System Specification, Verification and Synthesis (SSVS) – CS 4830/7485, Fall 2019

13: Formal Verification: Symbolic Methods

Stavros Tripakis

Northeastern University
Khoury College of Computer Sciences

SYMBOLIC METHODS

Symbolic Methods: Why?

Motivation: attack the state explosion problem.

A seminal paper: *Symbolic model checking: 10^{20} states and beyond.* [Burch et al., 1990].

10^{20} is less than 2^{67}, so far from adequate for real-world systems.

Nevertheless: a great leap forward at that time.

Ken McMillan

Symbolic Representation of State Spaces

Key idea:

Instead of reasoning about individual states, reason about **sets** *of states.*

How do we represent a set of states?

Symbolic representation:

Set = *predicate.*

Set of states = *predicate on state variables.*

Symbolic Representation of Sets of States

Examples:

1. Assume 3 state variables, p, q, r, of type boolean.

$$S_1 : \quad p \vee q$$

Symbolic Representation of Sets of States

Examples:

1. Assume 3 state variables, p, q, r, of type boolean.

$$S_1 : \qquad p \vee q = \{p\overline{q}r, p\overline{q}\overline{r}, \overline{p}qr, \overline{p}q\overline{r}, pqr, pq\overline{r}\}$$

Symbolic Representation of Sets of States

Examples:

1. Assume 3 state variables, p, q, r, of type boolean.

$$S_1 : \qquad p \vee q = \{p\overline{q}r, p\overline{qr}, \overline{p}qr, \overline{p}q\overline{r}, pqr, pq\overline{r}\}$$

2. Assume 3 state variables, x, i, b, of types real, integer, boolean.

$$S_2 : \qquad x > 0 \wedge (b \rightarrow i \geq 0)$$

How many states are in S_2?

Symbolic Representation of Transition Relations

Symbolic Representation of Transition Relations

Key idea:

> *Use a predicate on* **two copies** *of the state variables: unprimed (current state) + primed (next state).*

If $\vec{x}$ is the vector of state variables, then the transition relation R is a predicate on $\vec{x}$ and $\vec{x}'$:

$$R(\vec{x}, \vec{x}')$$

e.g., for three state variables, x, i, b:

$$R(x, i, b, x', i', b')$$

Symbolic Representation of Transition Relations

Examples:

1. Assume one state variable, p, of type boolean.

$$R_1 : \qquad (p \rightarrow \neg p') \land (\neg p \rightarrow p')$$

Which transition relation does this represent? Is it a relation or a function (deterministic)?

Symbolic Representation of Transition Relations

Examples:

1. Assume one state variable, p, of type boolean.

$$R_1 : \qquad (p \rightarrow \neg p') \wedge (\neg p \rightarrow p')$$

   Which transition relation does this represent? Is it a relation or a function (deterministic)?

2. Assume one state variable, n, of type integer.

$$R_2 : \qquad n' = n + 1 \vee n' = n$$

   Which transition relation does this represent? Is it a relation or a function (deterministic)?

Symbolic Representation of Kripke Structures

Kripke structure:

$$(P, S, S_0, L, R)$$

Symbolic representation:

$$(P, \mathit{Init}, \mathit{Trans})$$

where

- $P = \{x_1, x_2, ..., x_n\}$: set of (boolean) state variables, also taken to be the atomic propositions.[1]
- Predicate $\mathit{Init}(\vec{x})$ on vector $\vec{x} = (x_1, ..., x_n)$ represents the set S_0 of initial states.
- Predicate $\mathit{Trans}(\vec{x}, \vec{x}')$ represents the transition relation R.

[1] this is done for simplicity, the two could be separated

Symbolic Representation of Kripke Structures

Kripke structure:

$$(P, S, S_0, L, R)$$

Symbolic representation:

$$(P, \mathit{Init}, \mathit{Trans})$$

where

- $P = \{x_1, x_2, ..., x_n\}$: set of (boolean) state variables, also taken to be the atomic propositions.[1]
- Predicate $\mathit{Init}(\vec{x})$ on vector $\vec{x} = (x_1, ..., x_n)$ represents the set S_0 of initial states.
- Predicate $\mathit{Trans}(\vec{x}, \vec{x}')$ represents the transition relation R.

Basis of the language of SMV/NuSMV/NuXMV.

[1] this is done for simplicity, the two could be separated

Example: NuSMV model

```
MODULE inverter(input)
VAR
  output : boolean;
INIT
  output = FALSE
TRANS
  next(output) = !input | next(output) = output
```

What is the Kripke structure defined by this NuSMV program?

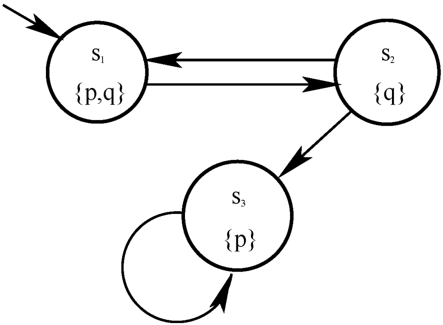
Example: NuSMV model

```
MODULE inverter(input)
VAR
  output : boolean;
INIT
  output = FALSE
TRANS
  next(output) = !input | next(output) = output
```

What is the Kripke structure defined by this NuSMV program?

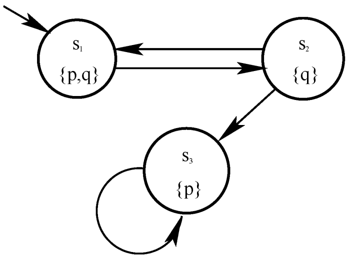
What about P and L?

Example: Kripke Structure



Represent this symbolically.

A subtlety



Transition relation – symbolic representation 1:

$$(s = s_1 \rightarrow s' = s_2) \wedge (s = s_2 \rightarrow (s' = s_1 \vee s' = s_3)) \wedge (s = s_3 \rightarrow s' = s_3)$$

Transition relation – symbolic representation 2:

$$(s = s_1 \wedge s' = s_2) \vee (s = s_2 \wedge (s' = s_1 \vee s' = s_3)) \vee (s = s_3 \wedge s' = s_3)$$

Which one is the right one?

A subtlety: a bit of propositional logic

Consider the two formulas:

$$\phi_1 = (a \to b) \wedge (c \to d)$$

$$\phi_2 = (a \wedge b) \vee (c \wedge d)$$

- Generally, they are not equivalent:
  - $\phi_1 \not\Rightarrow \phi_2$, e.g., when $a = c = 0$.
  - $\phi_2 \not\Rightarrow \phi_1$, e.g., when $a = b = c = 1$, $d = 0$.
- BUT:
  - $\phi_1 \Rightarrow \phi_2$ when $a \vee c$ is valid.
  - ϕ_1 and ϕ_2 are equivalent when both $a \vee c$ and $a \oplus c$ (a XOR c) are valid.

A subtlety: a bit of propositional logic

Consider the two formulas:

$$\phi_1 \quad = \quad (a \rightarrow b) \wedge (c \rightarrow d)$$

$$\phi_2 \quad = \quad (a \wedge b) \vee (c \wedge d)$$

- Generally, they are not equivalent:
  - $\phi_1 \not\Rightarrow \phi_2$, e.g., when $a = c = 0$.
  - $\phi_2 \not\Rightarrow \phi_1$, e.g., when $a = b = c = 1$, $d = 0$.
- BUT:
  - $\phi_1 \Rightarrow \phi_2$ when $a \vee c$ is valid.
  - ϕ_1 and ϕ_2 are equivalent when both $a \vee c$ and $a \oplus c$ (a XOR c) are valid.

So, if you cover all the cases for the current state s, and the cases are all mutually exclusive, both forms are equivalent.

SYMBOLIC REACHABILITY ANALYSIS

Recall: Symbolic Representation of Kripke Structures

$$(P, Init, Trans)$$

where

- $P = \{x_1, x_2, ..., x_n\}$: set of boolean state variables, also taken to be the atomic propositions.
- Predicate $Init(\vec{x})$ on vector $\vec{x} = (x_1, ..., x_n)$ represents the set S_0 of initial states.
- Predicate $Trans(\vec{x}, \vec{x}')$ represents the transition relation R.

Recall: Symbolic Representation

- Set of states = predicate $\phi(\vec{x})$ on vector of state variables $\vec{x}$. E.g.:
  - $Init(x, y, z) : x \wedge \neg y$
  - $Bad(x_1, x_2) : x_1 = crit \wedge x_2 = crit$
- Transition relation = predicate $Trans(\vec{x}, \vec{x}')$ on state variables and next-state variables. E.g.:
  - $Trans(x, y, x', y') : x' = x + 1 \wedge (y' = 0 \vee y' = 1)$

Recall: Symbolic Representation

- Set of states = predicate $\phi(\vec{x})$ on vector of state variables $\vec{x}$. E.g.:
  - $Init(x, y, z) \,:\, x \wedge \neg y$
  - $Bad(x_1, x_2) \,:\, x_1 = crit \wedge x_2 = crit$
- Transition relation = predicate $Trans(\vec{x}, \vec{x}')$ on state variables and next-state variables. E.g.:
  - $Trans(x, y, x', y') : x' = x + 1 \wedge (y' = 0 \vee y' = 1)$
- How do we perform set-theoretic operations with predicates?
  - Union of two sets represented by ϕ_1 and ϕ_2:

Recall: Symbolic Representation

- Set of states = predicate $\phi(\vec{x})$ on vector of state variables $\vec{x}$. E.g.:
  - $Init(x, y, z) : x \wedge \neg y$
  - $Bad(x_1, x_2) : x_1 = crit \wedge x_2 = crit$
- Transition relation = predicate $Trans(\vec{x}, \vec{x}')$ on state variables and next-state variables. E.g.:
  - $Trans(x, y, x', y') : x' = x + 1 \wedge (y' = 0 \vee y' = 1)$
- How do we perform set-theoretic operations with predicates?
  - Union of two sets represented by ϕ_1 and ϕ_2: $\phi_1 \vee \phi_2$.
  - Intersection of two sets represented by ϕ_1 and ϕ_2:

Recall: Symbolic Representation

- Set of states = predicate $\phi(\vec{x})$ on vector of state variables $\vec{x}$. E.g.:
  - $Init(x, y, z) : x \wedge \neg y$
  - $Bad(x_1, x_2) : x_1 = crit \wedge x_2 = crit$
- Transition relation = predicate $Trans(\vec{x}, \vec{x}')$ on state variables and next-state variables. E.g.:
  - $Trans(x, y, x', y') : x' = x + 1 \wedge (y' = 0 \vee y' = 1)$
- How do we perform set-theoretic operations with predicates?
  - Union of two sets represented by ϕ_1 and ϕ_2: $\phi_1 \vee \phi_2$.
  - Intersection of two sets represented by ϕ_1 and ϕ_2: $\phi_1 \wedge \phi_2$.
  - Complement of a set represented by ϕ:

Recall: Symbolic Representation

- Set of states = predicate $\phi(\vec{x})$ on vector of state variables $\vec{x}$. E.g.:
  - $Init(x, y, z) : x \wedge \neg y$
  - $Bad(x_1, x_2) : x_1 = crit \wedge x_2 = crit$
- Transition relation = predicate $Trans(\vec{x}, \vec{x}')$ on state variables and next-state variables. E.g.:
  - $Trans(x, y, x', y') : x' = x + 1 \wedge (y' = 0 \vee y' = 1)$
- How do we perform set-theoretic operations with predicates?
  - Union of two sets represented by ϕ_1 and ϕ_2: $\phi_1 \vee \phi_2$.
  - Intersection of two sets represented by ϕ_1 and ϕ_2: $\phi_1 \wedge \phi_2$.
  - Complement of a set represented by ϕ: $\neg\phi$.

Symbolic Reachability Analysis

Main idea:

- Start with set of initial states S_0.
- Compute $S_1 := S_0 \cup \{$all 1-step successors of $S_0\} = S_0 \cup \mathbf{post}(S_0)$.
- Compute $S_2 := S_1 \cup \{$all 1-step successors of $S_1\} = S_1 \cup \mathbf{post}(S_1)$.
- ...
- Until $S_{k+1} = S_k$.
- S_k contains all reachable states.

Computing Successors Symbolically

Given a set of states represented as a predicate $\phi(\vec{x})$.

We want to compute a new predicate ϕ', representing the set of **all 1-step successors** of states in $\phi(\vec{x})$.

Predicate Transformer

- Successors can be computed by a **predicate transformer** :

$$\mathbf{succ}\big(\phi(\vec{x})\big) := \big(\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')\big)[\vec{x}' \rightsquigarrow \vec{x}]$$

  - $\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')$: successors of states in ϕ
  - $[\vec{x}' \rightsquigarrow \vec{x}]$: renames variables so that resulting predicate is over current state variables

Predicate Transformer

- Successors can be computed by a **predicate transformer** :

$$\mathbf{succ}\big(\phi(\vec{x})\big) := \big(\exists \vec{x} : \phi(\vec{x}) \wedge Trans(\vec{x}, \vec{x}')\big)[\vec{x}' \rightsquigarrow \vec{x}]$$

  - $\exists \vec{x} : \phi(\vec{x}) \wedge Trans(\vec{x}, \vec{x}')$: successors of states in ϕ
  - $[\vec{x}' \rightsquigarrow \vec{x}]$: renames variables so that resulting predicate is over current state variables

Example:

$$\begin{array}{rcl} \phi & = & 0 \leq x \leq 5 \\ Trans & = & x \leq x' \leq x + 1 \\ \mathbf{succ}(\phi) & = & (\exists x : 0 \leq x \leq 5 \wedge x \leq x' \leq x + 1)[x' \rightsquigarrow x] \end{array}$$

Predicate Transformer

- Successors can be computed by a **predicate transformer** :

$$\mathbf{succ}\big(\phi(\vec{x})\big) := \big(\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')\big)[\vec{x}' \rightsquigarrow \vec{x}]$$

  - $\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')$: successors of states in ϕ
  - $[\vec{x}' \rightsquigarrow \vec{x}]$: renames variables so that resulting predicate is over current state variables

Example:

$$\begin{array}{rcl}
\phi & = & 0 \leq x \leq 5 \\
\mathit{Trans} & = & x \leq x' \leq x + 1 \\
\mathbf{succ}(\phi) & = & (\exists x : 0 \leq x \leq 5 \wedge x \leq x' \leq x + 1)[x' \rightsquigarrow x] \\
 & = & (\exists x : 0 \leq x \leq 5 \wedge 0 \leq x' \leq 5 + 1)[x' \rightsquigarrow x]
\end{array}$$

Predicate Transformer

- Successors can be computed by a **predicate transformer** :

$$\mathbf{succ}\big(\phi(\vec{x})\big) := \big(\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')\big)[\vec{x}' \rightsquigarrow \vec{x}]$$

  - $\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')$: successors of states in ϕ
  - $[\vec{x}' \rightsquigarrow \vec{x}]$: renames variables so that resulting predicate is over current state variables

Example:

$$\begin{aligned}
\phi &= 0 \leq x \leq 5 \\
\mathit{Trans} &= x \leq x' \leq x + 1 \\
\mathbf{succ}(\phi) &= (\exists x : 0 \leq x \leq 5 \wedge x \leq x' \leq x + 1)[x' \rightsquigarrow x] \\
&= (\exists x : 0 \leq x \leq 5 \wedge 0 \leq x' \leq 5 + 1)[x' \rightsquigarrow x] \\
&= (0 \leq x' \leq 6)[x' \rightsquigarrow x]
\end{aligned}$$

Predicate Transformer

- Successors can be computed by a **predicate transformer** :

$$\mathbf{succ}\big(\phi(\vec{x})\big) := \big(\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')\big)[\vec{x}' \rightsquigarrow \vec{x}]$$

  - $\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')$: successors of states in ϕ
  - $[\vec{x}' \rightsquigarrow \vec{x}]$: renames variables so that resulting predicate is over current state variables

Example:

$$\begin{aligned}
\phi &= 0 \le x \le 5 \\
\mathit{Trans} &= x \le x' \le x + 1 \\
\mathbf{succ}(\phi) &= (\exists x : 0 \le x \le 5 \wedge x \le x' \le x + 1)[x' \rightsquigarrow x] \\
&= (\exists x : 0 \le x \le 5 \wedge 0 \le x' \le 5 + 1)[x' \rightsquigarrow x] \\
&= (0 \le x' \le 6)[x' \rightsquigarrow x] \\
&= 0 \le x \le 6
\end{aligned}$$

Predicate Transformer

$$\mathbf{succ}\big(\phi(\vec{x})\big) := \big(\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')\big)[\vec{x}' \leadsto \vec{x}]$$

How to do quantifier elimination automatically?

In the case of propositional logic, quantifier elimination is simple. Suppose x is a boolean variable:

$$\exists x : \phi \quad \Leftrightarrow$$

Predicate Transformer

$$\mathbf{succ}\big(\phi(\vec{x})\big) := \big(\exists \vec{x} : \phi(\vec{x}) \wedge \mathit{Trans}(\vec{x}, \vec{x}')\big)[\vec{x}' \rightsquigarrow \vec{x}]$$

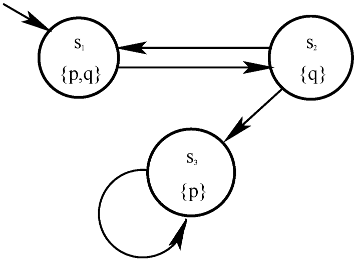
How to do quantifier elimination automatically?

In the case of propositional logic, quantifier elimination is simple. Suppose x is a boolean variable:

$$\exists x : \phi \quad \Leftrightarrow \quad \phi[x \rightsquigarrow 0] \vee \phi[x \rightsquigarrow 1]$$

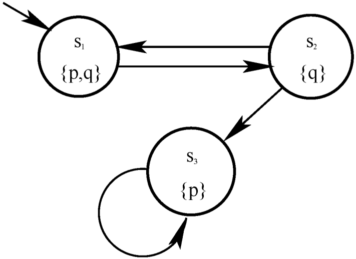
where $\phi[x \rightsquigarrow 0]$ is the formula obtained by ϕ after replacing all free occurrences of x by 0 (false), and similarly for $\phi[x \rightsquigarrow 1]$.

Predicate Transformer: Another Example



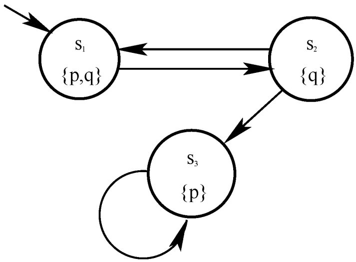
$$\mathbf{succ}(p \wedge q) \quad = \quad (\exists p, q : p \wedge q \wedge \mathit{Trans})[p' \leadsto p, q' \leadsto q]$$

Predicate Transformer: Another Example



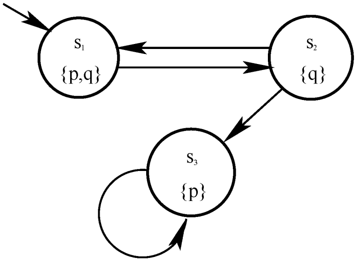
$$
\begin{aligned}
\mathbf{succ}(p \wedge q) &= (\exists p, q : p \wedge q \wedge \mathit{Trans})[p' \leadsto p, q' \leadsto q] \\
&= (\exists p, q : p \wedge q \wedge \overline{p}' \wedge q')[p' \leadsto p, q' \leadsto q]
\end{aligned}
$$

Predicate Transformer: Another Example



$$
\begin{aligned}
\mathbf{succ}(p \wedge q) &= (\exists p, q : p \wedge q \wedge \mathit{Trans})[p' \leadsto p, q' \leadsto q] \\
&= (\exists p, q : p \wedge q \wedge \overline{p}' \wedge q')[p' \leadsto p, q' \leadsto q] \\
&= (\overline{p}' \wedge q')[p' \leadsto p, q' \leadsto q]
\end{aligned}
$$

Predicate Transformer: Another Example



$$
\begin{aligned}
\mathbf{succ}(p \wedge q) &= (\exists p, q : p \wedge q \wedge \mathit{Trans})[p' \leadsto p, q' \leadsto q] \\
&= (\exists p, q : p \wedge q \wedge \overline{p}' \wedge q')[p' \leadsto p, q' \leadsto q] \\
&= (\overline{p}' \wedge q')[p' \leadsto p, q' \leadsto q] \\
&= \overline{p} \wedge q
\end{aligned}
$$

succ vs post

- **post** takes a set of states and returns a set of states:

$$\mathbf{post} : 2^S \to 2^S$$

where S is the set of states of the transition system.

- **succ** takes a formula and returns a formula:

$$\mathbf{succ} : \text{ Formula } \to \text{ Formula}$$

Symbolic Reachability Analysis Algorithm

$Reachable := Init$;
terminate := false;
repeat
    $tmp := Reachable \vee \mathbf{succ}(Reachable)$;
    if $tmp \Leftrightarrow Reachable$ **then**
        terminate := true;
    else
        $Reachable := tmp$;
    end if
until terminate
return $Reachable$;

Symbolic Reachability Analysis Algorithm

$Reachable := Init$;
terminate $:=$ false;
repeat
    $tmp := Reachable \vee \mathbf{succ}(Reachable)$;
    if $tmp \Leftrightarrow Reachable$ **then**
        terminate $:=$ true;
    else
        $Reachable := tmp$;
    end if
until terminate
return $Reachable$;

Does the algorithm terminate? Why?

Symbolic Reachability Analysis Algorithm

```
Reachable := Init;
terminate := false;
repeat
    tmp := Reachable ∨ succ(Reachable);
    if tmp ⇔ Reachable then
        terminate := true;
    else
        Reachable := tmp;
    end if
until terminate
return Reachable;
```

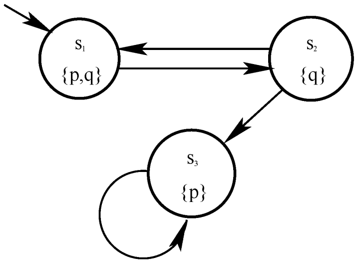
Does the algorithm terminate? Why?

Quiz: modify the algorithm to make it check reachability of a set of bad states characterized by predicate Bad.

Symbolic Reachability Algorithm: checking for Bad states

$Reachable := Init$;
terminate := false;
error := false;
repeat
    $tmp := Reachable \vee \mathbf{succ}(Reachable)$;
    if $tmp \Leftrightarrow Reachable$ **then**
        terminate := true;
    else
        $Reachable := tmp$;
    end if
    if SAT$(Reachable \wedge Bad)$ **then**
        error := true;
    end if
until terminate or error
return ($Reachable$,error);

Symbolic Reachability: Example



Let's check this system symbolically!

We want to check that all reachable states satisfy $p \lor q$.

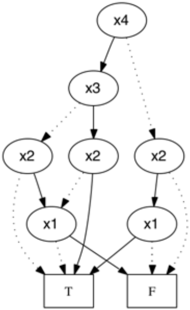
In temporal logic parlance:

$$\text{CTL:} \quad \mathbf{AG}(p \lor q)$$
$$\text{LTL:} \quad \mathbf{G}(p \lor q)$$

Symbolic Model-Checking: Implementation

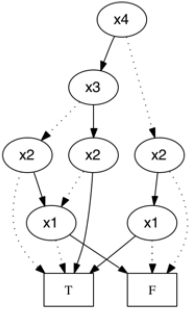
- For finite-state systems, boolean variables can be used to encode state.
- All predicates then become boolean expressions.
- Efficient data structures for boolean expressions:
  - **BDDs (Binary Decision Diagrams)**
- Efficient algorithms for implementing logical operations (conjunction, disjunction, satisfiability check, ...) on BDDs.
- Note: logical operations correspond to set-theoretic operations:
  - Conjunction: intersection
  - Disjunction: union
  - Satisfiability check: emptiness check
  - ...

Example: BDD



Can you guess which boolean expression this BDD represents?

Example: BDD



Can you guess which boolean expression this BDD represents?

$$x_4\big(\overline{x_3}(\overline{x_2} + x_2\overline{x_1}) + x_3(\overline{x_2}\,\overline{x_1} + x_2)\big) + \overline{x_4}x_2x_1$$

Bibliography

Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
MIT Press.

Bryant, R. E. (1992).
Symbolic boolean manipulation with ordered binary-decision diagrams.
ACM Comput. Surv., 24(3):293–318.

Burch, J., Clarke, E., Dill, D., Hwang, L., and McMillan, K. (1990).
Symbolic model checking: 10^{20} states and beyond.
In *5th LICS*, pages 428–439. IEEE.

Clarke, E., Grumberg, O., and Peled, D. (2000).
Model Checking.
MIT Press.

Huth, M. and Ryan, M. (2004).
Logic in Computer Science: Modelling and Reasoning about Systems.
Cambridge University Press.