Combining Bounded Model Checking with Highway Search

Maarten Meulen (0597519)
m.g.meulen@student.tue.nl

January 24, 2009

Abstract

A new technique is developed which can be used to reduce a finite state transition system by bounding its breadth, in a way the result can be used for model check purposes. The technique is based on the 'Highway' search algorithm and is inspired on already developed bounded depth model check techniques. The newly developed 'Highway' algorithm was implemented in the mCRL tool set where after some experiments have been conducted. The results show that the Highway algorithm is reasonably effective in generating finite state transitions systems which include witnesses for existentially quantified LTL formulas. Future research has to show how effective the technique is when used for larger finite state transition systems.

1 Introduction

Verification and testing of computer hardware and software has always been a hot issue in computer science. For example, in areas where computers are used to control life-critical systems, certain safety properties are of great importance. One would like to verify whether or not the design of these systems meets these requirements. But also in non life-critical systems, design verification can be useful. Think of any system for which the commercial value greatly depends on the correct behavior of the particular system (e.g. network protocol design, distributed control systems).

Over the past two decades, several techniques for automatic formal verification of state transition systems have been developed. The most general and widely used name for these techniques is 'Model Checking'. With the help of these techniques one can model a hardware or software design as a state transition system and encode one or more important aspects as 'temporal logic' properties. With these ingredients, a model check algorithm can verify whether or not the temporal logic properties hold for the given state transition system.

Besides the ability to check whether or not a property holds, model check algorithms also generate a counterexample in case a property does not hold. In general, two types of properties are of interest: 'safety' and 'liveness' properties. A 'safety' property declares what should never (or always) happen. A counterexample for a safety property is a finite prefix of a path which leads to a state in which the property is contradicted. On the other hand, a 'liveness' property declares what eventually should happen. A counterexample for this kind of property is, in its simplest form, a path which contains a loop in which the property is contradicted (e.g. a lasso). Such a loop represents an infinite path which will not lead to a specified state.

One of the major complications with model check techniques as they exist today, is that it often is impossible to verify the complete system due to space and time limitations. Even, at first hand, very simple system designs can grow exponentially large when they are modeled as a state transition system. In practice this means many system designs of industrial complexity can not be verified.

In many cases, model check techniques are not used to perform complete verification of the design. More often, model check techniques are used for finding errors that are unlikely to be found during simulation testing. In these cases, examining only a part of the system using only a part of the specification is acceptable. In other words, model check techniques are more often

used for 'falsification' rather than 'verification'. From this, one can conclude that model checking is still useful if it is incomplete.

The observation that incomplete model check techniques are indeed useful, provides a justification for the development of these techniques of which one will be discussed in this paper. If we consider a state transition system to have a breadth and depth, where the breadth is determined by the number of states that can be reached after a certain amount of transitions and depth is determined by the length of a prefix of a path in the transition system, then a restriction on these parameters can be used to make a bounded selection of the state transition system. This part of the transition system can then be used for model checking purposes.

In 2003, Biere et. al. introduced a technique called 'Bounded Model Checking' [1]. One of the goals of this technique is to provide a way of dealing with finite state transition systems that have a bounded depth. Although experiments have shown that this technique can be very effective, there exists no similar technique that can bound a finite state transition system on its breadth parameter. Therefore, inspired on the technique introduced by Biere et. al., this paper will discuss a way of reducing a finite state transition system by bounding its breadth parameter.

In the remainder of this paper, the standard model check technique 'linear temporal logic' (LTL) will be discussed. Thereafter, the incomplete 'Bounded Model Checking' (BMC) will be discussed briefly. Bounded Model Checking is a model check technique which restricts the depth of a finite state transition system for which properties can be checked. Section 4 will discuss the 'Highway' search algorithm which is adapted in the same section in such a way it can be used to limit the breadth of a finite state transition system. Section 5 will deal with some experiments which were conducted with the adapted highway algorithm. Finally, section 6 will give a brief overview of some topics that might be interesting for future research and section 7 will give a general conclusion.

2 Linear Temporal Logic

In the introduction, several notions such as 'state transition system', 'path', 'design' of a 'system' have been introduced without a proper explanation. In order to discuss model check techniques, this terminology requires a more detailed definition.

An engineer designing a real piece of software or hardware, has several techniques at his disposal to describe the functional behavior of the system in development. This design can be interpreted as some sort of automation which, for the sake of simplicity, is considered to be finite (the automation only has a finite number of distinguishable states). In the context of model checking, structures like finite automations are often represented by a 'Kripke structure'. With such a Kripke Structure, a finite state transition system is obtained that represents the functional behavior of the system, by defining states, each labeled with a unique set of atomic propositions that hold in each particular state, and a transition relation. Definition 1 gives a formal definition of a Kripke Structure.

Definition 1 (Kripke Structure)**.** *A Kripke Structure M is a quadruple (S, I, T, L) where:*

- *S: denotes the set of states*
- *I: denotes the set of initial states: $I \subseteq S$*
- *T: denotes the transition relation between states: $T \subseteq S \times S$*
- *L: denotes the labeling function which assigns every state $s \in S$ a unique set of atomic propositions that hold in s.*

In addition to the Kripke structure, a definition is needed of the sequential behavior of a finite state transition system. A so called 'path' is simply a sequence of states in which the order of states respects the transition relation. For the sake of simplicity, the transition relation is considered to be 'total', which means that every state must have an outgoing transition and thus all paths are infinitely long. For these paths, a prefix is considered to be a finite sequence of states, starting

with a state that is an element of the set of initial states. On the other hand, a suffix of a path is an infinite sequence of states which starts in a specified state. The notion of paths is captured more formally in definition 2.

Definition 2 (Path in a Kripke Structure)**.** *A path π in Kripke Structure M is a infinite sequence of states $(s_0, s_1, ...)$ for which the following property holds: $(\forall i : i \in \mathbb{N} : T(s_i, s_{i+1}))$*
Moreover:

- *The i-th state of a path π in M is denoted by $\pi(i)$*
- *A prefix of a path π of length $n+1$ is a sequence of states $(s_0, ..., s_n)$ which respects the transition relation T and $s_0 \in I$*
- *A suffix of a path π starting with in the i-th state is an infinite sequence $(s_i, s_{i+1}, ...)$ which respects the transition relation T and is denoted by π_i.*

With the system under consideration represented as a finite state transition system by a Kripke Structure and a definition of paths, one or more properties can be formalized in temporal logic. However, first a definition of temporal logic itself is needed. In the context of this paper, only a specific type of temporal logic is discussed: linear temporal logic (LTL). Temporal logic in general is an extension of classical propositional logic, and thus, includes all the standard concepts of propositional logic like Boolean variables and operators such as negation, conjunction, disjunction, implication and bi-implication. But moreover, linear temporal logic also includes temporal operators. In the next paragraphs, these operators are briefly discussed.

The first and most straightforward operator is the 'Next' operator **X**. This unary operator is used to declare that a certain property p holds for the path starting in the next state. Note that p itself is a property in linear temporal logic too. So, it can be a simple atomic proposition or a more complex property built using the linear temporal logic operators.

The next operator discussed is the 'Global' operator **G**. The global operator is used to encode properties that need to hold along the entire path. This in fact, sounds like the safety property that was already discussed in the introduction of this paper. The safety property states what should happen or what should not happen. With the help of the global operator, such properties can now be expressed: $\mathbf{G}p$ declares that along a path property p has to hold. In the same way $\mathbf{G}\neg p$ declares that along a path property p should not hold.

Another important operator in linear temporal logic is the 'Future' operator **F**. While the global operator is used to encode safety properties, the future operator is used to encode liveness properties. Informally this operator states that somewhere along a path, a certain property should hold. Or, as described in the introduction, what eventually should happen.

Besides the 'Next', 'Global' and 'Future' operators, two more temporal operators exist in LTL: the 'Until' (**U**) and 'Release' (**R**) operators. Since these operators play no role in this article, an informal explanation is omitted. Detailed explanation of the 'Until' and 'Release' operators as well as a thorough discussion of standard model checking techniques can be found in [2] and [5].

If one claims an infinite path π of Kripke Structure M satisfies a linear temporal logic formula f, then this is denoted as $\pi \vDash f$. In the same way one can state that a Kripke Structure M satisfies a LTL formula f, $M \vDash f$, iff $\pi \vDash f$ for all initialized paths π (paths of which the first state is in the set of initial states I) of M. What rests is a formal definition of the semantics of the LTL operators, In definition 3, $\pi \vDash f$ is defined recursively:

Definition 3 (Formal Semantics of LTL formulas)**.** *The semantics are recursively defined:*

$$
\begin{array}{lll}
\pi \vDash p & \textit{iff} & p \in L(\pi(0)), \textit{ where } p \textit{ is an atomic proposition} \\
\pi \vDash \neg p & \textit{iff} & \pi \nvDash p \\
\pi \vDash f \wedge g & \textit{iff} & \pi \vDash f \textit{ and } \pi \vDash g \\
\pi \vDash f \vee g & \textit{iff} & \pi \vDash f \textit{ or } \pi \vDash g \\
\pi \vDash f \rightarrow g & \textit{iff} & \pi \vDash g \textit{ whenever } \pi \vDash f \\
\pi \vDash \mathbf{X}f & \textit{iff} & \pi_1 \vDash f \\
\pi \vDash \mathbf{F}f & \textit{iff} & \pi_i \vDash f \textit{ for some } i \geq 0, \pi_i \vDash f \\
\pi \vDash \mathbf{G}f & \textit{iff} & \pi_i \vDash f \textit{ for all } i \geq 0, \pi_i \vDash f \\
\pi \vDash f\mathbf{U}g & \textit{iff} & \pi_i \vDash g \textit{ for some } i \geq 0 \textit{ and } \pi_j \vDash f \textit{ for all } 0 \leq j < i \\
\pi \vDash f\mathbf{R}g & \textit{iff} & \pi_i \vDash g \textit{ if for all } j < i,\ \pi_j \nvDash f
\end{array}
$$

Note that from the above definition it follows that $\neg\mathbf{F}\neg p \equiv \mathbf{G}p$ where the equivalence $\equiv$ is defined as: $f \equiv g$ iff $M \vDash f \leftrightarrow M \vDash g$ for all Kripke Structures M. Hence, $\mathbf{F}$ and $\mathbf{G}$ are dual operators.

Since LTL formulas are defined over paths in the finite state transition system, verification is done by searching for a counterexample that contradicts the property to be checked. A counterexample for a property that has to hold for all paths in the finite state transition system is in fact a single path contradicting the property. Such a path is called a 'witness'. For example, when the goal is to check whether or not a finite state transition system M satisfies LTL formula $\mathbf{G}p$, more formally $M \vDash \mathbf{G}p$, it suffices to search for a single path that satisfies $\mathbf{F}\neg p$. This follows directly from the semantics of LTL formula and the observation that $\mathbf{G}$ and $\mathbf{F}$ are dual operators. Because the standard LTL semantics dictates that LTL formulas are defined over all paths, there is no way to indicate only a witness is expected to be found. Therefore, 'path quantifiers' $\mathbf{A}$ and $\mathbf{E}$ are introduced. These path quantifiers are used as expected: a finite state transition system M that satisfies a LTL formula f over all paths is denoted by $M \vDash \mathbf{A}f$ and for a finite state transition system M that satisfies formula f over at least one path is denoted by $M \vDash \mathbf{E}f$.

3 Bounded Model Checking

The bounded model checking technique was first introduced in 2003 by Biere et. al. [1]. With bounded model checking a finite state transition system is only explored up and till a certain depth. This depth is determined by the amount of states in a prefix of a path. By doing so, the number of states in a finite state transition system can possibly be reduced to a level which makes model checking feasible in cases the unbounded finite state transition system is too large.

Since bounded model checking is performed on prefixes with a bounded length of infinite paths, this results in two situations: a prefix resembles a finite path, which is quite logical since a prefix is by definition finite (see definition 2), or the prefix resembles a infinite path because it contains a 'back loop' from the last state of the prefix to an earlier state. The two situations are illustrated in figure 1 and 2.

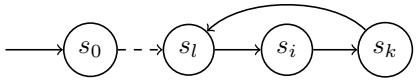

Figure 1: Path with a back jump from s_k to s_l

When observing the case of a prefix with bounded length $k+1$ as is illustrated in figure 2, it becomes clear that the bounded prefix does not provide any insight in the behavior of the path beyond state s_k. For example, in the situation the property $\mathbf{F}p$ is to be checked, the model check

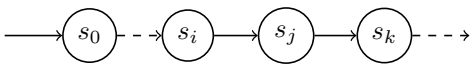

Figure 2: Path with no back jump up and till s_k

algorithm will search for a counterexample consisting of a witness for the property $\mathbf{G}\neg p$. Such a witness should be prefix with a loop in which p does not hold. With a prefix of a path containing no loop and $\neg p$ satisfied in every state, it still is impossible to conclude whether or not the entire path satisfies the property. It could be possible that the property fails in state s_{k+1}, which is not taken into consideration. On the other hand, in case the path did contain a loop as is illustrated in figure 1, it is possible to determine whether or not the path is a proper witness. In the later case nothing changes compared to standard LTL model checking.

Observing the example, it can be concluded that the standard LTL semantics is not applicable anymore in case a prefix of a path is observed with no loop. This seems to be somehow logical since standard LTL does not consider finite (prefixes of) paths. To distinguish paths with loops from path with no loops, a definition of a k,l-loop is needed. Informally a k,l-loop is a prefix of a path of length $k+1$ that has a back loop from state s_k to an earlier state s_l. A more formal definition can be given as follows:

Definition 4 (k,l-loops)**.** *For k and l, $0 \leq l \leq k$, a path π is called a k,l-loop if $T(\pi(k),\pi(l))$ and $\pi = u \cdot v^\omega$ with $u = (\pi(0),...,\pi(l-1))$ and $v = (\pi(l),...,\pi(k))$ where v^ω represents an infinite repetition of v.*

In [1] this problem with different types of paths, is solved by introducing a new set of semantics for the special situation in which a finite prefix of a path is considered that does not contain a loop. Moreover, in the same article, a new method for LTL model checking is introduced since standard LTL model check algorithms can not be used anymore. This new method transforms the finite state transition system and the LTL properties to a propositional formula (using the new semantics) where after a satisfiability solver is used to perform the actual model checking. Furthermore, experiments have shown that modern satisfiability solvers can outperform standard LTL model check techniques for finding a witness of an existential quantified LTL formula.

4 Highway Search

In this section, a new technique is presented which can be used to reduce the number of states of a finite state transition system. But, this time, instead of imposing a bound on the depth of the finite state transitions system, a bound is applied on the breadth. This novel technique is developed around the Highway search algorithm first presented in [3]. In subsection 4.1 the Highway algorithm is introduced as it can be found in the original paper. Thereafter, section 4.2 will discuss some modifications needed in order to perform model checking.

4.1 Standard Highway algorithm

The Highway algorithm as it is described in [3] is not meant for model checking on its own. In its essential form it is a search algorithm that can search for a particular state of interest in a state transition system. This means that it can be used to solve a specific type of model check problem: find a state in which a safety property is contradicted. Since the Highway algorithm only considers states and has no notion of paths, more complex properties can not be checked. For example, a counterexample for a liveness property, which is a prefix of a path which contains a back loop, can not be found.

As the introduction already made clear, where bounded model checking can be used to explore a state transition system up and till a certain depth and perform model checking, the Highway search algorithm only searches a state transition system for a certain breadth. In this article the Highway algorithm is used to cut down an existing state transition system by restricting its breadth. The cut down state transition system can then be used for model checking with the help of a standard model check technique.

Before the highway search algorithm can be discussed, several definitions are needed first. These definitions are introduced in a rather informal way. The Highway search algorithm, which is an iterative algorithm, traverses the state space in a very intuitive way. It starts at the initial

state (or states) and in every iteration it make a decision which states, that are reachable from the states of the previous iteration, will be visited next. This means that if the number of iterations is counted, a depth index is acquired. In the rest of this discussion this depth index is referred to by d. At any point in time during the execution of the algorithm, d will be the depth level that is currently processed. Once the algorithm has terminated, d automatically denotes the number of depth levels processed.

On each depth level d, for $d \geq 0$, the algorithm needs to store the states it will visit. The set of visited states at depth level d is denoted by Q_d. The union of all these sets represents all visited states $V = \bigcup_{i=0}^{d} Q_i$. The set of states that are reachable from the states of depth level d is denoted by $\texttt{succ}(Q_d)$. The number of states that can be chosen at each depth level is bounded by parameter N, for $N \geq 0$, this parameter is provided as input to the algorithm.

The highway algorithm is a combination of a breadth first search and random search algorithm. A typical random search implementation will determine the next state to visit by randomly picking a state that is reachable from the last state it has visited. The main difference between the random search and the Highway algorithm, is that the Highway algorithm tries to find N successor states that are reachable from one or more states visited on the current depth level. This is achieved by first computing all unique sets containing N states reachable from the current depth level and thereafter randomly pick one of them. Note that this description for finding the next states, deviates from the original discussion found in [3]. For analysis purposes, a more formal description is needed. This description and the analysis can both be found in [3] and will not be discussed here.

A typical random search implementation also is allowed to visit states it has already visited before. This does not seem to be very efficient since there is no benefit from visiting the same states over and over again. Moreover, in order to use the Highway algorithm to generate a reduced state transition system, revisiting of states is clearly not beneficial. Therefore, the set of electable states can only contain states that have not been visited before.

At this point an algorithm outline can be presented:

Definition 5 (Highway algorithm outline)**.** *For a search width of N:*

- $Q_0 = \{s_0\}$
- *Each Q_i (for $i > 0$) is obtained by randomly picking N states from the set:* $\texttt{succ}(Q_{i-1}) \setminus \bigcup_{j=0}^{i-1} Q_j$

Observing the algorithm outline in definition 5, a straightforward implementation is possible. Algorithm 1 provides an implementation in pseudo code:

Algorithm 1
Procedure *Highway_algorithm*(N, s_0)
1. $V, d, Q_0 \leftarrow \{s_0\}, 0, \{s_0\}$;
2. **while** $Q_d \neq \varnothing$
3. $\quad r = \varnothing$;
4. $\quad$ **for** $s \in Q_d$
5. $\quad\quad r \leftarrow r \cup (\texttt{succ(s)} \setminus V)$;
6. $\quad Q_{d+1} \leftarrow$ 'randomly pick $\texttt{min}(N, |r|)$ elements from r';
7. $\quad V, d \leftarrow V \cup Q_{d+1}, d + 1$;
8. **return** V

One may have noticed that the discussion in this section does not require a state transition system to be finite. There was no specific reason for this additional restriction up till now. Line 6 of algorithm 1 dictates that $\texttt{min}(N, |r|)$ states are picked from the set of not visited reachable states. From this it follows that the algorithm is terminated when no unvisited reachable states are left (hence, $Q_{d+1} = \varnothing$ which negates the guard of the main loop). This automatically means that in case the algorithm is presented with an infinite state transition system, it will possibly not terminate. Therefore, all state transition systems are assumed to be finite.

The Highway algorithm explores several states at each depth level simultaneously, this can be seen as traversing several 'lanes' of a 'highway'. This is why the algorithm is named: 'Highway'. What rests is a time analysis for algorithm 1:

- Line 7, 3, 1. Housekeeping actions are assumed to be: $O(1)$.
- Line 6. Selecting the states to be visited on the next depth level is assumed to be: $O(N)$.
- Line 5. Because V can contain at most $N * d$ states, hence this line is: $O(N * d)$.
- Line 4. At most N iterations are performed here. Together with line 5: $O(N^2 * d)$.
- Line 2. At most d operations are performed here. Together with lines 6, 5, 4: $O(N^2 * d^2)$.

Time complexity of algorithm 1: $O(N^2 * d^2)$.

4.2 Modified Highway for Model Checking

The standard highway algorithm can only be used to explore a finite state transition system and search for a particular state of interest. In this section, some modifications are presented to make the Highway algorithm suitable for model checking purposes. The goal is to provide an algorithm that can generate a reduced finite state transition system (bounded in breadth) that can be used for model checking with existing model check techniques.

In order to generate such a finite state transition system, the transitions from one state to another have to be recorded. Adapting the algorithm in such a way that all transitions from depth level i, for $i \geq 0$, to depth level $i + 1$ are recorded, is not very hard. But still, with this modification, the resulting finite state transition system can not be used for model checking. Some vital information in the form of transitions is missing: back loops. Because the Highway algorithm does not revisit states, these back loop transitions are not recorded, and thus, no infinite paths are created in the reduced finite state transition system. This, however, is a requirement for model check algorithms. Figure 3 illustrates a state transition system in which a back loop transition is not recorded.

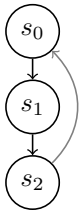

Figure 3: Graph with a back loop; the gray transition is not recorded

The problem of back loops (actually the absence thereof), was also a concern when a bound is applied on the depth of a finite state transition system (see section 3). The solution Biere et. al. presented, was to distinguish paths with loops and paths without loops and then redefine the semantics of the LTL operators. Although an interesting proposition, this solution is not applicable in the context of the Highway algorithm. The depth bounded model check technique as is described in section 3 does record back loops and the distinction between the different types of paths only applies in those situation when a path in the original finite state transition system contains no back loop up and till a certain depth. In the context of the Highway algorithm, back loop transitions are not recorded by definition. In other words, information in the form of transitions is lost within the breadth bound. To counter this problem a solution for recording back loop transitions is presented further down this section. A positive side effect is that there is no need to make a distinction between several types of paths and hence, the standard LTL semantics and model check techniques can be used.

However, there is another type of transition that will not be recorded by the Highway algorithm: the forward jump. Because only transitions from a particular depth level to the next (from depth level i, for $i \geq 0$ to depth level $i+1$) are recorded. Transitions that span over one or more depth levels (forward jumps), will not be present in the resulting state transition system. This type of transition is illustrated in figure 4.

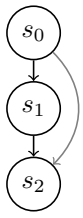

Figure 4: Graph with a forward jump; the gray transition is not recorded in case s_2 is not chosen to be in Q_{i+1} where s_0 is in Q_i

In order to record all possible transitions, some extra modifications are needed. First the different types of transitions that have to be recorded by the modified Highway algorithm are defined more formally. The transitions from one depth level to the next are called 'one-step' transitions. The other types of transitions ('back loop' and 'forward jump') have already been introduced.

Definition 6 (Transitions that are one-step transitions)**.**

$$\bigcup_{i=0}^{d-1} \{\langle s, r\rangle \mid s \in Q_i \wedge r \in Q_{i+1} \wedge s \to r\}$$

Definition 7 (Transitions that are back loops)**.**

$$\bigcup_{i=0}^{d} \bigcup_{j=0}^{i} \{\langle s, r\rangle \mid s \in Q_i \wedge r \in Q_j \wedge s \to r\}$$

Definition 8 (Transitions that are forward jumps)**.**

$$\bigcup_{i=0}^{d-2} \bigcup_{j=i+2}^{d} \{\langle s, r\rangle \mid s \in Q_i \wedge r \in Q_j \wedge s \to r\}$$

Note the -2 and $+2$ in definition 8. The need for this becomes clear when one imagines which transitions actually need to be considered here. The definition includes all transitions that start from a state at a certain depth level and go to a state on a depth level that is at least two levels higher (higher meaning a greater value of the depth level indicator). Definition 6 deals with all transition between states of consecutive depth levels (from depth level i, for $i \geq 0$, to depth level $i+1$).

With the above definitions, the standard Highway algorithm can be adapted. However there is a special situation that has to be avoided. The set containing the states that will be visited on the next depth level can be chosen freely as long as they have not been visited before. But this can give rise to the situation in which a state on depth level i, for $i \geq 0$, has no outgoing transition to a state chosen on depth level $i+1$ or to an already visited state. This situation, is illustrated in figure 5. In this situation, the Highway algorithm is applied with bound $N = 2$. The first two depth levels offer little choice: $Q_0 = \{s_0\}$ and $Q_1 = \{s_1, s_2\}$. But then there is a choice, either $Q_2 = \{s_3, s_4\}$ or $Q_2 = \{s_3, s_5\}$ or $Q_2 = \{s_4, s_5\}$. In case $Q_2 = \{s_3, s_4\}$ is chosen, a deadlock is created because there is no outgoing transition from s_2 to a state in Q_2 or to a state in Q_0 or Q_1.

An extra condition is needed to prevent the Highway algorithm from creating deadlocks. This conditions enforces that all states on depth level i, for $i \geq 0$, have an outgoing transition to either another visited state (the states that are contained in set V) or to a state that is to be visited on the next depth level $i+1$. Definition 9 make this more formal.

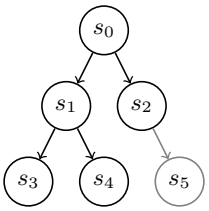

Figure 5: Situation in which a deadlock is created by Highway. Black states and transitions are recorded, gray ones are not

Definition 9 (Additional condition for electing reachable states)**.** *For all i, $0 \leq i < d$:*

$$\forall(s : s \in Q_i : \exists(r : r \in \bigcup_{j=0}^{i+1} Q_j : s \to r))$$

where $s \to r$ means there is a transition from state s to state r.

Now the modified Highway algorithm can be presented. The modifications in pseudo code are more or less a direct translation of definitions 6, 7, 8 and 9. In algorithm 2 a new variable T is used to store the transitions. A transition from state s to state r is denoted as $\langle s, r\rangle$. For the sake of clarity, the part of the algorithm that gathers all forward jumps is presented as a separate algorithm. In fact, this part is like a 'depth-first search' that is performed on the result of algorithm 2.

Algorithm 2
Procedure *Highway_one-step_backloop*(N, s_0)
1. $V, d, Q_0, T \leftarrow \{s_0\}, 0, \{s_0\}, \varnothing$;
2. **while** $Q_d \neq \varnothing$
3. $\quad t, r \leftarrow \varnothing, \varnothing$;
4. $\quad$ **for** $s \in Q_d$
5. $\quad\quad$ **for** $v \in$ `succ(s)`
6. $\quad\quad\quad$ **if** $v \notin V$
7. $\quad\quad\quad\quad$ **then** $t, r \leftarrow t \cup \{\langle s, v\rangle\}, r \cup \{v\}$;
8. $\quad\quad\quad\quad$ **else** $T \leftarrow T \cup \{\langle s, v\rangle\}$;
9. $\quad Q_{d+1} \leftarrow$ 'random `min`$(N, |r|)$ elem. from r' s.t. $\forall(s : s \in Q_d : \exists(r : r \in \bigcup_{j=0}^{d+1} Q_i : s \to r))$;
10. $\quad V, T, d \leftarrow V \cup Q_{d+1}, T \cup \{\langle s, v\rangle \mid v \in Q_{d+1} \wedge \langle s, v\rangle \in t\}, d + 1$;
11. **return** T

Algorithm 3 gathers all the additional forward jumps. For each state, all transition to states on higher depth levels (higher meaning a higher value of the depth level indicator) are found. Variable **Q** is a set containing all sets with stored states at each depth level, hence $\mathbf{Q} = \bigcup_{i=0}^{d}\{Q_i\}$. Note that this algorithm is a direct translation of definition 8.

Algorithm 3
Procedure *Highway_algorithm_with_basic_transition*(N, s_0)
1. $i \leftarrow 0$;
2. **while** $i \leq d - 2$
3. $\quad$ **for** $s \in Q_i$
4. $\quad\quad j \leftarrow i + 2$;
5. $\quad\quad$ **while** $j \leq d$
6. $\quad\quad\quad$ **for** $r \in Q_j$
7. $\quad\quad\quad\quad$ **if** $s \to r \wedge \langle s, r\rangle \notin T$
8. $\quad\quad\quad\quad\quad$ **then** $T \leftarrow T \cup \{\langle s, r\rangle\}$;
9. $\quad\quad\quad j \leftarrow j + 1$;

10. $i \leftarrow i + 1$;
11. **return** T

As is stated earlier, standard model check techniques assume all paths in a state transition system are infinite. The following theorem is introduced which declares that the modified Highway algorithm, as implemented by algorithm 2, preserves this property. Hence, the resulting finite state transition system can be used for model checking.

Theorem 1. *A finite state transition system generated by algorithm 2 will only contain infinite paths, whenever, the unbounded finite state transition system only contains infinite paths*

Proof. Assume that the unbounded finite state transition system only has infinite paths.
For a path to be infinite, all states in the path need to have an outgoing transition. Since all paths are assumed to be infinite, all states have an outgoing transition. For the Highway algorithm, definition 9 dictates that all states on depth level i, for $i \geq 0$, are required to have an outgoing transition either to a state on depth level $i + 1$ or to an already visited state. In other words, the definition requires that either an outgoing 'one-step' transition is recorded or a back loop. From definition 6 and 7 it can be concluded that for all visited states, all 'one-step' and back loop transitions are recorded. Summarizing, definition 9 requires that every state has at least one outgoing 'one-step' or back loop transition and definition 6 and 7 make sure that for all visited states, all 'one-step' and back loop transitions are recorded. Hence, algorithm 2 produces a finite state transition system in which all states have an outgoing transition and thus all paths will be infinite. □

It should be clear that algorithm 3 preserves this property: it is impossible to introduce finite paths by adding additional transitions to visited states.

However, as with bounded model checking, there is a small catch. Once the Highway algorithm has been used to reduce the finite state transition system, only a part of the original finite state transition system is left. This means that, as with bounded model checking, only existentially quantified LTL formulas can be checked. Since states and transitions of the original state transition system are lost by applying the Highway algorithm, it does not make sense trying to prove properties that are quantified over all paths. In case bound N is chosen to be sufficiently large (such that the highway algorithm will include all states of the original state transition system) universally quantified LTL formulas can be checked.

What rests is a complexity analysis of the modified highway algorithm. For this analysis we assume that checking whether or not a transition is possible ($s \rightarrow r$) has time complexity $O(1)$. Although the standard Highway algorithm is modified to store one-step and back loop transitions, the complexity analysis will remain unchanged: $O(N^2 * d^2)$. The time analysis of algorithm 3 is as follows:

- Line 8. Adding one element to set T is done in $O(1)$.
- Line 7. Here a check is performed whether or not the new transition is already contained in set T. If all states at depth level $i - 1$ are connected to all states at level i for $0 < i \leq d$, then there are $N^2 * d$ transitions in set T. Furthermore, in case every state at depth level i is connected via a back loop to every state at all depth levels j with $0 \leq j < i$, then there can be at most $N^2 * d^2$ additional transitions in T. Under the assumption it only takes constant time to check whether or not a transition from s to r is possible, this line has time complexity $O((N^2 * d) + (N^2 * d^2))$.
- Line 6. At most N iterations are performed here. Hence, together with lines 7 and 8: $O((N^3 * d) + (N^3 * d^2))$.
- Line 5. At most d iterations are performed here. Hence, together with lines 6, 7 and 8: $O((N^3 * d^2) + (N^3 * d^3))$.
- Line 4. Initializing a loop variable is done in $O(1)$.

- Line 3. At most N iterations are performed here. Hence, together with lines 5, 6, 7 and 8: $O((N^4 * d^2) + (N^4 * d^3))$.
- Line 2. At most d iterations are performed here. Hence, together with lines 3, 5, 6, 7 and 8: $O((N^4 * d^3) + (N^4 * d^4))$.

Time complexity for algorithm 3: $O(N^4 * d^4)$.

5 Experiments with modified Highway

In order to review the new Highway algorithm, several experiments were conducted. For these experiments, three basic network communication protocols are used: Alternating Bit Protocol (ABP), Concurrent Alternating Bit Protocol (CABP) and Bounded Retransmission Protocol (BRP). The goal of the experiments is to find a prefix of a path (with or without a loop) that satisfies a certain condition in the reduced finite state transition system. Note that since a state or transition can only be present in the reduced finite state transition system, if and only if the same state or transition is also present in the unbounded finite state transition system, it can be concluded that if a witness is found in the reduced finite state transition system, it will also be present in the unbounded one. However, the main question is: how likely will the prefix be found for several values of bound N? For each network protocol the Highway algorithm is used to generate one hundred reduced finite state transition systems for every bound N. The values for N are chosen manually depending on the size of the unbounded finite state transition system and the effectiveness of the Highway algorithm in generating a finite state transition system with the requested prefix.

All experiments described here, were conducted using the mCRL2 tool set. This tool set uses process algebra for specifying state transition systems and μ-calculus to formalize properties. Furthermore, in the mCRL2 tool set, model checking is concerned with sequences of actions instead of sequences of states. Dispite the differences, the mCRL2 tool set can be used to verify LTL properties once they are 'translated' to μ-calculus.

The specifications of the network protocols can be found in the mCRL2 distribution. More information about mCRL2 can be found on the website: `http://www.mcrl2.org`. Additional information about μ-calculus can be found in [2].

5.1 Alternating Bit Protocol

The Alternating Bit Protocol described in [**?**] is a very simple connectionless communication protocol. This protocol incorporates a sender and receiver process and two channel processes (one to deliver a message from the sender to the receiver and one to deliver the acknowledgment from the receiver to the sender). During transmission, messages and acknowledgments can be lost. To detect this, all messages are assigned a Boolean value and only when the sender receives the same Boolean value, used to sent a message, from the receiver, it knows for sure that the receiver has received the massage correctly. In all other cases, the message will be (possibly infinitely) resent.

Since the Alternating Bit Protocol can lose messages during transmission, and therefore a message possibly may never be received, it can be interesting to show that the channel is not broken. In other words, can a sent message always possibly be received by the receiver? This property can be encoded as a LTL formula as follows: $\mathbf{EF}(sent \wedge \mathbf{GF}receive)$. However, the mCRL tool set can not verify this property directly since it uses μ-calculus formulas instead of LTL formulas. The μ-calculus formula $f_\mu = \mu X.(\langle\top\rangle X \vee \langle c2(d1, true)\rangle \nu Y.(\mu Z.(\langle !s4(d1)\rangle Y \vee \langle s4(d1)\rangle Z))$ represents the LTL property $f_{LTL} = \mathbf{EF}(c2(d1, true) \wedge \mathbf{GF}s4(d1))$. Note that $c2(d1, true)$ denotes an action which represents that message $d1$ is sent and $s4(d1)$ denotes an action which represents that message $d1$ is received.

For each bound, starting at $N = 1$, the Highway algorithm is run one hundred times. For each resulting finite state transition system, the property f_μ is checked. Bound N is increased until the property is consistently satisfied for all finite state transition systems. The results are based

on the Alternating Bit Protocol with support for four different messages and are shown in table 1. Note that the number of states and transition can vary. These numbers are averages.

Bound N	**# Property satisfied**	**# States**	**# Transitions**
1	1 (of 100)	12	12
2	0 (of 100)	27	30
3	14 (of 100)	39	44
4	38 (of 100)	58	67
5	48 (of 100)	75	88
6	59 (of 100)	85	100
7	78 (of 100)	102	121
8	88 (of 100)	106	126
9	100 (of 100)	120	144
10	100 (of 100)	141	162

Table 1: ABP: a sent message can always possibly be received

As can be seen in table 1, the Highway algorithm is reasonably effective in generating a finite state transition system which includes a witness for the property that is checked. Already with bound 3 and a significant smaller finite state transition system, a reasonable number of witnesses is found. When the bound is chosen to be larger than 8, all generated finite state transition systems include a witness. Although it is assuring that specific witnesses can be found, the Alternating Bit Protocol with four messages only has 146 states and 184 transitions when no bound is used. Therefore it might be interesting to consider an experiment with a larger finite state transition system.

5.2 Concurrent Alternating Bit Protocol

The second experiment that was conducted for this paper deals with the Concurrent Alternating Bit Protocol. This protocol described in [6] is a variation on the standard Alternating Bit Protocol discussed earlier in this paper. The main difference is that instead of the sender waiting for an acknowledgment from the receiver, the sender just keeps on sending the message until an acknowledgment is received.

Although the protocol does not differ very much from original Alternating Bit Protocol, the finite state transition system of the protocol with support for four messages, is about 6.5 times larger (1040 states and 3808 transitions). For the Concurrent Alternating Bit Protocol, the same property can be checked. Hence, the property to check is: can a sent message always possibly be received by the receiver? Due to differences in the specification of the protocol, names and parameters of actions have changed. For completeness, the new LTL formula is as follows: $f_{LTL} = \mathbf{EF}(c3(frame(d1, bit0)) \wedge \mathbf{GF}s2(d1))$. This property is again translated to a μ-calculus formula, which leads to the following: $f_\mu = \mu X.(\langle\top\rangle X \vee \langle c3(frame(d1, bit0))\rangle \nu Y.(\mu Z.(\langle !s2(d1)\rangle Y \vee \langle s2(d1)\rangle Z))$. In these formulas, action $c3(frame(d1, bit0))$ denotes the sender sending a 'frame'. This frame consists of a message $d1$ and a Boolean value $bit0$ which is expected from the receiver as acknowledgment. Action $s2(d1)$ denotes the receiver actually delivering the message to the 'outside world'.

During the experiment, bound N, used to generate the bounded finite state transition systems, is increased with steps of three. For each bound, one hundred finite state transition systems are generated and checked. The results are shown in table 2

As can be seen from the results in table 2, it has already become harder to consistently find a witness. But, the increasing bound definitely has an effect on the effectiveness of the Highway algorithm. Starting with a bound of 22, consistently at least 50 out of 100 transition systems include the witness. Note that for larger bounds, the number of states and transitions visited does not increase much. This seems to be quite logical since an increase of the bound by three, only allows three more states to be visited at each depth level. It is expected that the bound has to be

significantly larger in order to equal the number of states and transitions of the unbounded finite state transition system. A consistent 100 out of 100 rating is only expected when the bounded finite state transition system approaches the unbounded one.

5.3 Bounded Retransmission Protocol

The Bounded Retransmission Protocol as it is described in [4] is a communication protocol designed for packages that are too large to be sent at once. In order to send such messages, the message is split into smaller packages which are sent one by one. Each package is assigned additional information to indicate if the package is the first, last or an intermediate package of the sequence. The network topology is very similar to that of the Alternating Bit Protocol with the exception that the Bounded Retransmission Protocol features two timers which provide a time-out mechanism for both the sender and receiver. Moreover, packages are only sent a bounded number of times. This means, that once the sender does not receive an acknowledgment message (or an incorrect one), the sender will eventually give up resending the message.

The finite state transition system of the Bounded Retransmission Protocol with support for messages containing three packages, is about 10 times larger than the finite state transition system of the Concurrent Alternating Bit protocol and 65 times larger than the standard Alternating Bit Protocol (10548 states and 12168 transitions). It would be interesting to see how effective the Highway algorithm is in generating a finite state transition system which includes a witness for a similar property as the property that was checked for the Alternating Bit Protocol. The property to be checked declares that there is a message read by the sender (read from the 'outside world') which can always possibly will be sent. This yields the following LTL formula: $f_{LTL} = \mathbf{EF}(r1([a,b,c]) \wedge \mathbf{GF}(s1(I\_ok)))$. Note that in this formula $r1([a,b,c])$ denotes that a message is read from the 'outside world' consisting of a sequence of packages (namely the list $[a,b,c]$). The action $s1(I\_ok)$ denotes the message delivered back to the 'outside world' indicating that the message was sent successfully. When this LTL formula is translated to μ-calculus, the following formula is obtained: $f_\mu = \exists_{a:D,b:D,c:D} \mu X.(\langle\top\rangle X \vee \langle r1([a,b,c])\rangle \nu Y.(\mu Z.(\langle !s1(I\_ok)\rangle Y \vee \langle s1(I\_ok)\rangle Z)))$

The bound is increased until the property is consistently satisfied for all finite state transition systems generated by the Highway algorithm. The results are shown in table 3

The results in table 3 show that a witness for property f_μ can be found fairly easy. The only exception here is the poor performance when a bound of 2 is used. One key difference between the Bounded Retransmission Protocol and the other protocols, is that the finite state transition system of the Bounded Retransmission Protocol has a larger set of initial states. Observing the reduced finite state transition systems (in particular for bound 2) it can be concluded that the

Bound N	**# Property satisfied**	**# States**	**# Transitions**
1	5 (of 100)	57	107
4	13 (of 100)	216	935
7	29 (of 100)	474	1444
10	26 (of 100)	527	1537
13	36 (of 100)	621	1826
16	43 (of 100)	663	1891
19	48 (of 100)	750	2367
22	52 (of 100)	807	2436
25	52 (of 100)	846	2787
28	65 (of 100)	851	2804
31	51 (of 100)	873	2910
34	72 (of 100)	893	3011
37	83 (of 100)	948	3280
40	89 (of 100)	950	3315

Table 2: CABP: a message can always possibly be received

states in the initial set are not picked randomly by the Highway algorithm implementation. This means that the results for the Bounded Retransmission Protocol are somehow unreliable. The finite state transition systems generated for the Alternating Bit Protocol and the Concurrent Alternating Bit Protocol show no signs of this problem. However, a more in depth analysis of the problem is needed in order to be absolutely sure. Due to time requirements, the results of this in depth analysis are not included in this paper.

5.4 Results

The conducted experiments suggest that, although the finite state transition systems are relatively small, the Highway algorithm is reasonably effective in generating a reduced finite state transition system in which a witness for a particular property is found. Even when a small bound is used, there is a fair chance of finding the witness, in particular when multiple runs are performed with the same bound. The effectiveness of the algorithm increases when the bound is increased too, quickly leading to a point where more than 50% of the generated finite state transition systems include the witness. Reaching a 100% rating is not in all cases self-evident without increasing the bound to a level on which the resulting finite state transition system approaches the unbounded one in size. This may seem to be a bit disappointing, but note that a 100% rating is not necessarily required. Finding a witness in a reduced finite state transition system, automatically means that this witness is also present in the unbounded finite state transition system. From this point of view, a high probability of finding the witness can be considered good enough.

6 Future work

Although the modified Highway algorithm can be used to perform model checking, there still are some related and open topics that may be worthwhile to explore.

Although the results from the experiments look very promising, more experiments should be conducted in order to see how useful the technique is when applied to very large finite state transition systems. Only with such experiments, one can conclude whether or not this technique is valuable enough to be used when finite state transition systems exceed the limits for which standard and complete model check techniques are feasible. Also, it would be interesting to see how this technique measures up to the bounded depth model check technique introduced by Biere et. al. in [1].

Moreover, the Highway algorithm as is described in this paper is only used to reduce the number of states of a finite state transitions system. Although standard model check techniques can be used on the resulting finite state transition system, it would be interesting to perform model checking while the Highway algorithm explores the finite state transition system. Since on-the-fly LTL model check techniques can be implemented as a nested depth-first search algorithm, it might be interesting to investigate whether or not such an algorithm can be integrated in the last part

Bound N	**# Property satisfied**	**# States**	**# Transitions**
1	89 (of 100)	42	42
2	7 (of 100)	134	138
3	98 (of 100)	192	203
4	98 (of 100)	248	258
5	100 (of 100)	309	326
6	100 (of 100)	377	402
7	100 (of 100)	424	447
8	100 (of 100)	485	518
9	100 (of 100)	490	519
10	100 (of 100)	640	678

Table 3: BRP: a sent message can always possibly be received

of the Highway algorithm. This part of the algorithm, algorithm 3, also performs a depth-first search in order to find all forward loops in the finite state transition system.

Another interesting direction would be to investigate how and if this technique can be combined with the bounded depth model check technique. Of course it is possible to use the technique presented in [1] after the Highway algorithm has reduced the finite state transition system using a breadth bound. However, it would be valuable to find a way to integrate both techniques. By doing so, a very versatile bounded model check technique is created which is capable of bounding the depth of a finite state transition system as well as the breadth.

7 Conclusion

Inspired by the work done in the field of bounded incomplete model check techniques [1], this paper has presented a technique that can be used to reduce a finite state transition system by bounding its breadth. With this technique, one can perform breadth bounded model checking, for which up till now, no technique exists in the literature. The theory for breadth bounded model check techniques is further developed into the Highway algorithm which is based on the Highway search algorithm found in [3]. This algorithm generates a breadth bounded reduced finite state transition system which can be considerably smaller then the unbounded one. The reduced finite state transition system can be used to find witnesses for existentially quantified LTL properties.

In order to conduct some experiments, the Highway algorithm has been implemented in the mCRL2 tool set. With this implementation, some test cases, based on the Alternating Bit Protocol, Concurrent Alternating Bit Protocol and Bounded Retransmission Protocol have been setup and executed. Although the unbounded finite state transition systems for these network communication protocols are relatively small, the results show that the Highway algorithm is reasonably effective in generating bounded finite state transition systems which include the witness searched for. Already with a small bound, witnesses are found and when the bound is slightly increased, more than 50% of the finite state transition systems satisfy the property checked for.

Although promising, future research has to show that the technique is equally effective when used with much larger state transition systems. This is particularly interesting since the Highway algorithm will most likely be used in settings in which model checking with unbounded finite state transition systems is not feasible. Furthermore, future research could investigate the possibility of integrating the Highway algorithm with existing depth-first search LTL model check algorithms. Another interesting proposition is to integrate the technique with existing bounded depth model check techniques.

References

[1] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58, 2003.

[2] E. Clarke, G. O., and D. Peled. *Model Checking*. MIT Press, Cambridge, 1999.

[3] T. Engels, J. Groote, M. Weerdenburg, van, and T. Willemse. Search algorithms for automated validation. *Approved for publication in: Journal of Logic and Algebraic Programming*.

[4] J. F. Groote and J. van de Pol. A bounded retransmission protocol for large data packets. In *AMAST '96: Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology*, pages 536–550, London, UK, 1996. Springer-Verlag.

[5] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL '85: Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 97–107, New York, NY, USA, 1985. ACM.

[6] S. Mauw and G. Veltink. *Algebraic Specification of Communication Protocols*. Cambridge University Press, 1993.