Maximal Synthesis for Hennessy-Milner Logic

A.C. van Hulst, Eindhoven University of Technology
M.A. Reniers, Eindhoven University of Technology
W.J. Fokkink, VU University Amsterdam

This paper concerns the maximal synthesis for Hennessy-Milner Logic on Kripke-structures with labeled transitions. We formally define, and prove the validity of, a theoretical framework which modifies a Kripke-model to the least possible extent, in order to satisfy a given HML formula. Applications of this work can be found in the field of controller synthesis and supervisory control for discrete-event systems. Synthesis is realized technically by first projecting the given Kripke-model onto a bisimulation-equivalent partial tree representation, thereby unfolding up to the depth of the synthesized formula. Operational rules then define the required adaptations upon this structure in order to achieve validity of the synthesized formula. Synthesis might result in multiple valid adaptations, which are all related to the original model via simulation. Each simulant of the original Kripke-model, which satisfies the synthesized formula, is also related to one of the synthesis results via simulation. This indicates maximality, or maximal permissiveness, in the context of supervisory control. In addition to the formal construction of synthesis as presented in this paper, we present it in an algorithmic form and analyze its computational complexity. Computer-verified proofs for two important theorems in this paper have been created using the Coq proof assistant.

Categories and Subject Descriptors: I.2.8 [Problem Solving, Control Methods, and Search]: Control theory

General Terms: Controller Synthesis

Additional Key Words and Phrases: Supervisory Control, Hennessy-Milner Logic, Maximal Permissiveness

ACM Reference Format:
DOI: http://dx.doi.org/10.1145/0000000.000000

1. INTRODUCTION

In this paper, we investigate a method to apply the least adaptations to a Kripke-structure with labeled transitions, in order to satisfy a formula in Hennessy-Milner logic [Hennessy and Milner 1985]. Synthesis, as defined in this way, preserves the most possible behavior under simulation, and is therefore maximally permissive. We view this article as a contribution to the field of controller synthesis and supervisory control theory for discrete event systems [Ramadge and Wonham 1987].

We model system behavior using Kripke-structures with added labels on transitions [Bull and Segerberg 1994]. This formalism eloquently captures both the process dynamics, as well as the state-based information of the system. This structure will be referred to in the remainder of this paper as KTS. A test for state-based properties is added to Hennessy-Milner Logic, to be able to refer to state-labels in formulas. Within this context, resolving the synthesis problem was not straightforward. A major difficulty, for instance, is formed by states playing different roles at various stages of synthesis, if loops are involved. Therefore, a partial tree representation of the original KTS is obtained under bisimulation equivalence. This partial tree allows an embedded unfolding up to the depth of the synthesized formula. Operational rules then define the required modifications within this unfolded structure, in order to satisfy the given HML formula.

Synthesis results in a set of outcomes, as multiple solutions are induced by disjunction and the existential modality $<> f$. Each result in the synthesized set satisfies the given HML formula, and is related to the original KTS via simulation. Each simulant of the original KTS, which satisfies the synthesized formula, is related to one of
the synthesis results via simulation. A general overview of the synthesis process is depicted in Fig. 1.

![Diagram of synthesis process]

Fig. 1. Illustrating key parts of the synthesis process. The first step consists of unfolding the KTS up to the depth of synthesized formula in the form of a partial tree representation of behavior. After the actual synthesis itself is applied, each obtained result is again converted back into a KTS.

In the field of modeling and verification, an approach based on labeled transition systems and logical expressions is often employed. Within this two-folded setup, discrete-event models are used to capture process dynamics as well as state-based information. Dually, desired behavior is often specified in a declarative way, where modal or temporal logic is used to state requirements. Synthesis as presented in this paper modifies a discrete-event model in order to satisfy a given logical expression, and is therefore in line with the existing modeling approach. Additionally, synthesis is defined upon non-deterministic models with no limitations on cyclic behavior. This allows for modeling the input-KTS in the broadest possible way.

The main technical contribution of this paper is a definition of maximally permissive synthesis on non-deterministic transition systems that connects to existing modeling approaches. Synthesis as presented in this paper modifies the transition system to the least possible extent, a property often referred to as maximal permissiveness [Cassandras and Lafortune 1999]. It does not remove a state or transition unless such removal is absolutely necessary to obtain validity of the synthesized requirement. If further analysis is to be applied to the synthesized KTS, for instance to investigate issues regarding liveness or optimization, it is of the utmost importance that as much behavior as possible is preserved. In terms of the simulation preorder, this is expressed as maximality with respect to all simulators which satisfy the synthesized formula.

Part of the work presented in this paper appeared earlier [van Hulst et al. 2013]. A notable difference is an extension of the result for maximality, since the presented synthesis technique is now proven to be maximal with respect to all non-deterministic simulators, as opposed to maximality with regard to only deterministic simulators, as in [van Hulst et al. 2013]. Also, computer verified proofs are now available [van Hulst 2013] for two fundamental theorems, in order to better assess the validity of the theories presented here.

The remainder of this paper is set up as follows. Section 2 considers the related work, regarding this research. In Section 3, a number of preliminary definitions for the synthesized logic and behavioral relations are introduced. In Section 4, we consider a number of specific cases to gain understanding of the synthesis construction at an intuitive level. A partial tree representation of a KTS is introduced in Section 5 as well as the conversion functions between this formalism and the standard KTS. Synthesis is then defined formally in Section 6, via a number of operational rules upon the partial tree representation of behaviors. Section 7 presents results for termination and a complexity analysis of the presented synthesis method. Two theorems show the validity of the synthesis approach in Section 8, while maximal permissiveness is shown...
Maximal Synthesis for Hennessy-Milner Logic

2. RELATED WORK

Previous and ongoing similar research considers several other modal and temporal logics. In [Antoniotti 1995], controller synthesis is investigated for a fragment of CTL, thereby imposing a restriction upon the expressibility of disjunctive formulas. The aforementioned approach is built upon a labeling algorithm from model checking introduced in [Kupferman et al. 2001]. The type of synthesis introduced in [Antoniotti 1995] is shown to be NP-complete in [Antoniotti and Mishra 1995]. The work in [Clarke and Emerson 2008] uses a quite different approach. It concerns a type of synthesis which consists of generating an abstract program description, called synchronization skeleton, based on a specification in branching time temporal logic. If a program requirement is satisfiable, a consistent specification is obtained via a bounded finite model property.

Model checking and generalized supervisory control are combined in [Ziller and Scheider 2005] in order to obtain a set of \( \mu \)-calculus equations representing a set of desired behaviors, according to a requirement specified in \( \mu \)-calculus. However, the approach in [Ziller and Scheider 2005] is based upon language-based supervisory control, which is unable to address non-deterministic system specifications, as shown in [Baeten et al. 2011]. Also, since the approach in [Ziller and Scheider 2005] results in a set of \( \mu \)-calculus equations, the post-synthesis formalism is different compared to that of the system specification, and therefore several issues may arise when converting the synthesis solution to a new, actual realization of the adapted system.

Again, a different approach is taken in [Deshpande and Varaiya 1996]. Here, semantic tableaux are used as part of the state of a controller constructed using a propositional LTL specification. Action inhibition via blocking of controllable signals is employed in order to gain validity of the synthesized requirement. The work in [Manna and Wolper 1984] also uses propositional temporal formulas in order to synthesize synchronizations in communicating processes, using a tableau-like method. Similar to the research presented in this paper, the technique described in [Manna and Wolper 1984] uses a separate unfolding step. In [Lüttgen and Vogler 2011], an alternative synthesis method is described, based on ready simulation as a behavioral relation. This method constructs a new behavioral model via embedding of a temporal specification in a Logic LTS. This introduces a technique for system specification via a mixed operational and specification formalism.

Several approaches to synthesis based on Rabin automata are described in the literature. In [Vardi 1996], a behavioral specification is mapped to such an automaton, upon which a program is extracted based on a proof that a given LTL specification is realizable. This approach is comparable, to a certain extent, with the technique introduced in [Jiang and Kumar 2006], where a restricted CTL* formula is mapped to a Rabin tree automaton, which is then converted algorithmically into a deterministic supervisor.

An important topic in the field of controller synthesis concerns the realization of liveness properties [D’Ippolito et al. 2010], [D’Ippolito et al. 2013]. Using the framework of the world machine model, a distinction is made between controlled and monitored behavior, and between system goals and environment assumptions in [D’Ippolito et al. 2010]. Synthesis for a relatively expressive set of liveness properties stated in fluent temporal logic is achieved in [D’Ippolito et al. 2010]. The employed method is to derive a controller from a winning strategy in a two-player game between original and required behavior, expressed in terms of the notion of generalized reactivity, as introduced in [D’Ippolito et al. 2013].
Synthesis, defined via an adaptation of existing behavior as in this paper, can be compared to incremental model checking, as introduced in [Sokolsky and Smolka 1994], which is based upon a linear-time algorithm [Cleaveland and Steffen 1993], which performs global (non-incremental) computation of fixed points. The main advantage of incremental model checking is an average-case linear time adjustment of the validity check of a modal formula. It might therefore be possible to construct an alternative implementation of synthesis for HML-formulas via ad-hoc transition removal and application of the incremental model checking algorithm as proposed in [Sokolsky and Smolka 1994]. However, the key difference is that incremental model checking takes as its input a set $\Delta$ of changes to the transition relation, while the behavioral modifications are computed during synthesis in this paper. It is therefore not entirely clear whether the synthesis setup presented here is compatible with incremental model checking, and whether these methods might be integrated in an efficient manner.

Reactive synthesis has been introduced in [Pnueli and Rosner 1989], and has been investigated for the $\mu$-calculus in [Kupferman and Vardi 2000]. In reactive synthesis, a system specification is constructed from a specification in modal or temporal logic of a relation between the input and output of the system. Further developments of reactive synthesis, such as the synthesis for recursive-components libraries [Lustig and Vardi 2013] and the related synthesis of hierarchical systems [Aminof et al. 2014], resemble, to a certain extent, the type of synthesis which is defined in this paper. In this work on reactive synthesis, a high-level type of synthesis is defined to perform synthesis via combination of a set of re-usable components, according to a given LTL-like specification. Via a deterministic transducer (i.e., a finite-state machine with outputs) a so-called “call-and-return” control flow structure is obtained, which combines the invocations of a set of existing components according to the requirement. As mentioned earlier, this type of language-based control is not suitable for supervisor synthesis for non-deterministic systems [Baeten et al. 2011]. Also, the approach taken in [Lustig and Vardi 2013] does not construct a satisfying solution, but instead complements the solution to the inverse synthesis problem in such a way that it accepts composition trees which do satisfy the specification. It is questionable whether the inverse approach, via augmenting the solution to the inverse synthesis problem, is compatible with the approach taken in this paper where multiple valid solutions may be obtained, even though the approach via tree-automata is comparable, to a certain extent.

The approach in this paper is different compared to earlier work mentioned here. Several methods have been described in the literature in which synthesis is much closer to requirement satisfaction [Clarke and Emerson 2008], [Kupferman and Vardi 2000], and [Vardi 1996], instead of defining synthesis as an adaptation of existing behaviors. Other approaches do so, but disjunctive formulas are considered to a lesser extent in [Antoniotti 1995] and [Jiang and Kumar 2006], among other logical restrictions. In the research presented in this paper, the type of behavioral model is the same before and after synthesis, while a different post-synthesis formalism is used in [Lütgten and Vogler 2011] and [Ziller and Scheider 2005]. Several other approaches do not take maximal permissiveness into account, such as [Manna and Wolper 1984], [Lütgten and Vogler 2011], and [Deshpande and Varaiya 1996]. We view the inclusion of non-determinism in behavioral models as an important improvement, while other research is limited to deterministic transition systems [Kupferman and Vardi 2000], [Jiang and Kumar 2006], and [Ziller and Scheider 2005].

Standard supervisory control theory, as described in in [Ramadge and Wonham 1987], is expressed in terms of absence of deadlock and marker state reachability. Furthermore, it differentiates between controllable and uncontrollable events. These properties are not taken into account in this paper. The main contribution of this research is a formal definition of maximal synthesis for non-deterministic system speci-
We require $L$ for all $e$. Assuming these definitions at a global level brings clarity to the definition of synthesis for more expressive modal specification formalisms [van Hulst et al. 2014], [van Hulst et al. 2015], but due to the nature of these approaches, a proper definition of synthesis for disjunction, as appears in this paper, is not possible.

3. DEFINITIONS

We assume the existence of a set of events $E$, a set of basic properties $P$ and a set of basic states $X$. The basic properties are related to states via a labeling function $L : X \mapsto 2^P$. Basic properties are used to capture the state-based information of a system. We say that basic property $p \in P$ holds in state $x \in X$ if and only if $p \in L(x)$. Assuming these definitions at a global level brings clarity to the definition of synthesis later on, while not limiting the scope of this definition. Since labels are not added or removed from states during synthesis, the decision to define $P$ at a global level does not affect the synthesis semantics.

In this paper, two types of structures are used to express structural behavior. The first is the Kripke-structure with labeled transitions (KTS), whose definition is reiterated here. The second is a partial tree representation upon which synthesis is defined. The latter structure will be defined in Section 5. We assume that every KTS is finitely branching, which means that each state has finitely many outgoing transitions. Both the KTS as well as the partial tree representation capture process dynamics via labeled transitions between states.

**Definition 3.1.** Given a state-space $X \subseteq X$, transition relation $\rightarrow \subseteq X \times E \times X$, and initial state $x$, a KTS $g$ is defined as the following tuple:

$$g = (X, \rightarrow, x)$$

The notation $G$ will be used to denote the universe of KTSs. For transition relation $\rightarrow \subseteq X \times E \times X$, we use $x \rightarrow x'$ to indicate that $(x, e, x') \in \rightarrow$. We use the standard simulation preorder and bisimulation equivalence [van Glabbeek 1993] to relate elements in $G$, according to the definitions shown hereafter. Note that unlabeled transition systems are used in [van Glabbeek 1993], but results and definitions can be adapted and applied straightforwardly.

**Definition 3.2.** For $g' = (X', \rightarrow', x') \in G$ and $g = (X, \rightarrow, x) \in G$ we say that $g'$ is simulated by $g$ (denoted by $g' \preceq g$) if there exists a relation $R \subseteq X' \times X$ such that $(x', x) \in R$ and for all $(y', y) \in R$ we have:

1. We require $L'(y') = L(y)$
2. For all $e \in E$ and $z' \in X'$ such that $y' \rightarrow' z'$, there exists a $z \in X$ such that $y \rightarrow z$ and $(z', z) \in R$.

We use $g' \preceq R g$ to indicate that $g' \preceq g$ as witnessed by $R$.

The first clause of Definition 3.2 requires equality of the sets of satisfied basic properties. This reflects the synthesis semantics where validity is enforced by removal of transitions, while state-based properties are not adjusted.

**Definition 3.3.** If $g' \preceq R g$ and $g' \preceq R^{-1} g$ according to Definition 3.2, then we say that $g'$ and $g$ are related via bisimulation (notation: $g' \leftrightarrow g$). Again, we will use the notation $g \leftrightarrow R g'$ to indicate that $g$ and $g'$ are related via bisimulation, as witnessed by $R$.
Simulation is reflexive as witnessed by the identity relation on $\mathcal{G}$ and transitive by composition of the two underlying witness relations. Bisimilarity is reflexive and transitive for the very same reasons but it is symmetric by the inverted witness relation as well, and therefore an equivalence. These are standard results [van Glabbeek 1993].

We extend the standard definition of formulas in Hennessy-Milner Logic (HML) [Hennessy and Milner 1985] with a test for basic properties $p$, for $p \in \mathcal{P}$, and its negation $\neg p$, as shown in Definition 3.4.

**Definition 3.4.** The set $\mathcal{F}$ is defined for $p \in \mathcal{P}$ and $e \in \mathcal{E}$ as follows:

$$\mathcal{F} := \{ \text{true} \mid \text{false} \mid p \mid \neg p \mid \mathcal{F} \land \mathcal{F} \mid \mathcal{F} \lor \mathcal{F} \mid [e] \mathcal{F} \mid <e> \mathcal{F} \}$$

The formulas $\text{true}$ and $\text{false}$ indicate truth and falsehood respectively, while the formula $p$, for $p \in \mathcal{P}$, can be used to test whether basic property $p$ holds in a specific state. Negation $\neg p$ is defined for state-based properties only. The meaning of the operators for conjunction $\land$ and disjunction $\lor$ is as expected. The universal modality $[e] f$ tests whether $f$ holds after every $e$-step, while $<e> f$ tests for the existence of an $e$-step after which $f$ holds. Negation at the level of basic properties is sufficient to extend the operator $\neg$ to the full set $\mathcal{F}$, as shown in [Hennessy and Milner 1985]. We use a specific measure on formulas, given in Definition 3.5.

**Definition 3.5.** We define $\text{depth} : \mathcal{F} \rightarrow \mathbb{N}$ for $p \in \mathcal{P}$, $e \in \mathcal{E}$ and $f, f_1, f_2 \in \mathcal{F}$ in the following way:

\[
\begin{align*}
\text{depth} (\text{true}) &= 0 \\
\text{depth} (\text{false}) &= 0 \\
\text{depth} (p) &= 0 \\
\text{depth} (\neg p) &= 0 \\
\text{depth} (f_1 \land f_2) &= \max (\text{depth} (f_1), \text{depth} (f_2)) \\
\text{depth} (f_1 \lor f_2) &= \max (\text{depth} (f_1), \text{depth} (f_2)) \\
\text{depth} ([e] f) &= 1 + \text{depth} (f) \\
\text{depth} (<e> f) &= 1 + \text{depth} (f)
\end{align*}
\]

We express the validity of formulas in $\mathcal{F}$ with respect to a KTS $g \in \mathcal{G}$ using the valuation function $\models$ given in Definition 3.6.

**Definition 3.6.** The predicate $\models$ over $\mathcal{G} \times \mathcal{F}$ is defined for $X \subseteq \mathcal{X}$, $\rightarrow \subseteq \mathcal{X} \times \mathcal{E} \times \mathcal{X}$, $g \in \mathcal{G}$, $f, f_1, f_2 \in \mathcal{F}$, $e \in \mathcal{E}$, $x \in \mathcal{X}$, $p \in \mathcal{P}$ by the following deduction rules:

\[
\begin{align*}
\frac{g \models \text{true}}{p \in \mathcal{L}(x)} \\
\frac{g \models \text{false}}{p \not\in \mathcal{L}(x)} \\
\frac{g \models f_1 \land f_2}{g \models f_1} \\
\frac{g \models f_1 \land f_2}{g \models f_2} \\
\frac{g \models f_1 \lor f_2}{g \models p} \\
\frac{g \models p}{g \not\models \neg p} \\
\frac{g \models f_1}{g \models <e> f} \\
\frac{g \models f_2}{g \models [e] f} \\
\frac{\forall x. e \xrightarrow{<e>} x' (X, \rightarrow, x) \models f}{g \models \langle e \rangle f}
\end{align*}
\]

If $g \models f$, then we say that $g$ satisfies the formula $f$. Note that validity for HML formulas is invariant under bisimulation, as shown in [Hennessy and Milner 1985].

4. INTRODUCTION TO SYNTHESIS

In this section we work towards an appropriate definition of synthesis, thereby illustrating the various constructions involved, as well as several caveats we encountered. A formal definition of synthesis will then follow in the later sections in terms of a partial tree representation of the KTS.

We first take a closer look at synthesis for the various elements in $\mathcal{F}$. It is clear that synthesis of $\text{true}$ should be neutral as no modification of the KTS is required to satisfy this formula. On the other hand, synthesis of the formula $\text{false}$ should not yield any result because no possible modification to the original structure exists which achieves validity for this formula. The formulas $p$, for $p \in \mathcal{P}$, are always evaluated and synthesized with respect to a single state $x$ in the state-space $\mathcal{X}$. If $p \in \mathcal{L}(x)$, then
Maximal Synthesis for Hennessy-Milner Logic

synthesis should be the same as if the formula were true, where no modification of the KTS is required. On the other hand, if \( p \not\in L(x) \), then the formula should be treated as if it were false and no synthesized result should be returned. Note that assigning the basic property \( p \) to \( x \) if \( p \not\in L(x) \) is not desired, as this would add information to the KTS, thereby invalidating the basic principle of synthesis of not introducing new behavior or new system properties. The inverse procedure is followed for the negation of basic properties \( \neg p \). If \( p \not\in L(x) \), then no modification needs to be applied to satisfy the formula \( \neg p \). However, if \( p \in L(x) \) then the formula \( \neg p \) can not be satisfied for \( x \) and therefore synthesis should not result in a satisfying model.

We continue by illustrating synthesis for additional elements of \( \mathcal{F} \), thereby first considering the operators \([e]f\) and \(\langle e \rangle f\), since any non-trivial example regarding the operators \(\land\) and \(\lor\) uses \([e]f\) or \(\langle e \rangle f\). For the operator \([e]f\), for \( f \in \mathcal{F} \), we apply synthesis for the formula \( f \) recursively after each \( e \)-step. If such synthesis can not be performed, for instance if \( f = false \), then we remove the corresponding \( e \)-step. On the other hand, if synthesis is successful, the \( e \)-step is retained and the KTS is modified recursively after the \( e \)-step in order to satisfy \( f \). At this point, it is important to stress that deleting a disallowed behavior does not contradict the maximality requirement, since maximality is defined with respect to all satisfying simulators. Synthesis for the formula \([a]p\) is shown in Fig. 2.

![Fig. 2. A straightforward example showing synthesis for the formula \([a]p\) for a basic property \( p \in \mathcal{P} \). The top-most transition is removed since its target state has not been assigned the label \( p \).](image)

For the operator \(\langle e \rangle f\) for \( f \in \mathcal{F} \), we attempt to synthesize \( f \) in each \( e \)-step. If none of these attempts is successful, synthesis of the operator \(\langle e \rangle f\) does not result in a valid outcome. Otherwise, synthesis proceeds recursively after an \( e \)-step while the all other transitions are left in place unmodified. In addition, we have to take into account the maximality requirement. Therefore, in order to give an appropriate definition of the synthesis for the existential modality \(\langle e \rangle\), we have to include the unmodified transitions following the corresponding \( e \)-step as well. Note that, analogous to the synthesis for \([e]f\), the synthesis for \(\langle e \rangle f\) might result in multiple solutions if \( f \) can be synthesized in multiple ways after the \( e \)-step. Synthesis for the formula \(\langle a \rangle [b]p\) is shown in Fig. 3. Note that copied behavior due to synthesis for the formula \(\langle a \rangle\) is indicated using dashed lines.

![Fig. 3. An example showing synthesis for the formula \(\langle a \rangle [b]p\). Retained behavior due to synthesis for the operator \(\langle e \rangle\) is indicated using dashed lines. The modification in the lower branch results in a witness \( a \)-step after which \([b]p\) is satisfied.](image)
We now proceed by considering the operators $\lor$ and $\land$. In Fig. 4a)-4c) it is shown how multiple valid adaptations might exist which all satisfy $\langle c \rangle p \lor \langle c \rangle q$. However, these solutions cannot be combined in any meaningful way in a single transition system, in such a way that the desired properties of our synthesis method are respected. The two results are incomparable and therefore it is unclear whether one should be preferred over the other. This should result in a definition of synthesis that includes multiple possible valid modifications to the input-KTS. Therefore, we will define the synthesis in the latter sections accordingly. Note that this remark also influences the definition of synthesis for the operators $\langle e \rangle f$ and $\langle e \rangle f$. As multiple valid adaptations might exist for $f$ after an $e$-step, each solution should result in a new instance where the $e$-step is combined with a synthesis result for $f$.

![Diagram](image)

Fig. 4. Synthesis for the formula $\langle e \rangle p \lor \langle e \rangle q$ on the system shown in a) might result in multiple valid adaptations b) and c) which are essentially incomparable. Therefore, synthesis is defined in such a way that it results in a set of synthesized products.

The operator $\land$ introduces additional complications. As shown in Fig. 5a)-5d), multiple applications of synthesis for each conjunct might be required to obtain a synthesis result which satisfies both formulas. The input-KTS as shown in 5a), is modified in order to satisfy the formula $\langle a \rangle [b]q \land [a] \langle b \rangle p$. The end result, as shown in Fig. 5d), is obtained via the intermediate steps 5b) and 5c). Synthesis of $\langle a \rangle [b]q$ is applied to the original in 5a), resulting in 5b). Consequently, we apply synthesis for $\langle a \rangle \langle b \rangle p$ in 5b), resulting in 5c). In the last step, synthesis for $\langle a \rangle [b]q$ in 5c) results in 5d), which also satisfies the second conjunct. Observe that both conjuncts have now become satisfied and we do not require any more applications of synthesis.

We generalize this process in the formal definition of synthesis later on, where synthesis for conjunction will be defined as a fixpoint construction that alternatingly applies synthesis for both conjuncts. Note that two possible intermediate results exist after the first synthesis step for the formula $\langle a \rangle [b]q$ on the model in Fig. 5a), but only one is shown for clarity.

Two important general aspects of synthesis need to be taken into account before we can proceed with a formal definition of synthesis: unfolding and maximality, or maximal permissiveness. As stated before, we require products of synthesis to be maximal in the sense that the least amount of modification is applied in order to satisfy the given formula. Maximality is reflected in two ways in the synthesis process:

1. Synthesis for a formula $\langle e \rangle f$, for $f \in \mathcal{F}$, should only remove an $e$-step if $f$ can not be satisfied in the state reached after the $e$-step.
2. The set of synthesis products should contain a maximal solution, with respect to all simulants of the original model which satisfy the synthesized formula.

The first property is illustrated in Fig. 6a)-6c) where a non-maximal as well as a maximal solution are given for the formula $\langle a \rangle p$. Synthesis for the a formula $\langle c \rangle f$, as defined formally in the next section, therefore excludes results such as in 6c.

Regarding the second property, it should be noted that non-maximal solutions can not always be avoided. As shown in Fig. 7a)-7c), the set of synthesis results for the
Fig. 5. Synthesis for the operator $\land$ is realized via alternating application of synthesis for both conjuncts. Fig. a) shows the input-KTS and d) the final result after synthesis for $\left< a \right> [b] q \land [a] < b > p$, via intermediate steps b) and c). The result in d) remains neutral for the synthesis of both conjuncts and thereby marks the final step in the alternating synthesis procedure. Note that retained behavior, as added by synthesis for $\left< a \right>$ and $\left< \diamond \right>$, is indicated using dashed lines. Also note that two possible intermediate results exist, after the first synthesis step for the formula $\left< a \right> [b] q$, on the model in a), but only one is shown for clarity.

Fig. 6. Synthesis of the formula $[a] p$ is applied to the original model in a), resulting in the modification shown in b). Fig. c) shows a non-maximal and therefore invalid solution.

The formula $[c] p \lor [c] (p \land q)$, contains a non-maximal solution that can not be excluded due to the nature of the synthesis for disjunction, where each operand is considered separately. In this regard it is, again, important to interpret maximality with respect to all satisfying simulants. That is, if a satisfying simulant of the input-KTS exists, then this simulant is also related via simulation to one of the synthesis outcomes.
This forms the justification why deleting a disallowed behavior does not contradict the maximality requirement.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig7}
\caption{Synthesis of the formula \([c] p \lor [c] (p \land q)\) on model \(a\) results in a maximal solution shown in \(b\), as well as non-maximal solution \(c\), due to the nature of the synthesis of disjunction, where each operand is considered separately and results are combined into a set of synthesis products.}
\end{figure}

The second general aspect is related to unfolding, a topic with significant implications that has been mentioned before, and justifies the usage of a non-standard behavioral model. Unfolding is induced by the fact that states may play multiple roles at various stages of synthesis, if loops are involved, or when states have multiple ingoing transitions. This is illustrated in Fig. 8a, where an obvious solution might seem to remove the self-loop at the initial state in order to satisfy the formula \([c] p\). However, this would violate the maximality requirement because not all possible behavior is retained. We therefore unfold the model for as far as the depth of \([c] p\), which is equal to 1, in this case. The result of this unfolding step is shown in Fig. 8b. A transition can now be safely removed from the model shown in 8b), while preserving maximality, resulting in the synthesis result shown in 8c). In Section 5, we will show how an alternative, albeit bisimilar, unfolded partial tree can be obtained for any given depth.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig8}
\caption{Before synthesis is applied, the KTS is unfolded up to the depth of the synthesized formula. This allows transitions to be removed safely, while retaining maximality. The model shown in \(a\) is unfolded up to depth \([c] p = 1\) in \(b\), after which synthesis for \([c] p\) is applied to obtain the model shown in \(c\).}
\end{figure}

5. TREE REPRESENTATION

As discussed previously, synthesis only manipulates transitions in the KTS within the finite reach of the synthesized formula. Additionally, unfolding is required in order to obtain a maximal solution. We therefore introduce a partial tree representation of a KTS which allows a clear and coherent definition of synthesis, and contains an embedded unfolding. One might wonder why a new formalism is required, while it would also seem possible to simply rely upon the KTS formalism and unfold up to a given depth. However, as examples in the previous section have shown, a single state may play multiple roles at various stages of synthesis. Therefore, it is not possible to specify synthesis directly as defined in this paper via \textit{in situ} changes to the transition.
The elements \( \{ \} \) allow for a clear and coherent operational definition of synthesis, since points of transition removal and recursive application of synthesis are directly clear from this structure. Formally, we represent the universe of these structures as \( K \rightarrow \), which may be interpreted as a dependent type, with regard to the transition relation \( \rightarrow \), via the construction in Definition 5.1.

**Definition 5.1.** Given a state-space \( X \subseteq X \), state \( x \in X \), and transition relation \( \rightarrow \subseteq X \times E \times X \), the dependent type \( K \rightarrow \) is defined in the following way:

\[
K \rightarrow := \langle x \rangle \rightarrow | \langle x, T \rangle \rightarrow \quad \text{for } T \subseteq E \times K \rightarrow
\]

To bring more clarity to Definition 5.1, we consider the two parts of this definition here separately.

1. The construct \( \langle x \rangle \rightarrow \) represents an end-node of the tree. This means that from this point on, behavior is modeled via the transition relation \( \rightarrow \).
2. The elements \( \langle x, T \rangle \rightarrow \) for \( T \subseteq E \times K \rightarrow \) represent the actual nodes of the tree. These consist of a state, combined with a continuation of behavior via underlying subtrees, and their corresponding events.

Steps between elements in \( K \rightarrow \) are created in the following way. Suppose \( k, k' \in K \rightarrow \), and \( e \in E \), then a step \( k \underset{e} {\rightarrow} k' \), can be obtained if one of the following two conditions is satisfied:

1. If \( k = \langle x \rangle \rightarrow \) and \( k' = \langle x' \rangle \rightarrow \) and \( x \underset{e} {\rightarrow} x' \), then \( k \underset{e} {\rightarrow} k' \)
2. If \( k = \langle x, T \rangle \rightarrow \) and \( (e, k') \in T \) then \( k \underset{e} {\rightarrow} k' \)

We realize that the construction for \( K \rightarrow \) is a non-standard and non-straightforward formalism. However, we deem this to be justified by the obtained clarity in the definition for synthesis, and the ability to capture embedded unfolding via this structure. In Fig. 9, an example is shown to illustrate a KTS as well as its unfolded partial tree representation. In Fig. 9a), a KTS is shown which is subsequently unfolded to depth two, and represented via the tree shown in Fig. 9b). Unfolding up to depth two would be applied if, for instance, the formula \(<a>[b]p\), were to be synthesized. The resulting formal definitions for this construction in \( K \rightarrow \) are shown in Fig. 9c).

Since steps are now defined with respect to \( K \rightarrow \), it is possible to transfer the standard behavioral relations of simulation and bisimulation to this structure. These are shown in Definitions 5.2 and 5.3 respectively.

**Definition 5.2.** For \( k' \in K \rightarrow \), \( k \in K \rightarrow \), we define simulation as a relation \( R \subseteq K \rightarrow \times K \rightarrow \), such that \( (k', k) \in R \) and the following holds for all \( (m', m) \in R \):

1. If \( x' \) and \( x \) are the respective states in \( m' \) and \( m \), then we require \( L(x') = L(x) \)
2. For all \( m' \overset{e} {\rightarrow} n' \), there exists a \( n \in K \rightarrow \) such that \( m \overset{e} {\rightarrow} n \) and \( (n', n) \in R \)

Again, we will use \( k' \overset{R} {\rightarrow} k \) to indicate that \( k' \overset{R} {\rightarrow} k \) as witnessed by \( R \). Note that the different transition relations \( \rightarrow \rightarrow \) and \( \rightarrow \rightarrow \) are used to highlight the fact that the
respective $\mathcal{K}$-structures are defined with respect to a different underlying transition relation.

**Definition 5.3.** If $k' \preceq_R k$ and $k'' \preceq_{R-1} k$ then $k'$ and $k$ are related via bisimulation (notation: $k' \leftrightarrow_R k$, or $k' \preceq_R k$, to indicate the witness relation $R$).

Validity for formulas in $\mathcal{F}$ can now be expressed with regard to the structure $\mathcal{K}_\rightarrow$, via Definition 5.4.

**Definition 5.4.** The predicate $\vdash$ over $\mathcal{K}_\rightarrow \times \mathcal{F}$ is defined for $k', k \in \mathcal{K}_\rightarrow$, $f, f_1, f_2, f \in \mathcal{F}$, $e \in \mathcal{E}$, $x \in \mathcal{X}$, $p \in \mathcal{P}$ by the following deduction rules:

\[ k \vdash \text{true} \quad \frac{p \in L(x)}{\langle x \rangle \rightarrow \vdash p} \quad \frac{p \notin L(x)}{(x, T) \rightarrow \vdash \lnot p} \quad \frac{p \notin L(x)}{(x, T) \rightarrow \vdash p} \]

\[ k \vdash f_1 \quad k \vdash f_2 \quad \frac{k \vdash f_1 \land f_2}{k \vdash f_1 \lor f_2} \quad \frac{k \vdash f_1 \lor f_2}{k \vdash f_1 \lor f_2} \quad \frac{k \vdash f}{k \vdash \langle c \rangle f} \]

It is clear that each KTS $g \in G$ can be represented as an element of $\mathcal{K}_\rightarrow$, since the structure $\langle x \rangle$, is isomorphic to $g$ if $g$ has $x$ as its initial state and $\rightarrow$ as its transition relation. To convert this structure into an unfolded $\mathcal{K}_\rightarrow$ representation, we formally define a function $\text{unfold} : \mathcal{K}_\rightarrow \times \mathbb{N} \mapsto \mathcal{K}_\rightarrow$, in Definition 5.5. In addition, we define a test for unfoldedness via the predicate $\text{unf} \subseteq \mathcal{K}_\rightarrow \times \mathbb{N}$ in Definition 5.6. Lemma 5.7 then shows that the unfolded structure is indeed bisimilar to the unmodified structure.

**Definition 5.5.** Let $\langle x \rangle \rightarrow \in \mathcal{K}_\rightarrow$, then for each $n \in \mathbb{N}$, a $k \in \mathcal{K}_\rightarrow$ can be constructed which is unfolded to depth $n$, using the following function $\text{unfold} : \mathcal{K}_\rightarrow \times \mathbb{N} \mapsto \mathcal{K}_\rightarrow$:

\[ \text{unfold}(k, 0) = k \]
\[ \text{unfold}(\langle x \rangle \rightarrow, n + 1) = \langle x, \{(e, \text{unfold}(\langle x' \rangle \rightarrow, n)) \mid x' \rightarrow x \} \rangle \rightarrow \]
\[ \text{unfold}(\langle x, T \rangle \rightarrow, n + 1) = \langle x, \{(e, \text{unfold}(k', n)) \mid e, k' \in T \} \rangle \rightarrow \]

We state here, without proof that every result of the $\text{unfold}$ function is indeed unfolded up to the given depth, as expressed by the predicate given in Definition 5.6.

**Definition 5.6.** The predicate $\text{unf} \subseteq \mathcal{K}_\rightarrow \times \mathbb{N}$ is defined for $x \in \mathcal{X}$, $T \subseteq \mathcal{E} \times \mathcal{K}_\rightarrow$, and $n \in \mathbb{N}$ by the following definition:

\[ \text{unf}(\langle x \rangle \rightarrow, n) \iff n = 0 \]
\[ \text{unf}(\langle x, T \rangle \rightarrow, 0) \iff \text{true} \]
\[ \text{unf}(\langle x, T \rangle \rightarrow, n + 1) \iff \forall (e, k) \in T. \text{unf}(k, n) \]
The following Lemma 5.7 shows that the result of the unfold function is indeed bisimilar to its original input.

**Lemma 5.7.** For \( k \in \mathcal{K}_\rightarrow \) and \( n \in \mathbb{N} \), it holds that \( k \leftrightarrow \text{unfold} \,(k, n) \).

**Proof.** We prove this property by induction towards \( n \). If \( n = 0 \) then \( \text{unfold} \,(k, 0) = k \), by Definition 5.5, and clearly \( k \leftrightarrow k \), by reflexivity of bisimulation. For the inductive case, assume that \( \text{unfold} \,(k', n) \leftrightarrow k' \), for all \( k' \in \mathcal{K}_\rightarrow \). We now need to show that \( \text{unfold} \,(k, n + 1) \leftrightarrow k \). A distinction is made between the two forms of \( k \), as given in Definition 5.1.

If \( k \equiv \langle x \rangle \rightarrow \), then for each step \( x \xrightarrow{c_i} x_i \), it holds that \( \text{unfold} \,(\langle x_i \rangle \rightarrow, n) \leftrightarrow R_i \langle x_i \rangle \rightarrow \), for some \( R_i \), by induction. Bisimulation is then shown by choosing \( R' = \bigcup_i R_i \cup \{ (\text{unfold} \,(k, n + 1), k) \} \), in order to prove that \( \text{unfold} \,(k, n + 1) \leftrightarrow R' \, k \).

If \( k \equiv \langle x, T \rangle \rightarrow \), then for each \((e, k_i) \in T \), it holds that \( \text{unfold} \,(k_i, n) \leftrightarrow R_i \, k_i \), for some \( R_i \), by induction. Again, we may choose \( R' = \bigcup_i R_i \cup \{ (\text{unfold} \,(k, n + 1), k) \} \), in order to show that \( \text{unfold} \,(k, n + 1) \leftrightarrow R' \, k \). \( \square \)

Since \( k \leftrightarrow \text{unfold} \,(k, n) \), we have \( k \equiv f \), if and only if \( \text{unfold} \,(k, f) \equiv f \), which is a standard property of bisimulation with respect to HML formulas [van Glabbeek 1993].

In the overview of the synthesis process as illustrated in Fig. 1, the unfolding step is the first step in the synthesis process. After synthesis is applied, each resulting partial tree representation \( k \in \mathcal{K}_\rightarrow \) is again converted into a KTS \( g \in \mathcal{G} \). This is indicated as the post-synthesis step \( \text{Tree2KTS} \) in Fig. 1. This function is stated formally in Definition 5.8.

The intuitive explanation behind Definition 5.8 is as follows. Due to the fact that a single state-element \( x \in X \) which occurs in a partial tree \( k \in \mathcal{K}_\rightarrow \) may play different roles at various stages of synthesis, as indicated by \( x \) occurring as state-element in multiple parts of \( k \), we can not directly convert \( k \) into a KTS having a transition relation defined over \( X \). Instead, we define a transition relation over the state-space \( X \times \mathbb{N} \), where the original transition relation is directly mapped to \( X \times \{0\} \). If the top of the partial tree \( k \) is unfolded to depth \( n \) and if \( x \) is the top-most state-element of \( k \), then we construct the new transition relation as \( (x, n) \xrightarrow{c} (x_i, n - 1) \), if each \( x_i \) is the state-element of a sub-tree of \( k \). We then continue recursively, as shown in Definition 5.8.

**Definition 5.8.** Let \( k \in \mathcal{K}_\rightarrow \) and let \( n \in \mathbb{N} \) be the greatest \( n \) such that \( \text{unf} \,(k, n) \). We construct a new KTS \( g \in \mathcal{G} \) having state space \( X \times \mathbb{N} \) if the state-elements in \( k \) are drawn from \( X \). The initial state of \( g \) is then defined as \( (x, n) \) and its transition relation \( \rightarrow' \subseteq (X \times \mathbb{N}) \times \mathcal{E} \times (X \times \mathbb{N}) \) is defined as:

\[
\rightarrow' = \{ (x, 0) \xrightarrow{c} (x', 0) \mid x \xrightarrow{c} x' \} \cup \text{Tree2KTS} \,(k, n)
\]

where the function \( \text{Tree2KTS} \) is defined in the following way:

\[
\text{Tree2KTS} \,(k, 0) = \emptyset
\]

\[
\text{Tree2KTS} \,(k, n) = \{ (x, n) \xrightarrow{c} (x', n - 1) \mid k \xrightarrow{c} k' \} \cup \bigcup_{k \xrightarrow{c} k'} \text{Tree2KTS} \,(k', n - 1)
\]

In the last clause of this definition, \( x \) and \( x' \) are the respective initial states of \( k \) and \( k' \). Note that in this particular situation we have to redefine the labeling function \( L \) into a new labeling function \( L' \), such that \( L'(x, n) = L(x) \), for all \( x \in X \) and \( n \in \mathbb{N} \).

6. **Operational Definition of Synthesis**

We define the synthesis function \( C : \mathcal{K}_\rightarrow \times \mathcal{F} \rightarrow 2^{\mathcal{K}_\rightarrow} \) inductively as a relation via deduction rules. Note that the function \( C \) is defined in such a way that it expects the first
argument $k \in \mathcal{K}$ to be unfolded to at least depth $depth(f)$. Since the synthesis function $C$ does not modify the underlying transition relation, we may omit this relation in the following definition of $C$ and write $\langle x \rangle$ and $\langle x, T \rangle$ instead of $\langle x \rangle \rightarrow$ and $\langle x, T \rangle \rightarrow$.

**Definition 6.1.** For $k \in \mathcal{K}$ and $f \in \mathcal{F}$, we define the set of synthesis results $C(k, f)$ by the following deduction rules for $x \in \mathcal{X}$, $p \in \mathcal{P}$, $T \subseteq \mathcal{E} \times \mathcal{K}$, $g \in \mathcal{F}$, $k', m \in \mathcal{K}$, and $e, e' \in \mathcal{E}$:

\[
\begin{align*}
\frac{k \in C(k, true)}{p \in L(x)} & \quad \frac{\langle x \rangle \in C(\langle x \rangle, p)}{\langle x \rangle} \quad \frac{\langle x, T \rangle \in C(\langle x, T \rangle, p)}{\langle x \rangle} \quad \frac{\langle x \rangle \in C(\langle x \rangle, \neg p)}{p \not\in L(x)} \\
\frac{p \not\in L(x)}{\langle x, T \rangle \in C(\langle x, T \rangle, \neg p)} & \quad \frac{m \in C(k, f)}{m \in C(k, f \lor g)} \quad \frac{m \in C(k, g)}{m \in C(k, f \land g)} \quad \frac{k' \in C(k, f)}{m \in C(k', g \land f)} \quad \frac{m \in C(k, f \land g)}{m \in C(k', g \land f)}
\end{align*}
\]

\[
\begin{align*}
\frac{\langle x, T' \rangle \in C(\langle x, T \rangle, \neg e f)}{\langle x \rangle \in C(\langle x \rangle, \neg e f)} \quad \frac{\langle x \rangle \in C(\langle x \rangle, \neg e f)}{\langle x \rangle, \langle e, \emptyset \rangle \cup T' \in C(\langle x \rangle, \emptyset \cup T \cup \emptyset, \neg e f)} \\
\frac{\langle x, T' \rangle \in C(\langle x, T \rangle, \neg e f)}{\langle x, \{e \} \cup T \rangle \in C(\langle x, \{e \} \cup T \rangle, \neg e f)} \quad \frac{\langle x, T' \rangle \in C(\langle x, T \rangle, \neg e f)}{\langle x, \{e, m \} \cup T \rangle \in C(\langle x, \{e, m \} \cup T \rangle, \neg e f)} \quad \frac{\langle x, \{e, m \} \cup T \rangle \in C(\langle x, \{e, m \} \cup T \rangle, \neg e f)}{m \in C(k, f)}
\end{align*}
\]

We briefly discuss the deduction rules for $C$ in Definition 6.1. Note that the set union operator $\cup$ is interpreted as a disjoint union within these rules. Synthesis is neutral for $true$ as this formula is always satisfied (rule 1). Synthesis for a basic property $p$ results in the same structure if $p$ is valid in the initial state (i.e. $p \in L(x)$), as shown in rules 2 and 3. Synthesis for the negated basic property $\neg p$ results in the same structure if $p \not\in L(x)$, as can be observed in rules 4 and 5. The rules 6 and 7 define a fixpoint construction for the synthesis of a conjunctive formula. The condition for termination as described in rule 6 applies when synthesis for both conjuncts results in the same structure. Otherwise, both conjuncts are synthesized alternately, as shown in rule 7. The rules 8 and 9 for disjunction are relatively straightforward: an element of the synthesized set is a result of the synthesis for one of the disjuncts. The operator $\{e\}f$ is covered in rules 10-13, which are defined inductively on the set $T$. Rule 10 describes the basic case for this induction where no transitions to underlying structures are present and no modification is required. Rule 11 details how an $e'$-transition, different from an $e$-transition is left in place for the operator $\{e\}f$, as this transition does not influence the validity of an $\{e\}f$ formula. Rule 12 removes an $e$-transition for the operator $\{e\}f$, if no synthesis candidate can be found for the corresponding transition. The last rule 13 for $\{e\}f$ ensures that the original sub-tree after an $e$-step is replaced by an appropriate synthesis product. Finally, we define a single rule 14 for the synthesis of the formula $\langle e \rangle f$. A single witness for a proper $e$-transition is added to the original structure, which is left unmodified for the rest of it. Note that we do not need to consider synthesis for $\langle x \rightarrow \rangle \in \mathcal{K}$ for the operators $\{e\}f$ and $\langle e \rangle f$ because that would invalidate the implicit unfoldedness condition.
7. TERMINATION AND COMPLEXITY

Three effective sequential steps are identified in the overview of the synthesis process, as seen in Fig. 1. These are the unfolding step, the actual synthesis itself, and the Tree2KTS step. We will illustrate the synthesis step here algorithmically and analyze its complexity, as well as the complexity of its preceding unfolding and succeeding Tree2KTS step. The synthesis algorithm is shown in Fig. 10, under the assumption that its input parameter \(k \in K_\rightarrow\) is adequately unfolded up to depth (\(f\)), for \(f \in \mathcal{F}\). The parameter \(H\) of the procedure \textit{synthesis} in Fig. 10 is used to guarantee termination of synthesis for conjunction.

<table>
<thead>
<tr>
<th>Rules (Def. 6.1)</th>
<th><strong>Algorithm 1:</strong> Synthesis Algorithm</th>
</tr>
</thead>
</table>
| \begin{align*}
\text{procedure} & \text{ synthesis} (k \in K_\rightarrow, f \in \mathcal{F}, \text{set } H \text{ of } K_\rightarrow) \text{ returns set of } K_\rightarrow \\
\text{begin} & \\
\text{set } R \text{ of } K_\rightarrow := \emptyset \\
1 & \text{case } (f = \text{true}) \\
& \quad R := \{k\} \\
2-3 & \text{case } (f = p \text{ and } (k = \langle x \rangle_\rightarrow \text{ or } k = \langle x, T \rangle_\rightarrow) \text{ and } p \in L(x)) \\
& \quad R := \{k\} \\
4-5 & \text{case } (f = \neg p \text{ and } (k = \langle x \rangle_\rightarrow \text{ or } k = \langle x, T \rangle_\rightarrow) \text{ and } p \notin L(x)) \\
& \quad R := \{k\} \\
6-7 & \text{case } (f = f_1 \land f_2) \\
& \quad R := \text{synthesis} (k, f_1, \emptyset) \cap \text{synthesis} (k, f_2, \emptyset) \\
& \quad \text{for each } k' \in \text{synthesis} (k, f_1, \emptyset) \setminus H \\
& \quad \quad R := R \cup \text{synthesis} (k', f_2 \land f_1, H \cup R) \\
8-9 & \text{case } (f = f_1 \lor f_2) \\
& \quad R := \text{synthesis} (k, f_1, \emptyset) \cup \text{synthesis} (k, f_2, \emptyset) \\
10 & \text{case } (f = [e] f' \text{ and } k = \langle x, \emptyset \rangle_\rightarrow) \\
& \quad R := \{\langle x, \emptyset \rangle_\rightarrow\} \\
11 & \text{case } (f = [e] f' \text{ and } k = \langle x, \{e', k'\} \cup T \rangle_\rightarrow \text{ and } e \neq e') \\
& \quad \text{for each } \langle x, T' \rangle_\rightarrow \in \text{synthesis} (\langle x, T \rangle_\rightarrow, [e] f', \emptyset) \\
& \quad \quad R := R \cup \{\langle x, \{e', k'\} \cup T' \rangle_\rightarrow\} \\
12 & \text{case } (f = [e] f' \text{ and } k = \langle x, \{e, k'\} \cup T \rangle_\rightarrow \text{ and } \text{synthesis} (k', f', \emptyset) = \emptyset) \\
& \quad R := \text{synthesis} (\langle x, T \rangle_\rightarrow, [e] f', \emptyset) \\
13 & \text{case } (f = [e] f' \text{ and } k = \langle x, \{e, k'\} \cup T \rangle_\rightarrow) \\
& \quad \text{for each } \langle x, T' \rangle_\rightarrow \in \text{synthesis} (\langle x, T \rangle_\rightarrow, [e] f', \emptyset) \\
& \quad \quad \text{for each } m \in \text{synthesis} (k', f', \emptyset) \\
& \quad \quad \quad R := R \cup \{\langle x, T \cup \{e, m\} \rangle_\rightarrow\} \\
14 & \text{case } (f = <e> f' \text{ and } k = \langle x, T \rangle_\rightarrow) \\
& \quad \text{for each } (e, k') \in T \\
& \quad \quad \text{for each } m \in \text{synthesis} (k', f', \emptyset) \\
& \quad \quad \quad R := R \cup \{\langle x, T \cup \{e, m\} \rangle_\rightarrow\} \\
\text{return } R \\
\text{end}
\end{align*} |

**Fig. 10.** Algorithmic representation of the synthesis procedure. This algorithm is a direct translation of the synthesis rules given in Definition 6.1. Corresponding rule numbers in Definition 6.1 are shown in the left row. The parameter \(H\) of the procedure \textit{synthesis} is used to guarantee termination of synthesis for conjunction. The synthesis method is invoked as \textit{synthesis} \((k, f, \emptyset)\).
As Fig. 10 shows, the recursive structure of the synthesis algorithm follows the inductive structure of the HML formulas for all cases except for \( f \equiv f_1 \land f_2 \). Due to the fact that synthesis for conjunction might involve multiple invocations of synthesis for the same conjunct, we consider this part of the algorithm the dominating factor in the time-complexity of the algorithm. This case also complicates the termination proof significantly. We show that the synthesis algorithm is terminating in Theorem 7.1, followed by the complexity result in Theorem 7.2.

**Theorem 7.1.** For each \( k \in K_\rightarrow \) and \( f \in F \), the synthesis procedure in Fig. 10 terminates in a finite number of steps.

**Proof.** We first show that only finitely possible synthesis results are obtained using the procedure \( \text{synthesis} \), by induction towards the structure of \( f \). As considered earlier, we assume the partial tree \( k \in K_\rightarrow \) to be finitely branching. Since the synthesis algorithm in Fig. 10 can be considered a direct implementation of the synthesis rules in Definition 6.1, it might be helpful to the reader to consider these rules as well. For the cases \( f \equiv \text{true}, f \equiv \text{false}, f \equiv p \), and \( f \equiv \neg p \), for \( p \in P \), it is clear from Fig. 10 that either zero or one results are returned. For the cases \( f \equiv f_1 \land f_2 \) and \( f \equiv f_1 \lor f_2 \), a finite number of synthesis results originates from a recursive call to the function \( \text{synthesis} \) for \( f_1 \) or \( f_2 \), so therefore a finite amount of results is obtained in these cases as well. If \( f \equiv [e] f' \), then the recursive finite synthesis results are combined over a finite number of branches, resulting again in a finite number of results. If \( f \equiv [e] f' \), then, again, a finite number of results originate from the recursive call. Also, due to retaining existing behavior, a sub-tree may be added. However, note that sub-trees are not duplicated, since addition of \( \{ (e, k) \} \) to the set \( T \) results in \( T \) if \( (e, k) \in T \). Therefore, synthesis for \( [e] f' \) also results in a finite number of synthesis outcomes.

We now show that an invocation of \( \text{synthesis} (k, f, \emptyset) \) terminates in a finite number of steps, via induction towards the structure of \( f \). The cases where \( f \equiv \text{true}, f \equiv \text{false}, f \equiv p \), and \( f \equiv \neg p \), for \( p \in P \), do not result in any recursive calls, so the function \( \text{synthesis} \) will terminate directly for these cases. For the cases \( f \equiv f_1 \lor f_2 \) or \( f = [e] f' \), the procedure only invokes a finite number of terminating recursive calls, and termination is therefore obtained via induction. Termination for the remaining case for \( f \equiv f_1 \land f_2 \) is derived as follows. Via induction we derive termination for the recursive calls to \( \text{synthesis} (k, f_1, \emptyset) \), \( \text{synthesis} (k, f_2, \emptyset) \) and \( \text{synthesis} (m, f_1, \emptyset) \). Since, for each recursive invocation of \( \text{synthesis} (m, f_2 \land f_1) \), the set \( H \) is extended with the set of synthesis results for \( f_1 \) and \( f_2 \), the recursive call to \( \text{synthesis} (f_2 \land f_1) \) will, at some recursion depth, not be invoked, due to finiteness of the number of possible synthesis results.

The number of affected transitions during synthesis is limited by \( \text{depth}(f) \), and may therefore be expressed as \( n \cdot \text{depth}(f) \), where \( n \) is linear in the number of transitions. Based upon this observation, we may express the upper bound for the number of solutions as \( 2^n \cdot \text{depth}(f) \). This upper bound is also derived as the upper bound of the computational complexity of the algorithm in Theorem 7.2. Note that this represents a worst-case scenario. For instance, a formula without conjunction may be synthesized in \( \Theta(n \cdot \text{depth}(f)) \).

**Theorem 7.2.** For \( k \in K_\rightarrow \) and \( f \in F \), the upper bound for the computational complexity of the procedure \( \text{synthesis} \) in Fig. 10 is determined as \( \Theta(2^n \cdot \text{depth}(f)) \), where \( n \) is linear in the number of transitions.

**Proof.** We apply induction towards the structure of \( f \). For the cases \( f \equiv \text{true}, f \equiv \text{false}, f \equiv p \), and \( f \equiv \neg p \), for \( p \in P \), the computational complexity can be stated as \( \Theta(1) < \Theta(2^n \cdot \text{depth}(f)) \). If \( f \equiv f_1 \lor f_2 \), then synthesis invokes two recursive calls, as
observed in Fig. 10. For this case, we therefore obtain $2 \cdot \Theta (2^n \cdot \text{depth}(f)) \approx \Theta (2^n \cdot \text{depth}(f))$. If $f \equiv [e] f'$ or $f \equiv <\!\!\!\!\!\!\!e\!\!\!\!\!\!\!\!>, then \(\frac{n}{m}\) for \(m \geq 1\) recursive calls of the \textit{synthesis} procedure are invoked. For these cases, we may therefore determine the upper bound for the computational complexity as $(n/m) \cdot \Theta (2^n \cdot \text{depth}(f)) \approx \Theta (2^n \cdot \text{depth}(f))$. The final case to consider is when $f \equiv f_1 \land f_2$. As the recursion depth for the synthesis invocation for $f_2 \land f_1$ is bounded by the number of possible synthesis results, we may express the computational complexity for this case as $2 \cdot \Theta (2^n \cdot \text{depth}(f)) + 2^n \cdot \text{depth}(f) \cdot \Theta (2^n \cdot \text{depth}(f)) \approx \Theta (2^n \cdot \text{depth}(f))$. \(\Box\)

Since the unfolding step only affects the KTS up to the depth of the synthesized formula, an actual realization of the function \textit{unfold} in Definition 5.5 can be implemented in time $\Theta (\text{depth}(f))$, for $f \in \mathcal{F}$. The \textit{Tree2KTS} function, as given in Definition 5.8, only involves a single operation for every constructed transition. Since the size of the transition structure is taken as a linear factor, this term can effectively be ignored in determining the time-complexity of the algorithm. We thus conclude that the computational complexity of the entire synthesis process is dominated by, and therefore equivalent to, the complexity of the synthesis method itself.

A final remark regarding the selection of the maximal candidates cannot be left unmentioned. As synthesis results in a set of satisfying structures, it would seem a natural part of such an algorithm to select the maximal candidates, among the synthesized results. Also, one might wonder why the selection of the maximal candidate is not considered in the analysis of the computational complexity of the algorithm. Multiple solutions arise due to a number of reasons. As Fig. 4 clearly shows, multiple maximal results may be a result of the synthesis of a disjunctive formula. As indicated in Fig. 4, these results are essentially incomparable, and therefore no selection is to be made. On the other hand, synthesis for a disjunction might result in multiple solutions of which a single maximal solution may be preferred, as shown in Fig. 4. However, in the general case, it is not clear how this may be efficiently determined, compared to a direct computation of the maximal candidate, based on the simulation preorder. Note that multiple results due to synthesis for a formula $<\!\!\!\!\!\!\!\!e\!\!\!\!\!\!\!\!>$ do not pose a problem in this respect, as all original behavior is copied when synthesizing for $<\!\!\!\!\!\!\!\!e\!\!\!\!\!\!\!\!>$, thus not invalidating maximality.

8. VALIDITY OF SYNTHESIS
We prove two theorems regarding the validity of the definition of synthesis. In Theorem 8.1 we show that every synthesis result satisfies the synthesized formula. Theorem 8.2 details how every synthesis result is related via simulation to the original structure.

**Theorem 8.1.** For $f \in \mathcal{F}$ and $k, m \in \mathcal{K}$, it holds that $m \in C(k, f)$ implies $m \models f$.

**Proof.** By induction towards the construction of $m \in C(k, f)$, via the deduction rules in Definition 6.1. If $m \in C(k, \text{true})$, then obviously $m \models \text{true}$. If $m \in C(k, p)$, for some $p \in \mathcal{P}$, then $m$ and $k$ have the same initial state, say $x$. Since $p \in L(x)$ this results in $m \models p$. If $m \in C(k, \neg p)$, then again we observe that $m$ and $k$ have the same initial state $x$, such that $p \notin L(x)$, and therefore $m \models \neg p$. For rules 6-7 we have the following analysis: If $m \in C(k, f_1)$ and $m \in C(k, f_2)$, then $m \models f_1 \land f_2$ by induction. For $k' \in C(k, f_1)$ and $m \in C(k', f_2 \land f_1)$, by induction and commutativity of the validity of $\land$ we have that $m \models f_1 \land f_2$. For rules 8-9 then again we have two cases. If $m \in C(k, f_1)$ then $m \models f_1 \lor f_2$, and if $m \in C(k, f_2)$ then $m \models f_1 \lor f_2$, both by induction. We have four cases corresponding to the rules 10-13. It trivially holds that \((x, \emptyset) \models [e] f'\). By induction, we have \((x, \{e', k\} \cup T') \models [e] f'\) for each $e \neq e'$ if \((x, T') \models [e] f'\), by induction. Rule 12 does not alter the structure of $m \in C(k, [e] f)$ and therefore preserves validity. If \((x, T') \models [e] f\), by induction, and $m \models f$ for $m \in \mathcal{K}$.
If there exists an \( m \in C(k, f') \) and therefore \( m \models f' \), then by induction it holds that \( \langle x, \{(e, m), (e, k)\} \cup T \rangle \models \langle e \mapsto f' \rangle. \]

**Theorem 8.2.** For \( f \in \mathcal{F} \) and \( k, m \in \mathcal{K} \), it holds that \( m \in C(k, f) \) implies \( m \leq k \).

**Proof.** We use the same proof strategy as in Theorem 8.1: induction to the construction of \( m \in C(k, f) \). Note that we only give a proof sketch here because no actual simulation relation is constructed. The cases for rules 1-5 and rule 10 are solved by reflexivity of simulation while rules 6-9 are covered by induction and transitivity of simulation. The four remaining cases consider the rules 11-14. For rule 11, we may assume \( \langle x, T' \rangle \leq \langle x, T \rangle \) as our induction hypothesis. This directly leads to \( \langle x, \{(e', k)\} \cup T' \rangle \leq \langle x, \{(e', k)\} \cup T \rangle \). For rule 12 we have \( \langle x, T' \rangle \leq \langle x, T \rangle \) by induction and therefore \( \langle x, T' \rangle \leq \langle x, \{(e, k)\} \cup T \rangle \). For the case corresponding to rule 13 we have as our induction hypotheses: \( \langle x, T' \rangle \leq \langle x, T \rangle \) and additionally \( m \leq k \) for \( m \in C(k, f) \). This leads to \( \langle x, \{(e, m)\} \cup T' \rangle \leq \langle x, \{(e, k)\} \cup T \rangle \). We conclude this proof by an analysis of the last rule 14 for which we have an \( m \in C(k, f) \) and therefore \( m \leq k \) as per induction hypothesis. Clearly this leads to \( \langle x, \{(e, m), (e, k)\} \cup T \rangle \leq \langle x, \{(e, k)\} \cup T \rangle \).

For Theorems 8.1 and 8.2, computer-verified proofs [van Hulst 2013] have been constructed using the Coq proof assistant [Barras et al. 1997]. A number of remarks should be noted with regard to these formal proofs. First and foremost, it is not possible to encode the dependent type \( \mathcal{K} \), as given in Definition 5.1, directly in Coq. Due to the strict positivity requirement of the inductive types in Coq, it is not possible to define the collection of underlying tree-elements as a set. Instead, a list is used which has some implications for the definition of equality on \( \mathcal{K} \), due to the occurrence of multiple equal elements. Strict positiveness for inductive types also implies that rule 12 can not be encoded precisely in Coq, since no test on emptiness of the type can be used during its definition. The implication is that a broader set of synthesis results is constructed. Still, each result satisfied the aforementioned two Theorems 8.1 and 8.2, and every synthesis result as constructed by Definition 6.1 is still present. Unfortunately, these peculiarities make it impossible to encode the full maximality proof, as shown in Section 9, in the Coq proof assistant.

**9. Maximality**

As indicated before, it is desirable for products of synthesis to be modified to the least extent possible in order to achieve a maximal solution. This is especially required if further analysis is to be applied to the model, for instance if liveness is investigated, or if some kind of optimization procedure is applied at a post-synthesis stage. Maximality within the context of supervisory control is also referred to as maximal permissiveness [Cassandras and Lafortune 1999]. The maximality proof is shown in Theorem 9.2.

**Lemma 9.1.** For each \( f \in \mathcal{F}, n \in \mathbb{N} \) and \( k, m \in \mathcal{K} \), such that \( m \in C(k, f) \) and \( \text{unf}(k, n) \) it holds that \( \text{unf}(m, n) \).

**Proof.** We apply induction towards the construction of \( m \in C(k, f) \). The four non-straightforward cases are the rules 11-14. The first case is resolved under the induction hypothesis \( \text{unf}(\langle x, T \rangle \to n) \Rightarrow \text{unf}(\langle x, T' \rangle \to n) \). Clearly the premise \( \text{unf}(\langle x, \{(e', k)\} \cup T' \rangle \to n) \) leads to \( \text{unf}(\langle x, \{(e', k)\} \cup T \rangle \to n) \). Rule 12 does not alter \( m \in C(k, \{e \mapsto f\}) \) and therefore preserves unfoldedness as shown by induction. For rule 13 we have two induction hypotheses: \( \text{unf}(m, n) \) and \( \text{unf}(\langle x, T \rangle \to n) \Rightarrow \text{unf}(\langle x, T' \rangle \to n) \). Based on the premise \( \text{unf}(\langle x, \{(e, k)\} \cup T \rangle \to n) \) we may immediately draw the con-
clusion that $\text{unf}(x, \{(e, m)\} \cup T'_{\rightarrow}, n)$. For the last case concerning rule 14 we have $\text{unf}(x, T_{\rightarrow}, n), \text{unf}(m, n)$ and therefore $\text{unf}(x, \{(e, m)\} \cup T'_{\rightarrow}, n)$ by induction. □

The maximality result is shown in Theorem 9.2. If $k'$ is a simulant of $k$ such that $k' \models f$ and $k$ is unfolded up to the depth of a formula $f$, then synthesis produces at least one result $m$ such that $k' \preceq m$. Note that this result indicates Pareto-optimality [Cassandras and Lafortune 1999].

**Theorem 9.2.** For each $f \in F$, $k' \in K_\omega$ and $k \in K_\omega$ with $k' \models f$, $k' \preceq k$ and $\text{unf}(k, \text{depth}(f))$, there exists an $m \in C(k, f)$ such that $k' \preceq m$.

**Proof.** We apply induction towards the structure of $f$. For all cases, the induction premise for unfoldedness after synthesis is satisfied by Lemma 9.1. If $f \equiv \text{true}$ then $k \in C(k, \text{true})$ by rule 1 in Definition 6.1 and clearly $k' \preceq k$. If $f \equiv \text{false}$ then this clearly contradicts the assumption that $k' \models f$. If $f \equiv p$ or $f \equiv \neg p$, for $p \in P$, then $k \models f$, since we assumed strict equality on labels in Definition 5.2. Application of the corresponding rule 2-5 from Definition 6.1 results in $k \in C(k, f)$, while $k' \preceq k$ was already assumed.

The case for $f \equiv f_1 \land f_2$ is not straightforward. Observe that we have the following induction hypotheses:

**IHF1:** For all $k' \preceq k$ such that $k' \models f_1$ and $\text{unf}(k, \text{depth}(f_1))$ there exists an $m \in C(k, f_1)$ such that $k' \preceq m$.

**IHF2:** For all $k' \preceq k$ such that $k' \models f_2$ and $\text{unf}(k, \text{depth}(f_2))$ there exists an $m \in C(k, f_2)$ such that $k' \preceq m$.

Using these induction hypotheses, we may create a sequence of alternating applications of synthesis for $f_1$ and $f_2$ of arbitrary length:

$$k_1 \in C(k, f_1), k_2 \in C(k_1, f_2), k_3 \in C(k_2, f_1), \ldots, k_n \in C(k_{n-1}, f_2)$$

This sequence is obtained in the following way. As $k' \models f_1$, we apply induction hypothesis IHF1 to obtain $k_1 \in C(k, f_1)$ and $k' \preceq k_1$. Also, we have $k' \models f_2$ which allows the application of IHF2 on $k_1$, resulting in $k_2 \in C(k_1, f_2)$ such that $k' \preceq k_2$. It is clear that this sequence of applications may be applied for an arbitrary number of times.

Assume that each $k_n \in C(k_{n-1}, f_i)$ for $i \in \{1, 2\}$ can be obtained using a finite derivation tree $T_n$. From Theorem 8.1, it follows that $k_n \models f_i$, so clearly the formulas $f_1$ and $f_2$, when considered separately, are not contradictory in themselves, since a synthesis result can be readily obtained for each of these conjuncts.

If we find an $n \geq 1$ such that $k_n \in C(k_{n-1}, f_1)$ and $k_n \in C(k_{n-1}, f_2)$ then $k_n \in C(k_{n-1}, f_1 \land f_2)$ can be obtained by $n - 1$ applications of rule 6 from Definition 6.1, followed by a single application of rule 5.

Assume the operator $<e>$ is not contained in both $f_1$ and $f_2$, then each derivation tree $T_n$ can be constructed using the rules 1-13 from Definition 6.1. Careful study of these rules shows that each rule either does not modify the model, or results in a synthesized product which has a strictly lower number of transitions. Therefore, in the restricted situation where only rules 1-13 apply, it is always possible to obtain $n \in \mathbb{N}$ such that $k_n \in C(k_{n-1}, f_1 \land f_2)$, because otherwise the number of transitions would decrease below zero, which is clearly impossible.

Application of rule 14 complicates this situation, since this rule introduces additional transitions via copying of original behavior when synthesizing a formula $<e>f$. However, if $k \models <e>f$ then $k \in C(k, <e>f)$, which seems to justify the conclusion that no more than two applications of rule 14 are required in order to obtain a stable point. Nevertheless, there is still the possibility that following the application of rule 14, rule 12 is applied to remove the just created witness for the formula $<e>f$ again. An example has been considered earlier in Fig. 5.
However, the key observation in Fig. 5 is the possibility to create a witness for a $\langle e \rangle f$ formula at multiple points, which can only be finitely many, in the general case. In detail, suppose that synthesis for a formula $\langle e \rangle f$ results in $\langle x, \{(e, m), (e, k) \cup T\} \rangle$, via the application of rule 14. Also, suppose that this $(e, m)$ part of the model is subsequently removed by application of rule 12. If, by the application of rule 14, $(e, m)$ is constructed again in a later synthesis step, then rule 12 was applied to synthesize a formula $\langle e \rangle f'$ such that $C(m, f') = 0$. This clearly indicates that the formulas $f_1 \land f_2$ are contradictory, which contradicts the $k' \models f_1 \land f_2$ assumption.

The next case to consider is when $f \equiv f_1 \lor f_2$. If $k' \models f_1$ then, by induction, there exists an $m \in C(k, f_1)$ such that $k' \leq m$. This results in $m \in C(k, f_1 \lor f_2)$ by application of rule 7. The case for $k' \models f_2$ is exactly symmetrical.

The next case for $f \equiv \langle e \rangle f'$ again requires some careful analysis. Observe that we have the following induction hypothesis:

IHf1: For all $k' \preceq k$ such that $k' \models f'$ and $\text{unf}(k, \text{depth}(f'))$ there exists an $m \in C(k, f')$ such that $k' \preceq m$.

It is clear that $k = \langle x, T \rangle$, since $k \not= \langle x \rangle$, because $\text{unf}(k, 1 + \text{depth}(f'))$. For each $k' \xrightarrow{e} k''$, we have $k'' \models f'$ and a corresponding $n \in \mathcal{K}_\rightarrow$ such that $(e, n) \in T$ and $k'' \preceq n$. By induction, we then have $m' \in C(n, f')$. Repeated application of rules 11-13, and a single application of rule 10, allows us to construct a set $U \subset \mathcal{E} \times \mathcal{K}_\rightarrow$ such that $(x, U) \in C(\langle x, T \rangle, \langle e \rangle f')$ and $k' \preceq (x, U)$.

The remaining case is when $f \equiv \langle e \rangle f'$. As there exists a $k' \xrightarrow{e} k''$ such that $k'' \models f'$, there exists a corresponding $(e, n) \in T$ such that $k'' \preceq n$. Application of the induction hypothesis then results in $m \in C(n, f')$. Following application of rule 14, we obtain $\langle x, \{(e, m), (e, n) \} \cup T \rangle \in C((x, (e, n) \cup T, \langle e \rangle f')$. The simulation requirement $k' \preceq (x, \{(e, m), (e, n) \} \cup T)$ is satisfied because original behavior is retained by rule 14.

10. CONCLUSIONS

In the research presented in this paper, we investigated the synthesis for Hennessy-Milner Logic on Kripke-structures with labeled transitions. A bisimilarity-preserving transformation is applied to transform a KTS into an equivalent partial tree representation, which is able to capture an embedded unfolding. Upon this structure, operational rules define the required modifications in order to satisfy the synthesized HML formula. Results in the set of synthesized models are shown to be valid in terms of satisfying the given HML formula, and simulation of the original input KTS. A maximal solution with regard to the simulation preorder is shown to be contained in this set. A significant part of definitions and proofs are computer verified, which contributes to the understanding and assessment of the validity of the proposed theory. Part of the material presented here appeared earlier as [van Hulst et al. 2013]. The extension of the maximality result for all non-deterministic simulants proves a key property of the synthesis method: the least amount of modifications is applied in order to satisfy the synthesized formula. This induces maximal permissiveness in the context of supervisory control.

HML is limited in its ability to specify many desired requirements. Nevertheless, the HML operators investigated in this work require a synthesis approach which is likely to be similar for those operators in more expressive logics. We therefore intend to extend the synthesized logic in future work by studying the synthesis problem for additional operators, notably those related to invariant or reachability expressions. Also, aspects related to controllability seem worthwhile to investigate, as it would bring the presented synthesis method closer to existing approaches in supervisory control.
ACKNOWLEDGMENTS

This work is supported by the EU FP7 Programme under grant agreement no. 295261 (MEALS).

REFERENCES


