
# Course Script

## IN 5110: Specification and Verification of Parallel Systems

IN5110, autumn 2021

Martin Steffen

# Contents

# Chapter 5
# SAT-based & bounded model checking

What is it about?

**Learning Targets of this Chapter**

Bounded-model checking

**Contents**



## 5.1 Intro

The slides here take inspiration from the presentation [4] and the article [3]. Another good (handbook-)article is [1].

We have talked about model checking, i.e., solving the problem whether or not a model satisfies a given formula, of $M \models^? \varphi$. The focus was on the model being a transition system $T$, typically finite state, and the formula expressing some temporal property, written in some some temporal logic or other. In particular we have covered so far LTL and automata-based model checking and symbolic CTL model checking, using BDDs.

**Definition 5.1.1** (Kripke structure)**.** A *Kripke structure* or transition system is a tuple $(S, I, \rightarrow, V)$ where $S$ is the set of states,$I \subseteq S$ the set of initial states, $\rightarrow \subseteq S \times S$ the transition relation, and $V : S \rightarrow 2^P$ the *valuation* function (aka. (state) labelling function).

The conceptual *idea* of bounded model checking is quite trivial. When facing the task of verifying a system, we simply give up on exploring all of the system, but content ourselves in exploring it only up-to some point, i.e., imposing an upper bound on the exploration. In the context of LTL-model checking, the upper bound is on the *depth* of exploration, i.e., one puts a upper bound on the length on the *paths* the model checker explores.

Of course, the idea of putting an bound on the amount of effort spent in analysing a system or model can take different forms. One could for instance limit the time the model checker runs (or stop it from running if one looses patience waiting for an outcome). That would also in a way be bounded model checking thought it would not fall into a what is conventionally be called a bounded model checking technique; it would be too ad-hoc. Characteristic for classical bounded model checking is also, that model checking is presented as a SAT-solving problem, and just shutting down a process when it runs too long does not qualify for that.

But besides bounding the length of the paths, other criteria have been studied, for instance the number of context switching in parallel processes etc. We will not cover alternatives, but focus on the classical situation: exploring a system only up-to a paths of a fixed length.

Above it was said that bounded model checking gives up the hope of exploring the whole system, basically giving up the hope of system verification. One should probably formulate that less categorical.

First, in principle bounded model checking may not give that up, one could increase the bound iteratively until it spots an error or if one knows the bound is big enough. For finite-state systems such may be possible. But on the other hand, even if possible, it is somehow not too natural and a bit against the spirit of bounded model checking, which focuses more on spotting errors within exactly specified bounds. Bounded model checking shares this spirit-ual focus of finding errors with testing (focusing on a finite set of test cases) and with run-time verification (looking just at one run).

Secondly, and probably more importantly: that unbounded model checking allows full and absolute verification of correct systems resp. finding bugs in faulty ones is also more a fiction. In the introductory part we discussed why that is is a naive view in practice. One is that model checking typically works with *models* of the real system and with certain abstractions. That alone may result in that some errors are overlooked (and/or lead to the detection of spurious errors, which are errors caused by the abstraction but absent in the more detailed system). And even in when abstracting away from certain details, the state space may still be infinite or too big. So, when doing "unbounded" model checking in such a situation, the analysis may find its natural bounds by the maximum memory usage, for instance, or the practically acceptable running time.

Back from the more philosophical remarks, back to the technical realization of the general idea of exploring all paths up to a fixed length of $k$. An important aspect of the approach we have encountered already in connection with CTL-model checking. In a quite different form, thought, but there is the general commonality of a

> symbolic treatment via a propositional encoding for the purpose of the exploration.

In the context of CTL model checking we have looked at how different propositional constructs, ultimatly how propositional constraints can be represented and worked with. The propositional encoding back then was to capture sets of states, and the transition system etc. At some level, those entities where described by (quantified) boolean formulas. The formulic constraints were interpreted as boolean (swiching) functions, ultimatelty represented as ROBDDs.

Here, we are concerned representing finite paths through a given transition system (not (the evolution of) sets of states as in the CTL case). That means, the boolean formulas look different, but the idea is similar.

In the CTL case, which does a backwards, breadth-first exploration, a crucial ingredient was to capture the set of predecessor states for a given set $A$ of states. That set $spre(A)$ was captured by a formula that relied on (propositional) existential quantification and

relied on the propositional representation of the states and the transition relation $\rightarrow$ of the transition system.

Following similar ideas, it's plausible that one can represent a *finite* path of the form

$$s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_m \ . \tag{5.1}$$

As mentioned earlier, in its basic form, bounded model checking focuses in finding errors, namely errors that can be spotted with a finite distance, say $k$, from the initial states. One could, if one prefers that, spot errors with a finite distance of a different point in the program, or errors doing the same thing backwards, i.e., errors at most $k$-step *before* a given point, maybe before the end of the program. Those variations are inessential for the idea, and let's discuss it by assuming we do it forward, and start at an initial state, and let's assume there's only one of those, which is a common set-up anyway.

Encoding the path from equation (5.1) as SAT problem descibes all paths of finite length $m$, i.e., the solutions to that SAT-problem are those paths. Indirectly that descibes also all states $s_m$ that are at a distance $m$ away from the initial state $s_0$.

Now, if we see bounded model checking as a form of bug-hunting with a bound how deep to look for a bug, then of course it's not good enough to encode equation (5.1). Assume that we are after a safety property, a simple invariance $\Box A'$, with $A'$ propositional. Seen as a state formula, it means we want to establish resp. refute $\forall\Box A'$. To refute it means to establish an existential state formula of the form

$$\exists\Diamond A \tag{5.2}$$

(with $A = \neg A'$). Taking the path specification from equation (5.1), i.e., its SAT encoding and with proposition $A$ at hand, one can formulate that

$$s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_m \ \wedge s_m \text{ satifies } A \tag{5.3}$$

The states are best understood as variables. As far as $s_0$ is concerned, there may only be one state that qualifies, under the assumption that the model has only one initial state. Checking equation (5.3) for a satifying solution mean, looking for states $s_0, \ldots s_m$ such that $s_m$ satisfies the proposition $A$. To say it differently but equivalently, the $s_i$ are best seen as exitentially quantified. At any rate, in this way, equation (5.3) does not represent a particular path with concrete states $s_0$ to $s_m$, but *all paths of length* $m$ that start at the initial state and with some end-state $s_m$ that satisfies $A$. That *symbolic* treatment is similar to what we have seen in CTL model checking where one crucial trick was to represent the *sets* of states and the set of predecessor states of a given set in a propositional manner.

So we know that we can capture paths of length $m$ and a witness to $\exists\Diamond A$ from equation (5.2). Let's abbreviate the corresponding formula from equation (5.3) by

$$\exists\Diamond_m A \tag{5.4}$$

with $m$ fixed. Now, we are looking for witnesses at an arbitrary distances, i.e., what we are actually after is something like $\exists m : \mathbb{N}.\exists\Diamond_m$. That's not a propositional formula. The inner existential quantifier is ok, we made use of existential quantifications also in the

encoding of $pre(A)$ for CTL model checking. It quantifies over finite entities, resp. their propositional representation. So, that can be done propistionally, in so-called quantified boolean or quantified propositional logics, which is ultimately not much different than propositional logic without that operation. In particular it's about *finite* structures.

But the outer quantification $\exists m : \mathbb{N}$, that's a different story, that's no longer propositional. Another way to see it, to look for a witness means to solve the sat-problem $\exists \Diamond A$ is to solve

$$\exists \Diamond_0 A \vee \exists \Diamond_1 A \vee \exists \Diamond_2 A \vee \ldots \tag{5.5}$$

That's just a different way of expression the existential quantification over $\mathbb{N}$. And of course neither that is propositionally allowed. Disjunctions, of course, are allowed, but equation (5.5) makes use of an *inifinite* disjunction, which one cannot do.

But that's where the boundedness of bounded model checking comes in. The sat-constraint $\exists \Diamond A$ captures standard, unbounded model checking, unfortunately in a non-propositinal way. But if we restrict ourselves to find a witness only up-to a fixed depth $k$, then the setting becomes finite. Instead of the existential quantification over all natural numbers, one has to deal only with a quantification $\exists m \in \{0, \ldots, k\}$, resp. the infinite disjunction from equation (5.5) is now defused to a propositional finite disjunction

$$\bigvee_{m=0}^{k} \exists \Diamond_m A \tag{5.6}$$

### 5.1.1 What else need to be taken care of?

We sketched the core idea behind sat-based bounded model checking. In the short remarks, we simplified the situation slightly, basically by focusing on finding a encoding for a the problem $\exists \Diamond_{m \leq k} A$ and looking for a satisfying witness for that. That corresponds to refuting the simple invariance property $\forall \Box \neg A$, at least in a bounded manner.

However, we want apply the approach not just to formulas $\forall \Box A'$, which is the simplest form of a safety property, but to all of LTL.

Generally, of course, that's not possible: one cannot hope to verify or refute a property by looking only at path of finite length. At least not the the way we told the story. Additional considerations would come it, which are somehow orthogonal, and thus not in the spirit of the story of bounded model checking as sat problem over finite paths. We leave those extra considerations out in this discussion (for now, or completely).

That actually also applies to the situation from above, when reasoning about $\forall \Box A'$ in a bounded manner. We can *refute* that property by finding a finite counter example. But one cannot *verify* it in the described manner (without additional tricks), i.e., we cannot *decide* the logical status of that formula.

The above formula is the simplest form of safety property. But the same can be said about liveness properties, neither their status can be decided. The situation is somehow dual,

however. Let's take as property to verify $\forall\Diamond\varphi$, which means $\exists\Box\neg\varphi$ for refutation resp. as sat problem. In a bounded form, it's about sat-solving

$$\exists\Box_{\leq k}\neg\varphi \ . \tag{5.7}$$

If there exists a solution to the constraint, i.e., a path as witness for that formula, resp. a counter-example for the original formula in its bounded form $\forall\Diamond_{\leq k}\varphi$, then we have a finite witness-path where always $\neg\varphi$. But having established the constraint (5.7) does not imply that also $\exists\Box\neg\varphi$, in other words, we cannot *dis*-prove $\forall\Diamond\varphi$. One the other hand, if the constraint (5.7) is not satisfiable, it means $\forall\Diamond_{\leq k}\varphi$ is the case, and that means also unboundedly $\forall\Diamond\varphi$ where it holds.

So the safety property could be disproved in a bounded manner (if it actually does not hold), and one gets counter-example for it. The sample liveness propert could be proved (if it actually holds).

In some way, that's not posible. There are properties that cannot be refuted in that manner (nor can they be verified that way). At least not the the way we told the story. Additional considerations would come it, which are somehow orthogonal, and thus not in the spirit of the story of bounded model checking as sat problem over finite paths. We leave those extra considerations out in this discussion (for now, or completely).

Without any extra tricks, it's clear that one cannot verify a problem $\forall\Diamond\varphi$, and one cannot refute it, i.e. find a counter-example as a witness for the negation $\exists\Box\neg\varphi$. Looking only at paths up-to length $k$ (and without extra tricks) gives **no** information about the status of $\forall\Diamond\varphi$. Looking at a single path (for the sake of argument) and assuming that $\varphi$ does not hold up to $k$, it's clear that without looking beyond the bound we cannot know

But we are of course not doing model checking on one path, but all paths up to a given bound. If, by some lucky chance, $\varphi$ would become true within the explored $k$-horizon *on all bounded paths*, then one could conclude that $\forall\Diamond\varphi$ actually holds. But we are not getting hold of all such paths. We are doing sat-solving and are working with $\exists\Box\neg\varphi$. The information we get is either

$$\exists\Box_{\leq k}\neg\varphi \quad \text{or} \quad \forall\Diamond_{\leq k}\varphi$$

and we only get to know that there exists bounded path where $\Box\neg\varphi$ or else not.

If there does not exist such bounded path where $\neg\varphi$ is always true, it means $\forall\Box_{\leq k}\varphi$forit has not information about the situation beyond the bound, and it does not help. If, on the other hand, there exists a bounded path where $\Box\varphi$ holds, it carries no information eather

## 5.2 Bounded semantics

We have covered the standard semantics of LTL earlier, interpreting LTL formulas over *infinite* paths. The interpretation of formulas over a transition system being a derived notion.

In the CTL-part, in particular in the model checking part, we restricted our interest to formulas in a particular normal form, it was CTL-formulas in ENF, existential normal form. We also mentioned that other normal forms exists, for CTL or other logics. We use a quite common such normal form here for LTL and bounded model checking, namely **negation normal form, NFF**.

In such a form, one restricts the use of negation in that only atomic propositions or propositional variables are allowed to occur negated and negation cannot be used on compound formulas. To compensate for that, we need both $\wedge$ *and* $\vee$, both $\Diamond$ and $\Box$ etc.

In the presentation (following [1]), we leave out, however, the until-operator and the other binary temporal operators. Not that it would be complex, but the presentation is slightly simpler.

An initialzed path satisfying $\varphi$ is called a *witness* for $\exists\varphi$.

or counter example, remember *refutation*

$\pi$ of a transition system $T$

*witness* of $\varphi$ in $T$ is a path starting in an initial states and that satisfies $\varphi$.

We assume the formulas in *negation normal form*.

Looking for witnesses for formulas of the form $\exists\Diamond\varphi$. In other words the approach can deal straightforwardly with finding *counter-examples* for formulas of the form $\forall\Box\varphi'$, generally counter-examples to *safety properties*.

In the dual case of looking for counter-examples for liveness properties, i.e., looking for witnesses of for instance formulas of the form $\exists\Box\varphi$, things get more involved. It's intuitively clear that a finite prefix of an infinite path cannot be used as witness for

$$\exists\Box\varphi \ . \tag{5.8}$$

Unless one adds one additional clever idea, a central one for bounded model checking in the form discussed here.

The idea is to take into account **looping behavior.** Consider Figure 5.1, depicting a loop or a lasso. Both words are used for that, in the original publication it was called loop, though in the handbook article, the word "lasso" is used. I prefer lasso, and a lasso consists of two parts, the *stem*, in the picture going from $s_0$ to $s_l$ and the loop part, the part depicted as cycle at the end.

The picture is a visualization of lasso or loop, but it's (also) understood as a *path*, i.e., an infinite sequence of states (see Definition 5.2.1). The infinite path it respesents has a finite prefix, the stem, and then an infinite repetition of the looping part (see also equation (5.9)).

Even if, technically, a lasso is an infinite path, it can be *finitely represented,* as in Figure 5.1 and it also can detected looking only at a finite prefix. Having reached state $s_k$ and detecting that one can continue to a state one has seen before, state $s_l$ in the picture,

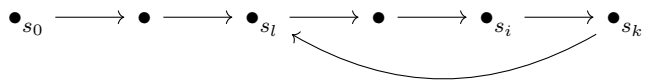


Figure 5.1: A lasso

shows that there is a cycle in the transition system. It shows also that the prefix *can* be extended to a infinite path by repeating the cycle forever. That means that in a situation with a lasso, a finite prefix can actually serve as a witness for the from equation (5.8). Note that it's important that in the approach, looking for witnesses, we are dealing with existentially quantified formulas $\exists\varphi$, so the detection of a loop shows there exists an infinite path as witness. So a finite sequence of $k+1$-states represents (the existance of) a lasso, if continues, and we use the term lasso also for that finite representation consisting of $s_0 \dots s_k$, with the last state $s_k$ as the one with the back-transition to some earlier $s_l$.

**Definition 5.2.1** (Lasso)**.** Assume $l \leq k$. A path $\pi$ is a $(k,l)$-*lasso* if $\pi_k \to \pi_l$ and

$$\pi = u \cdot v^\omega \tag{5.9}$$

with

$$u = \pi_0 \dots \pi_{l-1} \quad \text{and} \quad v = \pi_l \dots \pi_k$$

A path $\pi$ is a $k$-*lasso* if there exists an $l$ with $0 \leq l \leq k$ s.t. $\pi$ is a $(k,l)$-lasso

Of course, not all finite paths are $k$-lassos, for instance if all states are different. Technically, that's not the only reason, why a path of length $k+1$ is no lasso. The definition as given insists that, to be a lasso, the last state $s_k$ loops back, if an earlier state $s_m$ with $m < k$ is repeated, then technically that's not a loop. But its prefix $s_0 s_1 \dots s_m$ would be. In the larger picture, it's not so important, as the approach works with prefixes of paths of all length up-to a pre-defined upper bound say $k$, i.e., the lasso at $m$ will be detected and taken into account.

At any rate, some finite path of length $k+1$ are lassos (or contain lassos in their prefix) and some not. One could increase $k$, increasing also the chance to find some lassos. Having a path with a lasso is a good thing, because, as explained, in this favorable situation, we can hope to use it as witness for an $\Box$-style formula (or a counter-example for an original liveness property). Of course not all lassos are factually a witness for such a formula, the $\varphi$ must also hold at all positions.

But for non-loop finite paths, the situation is worse. They can serve as witness for $\exists\Diamond\varphi$ formulas; for that kind of properties the lassos play no role. But without loop, they don't contain information for $\forall\Box\varphi$-formulas.

When we will define the so-called *bounded* semantics for LTL, i.e., when a finite path satisfies an LTL formula, we distinguish the two situations: paths which are lassos and paths which are not. In both cases, sometimes the finite path can serve at a witness. In this case the semantic will report back that this is the case.

But there will be cases where the finite prefix contains not enough information to serve as a witness (and to serve as counter-example for the original $\forall$-formula). We discussed the case that obviously non-lasso prefixes can never serve as witness for $\Box$-formulas. But the same problem, the fact that a finite prefix does not contain enough information, also applies to $\Diamond$-formulas. If we want a witness for $\Diamond\varphi$, but the finite prefix contains no occurrence of $\varphi$, we likewise don't know.

The question is

What should the semantics give as an answer is such "don't know'-situations?

One possibility could be to work not with a binary logic with true and false, but perhaps with true, false, and "don't know". Some approaches for run-time verification use a 3-valued version of LTL for that.[1] But here we do something else. We stick to 2 boolean values, but in case of not knowing, the bounded semantics reacts pessimistically and gives *false* as judgement. The emphasis is on finding a witness for a formula (resp. a counter-example for the non-negated property), and if the length of exploration cannot be used as such a witness, then, with the current bound, there is no witness (yet) and the finite path does not satify the formula in the bounded semantics.

Let's now define the bounded semantics, and let's do that in the form of a satisfaction relation $\models$, as before. The bounded version is written $\models_k$, where $k$ is the bound up-to which we consider the path. As mentioned, the definition is given in two "variants", one for paths which are in the form of a lasso, and one for finite paths where that is not the case. In the latter case we will also work with a refined version of $\models_k$, written $\models^i_k$, see below. Let's start with the *lasso-case*.

**Definition 5.2.2** (Bounded semantics: with lasso)**.** Let $\pi$ be a $k$-loop. A formula $\varphi$ is *valid* along $\pi$ *with bound* $k$, written

$$\pi \models_k \varphi \quad \text{iff} \quad \pi \models \varphi \ .$$

The definition is quite trivial, it's the original, unchanged semantics $\models$. In the presence of a $k$-loop and with the bound at $k$, we have all the information we need. The loop case is exact, non approximative.

On the other hand, that's only the case if we consider the bounded semantics on one path (as the definition does). In the larger conceptual picture, we are exploring a transition system, and we are exploring paths up-to a bound of $k$. Thinking in particular about looking for a witness for $\exists\Box\varphi$, in the positive case, namely that the formula holds, the semantic is exact also in the larger picture of exploring the transition system. In the negative outcome, however, when the lasso-path does *not* satisfy the formula, of course it does not mean ...

[1] Actually also 4-valued version. It's 4 values in that they make use of 2 version of "unknown", somehow like "unknown, but it looks good", and "unknown, but it looks bad", a "tendency" towards true resp false. This is an informal description and not 100% accurate, and there are technical reasons which speak for 4 values instead of 3.

**Definition 5.2.3** (Bounded semantics: without lasso)**.**

$$
\begin{array}{lll}
\pi \models^i_k p & \text{iff} & p \in L(\pi_i) \\
\pi \models^i_k \neg p & \text{iff} & p \notin L(\pi_i) \\
\pi \models^i_k \varphi_1 \wedge \varphi_2 & \text{iff} & \pi \models^i_k \varphi_1 \quad \text{and} \quad \pi \models^i_k \varphi_2 \\
\pi \models^i_k \varphi_1 \vee \varphi_2 & \text{iff} & \pi \models^i_k \varphi_1 \quad \text{or} \quad \pi \models^i_k \varphi_2
\end{array}
$$

$$
\begin{array}{ll}
\pi \models^i_k \Box\varphi & \text{is always false} \\
\pi \models^i_k \Diamond\varphi & \text{iff} \quad \exists j. i \leq j \leq k. \quad \pi \models^j_k \varphi \\
\pi \models^i_k \bigcirc\varphi & \text{iff} \quad i < k \quad \text{and} \quad \pi \models^{i+1}_k \varphi \\
\pi \models^i_k \varphi_1 \; U \; \varphi_2 & \text{iff} \quad \exists j, i \leq j \leq k. \pi \models^j_k \varphi_2 \text{ and } \forall n, i \leq n < j. \pi \models^n_k \varphi_1 \\
\pi \models^i_k \varphi_1 \; R \; \varphi_2 & \text{iff} \quad \exists j, i \leq j \leq k. \pi \models^j_k \varphi_1 \text{ and } \forall n, i \leq n < j. \pi \models^n_k \varphi_2
\end{array}
$$

Now that we have established the bounded semantics, we look into it connection to the original one. Basically, in the limit, both semantics coincide. The bounded semantics is an approximation of the real semantics, in the sense that in case of a positive answer in the bounded semantics, that carries over to the real, unbounded semantics. In other words, the bounded semantics is a *sound* approximation of the unbounded semantics, and that for every bound $k$. See Lemma 5.2.4.

**Lemma 5.2.4** (Soundness (per path))**.**

$$\pi \models_k \varphi \quad \textit{implies} \quad \pi \models \varphi$$

That's not surprising, and was easy to achieve: the bounded semantics in case of not having enough information, stipulated that the formula does *not* hold. The reverse direction, completeness is harder, and it means that ultimately, if a formula for existential LTL holds in a transition system, it also holds for the bounded semantics, if only the $k$ is large enough. See Lemma 5.2.5.

**Lemma 5.2.5** (Completeness (for TSs/KSs))**.**

$$S \models \exists\varphi \quad \textit{implies} \quad S \models_k \exists\varphi \quad \textit{for some } k \geq 0$$

Both results together give the following

**Theorem 5.2.6.**

$$S \models \exists\varphi \quad \textit{iff} \quad S \models_k \exists\varphi \quad \textit{for some } k \geq 0$$

So, to use a bounded model checker, one has pick a large enough $k$ and the tool gives the correct answer. The $k$ may not be known, so one could guess one. Or better, model checker could increase the $k$ automaticall in case an attempt is unsuccessful, hoping that one finds an positive answer at some point and before running out of resources.

Besides that one may well run out of resouces before one gets a positive answer, banking in on soundness, the case where the $\exists\varphi$ does *not* hold is principally more problematic.

Completeness from Lemma 5.2.5 states that in this case, one need to check $\models_k$ **for all** $k \geq 0$**!** That's of course not doable.

The existential LTL formula $\exists\varphi$ is typically the negation of a standard $\forall$-formula, one tries to refute. That's why we said, bounded model checking focuses on finding bugs (i.e., refuting the original formula), but *not* on verification: one cannot check the bounded model checking problem for all $k$.

There are things one can do about that. For instance, in a finite-state system, if one knows the size of the transition system (more precisely its diameter), one can use that to know when there's no use in looking for larger $k$'s, and if until then one has not found a witness (a counter-example to the original $\forall$-formula), one is sure there is none. But we don't into that further.

## 5.3 Propositional encoding

Next we will present how a bounded model checking problem can be addressed by sat-solving by showing an encoding of such problems. We show the encoding as originally proposed, which is a pretty straightforward translation of the bounded LTL semantics and of course the transition system into propositional logics. Indeed, afterwards, other encodings and representation have been proposed, for instance some working with Büchi-automata or other techniques that lead to more efficient implementations. See for instance [5] which is also covered in [1]. Also explicit-state, non-symbolic method have been explored for bounded model checking. See for instance for some discussion of different techniques [2].

Here, as said, we stick to the original proposal of the encoding. Some of that we basically know already from symbolic model checking of CTL, namely how to encode the transition system. Here we don't need to encode the set of predecessors *pre*, but finite paths through the system, but the priciples are the same. Talking about CTL model checking: in that part we continued to explain how the encoding can be understood as boolean switching functions, and how those can be canonically represented as BDDs. That one popular way in CTL model checking.

Here we don't do that, we just show the formulaic representation as boolean formula. How to handle those we leave to a SAT-solver, which may not be based on BDDs; most SAT solvers are using variations of David-Putnam's procedure (DP).

**Goal:** $[\![T, \varphi]\!]_k$ is *satisfiable* by some $\pi$ iff $\pi$ is a witness for $\varphi$

The encoding will be done in three separate parts, one dealing with the (finite paths) of the given transition system. Secondly, capturing the loop or lasso condition, and finally, representing the formula $\varphi$, resp. its bounded satisfaction relation. The latter part is split into the semantics dealing with paths with a lasso, and those without (as it was defined earlier)

### Encoding the transition system

Let's start with the transition system, resp. bounded paths through it. Let $T$ be a transition system, then

$$[\![T]\!]_k \triangleq I(s_0) \wedge \bigwedge_{i=0}^{k-1} s_i \to s_{i+1} \tag{5.10}$$

### Loop condition

Remember the concept of $(k,l)$-loop or lasso and of $k$-lasso from Definition 5.2.1 (remember also Figure 5.1). Both are readily encodable. ${}_lL_k$, representing a $(k,l)$-loop is a simple abbreviation for the fact that there is an edge in the transition system between the to involved states.

$${}_lL_k \triangleq s_k \to s_l$$

With that the $k$-lasso is given as a finite disjunction (or bounded existential quantification) as follow:

**Definition 5.3.1** (Loop condition)**.**

$$L_k \triangleq \bigvee_{l=0}^{k} {}_lL_k$$

### Encoding (the semantics of) $\varphi$

Earlier, the bounded semantics was defined using the relation $\models$. Here we capture the semantics as the set of states where the formula holds, i.e., for encode $[\![\_]\!]$.

**Lasso-case: bounded semantics for a $(k,l)$-loop** Encoding the propositional part as propositnal formula is, of course, trivial

$$\begin{aligned}
{}_l[\![p]\!]_k^i &\triangleq p(s_i) \\
{}_l[\![\neg p]\!]_k^i &\triangleq \neg p(s_i) \\
{}_l[\![\varphi_1 \wedge \varphi_2]\!]_k^i &\triangleq {}_l[\![\varphi_1]\!]_k^i \wedge {}_l[\![\varphi_2]\!]_k^i \\
{}_l[\![\varphi_1 \vee \varphi_2]\!]_k^i &\triangleq {}_l[\![\varphi_1]\!]_k^i \vee {}_l[\![\varphi_2]\!]_k^i
\end{aligned}$$

The temporal part is a but more interesting. To do that, we also need to "define" the successor of a state, resp. we need to convince ourselves that one can propositionally refer

to the successor of a state in a path with a loop. We use $succ(i)$ to refer to the successor of $i$ in a $(k, l)$-loop as

$$succ(i) = \begin{cases} i+1 & \text{for } i < k \\ l & \text{for } k \end{cases} \tag{5.11}$$

$$\begin{array}{rcl} {}_l[\![\Box\varphi]\!]^i_k & \triangleq & {}_l[\![\varphi]\!]^i_k \wedge {}_l[\![\Box\varphi]\!]^{succ(i)}_k \\ {}_l[\![\Diamond\varphi]\!]^i_k & \triangleq & {}_l[\![\varphi]\!]^i_k \vee {}_l[\![\Diamond\varphi]\!]^{succ(i)}_k \\ {}_l[\![\bigcirc\varphi]\!]^i_k & \triangleq & {}_l[\![\varphi]\!]^{succ(i)}_k \\ {}_l[\![\varphi_1 \ U \ \varphi_2]\!]^i_k & \triangleq & {}_l[\![\varphi_1]\!]^i_k \vee {}_l[\![\varphi_1 \ U \ \varphi_2]\!]^{succ(i)}_k \\ {}_l[\![\varphi_1 \ R \ \varphi_2]\!]^i_k & \triangleq & {}_l[\![\varphi_2]\!]^i_k \wedge {}_l[\![\varphi_1 \ R \ \varphi_2]\!]^{succ(i)}_k \end{array}$$

**Translation for paths without a loop** The translation follows the same principles (the index $l$ is not needed obviously here). And instead of the more "complex" $succ(i)$ from before, we can simply use $i+1$, otherwise: the definition stays "the same"

The propositional part is boring again (and we don'd show it):

For the temporal case, we can make, we have $\forall i \leq k$:

$$\begin{array}{rcl} [\![\Box\varphi]\!]^i_k & \triangleq & [\![\varphi]\!]^i_k \wedge [\![\Box\varphi]\!]^{i+1}_k \\ [\![\Diamond\varphi]\!]^i_k & \triangleq & [\![\varphi]\!]^i_k \vee [\![\Diamond\varphi]\!]^{i+1}_k \\ [\![\bigcirc\varphi]\!]^i_k & \triangleq & [\![\varphi]\!]^{i+1}_k \\ [\![\varphi_1 \ U \ \varphi_2]\!]^i_k & \triangleq & [\![\varphi_1]\!]^i_k \vee [\![\varphi_1 \ U \ \varphi_2]\!]^{i+1}_k \\ [\![\varphi_1 \ R \ \varphi_2]\!]^i_k & \triangleq & [\![\varphi_2]\!]^i_k \wedge [\![\varphi_1 \ R \ \varphi_2]\!]^{i+1}_k \end{array}$$

We can see it as induction case. The "induction" goes backwards insofar the situation of $i$ is defined in terms of $i+1$. So the base case is the one at the end, namely for $k+1$.

$$[\![\varphi]\!]^{k+1}_k \triangleq \mathit{false} \tag{5.12}$$

**Putting it together** Finally, we can put the three ingredients together, the encoding of the paths, the loop condition, and the encoding of the formula.

$$\begin{array}{rcl} [\![S, \varphi]\!]_k \triangleq & [\![S]\!]_k \wedge & \\ & ( \ (\neg L_k \wedge [\![\varphi]\!]^0_k) & \\ & \quad \vee \quad (\bigvee_{l=0}^{k} ({}_l L_k \wedge \ {}_l[\![\varphi]\!]^0_k) \quad ) & \end{array} \tag{5.13}$$

**Theorem 5.3.2.**

$$[\![S, \varphi]\!]_k \ \mathit{satisfiable} \quad \mathit{iff} \quad S \models_k \exists\varphi \ .$$

# Bibliography

[1] Biere, A. (2009). Bounded model checking. In Biere, A., Heule, M., van Maaren, H., and Walsh, T., editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 457–481. IOS Press.

[2] Clarke, E., Kroening, D., Ouaknine, J., and Strichman, O. (2005). Computational challenges- in bounded model checking. *Software Tools for Technology Transfer (STTT)*, 7(2):174–184.

[3] Clarke, E. M. (2008). Model checking – my 27-year quest to overcome the state explosion problem. In Cervesato, I., Veith, H., and Voronkov, A., editors, *Logic for Programming, Artificial Intelligence, and Reasoning: 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings*, Lecture Notes in Artificial Intelligence, pages 182–182. Springer Verlag.

[4] Clarke, E. M. (2017). SAT-based bounded and unbounded model checking. Available electronically on the net. Data of publication unknown.

[5] Latvala, T., Biere, A., Heljanko, K., and Junttila, T. (2004). Simple bounded ltl model checking. In Hu, A. J. and Martin, A. K., editors, *Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD'04), Austin, Texas, USA, November 2004.*, volume 3312 of *Lecture Notes in Computer Science*. Springer Verlag. An extended version is available as University of Helsinki Technical Report UT-TCS-A92.

# Index