Maximal Synthesis for Hennessy-Milner Logic

A.C. van Hulst, M.A. Reniers, W.J. Fokkink
Eindhoven University of Technology
Eindhoven, The Netherlands
{ahulst,m.a.reniers,w.j.fokkink}@tue.nl

***Abstract*—We present a solution for the synthesis on Kripke-structures with labelled transitions, with respect to Hennessy-Milner Logic. This encompasses the definition of a theoretical framework that is able to express how such a transition system should be modified in order to satisfy a given HML-formula. The transition system is mapped under bisimulation equivalence onto a recursive structure, thereby unfolding up to the applicable reach of a given HML-formula. Operational rules define the required adaptations to ensure validity upon this structure. Synthesis might result in multiple valid adaptations which are all related to the original transition system via simulation. The set of synthesized products contains an outcome which is maximal with respect to all deterministic simulants which satisfy the HML-formula.**

I. Introduction

This paper concerns the synthesis on Kripke-structures with labelled transitions, for formulas in Hennessy-Milner Logic [13]. We formally define a theoretical framework which is able to modify a transition system in order to satisfy a given HML formula. Synthesis is thereby required to modify the transition structure to the least possible extent, and to preserve simulation of the original model.

We consider Kripke structures with labelled transitions because this combination eloquently captures both the process dynamics as well as the state-based information of a system. Such a model was introduced earlier as *KTS* in [19]. Basic properties are added to HML to allow full expressibility of formulas on the modelling structure. Within this context, resolving the synthesis problem already proved to be a quite challenging effort. Imposing no restrictions upon the general class of LTSes leads to states playing different roles at various stages of synthesis, for instance if loops are involved. Therefore, a recursive transition structure, which is unfolded up to the applicable reach of the synthesized formula, is derived under bisimulation equivalence. Operational rules define the required modifications within this unfolded structure in order to satisfy the given HML-formula.

Multiple outcomes might exist, for instance if the synthesized formula contains a disjunction. Each result in this synthesized set satisfies the given formula and is related via simulation to the original structure. The result set contains an outcome which is maximal with respect to all deterministic simulants which satisfy the HML-formula. Key elements of the synthesis process are shown in Figure 1. Note that our approach of using the simulation preorder does not guarantee deadlock-freeness. Computational complexity of the method described in this paper is potentially exponential in terms of the size of the HML formula subject to synthesis.

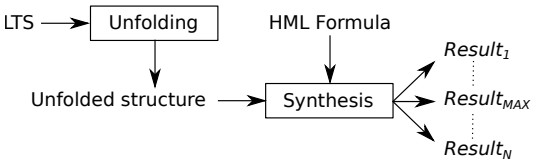

Figure 1. Key parts of the synthesis process include unfolding up to the applicable reach of the synthesized formula as well as the actual synthesis itself, which may result in multiple solutions.

The main contribution of this paper lies in a definition of synthesis on non-deterministic transition systems that connects to existing modelling approaches and is maximally permissive. In modelling and verification of discrete-event systems, a dual approach based on LTS-like structures and logical formulas is often used. Within this two-folded setup, discrete-event models are used to capture process dynamics as well as state-based information. On the other hand, requirements are often specified in a declarative way, based on some form of (temporal) logic. Our definition of synthesis modifies a discrete-event model in order to satisfy one or more logical requirements, and is therefore in line with the existing modelling approach. Additionally, synthesis is defined upon non-deterministic transition systems with no limitations on cyclic behavior. This allows for modelling the input LTS in the broadest possible way.

Synthesis should modify a transition system to the least possible extent. It should not remove a state or transition unless such removal is absolutely neccessary to gain validity of the requirement. If further analysis is to be applied to the synthesized model, for instance to investigate liveness properties, it is of the utmost importance that as much behavior as possible is preserved. Using the simulation preorder, this is expressed as maximality with respect to all simulants which satisfy the synthesized formula. For the specific definition of synthesis as outlined in this article, this property is shown for all *deterministic* simulants.

The research in this paper is part of the broader context of model based systems engineering (MBSE), a discipline which closely co-aligns with the field of supervisory control theory [18]. Within MBSE, model-based design techniques

such as automated verification, model-driven controller design and code generation are integrated into a single workflow. Various applications include the supervisory control synthesis of theme park vehicles [11] and a compositional interchange format for hybrid systems [20]. We view the type of synthesis in this paper as a future integral part of MBSE, in order to efficiently obtain a verified system.

Previous and ongoing similar research into this problem revolves around several extended logics. Notably the branching time temporal framework [8], the μ-calculus [16], unrestricted CTL [1], linear temporal logic [27], [24], PLTL-formulas [9] and CTL* [14]. The urge to resolve this matter originates from various fields. In a theoretical setting, operational and logical styles of specifications were combined in one unified framework [17], [6]. Several attempts using tableau-based methods cannot be left unmentioned [9], [10], and valuable knowledge about the complexity of generalized synthesis was gained [1]. A different way to tackle this problem is via (symbolic) model checking [12], [7], [2], where some similarity to planning approaches [4] exists. Another approach in the model checking area [25] concerns the alteration-free fragment of the μ-calculus, taking addition and removal of transitions into account.

The field of supervisory control [23] is by far the main contributor to solutions in line with our approach [14], [15], [28], [21], [1], [3], even maturing into various algorithms (and realizations thereof) [22], [29]. In this paper we do not take into account a number of specific aspects of supervisory control theory, such as marker state reachability, controllability and deadlock avoidance [23].

Our approach as presented in this paper differs in a number of aspects from the results of the aforementioned authors. Several authors apply synthesis only on requirements stated as formulas, in conjunction with unrestricted system behavior (see e.g. [16], [24]), thereby obtaining a transition system that satisfies the stated requirements. Instead, we apply synthesis on an existing discrete-event model according to declarative requirements. Other papers use seemingly more extended logics [8] but omit some key elements such as a complete coverage of the $\vee$-operator. Other approaches use a different formalism for the automaton before and after synthesis [21] or present an approach that selects control actions online [15]. A number of articles do not take into account the maximality or maximal permissiveness of the synthesized products, see for instance [14] and [15]. As a further difference between this paper and earlier work, we define synthesis upon non-deterministic transition systems, while other authors explicitly require the input LTS to be deterministic [12], [14], [16]. We also do not require the LTS model to be non-terminating as in [12].

The remainder of this paper is set up as follows: In Section II we give a number of preliminary definitions concerning an LTS-like structure which is able to capture inherent unfoldings as well as the corresponding behavioral relations of similarity and bisimilarity on this structure. In Section III we work towards an appropriate formal definition of synthesis by considering specific situations and several caveats we encountered while studying this matter. Section IV provides the theoretical framework for synthesis which is shown to be valid by two theorems in Section V. The aforementioned maximality requirement is the subject of Section VI and we conclude this paper by discussing future improvements in terms of extensions of the logic.

II. Definitions

We assume the existence of a set of events $\mathcal{E}$, a set of atomic propositions $\mathcal{P}$ and a set of basic states $\mathcal{X}$. We further assume the existence of a labelling function $L : \mathcal{X} \mapsto 2^{\mathcal{P}}$ relating states to properties. We say that atomic proposition $p \in \mathcal{P}$ holds in state $x \in \mathcal{X}$ if and only if $p \in L(x)$. By assuming these definitions at a global level, we are able to bring much clarity in the definition of synthesis later on, while not limiting the scope of this definition.

We use a non-standard definition to express structural behavior. Instead of the commonly used labelled transition system (LTS), we define a set $\mathcal{K}_{\rightarrow}$ of transition structures in Definition 1. This approach has the following advantages:

1) Elements $k \in \mathcal{K}_{\rightarrow}$ represent an unfolded part of a recursive transition structure, as well as a continuation via a transition relation. The synthesis process builds upon unfolding an LTS up to the applicable reach of a HML-formula. This particular choice of $\mathcal{K}_{\rightarrow}$ allows us to embed the unfolding directly into this structure.
2) Modifications on transitions are expressed more conveniently using this definition via addition of elements to a set. We found this approach to be much more transparent compared to modifications on transition relations.

The intuitive idea behind the following Definition 1 is that it represents a partially unfolded LTS and continues via a 'usual' transition relation.

Definition 1. We define the set $\mathcal{K}_{\rightarrow}$ for $\rightarrow \subseteq \mathcal{X} \times \mathcal{E} \times \mathcal{X}$ in the following way. An element $k \in \mathcal{K}_{\rightarrow}$ is either:

1) A single initial state $x \in \mathcal{X}$, denoted by $\langle x \rangle_{\rightarrow}$.
2) A pair $\langle x, T \rangle_{\rightarrow}$ where $T \subset \mathcal{E} \times \mathcal{K}_{\rightarrow}$. It specifies the initial state as well as a set of successive structures. Note that this definition holds for *syntactic* inclusion only. Therefore, definitions such as $k = \langle x, \{(e, k)\} \rangle_{\rightarrow}$ are invalid within this context.

Note that Definition 1 denotes the *least* set $\mathcal{K}_{\rightarrow}$, having a single initial state, which satisfies the properties in Definition 1. It is clear that Definition 1 is a generalization of the commonly used LTS because $\langle x \rangle_{\rightarrow}$ is isomorphic to such an LTS. We will use the following notations.

$$\langle x, T \rangle_{\rightarrow} \xrightarrow{e} k \quad \Longleftrightarrow \quad (e, k) \in T$$
$$\langle x \rangle_{\rightarrow} \xrightarrow{e} \langle x' \rangle_{\rightarrow} \quad \Longleftrightarrow \quad (x, e, x') \in \rightarrow$$

Often we will use the generalized notation $k \xrightarrow{e} k'$ which refers to one of these notations, depending on the form of k. An example instance of Definition 1 is shown in Figure 2. Note that this is just *one* particular representation of the transition system in terms of $\mathcal{K}_\rightarrow$ and that other representations are possible as well.

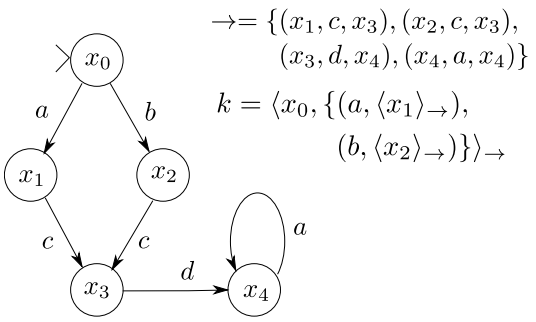

Figure 2. An example instance of Definition 1, which represents an unfolded structure $k \in \mathcal{K}_\rightarrow$. In this case, unfolding is captured up to depth 1, after which transitions are defined using the transition relation $\rightarrow$.

We use the simulation preorder and bisimulation equivalence to relate elements in $\mathcal{K}_\rightarrow$ according to the following definitions.

Definition 2. For $k' \in \mathcal{K}_{\rightsquigarrow}, k \in \mathcal{K}_\rightarrow$, we say that k' is *simulated* by k (denoted by $k' \preceq k$) if there exists a relation $R \subseteq \mathcal{K}_{\rightsquigarrow} \times \mathcal{K}_\rightarrow$ such that $(k', k) \in R$ and for all $(m', m) \in R$ we have:

- If x', x are the respective initial states of m', m, then for all $p \in \mathcal{P}$ we have $p \in L(x')$ if and only if $p \in L(x)$.
- For all $m' \xrightarrow{e} n'$ there exists an $n \in \mathcal{K}_\rightarrow$ such that $m \xrightarrow{e} n$ and $(n', n) \in R$.

We use $k' \preceq_R k$ to indicate that $k' \preceq k$ as witnessed by R. The notations $\rightsquigarrow$ and $\rightarrow$ are used specifically in Definition 2 to indicate that the two underlying transition relations for k' and k do not need to be the same.

The first clause of Definition 2 is non-standard in the sense that it requires equality of the sets of satisfied basic properties, instead of inclusion of these sets, as the reader might expect. This reflects the synthesis semantics where validity is enforced by removal of transitions, while state-based properties are not adjusted.

Definition 3. For $k' \in \mathcal{K}_{\rightsquigarrow}, k \in \mathcal{K}_\rightarrow$, we say that k' and k are related under *bisimulation* equivalence (denoted by $k' \underline{\leftrightarrow} k$) if there exists a relation $R \subseteq \mathcal{K}_{\rightsquigarrow} \times \mathcal{K}_\rightarrow$ such that $(k', k) \in R$ and for all $(m', m) \in R$ we have:

- If x', x are the respective initial states of m', m, then for all $p \in \mathcal{P}$ we have $p \in L(x')$ if and only if $p \in L(x)$.
- For all $m' \xrightarrow{e} n'$ there exists an $n \in \mathcal{K}_\rightarrow$ such that $m \xrightarrow{e} n$ and $(n', n) \in R$.
- For all $m \xrightarrow{e} n$ there exists an $n' \in \mathcal{K}_{\rightsquigarrow}$ such that $m' \xrightarrow{e} n'$ and $(n', n) \in R$.

We use $k' \underline{\leftrightarrow}_R k$ to indicate that $k' \underline{\leftrightarrow} k$ as witnessed by R.

Simulation is reflexive as witnessed by the identity relation on $\mathcal{K}_\rightarrow$ 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 (see e.g. [26]).

We define the following set of formulas. Note that this definition adds the atomic proposition $p \in \mathcal{P}$, as well as its negation $\neg p$, to HML [13].

Definition 4. The set $\mathcal{F}$ is defined for $p \in \mathcal{P}$ and $e \in \mathcal{E}$ as follows.

$$\mathcal{F} \Rightarrow tt \mid ff \mid p \mid \neg p \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid [e]\,\mathcal{F} \mid \langle e \rangle \mathcal{F}$$

The formulas tt and ff indicate truth and falsehood respectively, while the formula p, for $p \in \mathcal{P}$, can be used to test whether atomic proposition p holds in a specific state. Negation $\neg p$ is defined for state-based properties only. The meaning of the operators for conjunction $\wedge$ and disjunction $\vee$ is as expected. The one-step-lookahead operator $[e]\,f$ tests whether f holds after every e-step while $\langle e \rangle f$ tests for the existence of an e-step such that f holds. Negation at the level of atomic propositions is sufficient to extend the operator $\neg$ to the full set $\mathcal{F}$, as shown in [13].

We express the validity of formulas in $\mathcal{F}$ in terms of $\mathcal{K}_\rightarrow$ by defining a *valuation* function $\vDash$ as follows.

Definition 5. The predicate $\vDash$ over $\mathcal{K}_\rightarrow \times \mathcal{F}$ is defined for $k \in \mathcal{K}_\rightarrow$, $f, g \in \mathcal{F}$, $e \in \mathcal{E}$, $x \in \mathcal{X}$, $p \in \mathcal{P}$ and $T \subset \mathcal{E} \times \mathcal{K}_\rightarrow$ by the following deduction rules.

$$\frac{}{k \vDash tt} \qquad \frac{p \in L(x)}{\langle x \rangle_\rightarrow \vDash p} \qquad \frac{p \in L(x)}{\langle x, T \rangle_\rightarrow \vDash p}$$

$$\frac{p \notin L(x)}{\langle x \rangle_\rightarrow \vDash \neg p} \qquad \frac{p \notin L(x)}{\langle x, T \rangle_\rightarrow \vDash \neg p}$$

$$\frac{k \vDash f \quad k \vDash g}{k \vDash f \wedge g} \qquad \frac{k \vDash f}{k \vDash f \vee g} \qquad \frac{k \vDash g}{k \vDash f \vee g}$$

$$\frac{\forall k \xrightarrow{e} k'.\ k' \vDash f}{k \vDash [e]\,f} \qquad \frac{k \xrightarrow{e} k' \quad k' \vDash f}{k \vDash \langle e \rangle f}$$

If $k \vDash f$, then we say that k *satisfies* the formula f. Note that the arrow notation such as in $k \xrightarrow{e} k'$ refers to the notation as introduced after Definition 1.

III. 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 terms of elements in the set $\mathcal{K}_\rightarrow$.

We first take a closer look at synthesis for the various elements in $\mathcal{F}$. It is clear that synthesis of tt should be

neutral as this formula always evaluates to $true$ and therefore no modification of the underlying structure is required to satisfy this formula. On the other hand, synthesis of the formula ff should not yield any result because no possible modification to the original structure exists in order to satisfy this formula. The formulas p, for $p \in \mathcal{P}$, are always evaluated and synthesized with respect to a single state $x \in \mathcal{X}$. If $p \in L(x)$, then the synthesis should be the same as if the formula were tt, where no modification of the underlying structure is required. On the other hand, if $p \notin L(x)$, then the formula should be treated as if it were ff and synthesis should result in void or emptiness. Note that assigning the atomic proposition p to x if $p \notin L(x)$ is not desired, as this would add information to the transition structure, thereby invalidating the synthesis semantics, as indicated before. The inverse procedure is followed for the state-based negation $\neg p$. If $p \notin L(x)$, then no modification needs to be applied to satisfy the formula $\neg p$. However, if $p \in L(x)$ then the formula $\neg p$ cannot be satisfied for x and therefore synthesis should be empty.

We now consider the operators $[e]f$ and $\langle e\rangle f$ first because any non-trivial example regarding the operators $\wedge$ and $\vee$ makes use of either $[e]f$ or $\langle e\rangle f$. For the operator $[e]f$ for $f \in \mathcal{F}$ we apply synthesis recursively in each e-step for the formula f. If such synthesis cannot be performed, for instance if $f \equiv ff$, then we remove the corresponding e-step. On the other hand, if synthesis is successful, the e-step is retained and the transition system is modified recursively after the e-step in order to satisfy f. 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 $\langle e\rangle f$ should be empty. Otherwise, synthesis proceeds recursively after this one e-step while the rest of the transitions are left in place unmodified. 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 $[e]p$ is illustrated in Figure 3a) while synthesis for the formula $\langle e\rangle[e]p$ is shown in Figure 3b).

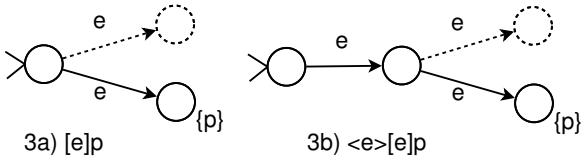

Figure 3. Two straightforward examples showing synthesis of the formulas $[e]p$ and $\langle e\rangle[e]p$, for atomic proposition $p \in \mathcal{P}$. Dotted lines and states are part of the original transition system but were removed during synthesis.

We now proceed by considering the operators $\wedge$ and $\vee$. In Figures 4a)-4c) it is shown how multiple valid adaptations might exist which all satisfy $[e]p \vee [e]q$. However, these solutions cannot be combined in any meaningful way in a single transition system. Without further information, these solutions are essentially 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 valid modifications to the original structure. Therefore, we will define the synthesis function in the next section as having the signature $\mathcal{K}_{\rightarrow} \times \mathcal{F} \mapsto 2^{\mathcal{K}_{\rightarrow}}$. Note that this remark also has its drawback on the definition of synthesis for the operators $[e]f$ 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.

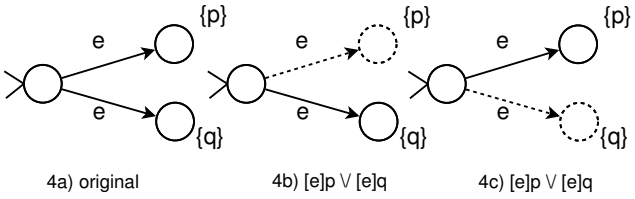

Figure 4. Synthesis for the operator $\vee$ might result in multiple valid adaptations which are essentially incomparable. Therefore, synthesis is defined in such a way that it results in a *set* of synthesized products.

The operator $\wedge$ introduces new complications. As shown in Figures 5a)-5d), multiple applications of the synthesis for each conjunct might be required to obtain a synthesis result that satisfies both formulas. The original transition system as shown in 5a), is modified in order to satisfy the formula $\langle e\rangle[e]q \wedge [e]\langle e\rangle p$. The end result, as shown in Figure 5d), is obtained via the intermediate steps 5b) and 5c). Synthesis of $\langle e\rangle[e]q$ is applied to the original in 5a), resulting in 5b). Consequently, we apply synthesis for $[e]\langle e\rangle p$ in 5b), resulting in 5c). In the last step, synthesis for $\langle e\rangle[e]q$ in 5c) results in 5d), which already satisfies the second conjunct. Observe that a stable point in the alternating application of synthesis has now been reached, and the entire conjunctive formula has become valid. We generalize this process in the next section, where synthesis for conjunction will be defined as a fixpoint definition that alternatingly applies synthesis for both conjuncts. Termination of this fixpoint is satisfied if synthesis for both conjuncts results in the same structure. As synthesis either leaves the structure unmodified or results in a strictly lower number of transitions, this fixpoint is guaranteed to terminate, and might give an empty result set if required. Note that two possible intermediate results exist after the first synthesis step for the formula $\langle e\rangle[e]q$ on the model in Figure 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 number of modifications is applied in order to satisfy the given formula. Maximality is reflected in two ways in the synthesis process: 1) If possible, synthesis should be defined to be

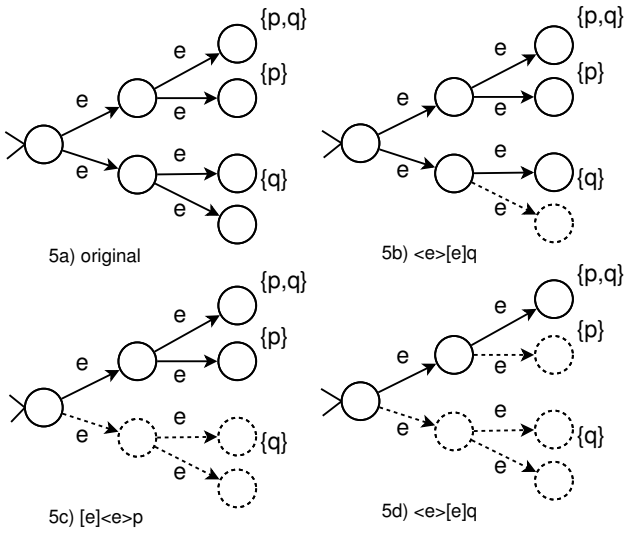

Figure 5. Synthesis for the operator $\wedge$ is realized via alternating application of synthesis for both conjuncts. Figure 5a) shows the original transition system and 5d) the final result after synthesis for $\langle e\rangle[e]q \wedge [e]\langle e\rangle p$, via intermediate steps 5b) and 5c). The result in 5d) remains neutral for the synthesis of both conjuncts and thereby marks the final step in the alternating synthesis procedure.

maximal for each operator, 2) The set of synthesis products should contain a maximal solution. The first property is illustrated in Figures 6a)-6c) where a non-maximal as well as a maximal solution is given for the formula $[e]p$. Synthesis for the operator $[e]$, as defined formally in the next section, therefore excludes results such as in 6c).

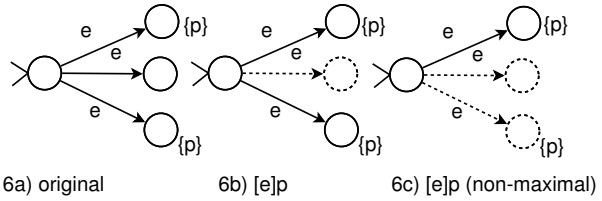

Figure 6. Synthesis of the formula $[e]p$ is applied to the original model in 6a), resulting in the modification shown in 6b). Figure 6c) shows a non-maximal and therefore invalid solution.

Regarding the second property, it should be noted that non-maximal solutions cannot always be avoided. As shown in Figures 7a)-7c), the set of synthesis results for the formula $[e]p \vee [e](p \wedge q)$, contains a non-maximal solution that cannot be excluded due to the nature of the synthesis for disjunction, where each operand is considered separately. However, in Section VI we will show that a maximal outcome, up to all deterministic simulants which satisfy the given formula, is always contained in the synthesized set.

The second general aspect is related to *unfolding*, a topic with severe implications that has been mentioned before, and justifies the definition of a non-standard transition structure. Unfolding is induced by the fact that states may play

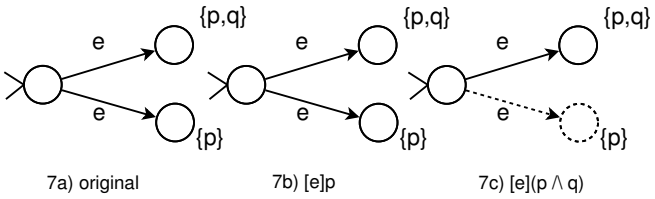

Figure 7. Synthesis of the formula $[e]p \vee [e](p \wedge q)$ contains a non-maximal solution due to the nature of the synthesis of disjunction, where each operand is considered separately and results are merged into a set of synthesis products

multiple roles in various stages of synthesis, for instance if loops are involved. This is illustrated in Figure 8a), where an obvious solution might seem to remove the loop in the initial state in order to satisfy the formula $[e]p$. However, this would violate the maximality requirement because not all possible behavior is retained. We therefore unfold the model for as far as the applicable reach of $[e]p$, which is one step, in this case. The result shown in Figure 8b) indicates a one-step unfolded structure, where a transition can be safely removed, while preserving maximality. In the next section, we will show how a bisimilar unfolded transition system can be obtained for any given depth.

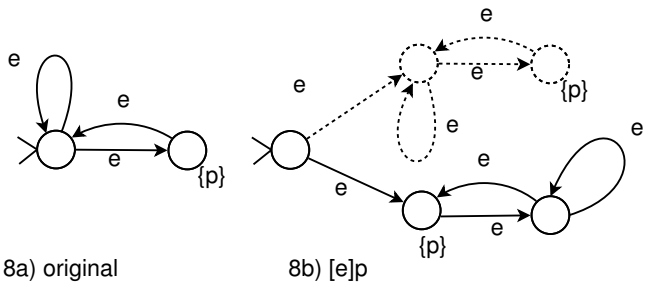

Figure 8. Before synthesis is applied, the transition structure is unfolded up to the applicable reach of the synthesized formula. This allows transitions to be removed safely, while retaining maximality.

IV. Operational Definition

A predicate unf which can be used to test for unfoldedness is used in several proofs and defined as follows.

Definition 6. The set $unf \subseteq \mathcal{K}_{\rightarrow} \times \mathbb{N}$ is defined for $x \in \mathcal{X}$, $T \subset \mathcal{E} \times \mathcal{K}_{\rightarrow}$ and $n \in \mathbb{N}$ by the following definition:

$$\begin{array}{lcl} unf(\langle x\rangle_{\rightarrow}, n) & \iff & n = 0 \\ unf(\langle x, T\rangle_{\rightarrow}, 0) & \iff & true \\ unf(\langle x, T\rangle_{\rightarrow}, n+1) & \iff & \forall (e,k) \in T.\ unf(k,n) \end{array}$$

We define an appropriate norm on formulas that will be used in accordance with the notion of unfoldedness. Before synthesis is applied to a transition structure, unfolding is applied for as far as the following norm indicates.

Definition 7. We define $size : \mathcal{F} \mapsto \mathbb{N}$ for $p \in \mathcal{P}$, $e \in \mathcal{E}$ and $f, f_1, f_2 \in \mathcal{F}$ in the following way.

$$\begin{array}{lcl} size(tt) & = & 0 \\ size(ff) & = & 0 \\ size(p) & = & 0 \\ size(\neg p) & = & 0 \\ size(f_1 \wedge f_2) & = & max(size(f_1), size(f_2)) \\ size(f_1 \vee f_2) & = & max(size(f_1), size(f_2)) \\ size(\,[e]\,f) & = & 1 + size(f) \\ size(<e>f) & = & 1 + size(f) \end{array}$$

For $\langle x \rangle_{\rightarrow} \in \mathcal{K}_{\rightarrow}$ it was already mentioned that this structure is isomorphic to an LTS in the traditional sense. We now show how a bisimilar structure exists that is unfolded for any given depth. Within the greater picture, this is the first step in the synthesis process.

Lemma 1. For each $k \in \mathcal{K}_{\rightarrow}$ and $n \in \mathbb{N}$ there exists a $k' \in \mathcal{K}_{\rightarrow}$ such that $k' \underline{\leftrightarrow} k$ and $unf(k', n)$.

Proof: We prove this property by induction on n, thereby generalizing over k. If $n = 0$, then obviously $k \underline{\leftrightarrow} k$ and $unf(k, 0)$. We continue by distinguishing between the two forms of k. Suppose that $k = \langle x \rangle_{\rightarrow}$. Then there exists a set $\{(e_1, x_1), \ldots, (e_m, x_m)\}$ such that for each $1 \leq i \leq m$ we have $\langle x \rangle_{\rightarrow} \xrightarrow{e_i} \langle x_i \rangle_{\rightarrow}$. We may apply the induction hypothesis in each $\langle x_i \rangle_{\rightarrow}$ to obtain a k_i such that $unf(k_i, n)$ and $\langle x_i \rangle_{\rightarrow} \underline{\leftrightarrow}_{R_i} k_i$. Let $T = \{(e_1, k_1), \ldots, (e_m, k_m)\}$. Then by Definition 6 we have $unf(\langle x, T \rangle_{\rightarrow}, n+1)$ and $\langle x, T \rangle_{\rightarrow} \underline{\leftrightarrow}$ $\langle x \rangle_{\rightarrow}$ as witnessed by: $\cup_{1 \leq i \leq m} R_i \cup \{(\langle x, T \rangle, \langle x \rangle)\}$. If $k = \langle x, T \rangle_{\rightarrow}$, then $T = \{(e_1, k_1), \ldots, (e_m, k_m)\}$. By the induction hypothesis for each k_i, this results in a k_i' such that $unf(k_i', n)$ and $k_i \underline{\leftrightarrow} k_i'$. Let $T' = \{(e_1, k_1'), \ldots, (e_m, k_m')\}$ then $unf(\langle x, T' \rangle_{\rightarrow}, n+1)$ and $\langle x, T \rangle_{\rightarrow} \underline{\leftrightarrow} \langle x, T' \rangle_{\rightarrow}$. ■

We define the synthesis function $C : \mathcal{K}_{\rightarrow} \times \mathcal{F} \mapsto 2^{\mathcal{K}_{\rightarrow}}$ inductively as a relation via deduction rules. Note that the function C is defined in such a way that it expects the first argument $k \in \mathcal{K}_{\rightarrow}$ to be *unfolded* to at least depth $size(f)$. 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 8. Let $k \in \mathcal{K}_{\rightarrow}$ and $f \in \mathcal{F}$ in the following rules.

$$\frac{}{k \in C(k, \mathrm{tt})}\textbf{[1]} \qquad \frac{p \in L(x)}{\langle x \rangle \in C(\langle x \rangle, p)}\textbf{[2]}$$

$$\frac{p \in L(x)}{\langle x, T \rangle \in C(\langle x, T \rangle, p)}\textbf{[3]} \qquad \frac{p \notin L(x)}{\langle x \rangle \in C(\langle x \rangle, \neg p)}\textbf{[4]}$$

$$\frac{p \notin L(x)}{\langle x, T \rangle \in C(\langle x, T \rangle, \neg p)}\textbf{[5]} \qquad \frac{m \in C(k, f) \quad m \in C(k, g)}{m \in C(k, f \wedge g)}\textbf{[6]}$$

$$\frac{k' \in C(k, f) \quad m \in C(k', g \wedge f)}{m \in C(k, f \wedge g)}\textbf{[7]} \qquad \frac{m \in C(k, f)}{m \in C(k, f \vee g)}\textbf{[8]}$$

$$\frac{m \in C(k, g)}{m \in C(k, f \vee g)}\textbf{[9]} \qquad \frac{}{\langle x, \emptyset \rangle \in C(\langle x, \emptyset \rangle, \,[e]\,f)}\textbf{[10]}$$

$$\frac{\langle x, T' \rangle \in C(\langle x, T \rangle, \,[e]\,f) \quad e \neq e'}{\langle x, \{(e', k)\} \cup T' \rangle \in C(\langle x, \{(e', k)\} \cup T \rangle, \,[e]\,f)}\textbf{[11]}$$

$$\frac{\langle x, T' \rangle \in C(\langle x, T \rangle, \,[e]\,f) \quad C(k, f) = \emptyset}{\langle x, T' \rangle \in C(\langle x, \{(e, k)\} \cup T \rangle, \,[e]\,f)}\textbf{[12]}$$

$$\frac{\langle x, T' \rangle \in C(\langle x, T \rangle, \,[e]\,f) \quad m \in C(k, f)}{\langle x, \{(e, m)\} \cup T' \rangle \in C(\langle x, \{(e, k)\} \cup T \rangle, \,[e]\,f)}\textbf{[13]}$$

$$\frac{m \in C(k, f)}{\langle x, \{(e, m)\} \cup T \rangle \in C(\langle x, \{(e, k)\} \cup T \rangle, <e>f)}\textbf{[14]}$$

We briefly discuss the deduction rules for C in Definition 8. Synthesis is neutral for tt as this formula is always satisfied (rule **1**). Synthesis for an atomic proposition 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 atomic proposition $\neg p$ results in the same structure if $p \notin L(x)$, as can be observed in rules **4** and **5**. The rules **6** and **7** define a fixpoint construction for the synthesis of a conjunction. The condition for termination as described in rule **6** applies when synthesis of both conjuncts results in the same structure. Otherwise, both conjuncts are synthesized alternatingly 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 modifications are required. Rule **11** details how an $e' \neq e$ transition is left in place for the operator $[e]f$, as this transition does not influence the satisfiability 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 structure after an e-step is replaced by an appropriate synthesis product. Finally, we define a single rule **14** for the synthesis of the formula $<e>f$. 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 $\langle x \rangle_{\rightarrow} \in \mathcal{K}_{\rightarrow}$ for the operators $[e]f$ and $<e>f$ because that would invalidate the implicit unfoldedness condition.

V. Validity of our Approach

We prove two theorems regarding the validity of the definition of synthesis. In Theorem 1 we show that every synthesis result satisfies the synthesized formula. Theorem 2 details how every synthesis result is related via simulation to the original structure. Two additional lemmas are important with regard to the next section on maximality. Lemma 2 shows that synthesis preserves unfoldedness and Lemma 3 details how synthesis does not modify a transition structure

if it already satisfies the required formula.

Theorem 1. For $f \in \mathcal{F}$ and $k, m \in \mathcal{K}_{\rightarrow}$ we have $m \in C(k, f)$ implies that $m \vDash f$.

Proof: The proof is by induction towards the construction of $m \in C(k, f)$, via the deduction rules in Definition 8. If $m \in C(k, tt)$, then obviously $m \vDash tt$. If $m \in C(k, p)$, for some $p \in \mathcal{P}$, then m and k have the same initial state resulting in $m \vDash p$. If $m \in C(k, \neg p)$, then again we observe that m and k have the same initial state and therefore $m \vDash \neg p$. For $f \equiv f_1 \wedge f_2$ we have the following analysis: If $m \in C(k, f_1)$ and $m \in C(k, f_2)$, then $m \vDash f_1 \wedge f_2$ by induction. For $k' \in C(k, f_1)$ and $m \in C(k', f_2 \wedge f_1)$, by induction and commutativity of the validity of $\wedge$ we have that $m \vDash f_1 \wedge f_2$. If $f \equiv f_1 \vee f_2$, then we again have two cases. When $m \in C(k, f_1)$ then $m \vDash f_1 \vee f_2$, and if $m \in C(k, f_2)$ then $m \vDash f_1 \vee f_2$, both by induction. We have four cases corresponding to the rules **10**-**13** for $f \equiv [e]\, f'$. By induction $\langle x, \emptyset \rangle_{\rightarrow} \vDash [e]\, f'$ and $\langle x, \{(e', k)\} \cup T' \rangle_{\rightarrow} \vDash [e]\, f'$ for each $e \neq e'$ if $\langle x, T' \rangle_{\rightarrow} \vDash [e]\, f'$. Rule **12** does not alter the structure of $m \in C(k, [e]\, f)$ and therefore preserves validity. If $\langle x, T' \rangle_{\rightarrow} \vDash [e]\, f$ and $m \vDash f$ for $m \in C(k, f)$, then we have $\langle x, \{(e, m)\} \cup T' \rangle_{\rightarrow} \vDash [e]\, f$. The last case is where $f \equiv \langle e \rangle f'$. If we have $m \in C(k, f')$ and therefore $m \vDash f'$, then by induction we certainly have that $\langle x, \{(e, m)\} \cup T' \rangle_{\rightarrow} \vDash \langle e \rangle f'$. ■

Theorem 2. For $f \in \mathcal{F}$ and $k, m \in \mathcal{K}_{\rightarrow}$ we have $m \in C(k, f)$ implies $m \preceq k$.

Proof: We use the same proof strategy as in Theorem 1: induction towards 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**-**6**,**8**-**10** are solved by reflexivity of simulation while rule **7** is covered by induction and transitivity of simulation. The four remaining cases consider the rules **11**-**14**. For rule **11**, we may assume $\langle x, T' \rangle_{\rightarrow} \preceq \langle x, T \rangle_{\rightarrow}$ as our induction hypothesis. This directly leads to $\langle x, \{(e', k)\} \cup T' \rangle_{\rightarrow} \preceq \langle x, \{(e', k)\} \cup T \rangle_{\rightarrow}$. For rule **12** we have $\langle x, T' \rangle_{\rightarrow} \preceq \langle x, T \rangle_{\rightarrow}$ by induction and therefore $\langle x, T' \rangle_{\rightarrow} \preceq \langle x, \{(e, k)\} \cup T \rangle_{\rightarrow}$. For the case corresponding to rule **13** we have as our induction hypothesis: $\langle x, T' \rangle_{\rightarrow} \preceq \langle x, T \rangle_{\rightarrow}$ and additionally $m \preceq k$ for $m \in C(k, f)$. This leads to $\langle x, \{(e, m)\} \cup T' \rangle_{\rightarrow} \preceq \langle x, \{(e, k)\} \cup T \rangle_{\rightarrow}$. We conclude this proof by an analysis of the last rule **14** for which we have $m \in C(k, f)$ and therefore $m \preceq k$ as induction hypothesis. Clearly this leads to $\langle x, \{(e, m))\} \cup T' \rangle_{\rightarrow} \preceq \langle x, \{(e, k)\} \cup T \rangle_{\rightarrow}$. ■

Lemma 2. For each $f \in \mathcal{F}$, $n \in \mathbb{N}$ and $k, m \in \mathcal{K}_{\rightarrow}$ such that $m \in C(k, f)$ and $unf(k, n)$ we have $unf(m, n)$.

Proof: Like in the previous two proofs in this section, 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 $unf(\langle x, T \rangle_{\rightarrow}, n) \Rightarrow unf(\langle x, T' \rangle_{\rightarrow}, n)$. Clearly the premise $unf(\langle x, \{(e', k)\} \cup T' \rangle_{\rightarrow}, n)$ leads to $unf(\langle x, \{(e', k)\} \cup T, \rangle_{\rightarrow}, n)$. Rule **12** does not modify the $m \in C(k, [e]\, f)$ and therefore preserves unfoldedness. For rule **13** we have two induction hypotheses: $unf(m, n)$ and $unf(\langle x, T \rangle_{\rightarrow}, n) \Rightarrow unf(\langle x, T' \rangle_{\rightarrow}, n)$. Based on the premise $unf(\langle x, \{(e, k)\} \cup T \rangle_{\rightarrow}, n)$ we may immediately draw the conclusion that $unf(\langle x, \{(e, m)\} \cup T' \rangle_{\rightarrow}, n)$. For the last case concerning rule **14** we have $unf(\langle x, T \rangle_{\rightarrow}, n)$, $unf(m, n)$ and therefore $unf(\langle x, \{(e, m)\} \cup T \rangle_{\rightarrow}, n)$ by induction. ■

Lemma 3. If $k \vDash f$ for $k \in \mathcal{K}_{\rightarrow}$ and $f \in \mathcal{F}$ such that $unf(k, size(f))$ then $k \in C(k, f)$.

Proof: We show this property by structural induction on f. If $f \equiv tt$, then $k \in C(k, f)$ according to rule **1**. If $f \equiv p$ or $f \equiv \neg p$, for $p \in \mathcal{P}$, then we may apply rules **2**-**5**, depending on the form of k, to obtain $k \in C(k, f)$. For the case $f \equiv f_1 \wedge f_2$ we have $k \in C(k, f_1)$ and $k \in C(k, f_2)$ by induction. The induction premises of $unf(k, size(f_1))$ and $unf(k, size(f_2))$ are easily resolved because $unf(k, size(f_1 \wedge f_2))$ (see Definition 7). Application of rule **6** then gives $k \in C(k, f_1 \wedge f_2)$. The case for $f \equiv f_1 \vee f_2$ is solved in a similar way by induction and application of the respective rules **8** and **9**. If $f \equiv [e]\, f'$, then we know that k is of the form $\langle x, T \rangle_{\rightarrow}$ because otherwise the unfoldedness condition $unf(k, 1 + size(f'))$ would be violated. We proceed by induction on the number of elements in T. If $T = \emptyset$, then $\langle x, \emptyset \rangle_{\rightarrow} \in C(\langle x, \emptyset \rangle_{\rightarrow}, [e]\, f')$ by rule **10**. Let $T = \{(e', k')\} \cup T'$. Then we apply rule **11** if $e \neq e'$. If $e = e'$, then $k' \vDash f'$ and we may apply rule **13**. The last case for f is when $f \equiv \langle e \rangle f'$. Since $k \vDash \langle e \rangle f'$ and k is of the form $\langle x, T \rangle_{\rightarrow}$, there exists an $(e, k') \in T$ with $k' \vDash f'$. The result follows by induction and **14**. ■

VI. Maximality

As indicated before, it is desirable for products of synthesis to be modified to the least extent 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. We refer to this property as *maximality* and it is formulated as Theorem 3. Due to the specific formulation of the deduction rules in Definition 8, we were able to show this property for all *deterministic* simulants. Maximality with respect to all non-deterministic simulants cannot be obtained for the set of deduction rules given in Definition 8 as shown by the counterexample in Figure 9. In Figure 9a) a non-deterministic simulant of the original LTS in Figure 9b) is shown to be unrelated (in terms of simulation) to both synthesis products for the formula $[a]\,([b]\,p \vee [b]\,q)$ shown in Figures 9c) and 9d). As indicated by this counterexample, maximal synthesis up to all non-deterministic simulants remains an open question. We therefore focus on deterministic simulants according to the following definition of determinism.

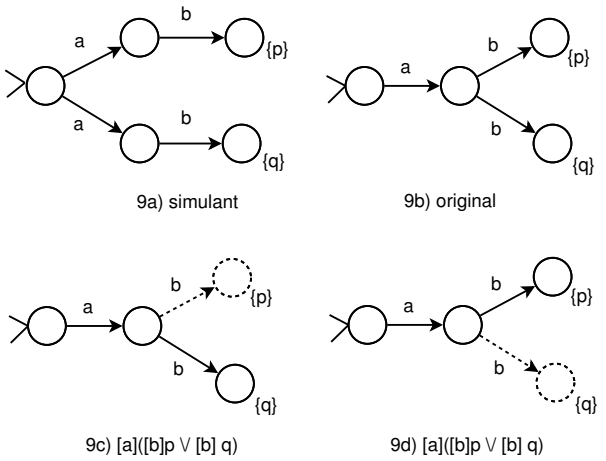

Figure 9. Counterexample indicating synthesis results which are non-maximal for non-deterministic simulants The original in 9b) has a non-deterministic simulant shown in 9a) which is not related via simulation to the synthesis results in 9c) and 9d) for the formula $[a]([b]p \vee [b]q)$.

Definition 9. We define the predicate $det \subset \mathcal{K}_{\rightarrow}$ for each $x \in \mathcal{X}$ and $T \subset \mathcal{E} \times \mathcal{K}_{\rightarrow}$ as follows.

$$det(\langle x, T\rangle_{\rightarrow}) \iff [\ \forall (e,k),(e,k') \in T.\ k = k'\ ] \wedge [\ \forall (e,k) \in T.\ det(k)\ ]$$

$$det(\langle x\rangle_{\rightarrow}) \iff \left[\ \forall x \xrightarrow{e} x'. \forall x \xrightarrow{e} x''. \ x' = x''\ \right] \wedge \left[\ \forall x \xrightarrow{e} x'.\ det(\langle x\rangle_{\rightarrow})\ \right]$$

In the maximality proof as shown below in Theorem 3, we need an appropriate norm to ensure termination of the fixpoint for conjunction.

Definition 10. We define $count : \mathcal{K}_{\rightarrow} \mapsto \mathbb{N}$ to compute the number of transitions of a $k \in \mathcal{K}_{\rightarrow}$ in the following way for $x \in \mathcal{X}$ and $T \subset \mathcal{E} \times \mathcal{K}_{\rightarrow}$:

$$\begin{aligned} count(\langle x\rangle_{\rightarrow}) &= 0 \\ count(\langle x, T\rangle_{\rightarrow}) &= \mid T \mid + \sum_{(e,k)\in T} count(k) \end{aligned}$$

where $\mid T \mid$ is the number of elements in T.

Note that we do not count beyond the unfolded part of any $k \in \mathcal{K}_{\rightarrow}$ because only the unfolded part is modified by the function C. The following Lemma 4 ensures that C results in either an unmodified structure or one with a decreased $count$ value.

Lemma 4. For each $f \in \mathcal{F}$ and $k, m \in \mathcal{K}_{\rightarrow}$ we have that $m \in C(k, f)$ implies either $k = m$ or $count(m) < count(k)$.

Proof: We again apply induction towards the construction of $m \in C(k, f)$. For rules **1**-**10**, this property follows directly. We consider the remaining rules.

11 If $\langle x, T'\rangle_{\rightarrow} = \langle x, T\rangle_{\rightarrow}$ or $count(\langle x, T'\rangle_{\rightarrow}) < count(\langle x, T\rangle_{\rightarrow})$, then this directly leads to $\langle x, \{(e', k)\} \cup T'\rangle_{\rightarrow} = \langle x, \{(e', k)\} \cup T\rangle_{\rightarrow}$ or $count(\langle x, \{(e', k)\} \cup T'\rangle_{\rightarrow}) < count(\langle x, \{(e', k)\} \cup T\rangle_{\rightarrow})$.

12 If $\langle x, T'\rangle_{\rightarrow} = \langle x, T\rangle_{\rightarrow}$ or $count(\langle x, T'\rangle_{\rightarrow}) < count(\langle x, T\rangle_{\rightarrow})$, then in both cases we have $count(\langle x, T'\rangle_{\rightarrow}) < count(\langle x, \{(e, k)\} \cup T\rangle_{\rightarrow}$.

13 Here we have $\langle x, T'\rangle_{\rightarrow} = \langle x, T\rangle_{\rightarrow}$ or $count(\langle x, T'\rangle_{\rightarrow}) < count(\langle x, T\rangle_{\rightarrow})$ as well as $m = k$ or $count(m) < count(k)$. The combination of the respective first disjuncts of these induction hypotheses leads to $\langle x, \{(e, m)\} \cup T'\rangle_{\rightarrow} = \langle x, \{(e, k)\} \cup T\rangle_{\rightarrow}$, while all other combinations of disjuncts lead to $count(\langle x, \{(e, m)\} \cup T'\rangle_{\rightarrow}) < count(\langle x, \{(e, k)\} \cup T)$.

14 If $k = m$ or $count(m) < count(k)$, then we have either $\langle x, \{(e, m)\} \cup T\rangle_{\rightarrow} = \langle x, \{(e, k)\} \cup T\rangle_{\rightarrow}$ or $count(\langle x, \{(e, m)\} \cup T\rangle_{\rightarrow}) < count(\langle x, \{(e, k)\} \cup T\rangle_{\rightarrow})$.

■

The maximality result follows below in Theorem 3. If k' is a deterministic simulant of k and k is unfolded up to the applicable reach of a formula f, then synthesis produces at least one result m such that $k' \preceq m$. Note that this result indicates Pareto-optimality [5].

Theorem 3. For each $f \in \mathcal{F}$, $k' \in \mathcal{K}_{\rightsquigarrow}$ and $k \in \mathcal{K}_{\rightarrow}$ with $k' \vDash f$, $k' \preceq k$, $unf(k, size(f))$ and $det(k')$ there exists an $m \in C(k, f)$ such that $k' \preceq m$.

Proof: This property is shown by structural induction on f. The first case to consider is when $f \equiv tt$, in which case we may choose $m = k$ as a witness. Clearly, $k \in C(k, tt)$ according to rule **1** in Definition 8 while $k' \preceq k$ was already assumed. We may omit the case for $f \equiv ff$ because $k' \nvDash ff$ and continue with the case for $f \equiv p$, for some $p \in \mathcal{P}$. Let x be the initial state of k. As $k' \vDash p$ we have $p \in L(x)$ by simulation (see Definition 2). We therefore have $m = k$ as a valid witness according to rules **2** and **3** while $k' \preceq k$ was already assumed. We may follow precisely the same reasoning for $f \equiv \neg p$. Since $k' \vDash \neg p$ and $k' \preceq k$, we have $k \vDash \neg p$, and $m = k$ as a valid witness by rules **4** and **5**.

The proof for $f \equiv f_1 \wedge f_2$ is less trivial. Assume that $k' \vDash f_1 \wedge f_2$ for a $k' \preceq k$ such that $unf(k, size(f_1 \wedge f_2))$ and $det(k')$. We have the following induction hypotheses:

$$\forall k', k.\ k' \preceq k \wedge k' \vDash f_1 \wedge unf(k, size(f_1)) \wedge det(k') \Rightarrow \exists m \in C(k, f_1) \wedge k' \preceq m \ \text{(IHf1)}$$

$$\forall k', k.\ k' \preceq k \wedge k' \vDash f_2 \wedge unf(k, size(f_2)) \wedge det(k') \Rightarrow \exists m \in C(k, f_2) \wedge k' \preceq m \ \text{(IHf2)}$$

As $k' \vDash f_1 \wedge f_2$, we clearly have $k' \vDash f_1$ and $k' \vDash f_2$. By observation of Definition 6 it is clear that $unf(k, size(f_1))$ as well as $unf(k, size(f_2))$. We proceed by induction on $count(k)$. If $count(k) = 0$, then we apply IHf1 and IHf2 to obtain $m_1 \in C(k, f_1)$ and $m_2 \in C(k, f_2)$ such that

$k' \preceq m_1$ and $k' \preceq m_2$. For this case, Lemma 4 directly gives $k = m_1 = m_2$ because otherwise $count(m_1) < 0$ or $count(m_2) < 0$. Since we now have $k \in C(k, f_1)$ and $k \in C(k, f_2)$, application of rule **6** gives $k \in C(k, f_1 \wedge f_2)$. Since $k' \preceq k$, this closes the case for $count(k) = 0$. We have the following induction hypothesis:

$$\forall p.\ count(p) < count(k) \wedge unf(p, size(f_1 \wedge f_2)) \wedge k' \preceq p \Rightarrow \exists m \in C(p, f_1 \wedge f_2) \wedge k' \preceq m$$

We clearly need to work towards an application of rule **7**. By application of IHf1 and IHf2 we obtain an $m_1 \in C(k, f_1)$ and $m_2 \in C(m_1, f_2)$. Following Lemma 4, we distinguish between several cases. If $k = m_1 = m_2$, then we immediately have $k \in C(k, f_1 \wedge f_2)$ as illustrated previously for $count(k) = 0$. If $k \in C(k, f_1)$ and $m_2 \in C(k, f_2)$ for $count(m_2) < count(k)$, then we may apply the induction hypothesis for $count(k)$ on m_2. The induction premise $unf(m_2, size(f_1 \wedge f_2))$ is satisfied by Lemma 2. By induction $m \in C(m_2, f_1 \wedge f_2)$ and $k' \preceq m_2$, and by applying rule **7** twice this gives us $m \in C(k, f_1 \wedge f_2)$. The remaining case to consider is when $m_1 \in C(k, f_1)$ and $m_2 \in C(m_1, f_2)$ such that $count(m_1) < count(k)$. Clearly we have $count(m_2) < count(k)$ by Lemma 4. Application of the induction hypothesis for $count(k)$ and using rule **7** twice gives an $m \in C(k, f_1 \wedge f_2)$ such that $k' \preceq m$.

For $f \equiv f_1 \vee f_2$ we only consider the case for $k' \vDash f_1$ as the situation for $k' \vDash f_2$ is very similar. By induction and because $unf(k, size(f_1))$ we have an $m \in C(k, f_1)$ such that $k' \preceq m$. Application of rule **8** immediately gives $m \in C(k, f_1 \vee f_2)$.

If $f \equiv [e]f'$ such that $k' \preceq k$, $k' \vDash [e]f'$, $unf(k, 1 + size(f'))$ and $det(k')$ we have the following induction hypothesis:

$$\forall k', k.\ k' \preceq k \wedge k' \vDash f' \wedge unf(k, size(f')) \wedge det(k') \Rightarrow \exists m \in C(k, f') \wedge k' \preceq m$$

As $unf(k, 1 + size(f'))$ we only need to consider $k = \langle x, T\rangle_\rightarrow$. We construct the set U as the *least* set according to the following definitions.

- If $(e', p) \in T$ for $e' \neq e$, then $(e', p) \in U$.
- If $(e, p) \in T$ and $C(p, f') = \emptyset$, then $(e, p) \notin U$.
- If $(e, p) \in T$ and $k' \xrightarrow{e} k''$ such that $k'' \preceq p$, then we apply the induction hypothesis for f' to obtain an $m \in C(p, f')$ such that $k'' \preceq m$. For this case we have $(e, m) \in U$.
- If $(e, p) \in T$ with no corresponding simulant in k', then we may choose an arbitrary $m \in C(p, f')$ for which we add $(e, m) \in U$.

It is clear that $k' \preceq \langle x, U\rangle_\rightarrow$. What remains to be shown is whether $\langle x, U\rangle_\rightarrow \in C(\langle x, T\rangle_\rightarrow, [e]f')$. We show this property by induction towards the number of elements in T. Clearly $\langle x, \emptyset\rangle_\rightarrow \in C(\langle x, \emptyset\rangle_\rightarrow, [e]f')$ according to rule **10**. Let $\langle x, T'\rangle_\rightarrow \in C(\langle x, T\rangle_\rightarrow, [e]f')$ such that $U = \{(e', p)\} \cup T'$. If $e' \neq e$, then $\langle x, U\rangle_\rightarrow \in C(\langle x, \{(e', p)\} \cup T\rangle_\rightarrow, [e]f')$ by rule **11**. If $e' = e$, then we either apply rule **12**, if $C(p, f') = \emptyset$, to obtain $\langle x, U\rangle_\rightarrow \in C(\langle x, \{(e, p)\} \cup T, [e]f')$, or rule **13**, in case $C(p, f') \neq \emptyset$.

If $f \equiv \langle e\rangle f'$ such that $k' \preceq k$, $k' \vDash \langle e\rangle f'$, $unf(k, 1 + size(f'))$ and $det(k')$ we have the following induction hypothesis:

$$\forall k', k.\ k' \preceq k \wedge k' \vDash f' \wedge unf(k, size(f')) \wedge det(k') \Rightarrow \exists m \in C(k, f') \wedge k' \preceq m$$

Since $unf(k, 1 + size(f'))$ it is clear that $k = \langle x, T\rangle_\rightarrow$ according to Definition 6. Since $k' \vDash \langle e\rangle f'$ there exists a $k'' \in \mathcal{K}$ such that $k' \xrightarrow{e} k''$ and $k'' \vDash f'$. Since $k' \preceq k$ there exists a $p \in \mathcal{K}$ such that $(e, p) \in T$ and $k'' \preceq p$. Clearly we have $det(k'')$ and $unf(p, size(f'))$ so we apply the induction hypothesis, resulting in an $m \in C(p, f')$ such that $k'' \preceq m$. Take $T' = (T \backslash \{(e, p)\}) \cup \{(e, m)\}$. We show that $\langle x, T\rangle_\rightarrow \in C(k, \langle e\rangle f')$ and $k' \preceq \langle x, T'\rangle_\rightarrow$. As there exists a $U \subseteq \mathcal{E} \times \mathcal{K}$ such that $T' = \{(e, m)\} \cup U$ and $T = \{(e, p)\} \cup U$ we can apply rule **14** on $\langle x, \{(e, m)\} \cup U\rangle_\rightarrow \in C(\langle x, \{(e, p)\} \cup U\rangle_\rightarrow, [e]f')$. What remains to be shown is whether $k' \preceq \langle x, T'\rangle_\rightarrow$. As $k' \preceq \langle x, T\rangle_\rightarrow$ and $k' \xrightarrow{e} k''$ is the only e-step in k' (because of $det(k')$) we clearly have $k' \preceq \langle x, T'\rangle_\rightarrow$ because $k'' \preceq m$. ■

VII. Conclusions

In this paper we studied the synthesis on Kripke-structures with labelled transitions, with respect to formulas in Hennessy-Milner Logic. A bisimilarity preserving transformation has been defined to transform an LTS into an equivalent recursive structure that is able to capture inherent unfolding. Upon this structure, operational rules define modifications in order to satisfy a given HML-formula. Results in the synthesized set are shown to be valid in terms of satisfiability and simulation by the original input LTS. A maximal solution for all deterministic simulants is contained within this set. Various examples show that this problem is far from trivial and most operators in HML require a dedicated approach.

Hennessy-Milner Logic is limited in its ability to specify all types of requirements. Therefore, we will seek to extend this logic in future work by studying the synthesis problem for additional operators. Notably, an invariant operator that can be used to test whether a property holds for every state might be useful to investigate. It should be noted that due to the nature of the unfolding approach as presented in this paper, it might be troublesome to extend the synthesis function C in its current form in such a way that it can handle an invariant operator. Therefore, a new insight will be required to handle formulas under invariance. A different aspect is related to controllability. Earlier work on synthesis divides the set of actions into a controllable and uncontrollable part. We plan to include this distinction in future work on synthesis. We will also attempt to solve the

open problem concerning a maximal synthesis up to nondeterministic simulation.

REFERENCES

[1] M. Antoniotti and B. Mishra. The supervisor synthesis problem for unrestricted CTL is NP-complete. Technical Report TR-95-062, International Computer Science Institute, Berkeley, 1995.

[2] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems*, pages 1–20. Springer, 1994.

[3] M. Barveau, F. Kabanza, and R. St.-Denis. A method for the synthesis of controllers to handle safety, liveness, and real-time constraints. *IEEE Transactions on Automatic Control*, 43(11):1543–1559, 1998.

[4] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a model based planner. In *Proceedings of the IJCAI01 Workshop on Planning under Uncertainty and Incomplete Information*, pages 54–71. Springer, 2001.

[5] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2006.

[6] A. Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35. Mittag-Leffler, 1963.

[7] A. Cimatti, M. Roveri, and P. Bertoli. Searching powerset automata by combining explicit-state and symbolic model checking. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 313–327. Springer, 2001.

[8] E. Clarke and A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs*, pages 52–71. Springer, 1982.

[9] A.R. Deshpande and P. Varaiya. Semantic tableau for control of PLTL formulae. In *Proceedings of the 35th IEEE Conference on Decision and Control*, pages 2243–2248. Elsevier, 1996.

[10] A. Emerson. Automata, tableaux, and temporal logics. In *Logics of Programs*, pages 79–88. Springer, 1985.

[11] S. Forschelen, J. De Mortel-Fronczak, R. Su, J.E. Rooda, et al. Application of supervisory control theory to theme park vehicles. In *Discrete Event Systems*, pages 293–299. Springer, 2010.

[12] A. Gromyko, M. Pistore, and P. Traverso. A tool for controller synthesis via symbolic model checking. In *8th International Workshop on Discrete Event Systems*, pages 475–476. Kluwer, 2006.

[13] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[14] S. Jiang and R. Kumar. Supervisory control of discrete event systems with CTL* temporal logic specifications. *SIAM Journal of Control and Optimization*, 44(6):2079–2103, 2006.

[15] R. Kumar, S. Jiang, C. Zhou, and W. Qiu. Polynomial synthesis of supervisor for partially observed discrete-event systems by allowing nondeterminism in control. *IEEE Transactions on Automatic Control*, 50(4):463–475, 2005.

[16] O. Kupferman and M. Vardi. μ-calculus synthesis. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*, pages 497–507. Springer, 2000.

[17] G. Lüttgen and W. Vogler. Safe reasoning with logic LTS. In *Software Seminar*, pages 376–387. Springer, 2009.

[18] J. Markovski, D. van Beek, R. Theunissen, K. Jacobs, and J. Rooda. A state-based framework for supervisory control synthesis and verification. In *Proceedings of the 49th IEEE Conference on Decision and Control*, pages 3481–3486. IEEE, 2010.

[19] M. Müller-Olm, D. Schmidt, and B. Steffen. Model-checking: A tutorial introduction. *Static Analysis*, pages 848–848, 1999.

[20] D. Nadales Agut, D. van Beek, and J. Rooda. Syntax and semantics of the compositional interchange format for hybrid systems. *The Journal of Logic and Algebraic Programming*, 82(1):1–52, 2013.

[21] J. Ostroff. Synthesis of controllers for real-time discrete event systems. In *Proceedings of the 28th IEEE Conference on Decision and Control, 1989.*, pages 138–144. IEEE, 1989.

[22] K. Åkesson, M. Fabian, H. Flordal, and A. Vahidi. Supremica – a tool for verification and synthesis of discrete event supervisors. In *Proceedings of the 11th Mediterranean Conference on Control and Automation*, pages 83–104. IEEE, 2003.

[23] P. Ramadge and W. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control Optimimization*, 25(1):206–230, 1987.

[24] K.T. Seow, M Gai, and T.L. Lim. A temporal logic specification interface for automata-theoretic finitary control synthesis. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 565–571. IEEE, 2005.

[25] O. Sokolsky and S. Smolka. Incremental model checking in the modal mu-calculus. In *Computer Aided Verification*, pages 351–363. Springer, 1994.

[26] R.J. van Glabbeek. The linear time branching time spectrum II. In *International Conference on Concurrency Theory*, pages 66–81. Springer, 1993.

[27] M. Vardi. An automata-theoretic approach to linear temporal logic. In *Proceedings of the VIII Banff Higher order workshop conference on Logics for concurrency*, pages 238–266. Springer, 1996.

[28] H. Wong-Toi and D. Dill. Synthesizing processes and schedulers from temporal specifications. In *Computer Aided Verification*, pages 272–281. Springer, 1991.

[29] Z. Zhang and W. Wonham. STCT: An efficient algorithm for supervisory control design. In *Symposium on supervisory control of discrete event systems*, pages 249–258. Springer, 2001.