Chapter 5

Model checking: with or without BDDs

This chapter presents two techniques to verify temporal properties. The first technique is based on BDDs. The second technique is based on SAT. Historically, BDD-based model checking was introduced first. The idea was to use the symbolic representation provided by BDDs to explore larger state spaces. In many cases, this approach turned out to be indeed more efficient than explicit techniques, where all states are explicitly represented in the memory. Still, there are cases where explicit state model checking out-performs BDD techniques. Later, SAT solvers became very good at solving large problems. The idea of SAT-based techniques was to leverage this new technology to verification. Today, SAT-based techniques are very often out-performing other ones. Still, there are cases where BDD-based solvers work better.

5.1 Symbolic model checking with BDD

An important concept in symbolic model-checking is the notion of fixpoints. The basic idea behind fixpoints is to recursively apply a function to a set of elements until two applications of the function return the same result, that is, the same set. We have reached a limit and this limit is exactly the fixpoint.

Formally, sssume a finite set S. $S' \subseteq S$ is a fixpoint of function $\tau : \mathcal{P}(S) \mapsto \mathcal{P}(S)$ if $\tau(S') = S'$.

5.1.1 Fixpoints and CTL

A predicate transformer is a function that transforms subsets of the set of states S into subsets of S. For instance, $\tau(Z) = f_2 \vee (f_1 \wedge \mathbf{EX}\ Z)$ is a predicate transformer that extract from set Z states that either satisfy f_2 or satisfy f_1 and have successors in Z. This corresponds exactly to states satisfying $\mathbf{E}(f_1\ \mathbf{U}\ f_2)$. This predicate has reached a fixpoint when two successive applications do not transform Z anymore. For instance, we will see that $\mathbf{E}(f_1\ \mathbf{U}\ f_2)$ is the least fixpoint of the predicate transformer τ defined as above. We consider two fixpoints: the least fixpoint noted μ and the greatest fixpoint noted ν.

Least fixpoint computation

The procedure to compute the least fixpoint of a predicate transformer τ starts with an empty set. It then applies τ on this set until no change occurs. The least fixpoint is the limit of the following increasing sequence:

$$\mathit{False} \subseteq \tau(\mathit{False}) \subseteq \tau(\tau(\mathit{False})) \subseteq \tau(\tau(\tau(\mathit{False}))) \;...$$

where False is the Boolean formula representing the empty set.

Figure 5.1 illustrates this computation. From the empty set, new elements are added using the predicate transformer until a point is reached where no new elements are introduced by applying the predicate transformer.

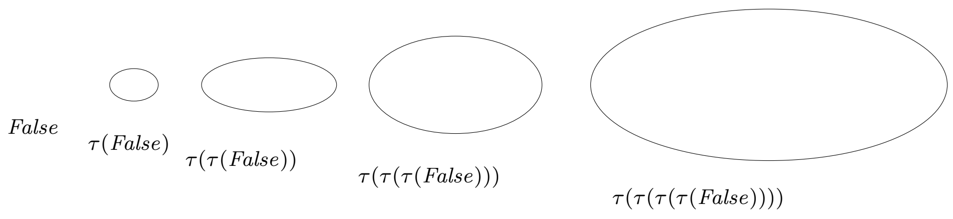
False
$\tau(\mathit{False})$
$\tau(\tau(\mathit{False}))$
$\tau(\tau(\tau(\mathit{False})))$
$\tau(\tau(\tau(\tau(\mathit{False}))))$

Figure 5.1: Least fixpoint computation.

Greatest fixpoint computation

The procedure to compute the greatest fixpoint of a predicate transformer τ starts with the entire set S and applies τ until no change occurs. The greatest fixpoint is the limit of the following decreasing sequence:

$$...\; \tau(\tau(\tau(\mathit{True}))) \subseteq \tau(\tau(\mathit{True})) \subseteq \tau(\mathit{True}) \subseteq \mathit{True}$$

where True is the Boolean formula representing the entire set S.

Figure 5.2 illustrates this computation. From the largest set, elements are removed using the predicate transformer until no more elements are removed by applying the predicate transformer.

CTL operators as fixpoints

All CTL operators can be expressed as fixpoints. The idea is to identify a formula f with the set of states of a Kripke structure M satisfying f, that is, we identify $\mathcal{P}(S)$ with the set $\{s|M, s \models f\}$ for function τ above. We then characterise the operators as follows:

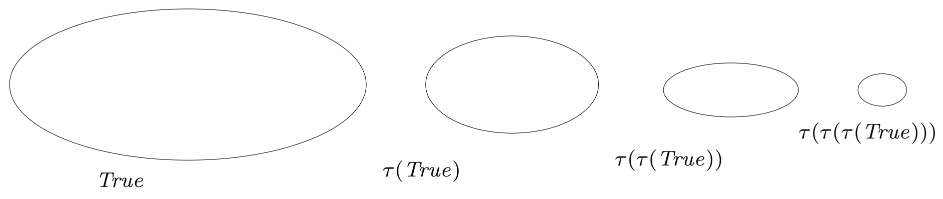

Figure 5.2: Greatest fixpoint computation.

- $\mathbf{AF}f = \mu Z.f \vee \mathbf{AX}\ Z$
- $\mathbf{EF}f = \mu Z.f \vee \mathbf{EX}\ Z$
- $\mathbf{AG}f = \nu Z.f_1 \wedge \mathbf{AX}\ Z$
- $\mathbf{EG}f = \nu Z.f_1 \wedge \mathbf{EX}\ Z$
- $\mathbf{A}[f_1 \mathbf{U} f_2] = \mu Z.f_2 \vee (f_1 \wedge \mathbf{AX}\ X)$
- $\mathbf{E}[f_1 \mathbf{U} f_2] = \mu Z.f_2 \vee (f_1 \wedge \mathbf{EX}\ X)$

Intuitively, least fixpoints correspond to properties that must hold at some time in the future while greatest fixpoints correspond to properties that must always hold. For instance, $\mathbf{AG}f$ is true if f is true now *and* $\mathbf{AG}f$ holds for *all* successor states. In contrast, $\mathbf{EF}f$ is true if f is true now *or* $\mathbf{EF}f$ holds in *one* successor state.

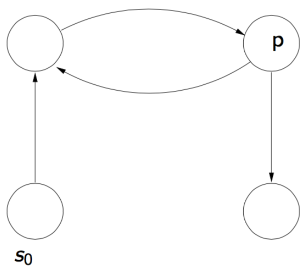

Figure 5.3: An example system.

To illustrate these fixpoints, consider the transition system in Figure 5.3. Let us check $\mathbf{EF}p$. We translate it to an equivalent property involving a predicate transformer. We will compute the least fixpoint of predicate transformer $\tau(Z) = p \vee \mathbf{EX}\ Z$. We start with an empty set and will add states satisfying p or with successors in Z.

Let us name s_2 the state satisfying p. As initially Z is empty, there is no state with successors in the empty set. We therefore only add s_2 to the set. We have:

$$Z = \tau^1(\mathit{false}) = \{s_2\}$$

No other state satisfies p. So, we can only add states with s_2 as a successor. There is only one such state. Let us name it s_1. Now we have:

$$Z = \tau^2(\mathit{false}) = \{s_1, s_2\}$$

There is one state with a successor in $\{s_1, s_2\}$. This state is s_0. We have:

$$Z = \tau^3(\mathit{false}) = \{s_0, s_1, s_2\}$$

There is no state with a successor in $\{s_0, s_1, s_2\}$. So, we have:

$$Z = \tau^4(\mathit{false}) = \tau^3(\mathit{false})$$

Because s_0 is in Z, we can conclude that the system satisfies $\mathbf{EF}p$.

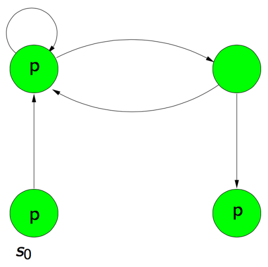

Figure 5.4: Another example system.

Consider the example in Figure 5.4. Consider the formula $\mathbf{EG}p$. We translate it to an equivalent formula involving a predicate transformer. We will compute the greatest fixpoint of predicate transformer $\tau(Z) = p \wedge \mathbf{EX}\ Z$. We start with the entire set of states. We will only keep states that do satisfy p and have a successor in Z. Let us name all states starting from the initial state s_0, that is, we define $S = \{s_0, s_1, s_2, s_3, s_4\}$.

Initially, $Z = \{s_0, s_1, s_2, s_3, s_4\}$. There is one state no satisfying p (s_2) and one state without a successor in Z (s_3). These states are removed. We have:

$$Z = \tau^1(S) = \{s_0, s_1\}$$

Now, all states satisfy p and have a successor in Z. We have:

$$Z = \tau^2(S) = \tau^1(S) = \{s_0, s_1\}$$

Because s_0 is in S we conclude that the system satisfies $\mathbf{EG}p$.

5.1.2 Fixpoints and BDDs

Let us consider existential properties, that is, CTL formulas of the form $\mathbf{EX}f$, $\mathbf{E}[f\mathbf{U}g]$, and $\mathbf{EG}f$. All fixpoints definitions of these formulas mention the operator $\mathbf{EX}$.

We now give a bit more details about the $\mathbf{EX}$ operator and its encoding as a quantified Boolean formula. Given a set of states satisfying predicate P, one can compute the

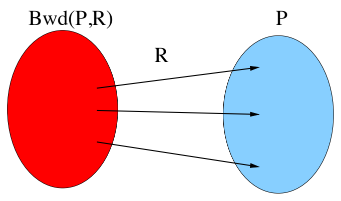

Figure 5.5: Backward computation.

set of states that can reach P with one application of the transition function T (noted R in the figure). Figure 5.5 illustrates this backward computation.

Formally, the backward computation is defined as follows:

$$Bwd(P,T) = \{\overline{v} | \exists \overline{v}' : \overline{v}' \in P \wedge (\overline{v}, \overline{v}') \in T\}$$

This can be encoded as a quantified Boolean formula:

$$f(\overline{v}) = \exists \overline{v}' : (P(\overline{v}) \wedge T(\overline{v}, \overline{v}'))$$

This Boolean can in turn be encoded as a BDD.

The backward operation exactly corresponds to the $\mathbf{EX}$ operator. The states satisfying $\mathbf{EX}\ p$ are the states with a successor state satisfying p. Starting from all states satisfying p, one computes all states satisfying $\mathbf{EX}\ p$ by applying the backward computation once.

This gives a symbolic encoding of the main operator for CTL model-checking. Checking $\mathbf{EU}$ properties is based on the fixpoint characterisation:

$$\mathbf{E}[f_1 \mathbf{U} f_2] = \mu Z.f_2 \vee (f_1 \wedge \mathbf{EX}\ X)$$

The idea is to compute the least fixpoint as a sequence of sets of states:

$$Z_0, Z_2, ...,$$

This converges to the solutions of $\mathbf{E}[f_1 \mathbf{U} f_2]$ in a finite number of steps. Using the Boolean encoding of $\mathbf{EX}$ above, we can compute the next approximation Z_{i+1} from the ROBDDs of Z_i, f_1, f_2 and the transition relation. After each iteration, equality $Z_i = Z_{i+1}$ is checked to know if convergence has been achieved. The canonical property for ROBDDs is here key.

For checking properties of the form $\mathbf{EG}$, the idea is to compute a greatest fixpoint:

$$\mathbf{EG}f = \nu Z.f_1 \wedge \mathbf{EX}\ Z$$

The procedure is similar to the previous case.

5.1.3 Witness and counter-examples

Counterexamples and witnesses constitute an essential feature of model-checking. The capability of exhibiting witnesses and counterexamples is crucial in the analysis of designs to find and correct errors.

Witnesses are naturally associated to "existential" formula, i.e., $\mathbf{EX}$,$\mathbf{EG}$, and $\mathbf{EU}$. The intuition is that $(\mathbf{EX}\ p)$ is true in state s if there exists a successor state satisfying property p. A witness is simply a path starting from s with one successor state satisfying p. Formally, a witnesspath for $(\mathbf{EX}\ p)$ in state s_0 is a path $\pi = s_0, s_1$ such that $s_1 \models p$. Figure 5.6 illustrates such a path.

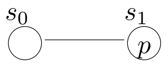

Figure 5.6: A witness for $\mathbf{EX}$.

Witnesses to $\mathbf{EU}$ formulas are similar. For instance, a witness for the formula $\mathbf{E}(p\ \mathbf{U}\ q)$ would be a path with a prefix where states satisfy p and the final state of the path satisfies q. Formally, a witness for $\mathbf{E}(p\ \mathbf{U}\ q)$ in state s_0 is a path $\pi = \pi_0, s_k$ such that $\forall j, \pi_0^j \models p$ and $s_k \models q$. Figure 5.7 illustrates such a path.

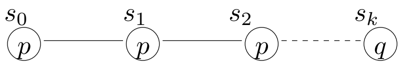

Figure 5.7: A witness for $\mathbf{EU}$.

Witnesses for $\mathbf{EG}$ have a specific form, named a *lasso*. Formula $(\mathbf{EG}\ p)$ is true if there exists an execution path where p is true in all states. The idea is that such a path must have a cycle and all states of the path must satisfy p. Formally, a witness path for $(\mathbf{EG}\ p)$ is a path $\pi = \pi_0, s_k, s_i$ such that $\forall j, \pi_0^j \models p$ and $s_k \models p$ and $s_i \in \pi_0$. Figure 5.8 illustrates such a path.

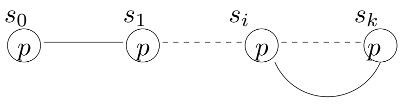

Figure 5.8: A witness for $\mathbf{EG}$.

A witness of an existentially quantified formula is a counter-example to the dual universally quantified formula. The next section about model checking with SAT heavily relies on this duality between witness and counter-examples.

5.2 Bounded model checking with SAT

The presentation is largely based on the original paper by Biere et al. [1] with some small modifications. The presentation of the completeness threshold differs more significantly from the original paper.

5.2.1 Principle

The main idea behind Bounded Model Checking (BMC) is to prove a property by showing that there exists no finite counter-examples to it. There actually are two ideas here: (1) prove that no counter-example exists and (2) restrict to finite paths.

Assume one wants to prove an LTL property of the form $\mathbf{G}p$. The semantics of LTL are such that this property holds if and only it holds on all paths. The negation of this property is $\neg\mathbf{G}p = \mathbf{F}\neg p$. This negation holds if all paths satisfy $\mathbf{F}\neg p$. We can also say that this negation *does not* hold if one can find *a single* path satisfying $\mathbf{F}p$. The idea of BMC is to create a propositional formula that is satisfiable if and only if such a path exists. If the formula is satisfiable, a counter-example to the original formula ($\mathbf{G}p$) is found. Otherwise, no counter-example exists and the formula holds. The same holds for an LTL formula of the form $\mathbf{F}p$. The BMC approach will create a propositional formula that is satisfiable if and only if a path satisfying $\mathbf{G}\neg p$ exists. If no such path exists, the property holds. Otherwise, a counter-example is found.

The restriction to finite paths does not reduce the power of BMC. We will see that if the bound is large enough, a formula is true in the bounded semantics if and only if it is true in the unbounded semantics. Here also, the fact that systems have finitely many states is crucial. The idea is that if paths are long enough, they capture all possible reachable states. In practice, the bound required for this is too large and BMC is used as a *bug finding* technique rather than a proof technique. This lack of completeness has not prevented BMC to become the de facto standard verification method.

5.2.2 Bounded semantics

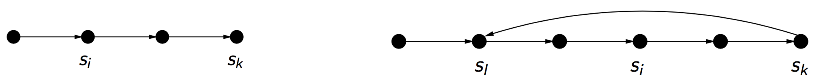

Figure 5.9: A path without a loop and a path with a loop.

Figure 5.9 shows a path without a loop and a path with a loop. Each path is of length k. The path without a loop can be a witness for properties of the form $\mathbf{F}p$. Indeed, if one of the states in the path satisfies p then $\mathbf{F}p$ holds. Note that extending the path beyond k does not modify the validity of the property. A path without a loop cannot be a witness for properties of the form $\mathbf{G}p$. If all states of a path without a loop satisfy p, nothing is known about the states after the end of the path. We cannot conclude that $\mathbf{G}p$ is valid. In contrast, a path with a loop is a witness for $\mathbf{G}p$ if all states satisfy p. Here the loop shows that p is true even if the path is unfolded infinitely often.

Definition 5.2.1. For $l \leq k$ we call a path π a (k, l)-loop if there exists a transition from $\pi(k)$ to $\pi(l)$ and $\pi = u \cdot v^{\omega}$, where u is a prefix without a loop $\pi(0)...\pi(l)$ and v is the loop prefix $\pi(l)...\pi(k)$. We call π a k-loop if there exists an $l \in \mathbb{N}$ with $l \leq k$ for which π is a (k, l)-loop.

The bounded semantics only consider the $k + 1$ states of a path to determine the validity of a formula along that path. In the case of a k-loop, the original LTL semantics are preserved. In that case, the semantics are the same as defined in Chapter 4 for LTL. See Definition 4.2.1.

In the case of a path without a loop, the formula has to be valid before the end of the path. That is, we should find an index along the path such that the property holds from that index. Because the $(k+1)$-th state of a path π has no successor, we can no longer define the semantics recursively over suffixes of path π. Instead, we will make the position in a path of a state explicit and define the semantics for an arbitrary position along the path. We shall use the notation $\pi \models_k^i f$ to denote that f is valid at position i along path π of length k.

Definition 5.2.2. Let $k \in \mathbb{N}$ and let π be a path that is not a k-loop. An LTL formula h holds for path π, notation $\pi \models_k h$ if and only if there exists an $i < k$ such that $\pi \models_k^i h$ where:

$$\begin{array}{lllc}
\pi \models_k^i & p & \text{iff} & p \in L(\pi(i)) \\
\pi \not\models_k^i & \neg p & \text{iff} & p \notin L(\pi(i)) \\
\pi \models_k^i & f \wedge g & \text{iff} & \pi \models_k^i f \text{ and } \pi \models_k^i g \\
\pi \models_k^i & \mathbf{X}f & \text{iff} & i < k \text{ and } \pi \models_k^{i+1} f \\
\pi \models_k^i & f\mathbf{U}g & \text{iff} & \exists j. i \leq j \leq k. \pi \models_k^j g \wedge \forall l. i \leq l < j. \pi \models_k^l f \\
\pi \models_k^i & \mathbf{G}p & \text{iff} & \text{is always false}
\end{array}$$

As usual, other connectives (like $\vee$) and operators (like $\mathbf{F}$) can be constructed from the connectives and operators of this definition. Note that because $\mathbf{G}f$ is always false, the duality between $\mathbf{F}$ and $\mathbf{G}$ – namely $\neg\mathbf{F}f = \mathbf{G}\neg f$ – no longer holds.

Lemma 5.2.3. Let h be an LTL formula and π a path, $\pi \models_k h$ implies $\pi \models h$.

Lemma 5.2.4. Let h be an LTL formula and M a Kripke structure. If $M \models \mathbf{E}h$, there exists a $k \in \mathbb{N}$ such that $M \models_k \mathbf{E}h$.

We now prove that if k is large enough, the bounded and unbounded semantics coincide.

Theorem 5.2.5. Let f be an LTL formula, M a Kripke structure.

$$M \models \mathbf{E}f \leftrightarrow \exists k \in \mathbb{N}. M \models_k \mathbf{E}f$$

5.2.3 Propositional translation

The previous section gave bounded semantics for LTL validity and proved that these semantics coincide with the unbounded case. We now will show how to translate validity checking to a propositional formula.

Valid paths

The first part of the translation is to constrain the solutions to valid paths, that is, paths that are according to the transition relation. This is achieved by first ensuring that s_0 is an initial state and second by ensuring that there is a transition for every successor in the path. Formally, the first part of the translation produces the following:

$$[\![M]\!]_k := I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \tag{5.1}$$

Loop conditions

A loop condition simply is a constraint stating that there exists a transition from the state at position k back to some previous state in the path at position l.

Definition 5.2.6. For $k, l \in \mathbb{N}$, let ${}_l\mathbf{L}_k := T(s_k, s_l)$ and $\mathbf{L}_k = \bigvee_{l=0}^{k} {}_l\mathbf{L}_k$.

Paths without a loop

Definition 5.2.7 below shows the variables created over paths without a loop. We can paraphrase them as follows. Let i be the current position in a path of length k. Proposition p is encoded by its value in the state at position i. Negation of p is the negated value in the state at position i. Conjunction and disjunction are encoded as the truth value of the conjunction or disjunction of the sub-formulas. On paths without a loop, the globally temporal operation ($\mathbf{G}$) is always false ($\bot$). Starting at position i, the eventually temporal operation ($\mathbf{F}$) is encoded as a disjunction over the indices between i and k. This encodes the fact that $\mathbf{F}g$ holds from i if there exists a state between position i and k where g holds. The encoding of the until operation also directly follows the bounded semantics. Sub-formula g must hold at a position j before k. For all positions before j, sub-formula f must hold.

Definition 5.2.7. Let f and g be LTL formulas, $k, i \in \mathbb{N}$, with $i \le k$.

$$\begin{array}{rcl}
[\![p]\!]_k^i & := & p(s_i) \\
[\![\neg p]\!]_k^i & := & \neg p(s_i) \\
[\![f \wedge g]\!]_k^i & := & [\![f]\!]_k^i \wedge [\![g]\!]_k^i \\
[\![f \vee g]\!]_k^i & := & [\![f]\!]_k^i \vee [\![g]\!]_k^i \\
[\![\mathbf{G}f]\!]_k^i & := & \bot \\
[\![\mathbf{F}g]\!]_k^i & := & \bigvee_{j=i}^{k} [\![g]\!]_k^i \\
[\![\mathbf{X}f]\!]_k^i & := & \text{if } i < k \text{ then } [\![f]\!]_k^{i+1} \text{ else } \bot \\
[\![f\mathbf{U}g]\!]_k^i & := & \bigvee_{j=i}^{k} \left([\![g]\!]_k^j \wedge \bigwedge_{n=i}^{j-1} [\![f]\!]_k^n \right)
\end{array}$$

Paths with a loop

First let us define the successor of a position in a loop.

Definition 5.2.8. Let $k, l, i \in \mathbb{N}$, with $i, l \le k$. Define the successor $succ(i)$ of i in a (k, l)-loop as $succ(i) := i + 1$ for $i < k$ and $succ(i) = l$ for $i = k$.

Definition 5.2.9 shows the encoding over paths with a (k, l)-loop. Let i be the position in a path of length k. Proposition p and its negation are encoded as the values of p or its negation is the state at position i. Conjunction and disjunction are encoded as the conjunction or disjunction of their sub-formulas. The globally temporal operation ($\mathbf{G}$) encodes the fact that all states must satisfy the sub-formula. This sub-formula must hold before the loop part of the path – that is for indices before l, or when i is smaller than l – and for all states in the loop[1]. In the encoding of the eventually operator ($\mathbf{F}$), conjunctions are replaced by disjunctions. The encoding of the until operator is more complicated. The first disjunct equals to the case of paths without a loop. This encodes the fact that the until might already hold in the prefix before the loop. The second part encodes $f\mathbf{U}g$ for positions i in the loop. Say g holds at position j in the loop. Then, f must hold at all position after j, but also at all positions before j in the loop.

Definition 5.2.9. Let f and g be LTL formulas, $k, l, i \in \mathbb{N}$, with $l, i \leq k$.

$$\begin{array}{rcl}
{}_l[\![p]\!]_k^i & := & p(s_i) \\
{}_l[\![\neg p]\!]_k^i & := & \neg p(s_i) \\
{}_l[\![f \wedge g]\!]_k^i & := & {}_l[\![f]\!]_k^i \wedge {}_l[\![g]\!]_k^i \\
{}_l[\![f \vee g]\!]_k^i & := & {}_l[\![f]\!]_k^i \vee {}_l[\![g]\!]_k^i \\
{}_l[\![\mathbf{G}f]\!]_k^i & := & \bigwedge_{j=min\{i,l\}}^k {}_l[\![f]\!]_k^i \\
{}_l[\![\mathbf{F}g]\!]_k^i & := & \bigvee_{j=min\{i,l\}}^k {}_l[\![g]\!]_k^i \\
{}_l[\![\mathbf{X}f]\!]_k^i & := & {}_l[\![f]\!]_k^{succ(i)} \\
{}_l[\![f\mathbf{U}g]\!]_k^i & := & \bigvee_{j=i}^k \left({}_l[\![g]\!]_k^j \wedge \bigwedge_{n=i}^{j-1} {}_l[\![f]\!]_k^n\right) \vee \\
 & & \bigvee_{j=l}^{i-1} \left({}_l[\![g]\!]_k^j \wedge \bigwedge_{n=i}^k {}_l]\!]f[\![_k^n \wedge \bigwedge_{n=l}^{j-1} {}_l[\![f]\!]_k^n\right)
\end{array}$$

General translation

Finally, the general translation consists in the conjunction of the constraints over path validity and the existence of a solution over either paths without a loop or paths with a loop. The part for paths with loop tries all possible start l of loops.

Definition 5.2.10. Let f be an LTL formula, M a Kripke structure and $k \in \mathbb{N}$:

$$[\![M, f]\!] := [\![M]\!]_k \wedge \left(\left(\neg\mathbf{L}_k \wedge [\![f]\!]_k^0\right) \vee \bigvee_{l=0}^{k} \left({}_l\mathbf{L}_k \wedge {}_l[\![f]\!]_k^0\right)\right)$$

Theorem 5.2.11. $[\![M, f]\!]$ is satisfiable if and only if $M \models_k \mathbf{E}f$.

Corollary 1 $M \models \mathbf{A}\neg f$ *if and only if* $[\![M, f]\!]$ *is unsatisfiable for all* $k \in \mathbb{N}$.

5.2.4 Completeness threshold

The previous section concluded with a Corollary stating that a formula holds in the LTL semantics if and only if it holds for all possible bounds in the bounded semantics.

[1]Note that the slides presented in the lecture omitted the initial prefix before the loop.

In practice, this means that BMC might never terminate. We now define a bound such that if there exists no witness path for all lengths within this bound, there is also no path longer than the bound. Formally, if $M \not\models_k \mathbf{E}f$ for all k within the bound, we conclude that $M \not\models \mathbf{E}f$.

For every *finite* system M and given a property f, there exists a number $\mathcal{CT}$ such that the absence of errors up to $\mathcal{CT}$ prove f for M. For instance, the longest "shortest" path from an initial state to any reachable state. We call $\mathcal{CT}$ the *completeness threshold* of M w.r.t. f and the translation scheme.

Consider formulas of the form $\mathbf{G}f$. The completeness threshold is the minimal number of steps to reach all states. We call it the *reachability diameter* of M. It is defined as follows:

$$rd(M) := min\{i|\forall s_0, ..., s_n.\exists s'_0, ..., s'_t, t \leq i.$$

$$I(s_0) \wedge \bigwedge_{j=0}^{i} T(s_j, s_{j+1}) \Rightarrow \left(I(s'_0) \wedge \bigwedge_{j=0}^{t-1} T(s'_j, s'_{j+1}) \wedge s'_t = s_n \right)\}$$

The left part of the implication expresses the minimal number of steps to reach n states. The right part states that these states can also be reached in t steps.

We still need to fix n in the above definition. A simple but not very practical option is the size of the state space, that is $n = 2^{|V|}$, where V is the set of Boolean variables of the Kripke structure. A better option is to take $n = i + 1$, that is, we check that all states that can be reached in i steps are also reachable in $i + 1$ steps.

Definition 5.2.12. (Reachability Diameter)

$$rd(M) := min\{i|\forall s_0, ..., s_{i+1}.\exists s'_0, ..., s'_i.$$

$$I(s_0) \wedge \bigwedge_{j=0}^{i} T(s_j, s_{j+1}) \Rightarrow \left(I(s'_0) \wedge \bigwedge_{j=0}^{t-1} T(s'_j, s'_{j+1}) \wedge \bigvee_{j=0}^{i} s'_j = s_{i+1} \right)\}$$

This formula involves alternation of quantifies ($\forall x.\exists y$). This is hard to compute on realistic problem. A possibility is to over-approximate the reachability diameter. This approximation is called the *recurrence diameter*, noted rdr. This diameter computes the longest loop-free path. The idea is that every shortest path is also a loop-free path.

Definition 5.2.13. (Recurrence Diameter)

$$rd(M) := max \left\{ i|\exists s_0, ..., s_i.I(s_0) \wedge \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k \right\}$$

5.2.5 The SMUTE example

We will illustrate the encoding of a model and a property into a SAT problem on a simple example of a mutual exclusion problem. Consider the code shown in Figure 5.10. This example has two processes. Each process tries to access a critical resource. Once

```
 process A                          process B
      forever                            forever
      A.pc = 0                           B.pc = 0
      wait for B.pc = 0                  wait for A.pc = 0
      A.pc = 1                           B.pc = 1
      access resource                    access resource
      end forever                        end forever
end process                        end process
```

Figure 5.10: Two processes accessing a shared resource.

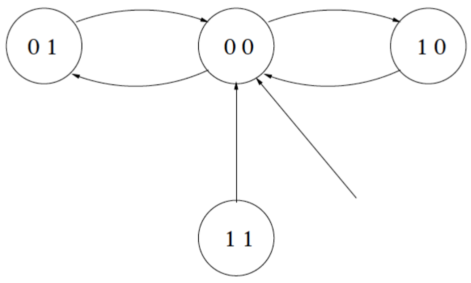

Figure 5.11: Kripke structure for the SMUTE example.

a process acquires a resource, it releases it. The choice to whom the resource is given is non-deterministic. The corresponding Kripke structure of this example is shown in Figure 5.11. The state space is $S = \{0, 1\}^2$. The transition relation is $T \subseteq S^2$.

The property we want to prove that is that it is never the case that both processes access the resource simultaneously. Assume two Boolean variables a and b. Variable a is true when process A owns the resource. It is false otherwise. Variable b is true when process B owns the resources. It is false otherwise. Formally we want to prove the following:

$$\mathbf{G}\neg(a \wedge b)$$

The idea of BMC is to show that there does not exist any path *not* satisfying this property, and therefore the property holds for the model; or a path violating the property is found and is returned as a counter-example. This means that BMC will create a CNF formula that is UNSAT if and only if no counter-example exists, that is, we will encode in CNF the existence of a path satisfying the following property:

$$\mathbf{F}(a \wedge b)$$

The transition relation induced by this example is illustrated by the Kripke structure shown in Figure 5.11. The formal definition of the transition relation is given by the following formula. Variables a' and b' represent the values of the variables at the next step.

$$\begin{array}{rcl} T(a, b, a', b') = & & \neg a \wedge \neg b \wedge \neg a' \wedge b' \\ & \vee & \neg a \wedge \neg b \wedge a' \wedge \neg b' \\ & \vee & \neg a \wedge b \wedge \neg a' \wedge \neg b' \\ & \vee & a \wedge \neg b \wedge \neg a' \wedge \neg b' \\ & \vee & a \wedge b \wedge \neg a' \wedge \neg b' \end{array}$$

Objective

The complete encoding is obtained by instantiating Definition 5.2.10 with k= 2, $f = \mathbf{F}(a \wedge b)$, and M is the Kripke structure shown in Figure 5.11:

$$[\![M]\!]_2 \wedge \left(\left(\neg\mathbf{L}_2 \wedge [\![\mathbf{F}(a \wedge b)]\!]_2^0\right) \vee \bigvee_{l=0}^{2} \left({}_l\mathbf{L}_2 \wedge \ {}_l[\![\mathbf{F}(a \wedge b)]\!]_2^0\right) \right)$$

Initial state and unfolding

The initial state is defined by having the two variables set to false. Predicate I recognising the initial state is defined as follows:

$$I(a, b) = \neg a \wedge \neg b$$

The first part of the SAT translation is to unfold the transition relation the number of times given by the bound k at which the proof is attempted. Assume $k = 2$.

We therefore needs to unfold the transition relation twice. We need variables to represent the state at each iteration. Let a_0 and b_0 represent the initial state and variables a_1, b_1, a_2, b_2 represent the first and second iterations. The unfolding is expressed by the following formula:

$$[\![M]\!]_2 = I(a_0, b_0) \wedge T(a_0, b_0, a_1, b_1) \wedge T(a_1, b_1, a_2, b_2)$$

Let us expand the transition from the initial state:

$$\begin{array}{rll} T(a_0, b_0, a_1, b_1) = & & \neg a_0 \wedge \neg b_0 \wedge \neg a_1 \wedge b_1 \\ & \vee & \neg a_0 \wedge \neg b_0 \wedge a_1 \wedge \neg b_1 \\ & \vee & \neg a_0 \wedge b_0 \wedge \neg a_1 \wedge \neg b_1 \\ & \vee & a_0 \wedge \neg b_0 \wedge \neg a_1 \wedge \neg b'_1 \\ & \vee & a_0 \wedge b_0 \wedge \neg a_1 \wedge \neg b_1 \end{array}$$

When taking the conjunction with the restriction to the initial state, all terms except the last one will contain a variable and its negation. Such terms are contradictions and can disappear. After these simplifications, we obtain the following:

$$I(a_0, b_0) \wedge T(a_0, b_0, a_1, b_1) = \neg a_0 \wedge \neg b_0 \wedge (a_1 \wedge \neg b_1 \vee \neg a_1 \wedge b_1)$$

This corresponds to the two transitions going out of the initial state: either A wins the resource or B wins the resource. Formally, only two states are reachable from the initial one and we have either $a_1 \wedge \neg b_1$ or $\neg a_1 \wedge b_1$.

Let us expand the last transition:

$$\begin{array}{rll} T(a_1, b_1, a_2, b_2) = & & \neg a_1 \wedge \neg b_1 \wedge \neg a_2 \wedge b_2 \\ & \vee & \neg a_1 \wedge \neg b_1 \wedge a_2 \wedge \neg b_2 \\ & \vee & \neg a_1 \wedge b_1 \wedge \neg a_2 \wedge \neg b_2 \\ & \vee & a_1 \wedge \neg b_1 \wedge \neg a_2 \wedge \neg b_2 \\ & \vee & a_1 \wedge b_1 \wedge \neg a_2 \wedge \neg b_2 \end{array}$$

We just saw that either $a_1 \wedge \neg b_1$ or $\neg a_1 \wedge b_1$. So, the only terms remaining in the conjunction of this term with the previous transitions are the third and fourth ones. We then obtain the following:

$$\begin{array}{rl} & I(a_0, b_0) \wedge T(a_0, b_0, a_1, b_1) \wedge T(a_1, b_1, a_2, b_2) \\ = & \neg a_0 \wedge \neg b_0 \wedge (a_1 \wedge \neg b_1 \vee \neg a_1 \wedge b_1) \wedge \neg a_2 \wedge \neg b_2 \end{array}$$

So, we have:

$$[\![M]\!]_2 = \neg a_0 \wedge \neg b_0 \wedge (a_1 \wedge \neg b_1 \vee \neg a_1 \wedge b_1) \wedge \neg a_2 \wedge \neg b_2$$

Loop conditions

The final encoding needs to distinguish between paths with a loop from paths without a loop. The loop conditions are used to express the fact that there exists a loop or not. Because $k = 2$, there are three loop conditions, expressing the fact that either the path loops from position 2 to 0, 1, or 2, that is, the path ends with a self loop. Formally, a loop condition simply states that there exists a transition between the states of the loop. We obtain the following three loop conditions:

$$\mathbf{L}_2 = T(a_2, b_2, a_0, b_0) \vee T(a_2, b_2, a_1, b_1) \vee T(a_2, b_2, a_2, b_2)$$

This expresses that either there is 2,0-loop, or a 2-1-loop, or a 2,2-loop.

Let us look at each loop separation. First consider the first loop:

$$_0\mathbf{L}_2 = T(a_2, b_2, a_0, b_0)$$

By simply expanding the transition relation, we obtain the following:

$$\begin{array}{rcl} T(a_2, b_2, a_0, b_0) = & & \neg a_2 \wedge \neg b_2 \wedge \neg a_0 \wedge b_0 \\ & \vee & \neg a_2 \wedge \neg b_2 \wedge a_0 \wedge \neg b_0 \\ & \vee & \neg a_2 \wedge b_2 \wedge \neg a_0 \wedge \neg b_0 \\ & \vee & a_2 \wedge \neg b_2 \wedge \neg a_0 \wedge \neg b_0 \\ & \vee & a_2 \wedge b_2 \wedge \neg a_0 \wedge \neg b_0 \end{array}$$

The final encoding will consider the conjunction of the above with $[\![M]\!]_2$. However, we know from $[\![M]\!]_2$ that a_0, b_0, b_2, a_2 are all false. We can therefore conclude that no transition from state 2 to state 0 is possible. Thus, we have:

$$_0\mathbf{L}_2 = \bot$$

Similarly, we also conclude that no transition from state 2 to itself is possible:

$$_2\mathbf{L}_2 = \bot$$

Finally, we expand the remaining loop condition $_1\mathbf{L}_2$:

$$\begin{array}{rcl} T(a_2, b_2, a_1, b_1) = & & \neg a_2 \wedge \neg b_2 \wedge \neg a_1 \wedge b_1 \\ & \vee & \neg a_2 \wedge \neg b_2 \wedge a_1 \wedge \neg b_1 \\ & \vee & \neg a_2 \wedge b_2 \wedge \neg a_1 \wedge \neg b_1 \\ & \vee & a_2 \wedge \neg b_2 \wedge \neg a_1 \wedge \neg b_1 \\ & \vee & a_2 \wedge b_2 \wedge \neg a_0 \wedge \neg b_0 \end{array}$$

Injecting the knowledge that a_2 and b_2 must be both false, this further simplifies to:

$$\begin{array}{rcl} T(a_2, b_2, a_1, b_1) = & & \neg a_2 \wedge \neg b_2 \wedge \neg a_1 \wedge b_1 \\ & \vee & \neg a_2 \wedge \neg b_2 \wedge a_1 \wedge \neg b_1 \end{array}$$

We can simplify to obtain:

$$_1\mathbf{L}_2 = \neg a_2 \wedge \neg b_2 \wedge (\neg a_1 \wedge b_1 \vee a_1 \wedge \neg b_1)$$

Finally, we also have $\mathbf{L}_2 = {}_1\mathbf{L}_2$, as there is only one possible loop according to the transition relation.

Encoding the property

As explained before, we are encoding the existence of a path satisfying $\mathbf{F}(a \wedge b)$. We first need to encode this property for paths with a loop. The only possible loop is a $(2, 1)$-loop. Here is the encoding for this loop:

$${}_1[\![\mathbf{F}(a \wedge b)]\!]_2^0 = a_0 \wedge b_0 \vee a_1 \wedge b_1 \vee a_2 \wedge b_2$$

For paths without a loop, the encoding is simply that the property must hold at one position in the path:

$$[\![\mathbf{F}(a \wedge b)]\!]_2^0 = a_0 \wedge b_0 \vee a_1 \wedge b_1 \vee a_2 \wedge b_2$$

Complete encoding

We now have all the elements to complete the encoding. Let us first look at the part over paths with a loop:

$$\bigvee_{l=0}^{2} \left({}_l\mathbf{L}_2 \wedge \ {}_l[\![\mathbf{F}(a \wedge b)]\!]_2^0\right)$$

As there is only one loop, this reduces to:

$$\left({}_1\mathbf{L}_2 \wedge \ {}_1[\![\mathbf{F}(a \wedge b)]\!]_2^0\right)$$

This expands to:

$$\neg a_2 \wedge \neg b_2 \wedge (\neg a_1 \wedge b_1 \vee a_1 \wedge \neg b_1) \wedge (a_1 \wedge b_1 \vee a_2 \wedge b_2)$$

This actually expands to $\perp$. We are left with the part considering paths without a loop. This means that the entire encoding reduces to:

$$[\![M]\!]_2 \wedge \neg \mathbf{L}_2 \wedge [\![\mathbf{F}(a \wedge b)]\!]_2^0$$

By simply expanding the first and last terms of the conjunct, this reduces to $\perp$. This proves that no witness for $\mathbf{F}(a \wedge b)$ exists. So, $\mathbf{G}\neg(a \wedge b)$ holds.

Completeness Threshold

The SMUTE Kripke structure has only two variables. The completeness threshold is therefore $n = 2^2 = 4$. We therefore know that if a property is true for all $k \leq 4$, then it is true in the unbounded semantics. We have seen that this bound can be improved by using a *reachability diameter* or a *recurrence diameter*. We will now illustrate these two notions on the SMUTE example.

Reachability diameter The definition of the reachability diameter (Definition 5.2.12) searches for the minimum i such that all states can be reached within i steps. Let first try with $i = 0$. We then obtain the following:

$$\forall s_0, s_1.\exists s_0'.I(s_0) \wedge T(s_0, s_1) \Rightarrow (I(s_0') \wedge s_0' = s_1)$$

This formula means that for all states that can be reached in one step can also be reached in zero step. This is not true for the SMUTE example. The reason is that the equality $s_0' = s_1$ expands to:

$$\neg a_0' \wedge \neg b_0' \wedge ((a_1 \wedge \neg b_1) \vee (\neg a_1 \wedge b_1))$$

This is UNSAT. Let's try with $i = 1$. We now get:

$$\forall s_0, s_1, s_2.\exists s_0', s_1'.I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \Rightarrow (I(s_0') \wedge T(s_0', s_1') \wedge (s_0' = s_2 \vee s_1' = s_2))$$

This is SAT because we have:

$$s_0' = \neg a_0' \wedge \neg b_0' \text{ and } s_2 = \neg a_2 \wedge \neg b_2$$

Conclusion: the reachability diameter for the SMUTE example is $n = 1 + 1 = 2$. So, doing BMC at depth 2 actually proves the property.

Exercise 5.2.14. The completeness threshold is *safe* bound in the sense that it might too large. Can you make an argument that a proof of the original safety property of SMUTE with $k = 1$ would be possible?

Recurrence diameter Definition 5.2.13 proposes an alternative to the reachability diameter by computing the longest loop-free path. This definition searches for the maximum i. Let's first try with $i = 2$. The definition rewrites to:

$$\exists s_0, s_1, s_2.I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge s_0 \neq s_1 \wedge s_0 \neq s_2 \wedge s_1 \neq s_2$$

This checks that there exists a loop-free path of length 2. This is not true for the SMUTE example, because we have $s_0 = s_2$.

Let now try $i = 1$. We obtain:

$$\exists s_0, s_1.I(s_0) \wedge T(s_0, s_1) \wedge s_0 \neq s_1$$

This is actually SAT because indeed $s_0 = \neg a_0 \wedge \neg b_0$ and $s_1 = (a_1 \wedge \neg b_1) \vee (\neg a_1 \wedge b_1)$. So, the recurrence diameter is $n = 1 + 1 = 2$.

5.2.6 Exercises

Exercise 5.2.15. As an exercise, you can add a transition from state 01 to state 11 in the SMUTE example in Figure 5.11 and do the BMC example again.

Exercise 5.2.16. Apply BMC to prove $\mathbf{F}(a \wedge b$ on the SMUTE example. At what depth must you perform BMC? Would $k = 1$ works in that case?