee (f \wedge \mathbf{AX}\, Z)$$
$$\mathbf{E}[f\, \mathbf{U}\, g] = \mu Z.\, g \vee (f \wedge \mathbf{AX}\, Z)$$
$$\mathbf{EG}\, f = \upsilon Z.\, f \wedge \mathbf{EX}\, Z$$
$$\mathbf{AG}\, f = \upsilon Z.\, f \wedge \mathbf{AX}\, Z$$
$$\mathbf{A}[f\, \mathbf{R}\, g] = \upsilon Z.\, g \wedge (f \vee \mathbf{AX}\, Z)$$
$$\mathbf{E}\, [f\, \mathbf{R}\, g] = \upsilon Z.\, g \wedge (f \vee \mathbf{EX}\, Z)$$

EG *as a greatest fixpoint*

Lemma: The predicate transformer $\tau(Z)=f \wedge \mathbf{EX}\ Z$ is monotonic.

Proof: Let $Z_1 \subseteq Z_2$. Let $s \in \tau(Z_1)$. Then $s \vDash f$ and there exists a successor s' of s, such that $s' \in Z_1$. But then $s' \in Z_2$ and this implies that also $s \in \tau(Z_2)$. □

Lemma: **EG** f is a fixpoint of the predicate transformer $\tau(Z)=f \wedge \mathbf{EX}\ Z$.

Proof: Suppose $s_0 \vDash \mathbf{EG}\ f$. Then there exists an infinite path $\pi=s_0 s_1 s_2\ldots$ such that for all k, $s_k \vDash f$. This implies that $s_0 \vDash \mathbf{EG}\ f$ and $s_1 \vDash \mathbf{EG}\ f$, that is $s_0 \vDash \mathbf{EX}\ \mathbf{EG}\ f$. Thus Sat(**EG** f) $\subseteq$ Sat($f \wedge$ **EX EG** f). Clearly Sat($f \wedge$ **EX EG** f) $\subseteq$ Sat(**EG** f) and hence they are equal. □

Lemma: **EG** f is the greatest fixpoint of the predicate transformer $\tau(Z)=f \wedge \mathbf{EX}\ Z$.

Proof: Being a fixpoint, $\mathbf{EG}\ f \subseteq \nu Z.\ f \wedge \mathbf{EX}\ Z = \cap_k \tau_k(S)$ for some k. Let $s \in \cap_k \tau_k(S)$. Since it is a fixpoint, $s \in \tau(\cap_k \tau_k(S))$. This implies that $s \vDash f$ and $\exists s'.R(s, s')$ and $s' \in \cap_k \tau_k(S)$. Applying this argument to s' we find an infinite sequence of states that belong to $\cap_k \tau_k(S)$ starting in s and thus $s \in \mathbf{EG}\ f$. □

EU *as a least fixpoint*

Lemma: The operator $\tau(Z) = h \vee (g \wedge \mathbf{EX}\ Z)$ is monotonic.

Proof: Let $Z_1 \subseteq Z_2$. Let $s \in \tau(Z_1)$. If $s \models h$ then $s \in \tau(Z_2)$. Otherwise $s \models g$ and there exists a successor s' of s such that $s' \in Z_1$. Since $s' \in Z_2$, we have also that $s \in \tau(Z_2)$. □

Lemma: $\mathbf{E}\ [g\ \mathbf{U}\ h]$ is a fixpoint of $\tau(Z) = h \vee (g \wedge \mathbf{EX}\ Z)$.

Proof: We have to show that $\mathrm{Sat}(\mathbf{E}\ [g\ \mathbf{U}\ h]) = \mathrm{Sat}(g \vee (h \wedge \mathbf{EX}\ \mathbf{E}\ [g\ \mathbf{U}\ h]))$. $s_0 \in \mathrm{Sat}(\mathbf{E}\ [g\ \mathbf{U}\ h])$ if and only if there exists a path of length $k \geq 0$ such that $s_k \models h$ and $s_i \models g$ for $0 \leq i < k$ if and only if $s_0 \in \mathrm{Sat}(h \vee (g \wedge \mathbf{EX}\ \mathbf{E}\ [g\ \mathbf{U}\ h])$. □

Lemma: $\mathbf{E}\ [g\ \mathbf{U}\ h]$ is the least fixpoint of the predicate transformer $\tau(Z) = h \vee (g \wedge \mathbf{EX}\ Z)$.

Proof: Being a fixpoint, $\cup_i \tau_i(\varnothing) = \mu Z.\ h \vee (g \wedge \mathbf{EX}\ Z) \subseteq \mathbf{E}\ [g\ \mathbf{U}\ h]$. Let $s \in \mathbf{E}\ [g\ \mathbf{U}\ h]$. If $s \models h$ then $s \in \tau(Z)$ for any Z and so $s \in \cup_i \tau_i(\varnothing)$. Otherwise $s \models g$ and there exists a path of length $k \geq 0$ such that $s_k \models h$ and $s_j \models g$ for $0 \leq j < k$. It is easy to see that $s \in \tau_k(\varnothing)$ by induction on k. □

Exercises on fixpoints

If you want to understand deeply fixpoints and semantics of temporal operators…try to solve the following exercises:

- **Exercise 1**: Which is the least fixpoint of the operator $\tau(Z)=f \wedge \mathbf{EX}\ Z$?
- **Exercise 2**: Find a Kripke structure in which the greatest fixpoint of the operator $\tau(Z) = h \vee (g \wedge \mathbf{EX}\ Z)$ contains at least a state s such that $s \nvDash \mathbf{E}\,[g\ \mathbf{U}\ h]$.
- **Exercise 2**: Find a Kripke structure in which the leatest fixpoint of the operator $\tau(Z) = g \wedge \mathbf{EX}\ Z)$ does not contain some state s such that $s \nvDash \mathbf{E}\ \mathbf{G}\ g$.
- **Exercise 3**: Find a monotonic but not continuous operator (of course you must deal with infinite sets!)
- **Exercise 4**: Which is the dual notion of finitary? That ensure that for a monotonic operator τ we have $\upsilon\tau = \cap_{i \in \mathrm{N}} \tau_i(\varnothing) = \tau_\omega$?

CTL model checking

The problem is to find three functions such that:

$$check(\mathbf{EX}\ f) = checkEX(check(f))$$

$$check(\mathbf{E}[f\ \mathbf{U}\ g]) = checkEU(check(f), Check(g))$$

$$check(\mathbf{EG}\ f) = checkEG(check(f))$$

Observe that the parameter of *check* is a CTL formula φ, its result is an OBDD representing the set of states satisfying φ.

The parameters of *checkEX*, *checkEU,* and *checkEG* **are OBDDs**.

CTL model checking

❖ *checkEX*($f(v)$) is strighforward. It is equivalent to $\exists v'. f(v') \wedge R(v,v')$.

❖ *checkEU*($f_1(v), f_2(v)$) is based on the characterization of **EU** as the least fixpoint of the predicate transformer $\mu Z. f_2(v) \vee (f_1(v) \wedge$ **EX** $Z)$.

It is computed a converging sequence of states $Q_1, \ldots, Q_i, \ldots$ Having the OBDD for Q_i and those for $f_1(v)$ and $f_2(v)$ one can easily compute those for Q_{i+1}. Observe that checking $Q_i = Q_{i+1}$ is strighforward.

❖ *checkEG*($f(v)$) is based on the characterization of **EG** as the greatest fixpoint of the predicate transformer $\upsilon Z. f_1(v) \wedge$ **EX** Z.

Quantified Boolean Formulas

In the previous slides we use formulas such as: $\exists v'.f(v') \wedge R(v,v')$.

They are **quantified boolean formulas**: They are equivalent to propositional formulas, but they allow a more succint representation.

Semantics:

- $\sigma \models \exists v f$ iff $\sigma\langle v \leftarrow 0\rangle \models f$ or $\sigma\langle v \leftarrow 1\rangle \models f$, and
- $\sigma \models \forall v f$ iff $\sigma\langle v \leftarrow 0\rangle \models f$ and $\sigma\langle v \leftarrow 1\rangle \models f$.

They can be represented as OBDD using restriction:

- $\exists x f = f|_{x\leftarrow 0} \vee f|_{x\leftarrow 1}$
- $\forall x f = f|_{x\leftarrow 0} \wedge f|_{x\leftarrow 1}$

That's all Folks!

Thanks for your attention…

…Questions?