Lattices and Fixpoints for Symbolic Model Checking
Introduction

❖ We are now working towards an algorithm for symbolic ROBDD-based CTL model checking (symbolic model checking algorithms for other temporal logics may be found in the literature).

❖ Symbolic ROBDD-based CTL MC uses ROBDDs to represent sets of states in which some formula holds as well as the transition relation of the system being examined.

❖ We need to express CTL operators as operators on ROBDD-represented sets in order to be able to compute by structural induction the sets of states in which the different subformulae of the given CTL formula hold.

❖ The effect of some of the CTL operators can be naturally computed as the limit of gradually improving approximations of their meaning.

❖ The limit will be obtained by iteratively computing least/greatest fixpoints of the transformation used to improve the approximations (obtained when no further improvement is possible).

❖ We need a structure on which to compute fixpoints, which takes us to lattices.
Partial Orders

❖ A tuple \((A, \leq_A)\) is a poset (partially-ordered set) iff \(A\) is a set and \(\leq_A \subseteq A \times A\) is a partial order (i.e., a reflexive, transitive, and antisymmetric binary relation) on \(A\).

❖ An example: Given a set \(S\), \(\langle 2^S, \subseteq \rangle\) is a poset.
Partial Orders

❖ A tuple $(A, \leq_A)$ is a poset (partially-ordered set) iff $A$ is a set and $\leq_A \subseteq A \times A$ is a partial order (i.e., a reflexive, transitive, and antisymmetric binary relation) on $A$.

❖ An example: Given a set $S$, $(2^S, \subseteq)$ is a poset.

❖ Given a poset $(A, \leq_A)$ and a set $B \subseteq A$,

- an element $a \in A$ is the greatest lower bound of $B$ (glb/infirmum/meet of $B$, $\cap B$) iff
  1. $\forall b \in B. \ a \leq_A b$ ("lower bound") and
  2. $\forall a' \in A. \ (\forall b \in B. \ a' \leq_A b) \implies a' \leq_A a$ ("greatest"),
Partial Orders

❖ A tuple \((A, \leq_A)\) is a poset (partially-ordered set) iff \(A\) is a set and \(\leq_A \subseteq A \times A\) is a partial order (i.e., a reflexive, transitive, and antisymmetric binary relation) on \(A\).

❖ An example: Given a set \(S\), \((2^S, \subseteq)\) is a poset.

❖ Given a poset \((A, \leq_A)\) and a set \(B \subseteq A\),
  - an element \(a \in A\) is the greatest lower bound of \(B\) (glb/infimum/meet of \(B\), \(\cap B\)) iff
    1. \(\forall b \in B. \ a \leq_A b\) (“lower bound”) and
    2. \(\forall a' \in A. \ (\forall b \in B. \ a' \leq_A b) \implies a' \leq_A a\) (“greatest”),
  - an element \(a \in A\) is the least upper bound of \(B\) (lub/supremum/join of \(B\), \(\cup B\)) iff
    1. \(\forall b \in B. \ b \leq_A a\) (“upper bound”) and
    2. \(\forall a' \in A. \ (\forall b \in B. \ b \leq_A a') \implies a \leq_A a'\) (“least”),

FIT.
Partial Orders

A tuple \((A, \leq_A)\) is a poset (partially-ordered set) iff \(A\) is a set and \(\leq_A \subseteq A \times A\) is a partial order (i.e., a reflexive, transitive, and antisymmetric binary relation) on \(A\).

An example: Given a set \(S\), \((2^S, \subseteq)\) is a poset.

Given a poset \((A, \leq_A)\) and a set \(B \subseteq A\),

- an element \(a \in A\) is the greatest lower bound of \(B\) (glb/infimum/meet of \(B\), \(\cap B\)) iff
  1. \(\forall b \in B. \ a \leq_A b\) ("lower bound") and
  2. \(\forall a' \in A. \ (\forall b \in B. \ a' \leq_A b) \implies a' \leq_A a\) ("greatest"),

- an element \(a \in A\) is the least upper bound of \(B\) (lub/supremum/join of \(B\), \(\cup B\)) iff
  1. \(\forall b \in B. \ b \leq_A a\) ("upper bound") and
  2. \(\forall a' \in A. \ (\forall b \in B. \ b \leq_A a') \implies a \leq_A a'\) ("least"),

An example: For \((2^{\{a,b,c\}, \subseteq})\), \(\cup\emptyset, \{a\}, \{b\}\) = \(\{a, b\}\), and \(\{a\} \cap \{b, c\} = \emptyset\).
**Lattices**

- A poset \((A, \leq_A)\) is a **lattice** iff each *non-empty, finite* subset \(B\) of \(A\) has a **lub** as well as a **glb** in \(A\).

- A poset \((A, \leq_A)\) is a **complete lattice** iff each subset \(B\) of \(A\) has a **lub** as well as a **glb** in \(A\).
**Lattices**

- A poset \((A, \leq_A)\) is a **lattice** iff each *non-empty, finite* subset \(B\) of \(A\) has a lub as well as a glb in \(A\).

- A poset \((A, \leq_A)\) is a **complete lattice** iff each subset \(B\) of \(A\) has a lub as well as a glb in \(A\),
  - \(\bot_A = \cap A\) and \(\top_A = \cup A\) are the least and greatest elements of a complete lattice, respectively.
A poset \((A, \leq_A)\) is a lattice iff each non-empty, finite subset \(B\) of \(A\) has a lub as well as a glb in \(A\).

A poset \((A, \leq_A)\) is a complete lattice iff each subset \(B\) of \(A\) has a lub as well as a glb in \(A\).

- \(\perp_A = \cap A\) and \(\top_A = \cup A\) are the least and greatest elements of a complete lattice, respectively.

Examples:
- \((2\{a,b,c\}, \subseteq)\) is a complete lattice, \(\cap\) corresponds to \(\cap\), \(\cup\) to \(\cup\), \(\perp\) to \(\emptyset\), and \(\top\) to \(\{a, b, c\}\).
Lattices

A poset \((A, \leq_A)\) is a **lattice** iff each *non-empty, finite* subset \(B\) of \(A\) has a **lub** as well as a **glb** in \(A\).

A poset \((A, \leq_A)\) is a **complete lattice** iff each subset \(B\) of \(A\) has a **lub** as well as a **glb** in \(A\):

- \(\bot_A = \cap A\) and \(\top_A = \cup A\) are the least and greatest elements of a complete lattice, respectively.

Examples:

- \((2\{a,b,c\}, \subseteq)\) is a complete lattice, \(\cap\) corresponds to \(\cap\), \(\cup\) to \(\cup\), \(\bot\) to \(\emptyset\), and \(\top\) to \(\{a, b, c\}\).
- \((\mathbb{N}, \leq)\) is a lattice with \(\cup\) being *max* and \(\cap\) being *min*, but not a complete lattice since \(\cup\mathbb{N}\) does not exist in \(\mathbb{N}\).
Lattices

❖ A poset \((A, \leq_A)\) is a lattice iff each non-empty, finite subset \(B\) of \(A\) has a lub as well as a glb in \(A\).

❖ A poset \((A, \leq_A)\) is a complete lattice iff each subset \(B\) of \(A\) has a lub as well as a glb in \(A\).

- \(\bot_A = \bigcap A\) and \(\top_A = \bigcup A\) are the least and greatest elements of a complete lattice, respectively.

❖ Examples:
  - \((\{a,b,c\}, \subseteq)\) is a complete lattice, \(\cap\) corresponds to \(\cap\), \(\cup\) to \(\cup\), \(\bot\) to \(\emptyset\), and \(\top\) to \(\{a, b, c\}\).
  - \((\mathbb{N}, \leq)\) is a lattice with \(\cup\) being max and \(\cap\) being min, but not a complete lattice since \(\cup \mathbb{N}\) does not exist in \(\mathbb{N}\).
  - \((\mathbb{N}_\infty, \leq)\), where \(\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}\) and \(\forall n \in \mathbb{N}. \ n \leq \infty\), is a complete lattice.
Lattices

- A poset \((A, \leq_A)\) is a lattice iff each non-empty, finite subset \(B\) of \(A\) has a lub as well as a glb in \(A\).

- A poset \((A, \leq_A)\) is a complete lattice iff each subset \(B\) of \(A\) has a lub as well as a glb in \(A\).
  - \(\bot_A = \cap A\) and \(\top_A = \cup A\) are the least and greatest elements of a complete lattice, respectively.

- Examples:
  - \((\{a,b,c\}, \subseteq)\) is a complete lattice, \(\cap\) corresponds to \(\cap\), \(\cup\) to \(\cup\), \(\bot\) to \(\emptyset\), and \(\top\) to \(\{a, b, c\}\).
  - \((\mathbb{N}, \leq)\) is a lattice with \(\cup\) being \(\text{max}\) and \(\cap\) being \(\text{min}\), but not a complete lattice since \(\cup \mathbb{N}\) does not exist in \(\mathbb{N}\).
  - \((\mathbb{N}_\infty, \leq)\), where \(\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}\) and \(\forall n \in \mathbb{N}. \ n \leq \infty\), is a complete lattice.

- Given a poset \((A, \leq_A)\), a set \(B \subseteq A\) is a chain iff \(\forall b, b' \in B. \ b \leq_A b' \lor b' \leq_A b\).
  - E.g., \(\emptyset, \{a\}, \{a, b\}, \{a, b, c\}\) is a chain wrt. \((2\{a, b, c\}, \subseteq)\).
Functions on Lattices

❖ Let \((A, \leq_A)\) and \((B, \leq_B)\) be lattices.

❖ A function \(f : A \rightarrow B\) is monotonic iff \(\forall a, a' \in A. a \leq_A a' \implies f(a) \leq_B f(a')\).
Functions on Lattices

❖ Let \((A, \leq_A)\) and \((B, \leq_B)\) be lattices.

❖ A function \(f : A \rightarrow B\) is monotonic iff \(\forall a, a' \in A. a \leq_A a' \implies f(a) \leq_B f(a')\).

❖ A function \(f : A \rightarrow B\) is \(\sqcup\)-continuous iff for every chain \(C \subseteq A\), we have \(f(\sqcup C) = \sqcup \{ f(c) \mid c \in C \}\). Analogously, one can define \(\sqcap\)-continuous functions.
Functions on Lattices

❖ Let \((A, \leq_A)\) and \((B, \leq_B)\) be lattices.

❖ A function \(f : A \rightarrow B\) is **monotonic** iff \(\forall a, a' \in A.\ a \leq_A a' \implies f(a) \leq_B f(a')\).

❖ A function \(f : A \rightarrow B\) is **⊔-continuous** iff for every chain \(C \subseteq A\), we have \(f(\sqcup C) = \sqcup \{f(c) \mid c \in C\}\). Analogously, one can define **\(\sqcap\)-continuous** functions.

❖ An element \(a \in A\) is a **fixpoint** of a function \(f : A \rightarrow A\) iff \(f(a) = a\).
Functions on Lattices

❖ Let \((A, \leq_A)\) and \((B, \leq_B)\) be lattices.

❖ A function \(f : A \rightarrow B\) is **monotonic** iff \(\forall a, a' \in A. \ a \leq_A a' \implies f(a) \leq_B f(a')\).

❖ A function \(f : A \rightarrow B\) is **⊔-continuous** iff for every chain \(C \subseteq A\), we have 
\[f(\sqcup C) = \sqcup \{f(c) \mid c \in C\}\]. Analogously, one can define **∩-continuous** functions.

❖ An element \(a \in A\) is a **fixpoint** of a function \(f : A \rightarrow A\) iff \(f(a) = a\).

❖ An example: Consider the function \(f : \mathbb{N}_\infty \rightarrow \mathbb{N}_\infty\) defined such that \(\forall n \in \mathbb{N}. \ f(n) = 0\) and \(f(\infty) = \infty\).
Functions on Lattices

❖ Let \((A, \leq_A)\) and \((B, \leq_B)\) be lattices.

❖ A function \(f : A \rightarrow B\) is monotonic iff \(\forall a, a' \in A. \ a \leq_A a' \implies f(a) \leq_B f(a')\).

❖ A function \(f : A \rightarrow B\) is \(\uparrow\)-continuous iff for every chain \(C \subseteq A\), we have \(f(\uparrow C) = \uparrow\{f(c) \mid c \in C\}\). Analogously, one can define \(\sqcap\)-continuous functions.

❖ An element \(a \in A\) is a fixpoint of a function \(f : A \rightarrow A\) iff \(f(a) = a\).

❖ An example: Consider the function \(f : \mathbb{N}_\infty \rightarrow \mathbb{N}_\infty\) defined such that \(\forall n \in \mathbb{N}. \ f(n) = 0\) and \(f(\infty) = \infty\).

  • \(f\) is monotonic since (1) \(\forall n_1, n_2 \in \mathbb{N}. \ f(n_1) = 0 \leq 0 = f(n_2)\) and (2) \(\forall n \in \mathbb{N}. \ f(n) = 0 \leq \infty = f(\infty)\).
Functions on Lattices

❖ Let \((A, \leq_A)\) and \((B, \leq_B)\) be lattices.

❖ A function \(f : A \to B\) is monotonic iff \(\forall a, a' \in A. a \leq_A a' \implies f(a) \leq_B f(a')\).

❖ A function \(f : A \to B\) is \(\sqcup\)-continuous iff for every chain \(C \subseteq A\), we have \(f(\sqcup C) = \sqcup \{f(c) \mid c \in C\}\). Analogously, one can define \(\sqcap\)-continuous functions.

❖ An element \(a \in A\) is a fixpoint of a function \(f : A \to A\) iff \(f(a) = a\).

❖ An example: Consider the function \(f : \mathbb{N}_\infty \to \mathbb{N}_\infty\) defined such that \(\forall n \in \mathbb{N}. f(n) = 0\) and \(f(\infty) = \infty\).
  
  • \(f\) is monotonic since (1) \(\forall n_1, n_2 \in \mathbb{N}. f(n_1) = 0 \leq 0 = f(n_2)\) and (2) \(\forall n \in \mathbb{N}. f(n) = 0 \leq \infty = f(\infty)\).
  
  • \(f\) is not \(\sqcup\)-continuous since \(\mathbb{N}\) is a chain and \(f(\sqcup \mathbb{N}) = f(\infty) = \infty\), but \(\sqcup \{f(n) \mid n \in \mathbb{N}\} = \sqcup \{0\} = 0\).
Functions on Lattices

❖ Let \((A, \leq_A)\) and \((B, \leq_B)\) be lattices.

❖ A function \(f : A \rightarrow B\) is monotonic iff \(\forall a, a' \in A. \ a \leq_A a' \implies f(a) \leq_B f(a')\).

❖ A function \(f : A \rightarrow B\) is \(\sqcup\)-continuous iff for every chain \(C \subseteq A\), we have \(f(\sqcup C) = \sqcup\{f(c) \mid c \in C\}\). Analogously, one can define \(\sqcap\)-continuous functions.

❖ An element \(a \in A\) is a fixpoint of a function \(f : A \rightarrow A\) iff \(f(a) = a\).

❖ An example: Consider the function \(f : \mathbb{N}_\infty \rightarrow \mathbb{N}_\infty\) defined such that \(\forall n \in \mathbb{N}. f(n) = 0\) and \(f(\infty) = \infty\).
   • \(f\) is monotonic since (1) \(\forall n_1, n_2 \in \mathbb{N}. f(n_1) = 0 \leq 0 = f(n_2)\) and (2) \(\forall n \in \mathbb{N}. f(n) = 0 \leq \infty = f(\infty)\).
   • \(f\) is not \(\sqcup\)-continuous since \(\mathbb{N}\) is a chain and \(f(\sqcup \mathbb{N}) = f(\infty) = \infty\), but \(\sqcup\{f(n) \mid n \in \mathbb{N}\} = \sqcup\{0\} = 0\).

\(~\text{Fit}\) • 0 is the least fixpoint of \(f\) and \(\infty\) is the greatest fixpoint of \(f\).
Knaster–Tarski Theorem

Knaster–Tarski Theorem. Let $(A, \leq_A)$ be a complete lattice and let $f : A \rightarrow A$ be a monotonic function. Then the set of fixpoints of $f$ in $(A, \leq_A)$ is also a complete lattice.

Since complete lattices have the least and the greatest element, the theorem in particular guarantees the existence of a least and greatest fixpoint of $f$ in $A$. 

FIT.
Knaster–Tarski Theorem

❖ Knaster–Tarski Theorem. Let \((A, \leq_A)\) be a complete lattice and let \(f : A \rightarrow A\) be a monotonic function. Then the set of fixpoints of \(f\) in \((A, \leq_A)\) is also a complete lattice.

❖ Since complete lattices have the least and the greatest element, the theorem in particular guarantees the existence of a least and greatest fixpoint of \(f\) in \(A\).

For the more curious:

❖ In more constructive terms, the least fixpoint of \(f\) is the stationary limit of \(f^\alpha(\perp_A)\) for \(\alpha\) ranging over the ordinals.
  
  • An ordinal is the order type of a well-ordered set.
Knaster–Tarski Theorem

- **Knaster–Tarski Theorem.** Let \((A, \leq_A)\) be a complete lattice and let \(f : A \rightarrow A\) be a monotonic function. Then the set of fixpoints of \(f\) in \((A, \leq_A)\) is also a complete lattice.

- Since complete lattices have the least and the greatest element, the theorem in particular guarantees the existence of a least and greatest fixpoint of \(f\) in \(A\).

*For the more curious:*

- In more constructive terms, the least fixpoint of \(f\) is the stationary limit of \(f^\alpha(\bot_A)\) for \(\alpha\) ranging over the ordinals.
  - An ordinal is the order type of a well-ordered set.
  - Every ordinal can be represented as the set of all smaller ordinals. There is the zero ordinal, successor ordinals, and limit ordinals. Natural numbers correspond to the so called finite ordinals (ordering types of finite sets), the set of natural numbers is the first infinite ordinal, and so on.

FIT.
Knaster–Tarski Theorem

Knaster–Tarski Theorem. Let \((A, \leq_A)\) be a complete lattice and let \(f : A \rightarrow A\) be a monotonic function. Then the set of fixpoints of \(f\) in \((A, \leq_A)\) is also a complete lattice.

Since complete lattices have the least and the greatest element, the theorem in particular guarantees the existence of a least and greatest fixpoint of \(f\) in \(A\).

For the more curious:

In more constructive terms, the least fixpoint of \(f\) is the stationary limit of \(f^\alpha(\bot_A)\) for \(\alpha\) ranging over the ordinals.

- An ordinal is the order type of a well-ordered set.
- Every ordinal can be represented as the set of all smaller ordinals. There is the zero ordinal, successor ordinals, and limit ordinals. Natural numbers correspond to the so called finite ordinals (ordering types of finite sets), the set of natural numbers is the first infinite ordinal, and so on.
- \(f^\alpha\) is defined by transfinite induction: \(f^{\alpha+1} = f(f^\alpha)\) and \(f^\gamma\) for a limit ordinal \(\gamma\) is the least upper bound of \(f^\beta\) for all ordinals \(\beta\) smaller than \(\gamma\).
**Knaster–Tarski Theorem**

- **Knaster–Tarski Theorem.** Let \((A, \leq_A)\) be a complete lattice and let \(f : A \rightarrow A\) be a monotonic function. Then the set of fixpoints of \(f\) in \((A, \leq_A)\) is also a complete lattice.

- Since complete lattices have the least and the greatest element, the theorem in particular guarantees the existence of a least and greatest fixpoint of \(f\) in \(A\).

*For the more curious:*

- In more constructive terms, the least fixpoint of \(f\) is the stationary limit of \(f^\alpha(\bot_A)\) for \(\alpha\) ranging over the ordinals.
  - An ordinal is the order type of a well-ordered set.
  - Every ordinal can be represented as the set of all smaller ordinals. There is the zero ordinal, successor ordinals, and limit ordinals. Natural numbers correspond to the so called finite ordinals (ordering types of finite sets), the set of natural numbers is the first infinite ordinal, and so on.
  - \(f^\alpha\) is defined by transfinite induction: \(f^{\alpha+1} = f(f^\alpha)\) and \(f^\gamma\) for a limit ordinal \(\gamma\) is the least upper bound of \(f^\beta\) for all ordinals \(\beta\) smaller than \(\gamma\).

- A dual result holds for the greatest fixpoint.
Kleene Fixpoint Theorem

- Kleene Fixpoint Theorem. Let \((A, \leq_A)\) be a complete lattice and \(f : A \rightarrow A\) a function.
  - If \(f\) is \(\sqcup\)-continuous, the least fixpoint of \(f\) is \(\mu f = \sqcup\{f^i(\bot_A) \mid i \geq 0\}\).
Kleene Fixpoint Theorem

Kleene Fixpoint Theorem. Let \((A, \leq_A)\) be a complete lattice and \(f : A \rightarrow A\) a function.

- If \(f\) is \(\sqcup\)-continuous, the least fixpoint of \(f\) is \(\mu f = \sqcup \{f^i(\bot_A) \mid i \geq 0\}\).
  - Moreover, a \(\sqcup\)-continuous function is monotone, and hence one in fact computes the supremum of the ascending chain
    \(\bot_A \leq_A f(\bot_A) \leq_A f(f(\bot_A)) \leq \ldots\).
**Kleene Fixpoint Theorem.** Let \((A, \leq_A)\) be a complete lattice and \(f : A \rightarrow A\) a function.

- If \(f\) is \(\sqcup\)-continuous, the least fixpoint of \(f\) is \(\mu f = \sqcup\{f^i(\bot_A) \mid i \geq 0\}\).
  - Moreover, a \(\sqcup\)-continuous function is monotone, and hence one in fact computes the supremum of the ascending chain
    \[\bot_A \leq_A f(\bot_A) \leq_A f(f(\bot_A)) \leq \ldots\]

- If \(f\) is \(\sqcap\)-continuous, the greatest fixpoint of \(f\) is \(\nu f = \sqcap\{f^i(\top_A) \mid i \geq 0\}\).
  - Moreover, a \(\sqcap\)-continuous function is monotone, and hence one in fact computes the infimum of the descending chain
    \[\top_A \geq_A f(\top_A) \geq_A f(f(\top_A)) \geq \ldots\]
**Kleene Fixpoint Theorem**

- **Kleene Fixpoint Theorem.** Let \((A, \leq_A)\) be a complete lattice and \(f : A \rightarrow A\) a function.
  - If \(f\) is \(\sqcup\)-continuous, the least fixpoint of \(f\) is \(\mu f = \sqcup\{f^i(\bot_A) \mid i \geq 0\}\).
    - Moreover, a \(\sqcup\)-continuous function is monotone, and hence one in fact computes the supremum of the ascending chain \(\bot_A \leq_A f(\bot_A) \leq_A f(f(\bot_A)) \leq \ldots\).
  - If \(f\) is \(\sqcap\)-continuous, the greatest fixpoint of \(f\) is \(\nu f = \sqcap\{f^i(\top_A) \mid i \geq 0\}\).
    - Moreover, a \(\sqcap\)-continuous function is monotone, and hence one in fact computes the infimum of the descending chain \(\top_A \geq_A f(\top_A) \geq_A f(f(\top_A)) \geq \ldots\).

- **Theorem.** For finite complete lattices, every monotonic function is \(\sqcap\)- and \(\sqcup\)-continuous.
Kleene Fixpoint Theorem

❖ **Kleene Fixpoint Theorem.** Let \((A, \leq_A)\) be a complete lattice and \(f : A \rightarrow A\) a function.

- If \(f\) is \(\sqcup\)-continuous, the **least fixpoint** of \(f\) is \(\mu f = \sqcup\{f^i(\bot_A) \mid i \geq 0\}\).
  - Moreover, a \(\sqcup\)-continuous function is monotone, and hence one in fact computes the supremum of the ascending chain
    \[\bot_A \leq_A f(\bot_A) \leq_A f(f(\bot_A)) \leq \ldots\]

- If \(f\) is \(\sqcap\)-continuous, the **greatest fixpoint** of \(f\) is \(\nu f = \sqcap\{f^i(\top_A) \mid i \geq 0\}\).
  - Moreover, a \(\sqcap\)-continuous function is monotone, and hence one in fact computes the infimum of the descending chain
    \[\top_A \geq_A f(\top_A) \geq_A f(f(\top_A)) \geq \ldots\]

❖ **Theorem.** For finite complete lattices, every monotonic function is \(\sqcap\)- and \(\sqcup\)-continuous.

❖ **Corollary.** On finite lattices, the Kleene fixpoint theorem is applicable, hence,

- to compute the least fixpoint, start with \(\bot_A\) and iteratively apply \(f\) till
  \[f^i(\bot_A) = f^{i+1}(\bot_A) = \mu f,\]

- to compute the greatest fixpoint, start with \(\top_A\) and iteratively apply \(f\) till
  \[f^i(\top_A) = f^{i+1}(\top_A) = \nu f.\]
CTL Operators
as Predicate Transformers
CTL Operators as Predicate Transformers

- A CTL formula $\varphi$ defines a (unary) predicate on states (namely, "being a state in which $\varphi$ holds") characterising the set of states $[\varphi] = \{ s \in S \mid M, s \models \varphi \}$ in which it holds.

- A (nullary, unary, or binary) CTL operator $op$ can thus be viewed as a predicate transformer $\tau_{op}()$/$\tau_{op}(.)$/$\tau_{op}(., .)$, respectively, transforming 0–2 predicates (given by the set of states they characterise) into a new predicate (again given by the set of states it characterises).
  - Taking into account the correspondence between unary predicates on states and sets of states, one could also speak about set transformers.
  - So far, we do not care how the sets are represented—later, we will represent them using ROBDDs.

- Some of the CTL operators having an iterative nature (e.g., EU, EG, etc.) may be defined as least/greatest fixpoints of some auxiliary predicate transformers which, intuitively, represent one step in the iterative computation of the appropriate operation.
**CTL Predicate Transformers**

Consider a Kripke structure $M = (S, S_0, R, L)$. The meaning of the **CTL operators** (including atomic formulae viewed as nullary operators) over $M$ can be defined in terms of **predicate transformers** as follows (for $S', S_1, S_2 \subseteq S$):

- $\tau_p() = \llbracket p \rrbracket$
- $\tau_{\neg}(S') = S \setminus S'$
- $\tau_\lor(S_1, S_2) = S_1 \cup S_2$
- $\tau_{EX}(S') = \{ s \in S | \exists s' \in S'. (s, s') \in R \}$
- $\tau_{AX}(S') = \{ s \in S | \forall s' \in S. (s, s') \in R \implies s' \in S' \}$
- $\tau_{AF}(S') = \mu Z. \tau(Z) \text{ where } \tau(Z) = S' \cup \tau_{AX}(Z)$
- $\tau_{EF}(S') = \mu Z. S' \cup \tau_{EX}(Z)$
- $\tau_{AG}(S') = \nu Z. S' \cap \tau_{AX}(Z)$
- $\tau_{EG}(S') = \nu Z. S' \cap \tau_{EX}(Z)$
- $\tau_{A[U]}(S_1, S_2) = \mu Z. S_2 \cup (S_1 \cap \tau_{AX}(Z))$
- $\tau_{E[U]}(S_1, S_2) = \mu Z. S_2 \cup (S_1 \cap \tau_{EX}(Z))$
Intuition behind Some of the Transformers

Consider $\tau_{EG}$:

- Clearly, $EG \varphi \equiv \varphi \land EX \ EG \varphi$.

- Hence, $[EG \varphi]$ is the greatest solution of the equation $Z = [\varphi] \cap R^{-1}(Z)$.
  - We want to exclude only those states from $Z$ which either do not satisfy $\varphi$ or cannot continue into $Z$.

- Hence, $[EG \varphi]$ is the greatest fixpoint of the predicate transformer $\tau(Z) = [\varphi] \cap \tau_{EX}(Z)$.
Intuition behind Some of the Transformers

❖ Consider $\tau_{EG}$:

- Clearly, $EG \varphi \equiv \varphi \land EX EG \varphi$.
- Hence, $[EG \varphi]$ is the greatest solution of the equation $Z = [\varphi] \cap R^{-1}(Z)$.
  - We want to exclude only those states from $Z$ which either do not satisfy $\varphi$ or cannot continue into $Z$.
- Hence, $[EG \varphi]$ is the greatest fixpoint of the predicate transformer $\tau(Z) = [\varphi] \cap \tau_{EX}(Z)$.

❖ Consider $\tau_{E[U]}$:

- Clearly, $E[\varphi_1 U \varphi_2] \equiv \varphi_2 \lor (\varphi_1 \land EX E[\varphi_1 U \varphi_2])$.
- Hence, $[E[\varphi_1 U \varphi_2]]$ is the least solution of the equation $Z = [\varphi_2] \cup ([\varphi_1] \cap R^{-1}(Z))$.
  - We want to consider only those states which either satisfy $\varphi_2$ or satisfy $\varphi_1$ and can continue within $Z$ towards some state which satisfies $\varphi_2$.
- Hence, $[E[\varphi_1 U \varphi_2]]$ is the least fixpoint of the predicate transformer $\tau(Z) = [\varphi_2] \cup ([\varphi_1] \cap \tau_{EX}(Z))$. 

Introduction – p.12/18
An Example of the CTL Fixpoint Semantics

\[ \llbracket EG \, p \rrbracket = \nu Z. \tau(Z) \text{ where } \tau(Z) = \llbracket p \rrbracket \cap \tau_{EX}(Z): \]

\[ s_1 \xrightarrow{p, \neg q} s_3 \]
\[ s_3 \xrightarrow{p, \neg q} s_1 \]
\[ s_2 \xrightarrow{\neg p, \neg q} s_4 \]
\[ s_4 \xrightarrow{p, q} s_2 \]
An Example of the CTL Fixpoint Semantics

$$[EG \ p] = \nu Z. \tau (Z) \text{ where } \tau (Z) = \llbracket p \rrbracket \cap \tau _{EX} (Z):$$

- $$Z_0 = \{s_1, s_2, s_3, s_4\}$$
An Example of the CTL Fixpoint Semantics

\[ [EG\ p] = \nu Z.\tau(Z) \] where \( \tau(Z) = [p] \cap \tau_{EX}(Z) \):

• \( Z_0 = \{s_1, s_2, s_3, s_4\} \)
• \( Z_1 = \tau(Z_0) = \{s_1, s_3, s_4\} \)
An Example of the CTL Fixpoint Semantics

\[ [EG\ p] = \nu Z.\tau(Z) \text{ where } \tau(Z) = [p] \cap \tau_{EX}(Z): \]

- \( Z_0 = \{s_1, s_2, s_3, s_4\} \)
- \( Z_1 = \tau(Z_0) = \{s_1, s_3, s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_1, s_3\} \)
An Example of the CTL Fixpoint Semantics

\[ [EG \ p] = \nu Z. \tau(Z) \text{ where } \tau(Z) = \llbracket p \rrbracket \cap \tau_{EX}(Z): \]

- \( Z_0 = \{s_1, s_2, s_3, s_4\} \)
- \( Z_1 = \tau(Z_0) = \{s_1, s_3, s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_1, s_3\} \)
- \( Z_3 = \tau(Z_2) = \{s_1, s_3\} = Z_2 = [EG \ p] \)
An Example of the CTL Fixpoint Semantics

\[ [EG \ p] = \nu Z. \tau(Z) \text{ where } \tau(Z) = [p] \cap \tau_{EX}(Z): \]

- \( Z_0 = \{s_1, s_2, s_3, s_4\} \)
- \( Z_1 = \tau(Z_0) = \{s_1, s_3, s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_1, s_3\} \)
- \( Z_3 = \tau(Z_2) = \{s_1, s_3\} = Z_2 = [EG \ p] \)

\[ [E[pUq]] = \mu Z. \tau(Z) \text{ where } \tau(Z) = [q] \cup ([p] \cap \tau_{EX}(Z)): \]
An Example of the CTL Fixpoint Semantics

\[
\llbracket EG p \rrbracket = \nu Z. \tau(Z) \text{ where } \tau(Z) = \llbracket p \rrbracket \cap \tau_{EX}(Z):
\]

- \( Z_0 = \{s_1, s_2, s_3, s_4\} \)
- \( Z_1 = \tau(Z_0) = \{s_1, s_3, s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_1, s_3\} \)
- \( Z_3 = \tau(Z_2) = \{s_1, s_3\} = Z_2 = \llbracket EG p \rrbracket \)

\[
\llbracket E[pUq] \rrbracket = \mu Z. \tau(Z) \text{ where } \tau(Z) = \llbracket q \rrbracket \cup (\llbracket p \rrbracket \cap \tau_{EX}(Z)):
\]

- \( Z_0 = \emptyset \)
An Example of the CTL Fixpoint Semantics

\[
\llbracket EG\, p \rrbracket = \nu Z.\tau(Z)\text{ where }\tau(Z) = \llbracket p \rrbracket \cap \tau_{EX}(Z):
\]

- \( Z_0 = \{s_1, s_2, s_3, s_4\} \)
- \( Z_1 = \tau(Z_0) = \{s_1, s_3, s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_1, s_3\} \)
- \( Z_3 = \tau(Z_2) = \{s_1, s_3\} = Z_2 = \llbracket EG\, p \rrbracket \)

\[
\llbracket E[pUq] \rrbracket = \mu Z.\tau(Z)\text{ where }\tau(Z) = \llbracket q \rrbracket \cup (\llbracket p \rrbracket \cap \tau_{EX}(Z)):
\]

- \( Z_0 = \emptyset \)
- \( Z_1 = \tau(Z_0) = \{s_4\} \)
An Example of the CTL Fixpoint Semantics

\[ [EG \ p] = \nu Z. \tau(Z) \quad \text{where} \quad \tau(Z) = [p] \cap \tau_{EX}(Z): \]

- \( Z_0 = \{s_1, s_2, s_3, s_4\} \)
- \( Z_1 = \tau(Z_0) = \{s_1, s_3, s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_1, s_3\} \)
- \( Z_3 = \tau(Z_2) = \{s_1, s_3\} = Z_2 = [EG \ p] \)

\[ [E[pUq]] = \mu Z. \tau(Z) \quad \text{where} \quad \tau(Z) = [q] \cup ([p] \cap \tau_{EX}(Z)): \]

- \( Z_0 = \emptyset \)
- \( Z_1 = \tau(Z_0) = \{s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_3, s_4\} \)
An Example of the CTL Fixpoint Semantics

\[ [EG \ p] = \nu Z. \tau(Z) \text{ where } \tau(Z) = [p] \cap \tau_{EX}(Z): \]

- \( Z_0 = \{s_1, s_2, s_3, s_4\} \)
- \( Z_1 = \tau(Z_0) = \{s_1, s_3, s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_1, s_3\} \)
- \( Z_3 = \tau(Z_2) = \{s_1, s_3\} = Z_2 = [EG \ p] \)

\[ [E[pUq]] = \mu Z. \tau(Z) \text{ where } \tau(Z) = [q] \cup ([p] \cap \tau_{EX}(Z)): \]

- \( Z_0 = \emptyset \)
- \( Z_1 = \tau(Z_0) = \{s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_3, s_4\} \)
- \( Z_3 = \tau(Z_2) = \{s_1, s_3, s_4\} \)

Introduction – p.13/18
An Example of the CTL Fixpoint Semantics

\[
\llbracket EG \ p \rrbracket = \nu Z. \tau(Z) \text{ where } \tau(Z) = \llbracket p \rrbracket \cap \tau_{EX}(Z):
\]
- \( Z_0 = \{s_1, s_2, s_3, s_4\} \)
- \( Z_1 = \tau(Z_0) = \{s_1, s_3, s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_1, s_3\} \)
- \( Z_3 = \tau(Z_2) = \{s_1, s_3\} = Z_2 = \llbracket EG \ p \rrbracket \)

\[
\llbracket E[pUq]\rrbracket = \mu Z. \tau(Z) \text{ where } \tau(Z) = \llbracket q \rrbracket \cup (\llbracket p \rrbracket \cap \tau_{EX}(Z)):\n\]
- \( Z_0 = \emptyset \)
- \( Z_1 = \tau(Z_0) = \{s_4\} \)
- \( Z_2 = \tau(Z_1) = \{s_3, s_4\} \)
- \( Z_3 = \tau(Z_2) = \{s_1, s_3, s_4\} \)
- \( Z_4 = \tau(Z_3) = \{s_1, s_3, s_4\} = Z_3 = \llbracket E[pUq]\rrbracket \)
The CTL Fixpoint Semantics and BDDs

- The operations used within the CTL fixpoint semantics include:
  - set operations on sets of states (like union, intersection, and set complement) which directly map to the corresponding operations on propositional formulae representing the sets wrt. some bit-vector encoding of the states (disjunction, conjunction, negation) and which are easy to implement on BDDs,
The CTL Fixpoint Semantics and BDDs

- The operations used within the CTL fixpoint semantics include:
  - set operations on sets of states (like union, intersection, and set complement) which directly map to the corresponding operations on propositional formulae representing the sets wrt. some bit-vector encoding of the states (disjunction, conjunction, negation) and which are easy to implement on BDDs,
  - fixpoint computations which (as we have discussed) can be implemented by iteratively applying the appropriate transformers starting from true (νf) or false (μf) till the result stops changing,
The CTL Fixpoint Semantics and BDDs

- The operations used within the CTL fixpoint semantics include:
  - **set operations** on sets of states (like union, intersection, and set complement) which directly map to the corresponding operations on propositional formulae representing the sets wrt. some bit-vector encoding of the states (disjunction, conjunction, negation) and which are easy to implement on BDDs,
  - **fixpoint computations** which (as we have discussed) can be implemented by iteratively applying the appropriate transformers starting from \( \text{true} (\nu f) \) or \( \text{false} (\mu f) \) till the result stops changing,
  - **application of the transition relation** which maps to a conjunction of the formulae representing a set of states and the relation,
The CTL Fixpoint Semantics and BDDs

- The operations used within the CTL fixpoint semantics include:
  - **set operations** on sets of states (like union, intersection, and set complement) which directly map to the corresponding operations on propositional formulae representing the sets wrt. some bit-vector encoding of the states (disjunction, conjunction, negation) and which are easy to implement on BDDs,
  - **fixpoint computations** which (as we have discussed) can be implemented by iteratively applying the appropriate transformers starting from true ($\nu f$) or false ($\mu f$) till the result stops changing,
  - **application of the transition relation** which maps to a conjunction of the formulae representing a set of states and the relation,
  - **quantification on Boolean variables** (we are dealing with quantified Boolean formulae, abbreviated as QBF)—can be done easily on ROBDDs using restriction and Apply:
The operations used within the CTL fixpoint semantics include:

- **set operations** on sets of states (like union, intersection, and set complement) which directly map to the corresponding operations on propositional formulae representing the sets wrt. some bit-vector encoding of the states (disjunction, conjunction, negation) and which are easy to implement on BDDs,

- **fixpoint computations** which (as we have discussed) can be implemented by iteratively applying the appropriate transformers starting from true (νf) or false (µf) till the result stops changing,

- **application of the transition relation** which maps to a conjunction of the formulae representing a set of states and the relation,

- **quantification on Boolean variables** (we are dealing with quantified Boolean formulae, abbreviated as QBF)—can be done easily on ROBDDs using restriction and Apply:
  
  - $\exists v. f \equiv f|_{v \leftarrow 0} \lor f|_{v \leftarrow 1}$,
  
  - $\forall v. f \equiv f|_{v \leftarrow 0} \land f|_{v \leftarrow 1}$.

*FIT.*
The CTL Fixpoint Semantics and BDDs

❖ The operations used within the CTL fixpoint semantics include:

- **set operations** on sets of states (like union, intersection, and set complement) which directly map to the corresponding operations on propositional formulae representing the sets wrt. some bit-vector encoding of the states (disjunction, conjunction, negation) and which are easy to implement on BDDs,

- **fixpoint computations** which (as we have discussed) can be implemented by iteratively applying the appropriate transformers starting from *true* \((\nu f)\) or *false* \((\mu f)\) till the result stops changing,

- **application of the transition relation** which maps to a conjunction of the formulae representing a set of states and the relation,

- **quantification on Boolean variables** (we are dealing with quantified Boolean formulae, abbreviated as QBF)—can be done easily on ROBDDs using restriction and Apply:
  - \(\exists v. f \equiv f|_{v \leftarrow 0} \lor f|_{v \leftarrow 1}\),
  - \(\forall v. f \equiv f|_{v \leftarrow 0} \land f|_{v \leftarrow 1}\).

- **renaming of primed variables to unprimed** (after quantification): trivial.
The Symbolic CTL MC Algorithm
The Symbolic CTL MC Algorithm

- **Input:**
  - a formula $\varphi$ to be checked and
  - ROBDDs representing
    - the initial states $S_0$,
    - the transition relation $R$ of the considered Kripke structure (can be obtained from the system being verified without generating all states reachable from $S_0$), and
    - the sets of states in which particular atomic propositions hold (can be obtained from the structure of states and the meaning of the propositions).
The Symbolic CTL MC Algorithm

❖ **Input:**

- a formula $\varphi$ to be checked and
- ROBDDs representing
  - the initial states $S_0$,  
  - the transition relation $R$ of the considered Kripke structure (can be obtained from the system being verified without generating all states reachable from $S_0$), and
  - the sets of states in which particular atomic propositions hold (can be obtained from the structure of states and the meaning of the propositions).

❖ **Output:** the formula holds or does not hold.
The Symbolic CTL MC Algorithm

❖ Input:
  • a formula \( \varphi \) to be checked and
  • ROBDDs representing
    – the initial states \( S_0 \),
    – the transition relation \( R \) of the considered Kripke structure (can be obtained from the system being verified without generating all states reachable from \( S_0 \)), and
    – the sets of states in which particular atomic propositions hold (can be obtained from the structure of states and the meaning of the propositions).

❖ Output: the formula holds or does not hold.

❖ Method: identical to the explicit state model checking algorithm up to:
  • the function \texttt{Check} must be implemented on ROBDDs,
  • the final check that \( \varphi \) holds in all initial states is implemented by taking an implication between the ROBDD representing the initial states and the one representing states where \( \varphi \) holds and checking that the result is the ROBDD 1.
The Symbolic CTL MC Algorithm

❖ **Input:**
- a formula $\varphi$ to be checked and
- ROBDDs representing
  - the initial states $S_0$,
  - the transition relation $R$ of the considered Kripke structure (can be obtained from the system being verified without generating all states reachable from $S_0$), and
  - the sets of states in which particular atomic propositions hold (can be obtained from the structure of states and the meaning of the propositions).

❖ **Output:** the formula holds or does not hold.

❖ **Method:** identical to the explicit state model checking algorithm up to:
- the function $\text{Check}$ must be implemented on ROBDDs,
- the final check that $\varphi$ holds in all initial states is implemented by taking an implication between the ROBDD representing the initial states and the one representing states where $\varphi$ holds and checking that the result is the ROBDD $1$.

❖ **Advantage:** may be much more efficient due to at the same time manipulating entire sets of states represented in a compact way!
Check on ROBDDs

- \text{Check}(p) \text{ for some } p \in AP: \text{ Return the ROBDD associated with } p.
Check on ROBDDs

- Check($p$) for some $p \in AP$: Return the ROBDD associated with $p$.

- Check($\neg \varphi$): Call Check($\varphi$) to compute the ROBDD encoding the states where $\varphi$ holds and then swap the meaning of the leaves.
**Check on ROBDDs**

- **Check**($p$) for some $p \in AP$: Return the ROBDD associated with $p$.

- **Check**($\neg \varphi$): Call **Check**($\varphi$) to compute the ROBDD encoding the states where $\varphi$ holds and then swap the meaning of the leaves.

- **Check**($\varphi_1 \lor \varphi_2$): Call **Check**($\varphi_1$) and **Check**($\varphi_2$) to compute the ROBDDs $G_1$, $G_2$ encoding the states where $\varphi_1$ and $\varphi_2$ hold and return the result of $\text{Apply}(\lor, G_1, G_2)$.
**Check on ROBDDs**

- **Check**(\(p\)) for some \(p \in AP\): Return the ROBDD associated with \(p\).

- **Check**(\(\neg \varphi\)): Call **Check**(\(\varphi\)) to compute the ROBDD encoding the states where \(\varphi\) holds and then swap the meaning of the leaves.

- **Check**(\(\varphi_1 \lor \varphi_2\)): Call **Check**(\(\varphi_1\)) and **Check**(\(\varphi_2\)) to compute the ROBDDs \(G_1\), \(G_2\) encoding the states where \(\varphi_1\) and \(\varphi_2\) hold and return the result of **Apply**(\(\lor\), \(G_1\), \(G_2\)).

- **Check**(\(EX \, \varphi\)):
  1. Call **Check**(\(\varphi\)) yielding the ROBDD \(G\) representing a propositional formula \(\psi\) encoding states in which \(\varphi\) holds in the given system.
**Check on ROBDDs**

- **Check**(p) for some p ∈ AP: Return the ROBDD associated with p.

- **Check**(¬φ): Call **Check**(φ) to compute the ROBDD encoding the states where φ holds and then swap the meaning of the leaves.

- **Check**(φ₁ ∨ φ₂): Call **Check**(φ₁) and **Check**(φ₂) to compute the ROBDDs G₁, G₂ encoding the states where φ₁ and φ₂ hold and return the result of **Apply**(∨, G₁, G₂).

- **Check**(EX φ):
  1. Call **Check**(φ) yielding the ROBDD G representing a propositional formula ψ encoding states in which φ holds in the given system.
  2. G is over variables Var = {v₁, ..., vₙ} (particular bit variables of the state vector), rename them to their primed counterparts Var' = {v'₁, ..., v'ₙ}, yielding a ROBDD G' representing a propositional formula ψ'.
Check on ROBDDs

❖ Check\((p)\) for some \(p \in AP\): Return the ROBDD associated with \(p\).

❖ Check\((¬\varphi)\): Call Check\((\varphi)\) to compute the ROBDD encoding the states where \(\varphi\) holds and then swap the meaning of the leaves.

❖ Check\((\varphi_1 \lor \varphi_2)\): Call Check\((\varphi_1)\) and Check\((\varphi_2)\) to compute the ROBDDs \(G_1, G_2\) encoding the states where \(\varphi_1\) and \(\varphi_2\) hold and return the result of Apply\((\lor, G_1, G_2)\).

❖ Check\((\exists X \varphi)\):
  1. Call Check\((\varphi)\) yielding the ROBDD \(G\) representing a propositional formula \(\psi\) encoding states in which \(\varphi\) holds in the given system.
  2. \(G\) is over variables \(Var = \{v_1, ..., v_n\}\) (particular bit variables of the state vector), rename them to their primed counterparts \(Var' = \{v'_1, ..., v'_n\}\), yielding a ROBDD \(G'\) representing a propositional formula \(\psi'\).
  3. Return the ROBDD corresponding to the formula \(\exists v'_1, ..., \exists v'_n. \psi' \land \varrho\) where \(\varrho\) is the (ROBDD-represented) propositional formula encoding the transition relation over variables \(Var \cup Var'\).
Check on ROBDDs—continued

- **Check**$^\ast$(E[ϕ₁ U ϕ₂]):
  1. Call Check(ϕ₁) and Check(ϕ₂) to compute the ROBDDs $G_1$, $G_2$ encoding the states where ϕ₁ and ϕ₂ hold.
  2. Let $Z_0$ be the ROBDD 0 (false) and let $Z_1$ be $G_2$.
  3. If $Z_0 = Z_1$, return $Z_1$, otherwise:
     (a) Let $Z_0$ be $Z_1$.
     (b) Let $Z_1$ be $\text{Apply}(\lor, G_2, \text{Apply}(\land, G_1, \text{CheckEX}(Z_0)))$ where CheckEX performs Points 2 and 3 from Check(EX ϕ).
     (c) Go back to Point 3.

*FIT.*
Check on ROBDDs—continued

❖ Check($E[\varphi_1 U \varphi_2]$):

1. Call Check($\varphi_1$) and Check($\varphi_2$) to compute the ROBDDs $G_1$, $G_2$ encoding the states where $\varphi_1$ and $\varphi_2$ hold.

2. Let $Z_0$ be the ROBDD $0$ (false) and let $Z_1$ be $G_2$.

3. If $Z_0 = Z_1$, return $Z_1$, otherwise:
   (a) Let $Z_0$ be $Z_1$.
   (b) Let $Z_1$ be $\text{Apply}(\lor, G_2, \text{Apply}(\land, G_1, \text{CheckEX}(Z_0)))$ where CheckEX performs Points 2 and 3 from Check($EX \varphi$).
   (c) Go back to Point 3.

❖ Check($EG \varphi$):

1. Call Check($\varphi$) to compute the ROBDD $G$ encoding the states where $\varphi$ holds.

2. Let $Z_0$ be the ROBDD $1$ (true) and let $Z_1$ be $G$.

3. If $Z_0 = Z_1$, return $Z_1$, otherwise:
   (a) Let $Z_0$ be $Z_1$.
   (b) Let $Z_1$ be $\text{Apply}(\land, G, \text{CheckEX}(Z_0))$.
   (c) Go back to Point 3.