Chapter 11
Learning Safe Interactions and Full-Control

Guillaume Maudoux, Charles Pecheur and Sébastien Combéfis

Abstract This chapter is concerned with the problem of learning how to interact safely with complex automated systems. With large systems, human–machine interaction errors like automation surprises are more likely to happen. Full-control mental models are formal system abstractions embedding the required information to completely control a system and avoid interaction surprises. They represent the internal system understanding that should be achieved by perfect operators. However, this concept provides no information about how operators should reach that level of competence. This work investigates the problem of splitting the teaching of full-control mental models into smaller independent learning units. These units each allow to control a subset of the system and can be learned incrementally to control more and more features of the system. This chapter explains how to formalize the learning process based on an operator that merges mental models. On that basis, we show how to generate a set of learning units with the required properties.

11.1 Introduction

The field of human–computer interaction (HCI) studies how humans interact with automated systems and seeks to improve the quality of these interactions. In HCI, formal methods allow to unambiguously describe behaviours of both humans and interactive systems. Given a formal description of humans and/or computer systems, it is possible to mechanically check properties on their interactions such as the *control* property, which we will describe later on. The automated verification of properties is called model checking. Model checking techniques can also be used to generate

G. Maudoux (✉) · C. Pecheur
Université catholique de Louvain, Louvain-la-Neuve, Belgium
e-mail: guillaume.maudoux@uclouvain.be

C. Pecheur
e-mail: charles.pecheur@uclouvain.be

S. Combéfis
École Centrale des Arts et Métiers, Brussels, Belgium
e-mail: s.combefis@ecam.be

B. Weyers et al. (eds.), *The Handbook of Formal Methods in Human-Computer Interaction*, Human-Computer Interaction Series,
DOI 10.1007/978-3-319-51838-1_11

models that satisfy desirable properties. In this chapter, we will use that capability to produce formal descriptions of the knowledge required to interact safely with a formally defined system.

Safe interactions are achieved when a system behaves in accordance with the user expectations. Any discrepancy between these expectations and the actual behaviour of the system is called an *automation surprise* (Sarter et al. 1997; Palmer 1995). To formalize our analysis of automation surprises, two models are used: one model describes the behaviour of the system and the other describes how the system is assumed to behave from the point of view of the user. The latter is called a *mental model* (Rushby 2002). A user can build a mental model of a system from training, by experimenting with the system or by reusing knowledge applicable to other similar systems. In this work, a user is assumed to behave according to her mental model. This assumption does not always hold as users may get distracted or make errors when manipulating a system, but we are not considering these kinds of faults here.

The *control* property describes the relation between a mental model and a system such that their interactions cannot lead to surprises. It is important to ensure that the mental model of the user allows control of the system in use. Or, less formally, that the user knows enough about the features she uses to avoid any surprise. To interact safely with a system, a user need not know all of its features. With most systems, knowing nothing about their behaviour is sufficient to never get surprised. These devices remain idle or turned off while waiting for a human to initiate the interaction. In the case of a plane at rest, for example, not interacting with it ensures that it will not behave unexpectedly. Some systems however require some knowledge in order to avoid surprises. Phones are an excellent example, as one could get surprised by an incoming ringtone if not warned that it may happen.

Generating mental models that ensure proper control provides a mean to train operators of critical systems. If the learning process ensures that their mental model controls the system, we can avoid automation surprises. In particular, new system features should be taught in such a way that the new mental model of the operators still ensures proper control of the system. This also means that operators must not interact with the system until the end of a learning phase and cannot perform their duties during that period.

Operators that have learnt all the possible behaviours of a system have built a full-control mental model. Such models have been defined by Combéfis and Pecheur (2009), and techniques to build minimal ones have been described in Combéfis and Pecheur (2011a, b). These mental models allow users to safely control all the features of a system. However, teaching a full-control mental model is impractical as it is equivalent to teaching all the features of the system at once, in one big step. For example, newly hired operators would be useless until they master the full complexity of the system. Large systems might even be too complex for one operator to manage. In that case, the operation of the system must be split in tasks dedicated to different operators and the generated full-control mental model cannot be used at all.

To be practical, learning processes should provide a set of learning units in the form of small, compatible mental models that can be combined incrementally into bigger models. Each intermediate mental model should ensure safe interactions with the system without necessarily describing all of its features. Learned sequentially, these units should increase the mental model of the operator until her mental model reaches full-control over the system. We will see that it is possible to decompose a full-control mental model into learning units units with such properties.

In this chapter, we first summarize the necessary concepts. Section 11.2 will describe *Labelled Transition Systems for Human Machine Interactions* (HMI-LTSs) as a specialization of *Labelled Transition Systems* (LTSs) and use these to formally define mental models, control, and full-control. In Sect. 11.3, we introduce a *merge* operator that describes the mental model resulting of learning two other mental models, and we argue that it is coherent with the intuition of learning a system. In Sect. 11.4, we explore the decomposition induced by the *merge* operator on HMI-LTSs. We show that some HMI-LTSs are too basic to be worth further decomposing, and that decomposing mental models into such basic elements is a finite process. We also provide an algorithm to automate this decomposition. Finally, we demonstrate that it is possible to generate a set of learning units with the desired properties by decomposing full-control mental models. This is presented with many examples in Sect. 11.5.

11.2 Background

In this section, we define HMI-LTSs, mental models, and the (full-)control property. We also formally introduce full-control mental models, a concept that lies at the intersection of these three notions. A deeper discussion of the background information provided in this section can be found in Combéfis (2013).

To formalize systems and mental models, we use *Labelled Transition Systems for Human–Machine Interactions* (HMI-LTSs) which are slightly modified *Labelled Transition Systems* (LTSs). An LTS is a state transition system where each transition has a *label*, also called *action*. LTSs interact with their environment based on this set of actions. Additionally, LTSs can have an internal τ action that cannot be observed by the environment. The representations of three different LTSs are shown in Figs. 11.1 and 11.2.

Definition 1 (*Labelled Transition System*) A labelled transition system (LTS) is a tuple $\langle S, \mathscr{L}, s_0, \rightarrow\rangle$ where S is a finite set of states, $\mathscr{L}$ is a finite set of labels representing visible actions, $s_0 \in S$ is the initial state and $\rightarrow \subseteq S \times (\mathscr{L} \cup \{\tau\}) \times S$ is the transition relation, where $\tau \notin \mathscr{L}$ is the label for the internal action.

The executions of LTSs can be observed from the environment via traces. An *execution* of an LTS is a sequence of transitions $s_0 \overset{a_1}{\rightarrow} s_1 \dots s_{n-1} \overset{a_n}{\rightarrow} s_n$ where each $(s_{i-1}, a_i, s_i) \in \rightarrow$. It represents the system moving from state to state by firing transitions. A *trace* of an LTS is a sequence $\sigma = a_1, a_2, \dots, a_n \in \mathscr{L}^\omega$ such that there exists an execution $s_0 \xrightarrow{\tau^* a_1 \tau^*} s_1 \cdots s_{n-1} \xrightarrow{\tau^* a_n \tau^*} s_n$. The notation $s \xrightarrow{\tau^* a \tau^*} s'$ represents a transitionlabelled a preceded and followed by any number of invisible τ transitions. Its

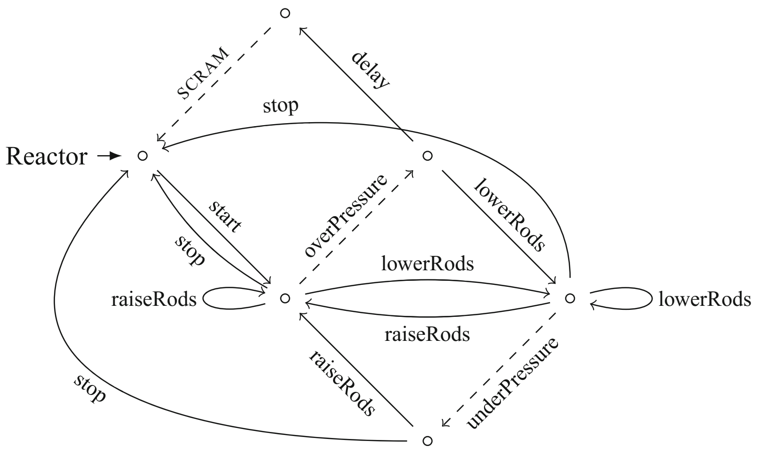

Fig. 11.1 HMI-LTS model of a reactor inspired from the nuclear power plant example (cf. Chap. 4). Observations are represented with a *dashed* edge. This model will be used as the main example through the remainder of this chapter

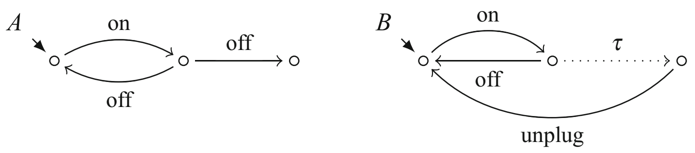

Fig. 11.2 Two examples of nondeterministic systems. *A* can be turned on then off at least once, but it is impossible to determine if it can be turned on again. *B* can be turned on and off, but it can also unobservably change to a state where the only way to restart it is to unplug it

only observable action is a. A trace is a sequence of visible actions that the system may produce in one of its executions. For example, the Reactor system of Fig. 11.1 can produce the trace “start, lowerRods, underPressure” and the trace “start, overPressure, delay, SCRAM” among infinitely many others.

To model interactions, we need to distinguish inputs from outputs. HMI-LTSs refine LTSs by distinguishing two kinds of actions, *commands* and *observations*. Like any I/O transition system, observations are uncontrollable outputs generated by the system and commands are controllable inputs. HMI-LTSs are very similar to LTS/IOs described by Tretmans (2008).

Definition 2 (*Human-Machine Interaction LTS*) A human-machine interaction labelled transition system (HMI-LTS) is a tuple $\langle S, \mathscr{L}^c, \mathscr{L}^o, s_0, \rightarrow\rangle$ where $\langle S, \mathscr{L}^c \cup \mathscr{L}^o, s_0, \rightarrow\rangle$ is a labelled transition system, $\mathscr{L}^c$ is a finite set of command labels and $\mathscr{L}^o$ is a finite set of observation labels. The two sets $\mathscr{L}^c$ and $\mathscr{L}^o$ are disjoint and the set of visible actions is $\mathscr{L} = \mathscr{L}^c \cup \mathscr{L}^o$.

HMI-LTSs are used to describe both systems and mental models. Mental models represent the knowledge an operator has about the system she controls. It is important to note that mental models do not represent the behaviour of a user, but the behaviour of a system as seen by a user. Therefore, a command in a mental model corresponds exactly to the same command on the system. The interactions between a system $\mathscr{S}$ and an operator behaving according to its mental model $\mathscr{M}$ are defined by the synchronous parallel composition $\mathscr{S} \parallel \mathscr{M}$. This distinguishes HMI-LTSs from LTS/IOs where inputs of the system must be synchronized on the outputs of the user and vice versa.

Notions on LTSs can be easily lifted to HMI-LTSs due to their high similarity. The set of states that can be reached from state s with an observable trace σ is represented as s **after** σ. This definition applies as is to LTSs and HMI-LTSs. We also use notations specific to HMI-LTSs. $A^c(s)$ (resp. $A^o(s)$) is the set of possible commands (resp. observations) of s. An action is possible in s if it is the first action of some trace starting at s.

An LTS is deterministic if $|s\,\textbf{after}\,\sigma| \leq 1$ for any σ. For example, the HMI-LTS A from Fig. 11.2 can be in two states after the trace "on, off" and is therefore not deterministic. Also, the HMI-LTS B has two possible actions in its middle state ('off' and 'unplug') because unplug can be the next visible action in executions firing the τ transition. It is of course also nondeterministic because after the "on" trace the system can be in two different states.

We want mental models to control systems without surprises. In particular, we want to avoid mental models that contain commands that are impossible on the system and to ignore observations that the system could produce. This motivates the introduction of the control property.

Definition 3 (*Control Property*) Given two HMI-LTSs $\mathscr{S} = \langle S_{\mathscr{S}}, \mathscr{L}^c, \mathscr{L}^o, s_{0\mathscr{S}}, \rightarrow_{\mathscr{S}} \rangle$ and $\mathscr{M} = \langle S_{\mathscr{M}}, \mathscr{L}^c, \mathscr{L}^o, s_{0\mathscr{M}}, \rightarrow_{\mathscr{M}} \rangle$, $\mathscr{M}$ controls $\mathscr{S}$ if $\mathscr{M}$ is deterministic and for all traces $\sigma \in \mathscr{L}^*$ such that $s_{\mathscr{S}} \in s_{0\mathscr{S}}$ **after** σ and $\{s_{\mathscr{M}}\} = s_{0\mathscr{M}}$ **after** σ:

$$A^c(s_{\mathscr{S}}) \supseteq A^c(s_{\mathscr{M}}) \text{ and } A^o(s_{\mathscr{S}}) \subseteq A^o(s_{\mathscr{M}}).$$

This definition is symmetric because it allows the mental model not to know the full set of available commands, and it also allows the system to produce fewer observations than expected by the mental model. From now on, this is the formal definition we refer to when we say that a mental model controls a system.

For a given system, there always exists a mental model that contains no commands and still allows control of the system. That mental model contains only the traces of observations available from the initial state and corresponds to the mental model needed by an agent to avoid surprises when not interacting with a system. Think back to the example of the phone given in the introduction. You need to know that your desk phone may ring even when you do not want to interact with it. Someone who ignores that fact will be surprised whenever the phone rings.

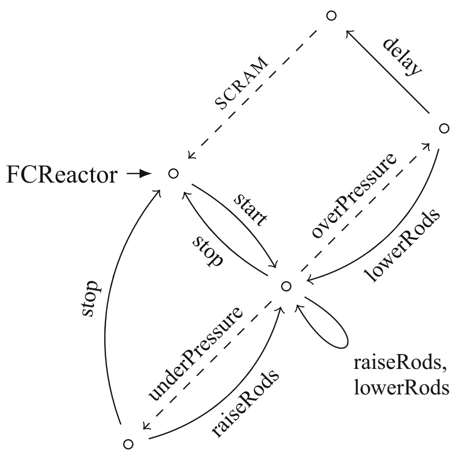

Fig. 11.3 The only minimal full-control mental model of the reactor system

We see that a mental model that controls a system does not necessarily explore the full range of possible behaviours of that system. When a mental model ensures control over a system and allows access to all the available commands of the system, we say that the model fully controls the system.

Definition 4 (*Full-Control Property*) Given two HMI-LTSs $\mathscr{S} = \langle S_{\mathscr{S}}, \mathscr{L}^c, \mathscr{L}^o, s_{0\mathscr{S}}, \rightarrow_{\mathscr{S}} \rangle$ and $\mathscr{M} = \langle S_{\mathscr{M}}, \mathscr{L}^c, \mathscr{L}^o, s_{0\mathscr{M}}, \rightarrow_{\mathscr{M}} \rangle$, $\mathscr{M}$ is a full-control mental model for $\mathscr{S}$, which is denoted $\mathscr{M}$ **fc** $\mathscr{S}$, if $\mathscr{M}$ is deterministic and for all traces $\sigma \in \mathscr{L}^*$ and for all $s_{\mathscr{S}} \in (s_{0\mathscr{S}}$ **after** $\sigma)$ we have

$$A^c(s_{\mathscr{S}}) = A^c(s_{\mathscr{M}}) \text{ and } A^o(s_{\mathscr{S}}) \subseteq A^o(s_{\mathscr{M}}).$$

where $\{s_{\mathscr{M}}\} = (s_{0\mathscr{M}}$ **after** $\sigma)$ is the only state reached in $\mathscr{M}$ by the trace σ.

A full-control mental model is therefore a deterministic HMI-LTS representing the required information for an operator to interact with a system to the full extent of its possibilities, and without surprises. Full-control mental models are minimal if they have a minimal number of states compared to other full-control mental models of the same system. Also, being *full-control deterministic* is a property shared by all the systems for which there exists a full-control mental model (Combéfis and Pecheur 2009). The property states that nondeterminism in the system does not impact controllability. Different algorithms exist to generate such models (Combéfis and Pecheur 2009; Combéfis et al. 2011a; Delp et al. 2013).

Figure 11.3 presents FCReactor, the only minimal full-control mental model of the Reactor system from Fig. 11.1. The two active states with no pressure warning have been merged as they are undistinguishable from a control point of view: they allow exactly the same commands, and their observations are compatible.

Minimal full-control mental models are important because they represent the minimal knowledge that a perfect operator should master. Compact training material and user guides should therefore describe a minimal full-control mental model.

11.3 Modelling the Learning Process with the Merge Operator

While minimal full-control mental models are perfect in terms of control, they are inefficient when training operators as they require to be completely mastered before using a system. But to optimize this process, we need to describe how the mental model of a user can be augmented with a learning unit. In this section, we define the new merge operator that combines two mental models into a broader one, and we claim that this operator is a natural way to encode the learning process.

The merge of two HMI-LTSs is obtained by superimposing their graphs and merging identical paths. This is a kind of lazy choice as the final behaviour does not commit to behave like the first or the second operand until a decision is required. The result may even alternate between the behaviour of its two operands. As the definition of the merge operator does not rely on observations and commands, it can easily be generalized to LTSs. An example of the action of the merge operator is given in Fig. 11.4.

Definition 5 (*Merge*)
The *merge* of two deterministic HMI-LTSs $A = \langle S_A, \mathscr{L}^c_A, \mathscr{L}^o_A, s_{0A}, \rightarrow_A \rangle$ and $B = \langle S_B, \mathscr{L}^c_B, \mathscr{L}^o_B, s_{0B}, \rightarrow_B \rangle$, denoted $A \oplus B$, is an HMI-LTS $\langle S, \mathscr{L}^c_A \cup \mathscr{L}^c_B, \mathscr{L}^o_A \cup \mathscr{L}^o_B, s_0, \rightarrow \rangle$ where

1. $S \in \mathscr{P}(S_A \uplus S_B)$ is the set partition defined by the equivalence relation $\sim$.
2. $\sim\, = \cup_{i=0}^{\infty} \sim_i$ is the equivalence relation on $S_A \uplus S_B$ such that
   a. s_{0A} and s_{0B} are the only equivalent nodes in $\sim_0$;
   b. $\sim_k \subset \sim_{k+1}$ and

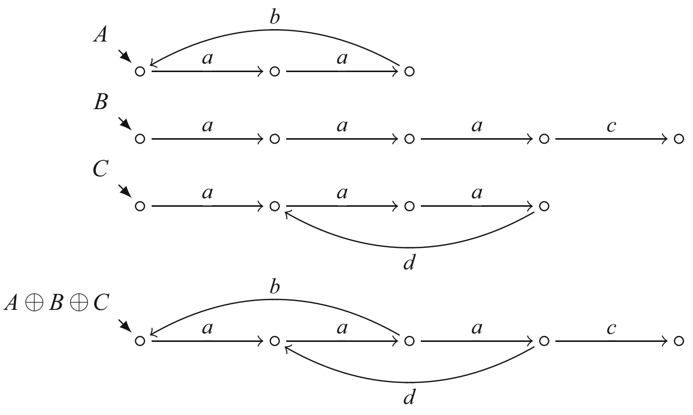

Fig. 11.4 Example of the merge operation on three HMI-LTSs

c. $\sim_{k+1}$ is the finest equivalence relation such that $m' \sim_{k+1} n'$ if there exists $m \sim_k n$ and a such that $m \xrightarrow{a}_A m'$ and $n \xrightarrow{a}_B n'$.

3. $\rightarrow$ is the set of transitions $[s]_\sim \xrightarrow{a} [s']_\sim$ such that either $s \xrightarrow{a}_A s'$ or $s \xrightarrow{a}_B s'$.

In this definition, S is always well defined. While the equivalence relation $\sim$ is an infinite union, we can see that the intermediate equivalence relations are monotonically increasing as they embed the previous one and can only add new equivalent states, not remove them. If no states are added at some step, then a fixpoint is reached. And this must happen within a finite number of steps as it must end when all the states are equivalent (in the worst case).

From this definition, we can also see that the merge of two deterministic HMI-LTS is unique. Were it not the case, there would be two different ways to build $\sim_{k+1}$ for some k. But each $\sim_{k+1}$ is uniquely defined with respect to $\sim_k$, and $\sim_0$ is uniquely defined. Therefore $\sim$ must be unique.

The example in Fig. 11.4 uses the fact that the merge operator is associative to write $A \oplus B \oplus C$ instead of $(A \oplus B) \oplus C$ or $A \oplus (B \oplus C)$. Associativity is tedious to prove because we need to show that $(A \oplus B) \oplus C$ is isomorphic to $A \oplus (B \oplus C)$. Instead, we draw attention to the extensibility of the definition to more than two models. It only requires minor adjustments to define the merge of three or even n HMI-LTSs. The operator is also commutative. This property may be assumed from the symmetry of the definition.

We can show that the result of merging two deterministic HMI-LTSs is deterministic. Indeed, as the two operands of the merge are deterministic, they cannot contain τ transitions and so their merge is free of τ transitions too. Neither can the result contain two transitions with the same label leaving the same state. Let us assume that the result contains two transitions outgoing from the same state, with the same label, and leading to different states. By the definition of $\rightarrow$, this means that there exists $(m, a, m') \in \rightarrow_A$ and $(n, a, n') \in \rightarrow_B$ such that $m \sim n$ and $m' \nsim n'$, which violates the recursive property on $\sim$. Also, the resulting HMI-LTS can contain no τ transitions and no fork where a transition with the same label leads to two different states. These two conditions are sufficient to prove that the merge is deterministic.

The HMI-LTS $A \oplus B$ can switch its behaviour from A to B provided A can reach a state that was merged with a state of B. This conversely holds from B to A. If the HMI-LTS can switch from A to B and from B to A, then it can alternate its behaviour arbitrarily often. We can see that this operator is different from the traditional choice operator because it is more than the union of the traces. It can build complex behaviours from two simple models. In Fig. 11.4, we can see that the trace $a, a, a, d, a, b, a, a, a, c$ is not possible on the different models but is valid on their merge.

The merge operator is useful because the set of traces of a merge is always larger than or equal to the union of the traces of the merged transition systems. This means that the possible behaviours of a merge can be richer than the union of the behaviours of its operands. This is needed to ensure that the decomposition of a big system is a small set of small systems. Indeed, if the behaviour of a merge was exactly the sum

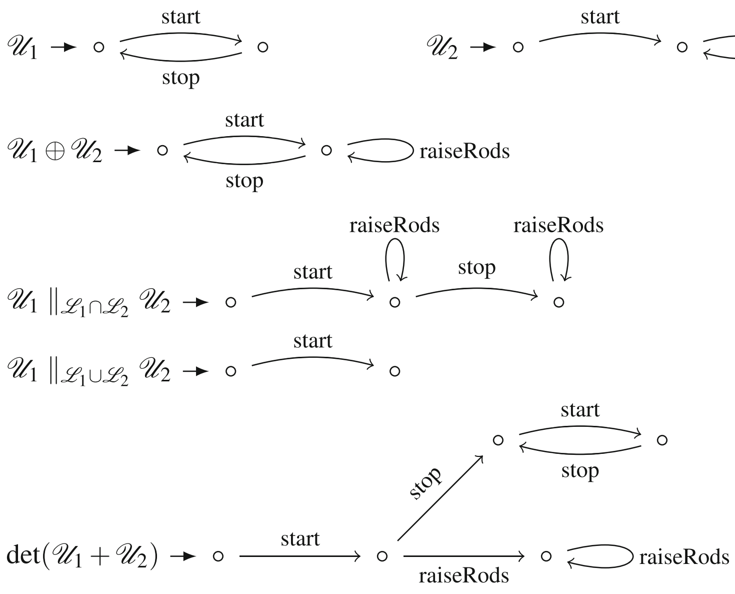
$\mathscr{U}_1$
start
stop
$\mathscr{U}_2$
start
raiseRods
$\mathscr{U}_1 \oplus \mathscr{U}_2$
start
stop
raiseRods
$\mathscr{U}_1 \parallel_{\mathscr{L}_1 \cap \mathscr{L}_2} \mathscr{U}_2$
start
raiseRods
stop
raiseRods
$\mathscr{U}_1 \parallel_{\mathscr{L}_1 \cup \mathscr{L}_2} \mathscr{U}_2$
start
$\det(\mathscr{U}_1 + \mathscr{U}_2)$
start
stop
start
stop
raiseRods
raiseRods

Fig. 11.5 The merge operation compared to the parallel synchronization (‖) and the choice (+) operators on (HMI-)LTSs. The result of the choice between $\mathscr{U}_1$ and $\mathscr{U}_2$ has been determinised for readability

of the behaviour of its operands, then learning a system would reduce to enumerating all the possible traces on it.

Figure 11.5 shows the effect of three operators on two simple models. We can see that the merged model can perform the actions "stop" and "raiseRods" infinitely often. This corresponds to the combined knowledge that the system can be started and stopped infinitely often, and that when started, the rods can be raised as many times as wished.

By comparison, the synchronous parallel composition requires the two operands to be executed in parallel, and to be synchronized on some actions. Usually, systems are synchronized on their common alphabet. This means that $\mathscr{U}_1 \parallel_{\mathscr{L}_1 \cap \mathscr{L}_2} \mathscr{U}_2$ cannot execute "start" more than once because $\mathscr{U}_2$ can only do it once. Unsynchronized actions like "raiseRods" can however fire at any time, even when the reactor is shut down. These two aspects are not desirable when modelling combined knowledge.

If we synchronize the two models on the unions of their alphabets, then $\mathscr{U}_1 \parallel_{\mathscr{L}_1 \cup \mathscr{L}_2} \mathscr{U}_2$ cannot perform any "stop" action because that action $\mathscr{U}_2$. This case is even worse for modelling knowledge increase as it removes existing known behaviours from their combined learning.

Finally, the choice operator does not allow to knowledge of multiple mental models to be combined either. It forms a new model that can use the knowledge of either operand, but that prevents any interaction between the two. This is like learning addition and multiplication but not being able to use both in the same calculation.

With these examples, we have shown that existing operators are inappropriate for our mathematical model of learning. The new merge operator remedies to these shortcomings. If two HMI-LTSs each represent some knowledge about a system, then their merge represents the combined knowledge of an operator who knows these two facts.

Furthermore, the merge operator is consistent with the interpretation of HMI-LTSs as scenarios. When a scenario loops, the system is assumed to have returned in a state strictly equivalent to the initial one. In particular, the scenario is assumed to be repeatable infinitely often unless explicitly stated. When a learning unit loops to a given state, it means that state is completely equivalent to the initial one for controllability purposes.

While there is no way to prove that the merge operator is perfect, we have provided examples and intuition on why it is a good way to encode how mental models grow during the learning of a system.

11.4 Basic Learning Units

Within our formal theory of knowledge and learning, we are now able to split a learning objective into smaller elements that can be learned independently. Decomposing a full-control mental model into independent elements amounts to find a set of mental models such that their merge by the merge operator is exactly that full-control mental model.

To define a good decomposition, we first need an order relation to tell if some sub-elements form a valid decomposition. We need our decomposition to split a model into elements that can be merged back into the original model, but we also need the elements to be smaller than the original model. Were it not the case, the decomposition would not be well-founded: systems could be decomposed forever into other systems. This goes against the idea of a decomposition into simpler elements.

In this section, we first explore the order induced by the merge operator on HMI-LTSs. We then show that this order is not well-founded and how an order based on the size of the graphs fixes it. Finally, we introduce *basic* learning units, small learning units that cannot be decomposed into smaller elements.

The merge operator naturally defines a partial order on the HMI-LTSs. The merge order is such that $A \leq_{\oplus} B$ if and only if $A \oplus B$ is isomorphic to B, which we denote $A \oplus B \simeq B$. The strict partial order relation also requires A to be different from B (i.e. not isomorphic to B). Figure 11.6 shows four HMI-LTSs ordered according to $<_{\oplus}$. The merge order captures the idea that B has more behaviours than A because $A \leq_{\oplus} B$ implies that $\text{Traces}(A) \subseteq \text{Traces}(B)$.

Furthermore, due to the definition of the merge order, the set of deterministic HMI-LTSs forms a join-semilattice: any two HMI-LTSs A and B are (upper) bounded by $A \oplus B$. In a such a lattice, the decomposition is performed by finding two elements strictly smaller than an HMI-LTS and such that their upper bound is exactly that HMI-LTS. However, this lattice has the undesirable property of allowing infinite

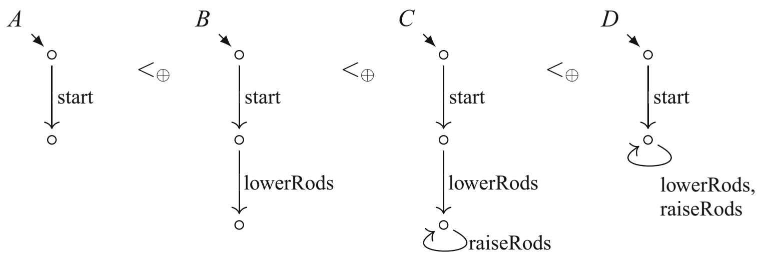

Fig. 11.6 Illustration of the merge-order relation on HMI-LTSs. For example, we have $C <_\oplus D$ because $C \oplus D = D$ and $C \neq D$

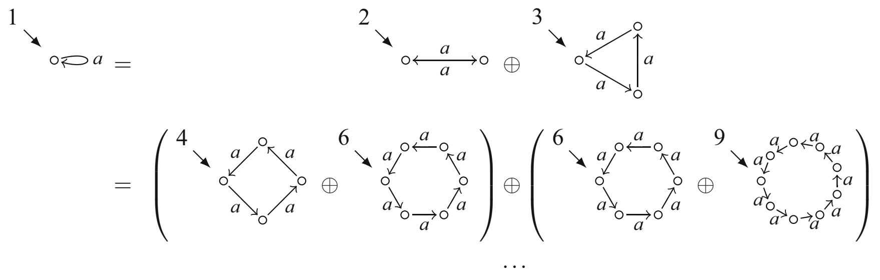

Fig. 11.7 A decomposition based on the merge order alone can lead to infinite decompositions into HMI-LTSs with increasing size

decomposition chains. As illustrated in Fig. 11.7, we can see that a simple loop can be decomposed into the merge of a 2- and a 3-loop, which can in turn be decomposed into the merge of a 4- and 6-loop, and a 6- and 9-loop, respectively, and so forth.

To encode the fact that a decomposition should produce simpler models than the model it comes from, we define the learning order. It restricts the merge order with the constraint that smaller models must have a lower number of states. Ties are broken based on the number of edges. This structural size order is denoted $\leq$.

Definition 6 (*Learning order*) A deterministic HMI-LTS A is said to be smaller than an HMI-LTS B according to the learning order if and only if $A \leq B$ and $A \leq_\oplus B$. This is denoted $A \leq_{\text{learn}} B$.

A decomposition will stop when we reach a model not worth splitting, which happens when we cannot find two strictly smaller models (in the sense of the learning order) that merge into the current model. We say that such HMI-LTSs are *basic*.

Definition 7 (*Basic HMI-LTS*) A deterministic HMI-LTS M is *basic* if there does not exist two HMI-LTSs A and B such that $A <_{\text{learn}} M$, $B <_{\text{learn}} M$ and $A \oplus B = M$.

It turns out that such basic HMI-LTS take the form of single sequences, single loops, lassos or tulips as shown in Fig. 11.8. Loops and sequences can be seen as

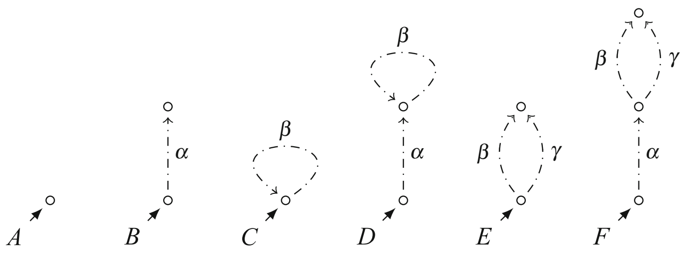

Fig. 11.8 Different shapes of basic HMI-LTSs. They can be A empty, B sequences, C loops, D lassos and E, F tulips with and without stem. *Dotted lines* represent any oriented sequence of states and transitions. All these shapes are degenerated tulips where action sequences α, β and γ can be empty

degenerated lassos with no stem or no loop. The fully degenerated lasso is the HMI-LTS with no transitions at all. Finally, a tulip is a branching HMI-LTS where the two branches reunite in the last state. Like lassos, they may have no stem. When they are comparable, lassos and tulips are always strictly greater than sequences. Lassos and tulips are never comparable.

Any finite deterministic HMI-LTS can be decomposed into a finite set of basic HMI-LTS. This arises from the fact that any HMI-LTS is the merge of a basic HMI-LTS and another HMI-LTS strictly smaller than the previous one. Were it not the case, that HMI-LTS would be basic itself. By induction on the remaining HMI-LTS, we show that it eventually reduces to a basic HMI-LTS after a finite number of basic HMI-LTS removal. All these basic elements form a set that we call the decomposition of the HMI-LTS. This algorithm is shown with the FCReactor model on Fig. 11.9 and formally defined hereafter.

Algorithm 1 formalizes the enumeration of the basic units forming the decomposition of a mental model. At line 5, this algorithm performs an unspecified exploration of the graph, edge by edge. It could be a depth-first search, a breadth-first search or any other exploration strategy. The *pre* mapping is used to build a spanning tree. The algorithm looks for edges that are outside the spanning tree and removes them permanently by calling the "extract_unit" procedure defined in Algorithm 2. For each such edge, a basic HMI-LTS is extracted. A special case is added in line 13 to handle states with only one adjacent edge. Such states are part of a basic sequence that does not contain edges outside of the spanning tree and a basic sequence must be extracted at that place. That way, all the paths in the graph are covered by some basic HMI-LTS, and basic HMI-LTSs are disjoint because they contain at least one edge that no other unit contains.

Algorithm 2 describes "extract_unit," which splits a model into a basic HMI-LTS and a smaller model by removing a given edge from the model. For correctness and efficiency, it also removes all the edges that are completely described by the extracted basic unit. That way, the resulting model $\mathscr{M}$' is the minimal (size-wise) model such that $\mathscr{M}' \oplus \mathscr{U} = \mathscr{M}$. Note that *pre* is guaranteed to be defined for the nodes used at line 6 because it forms a partial spanning tree spanning at least to x

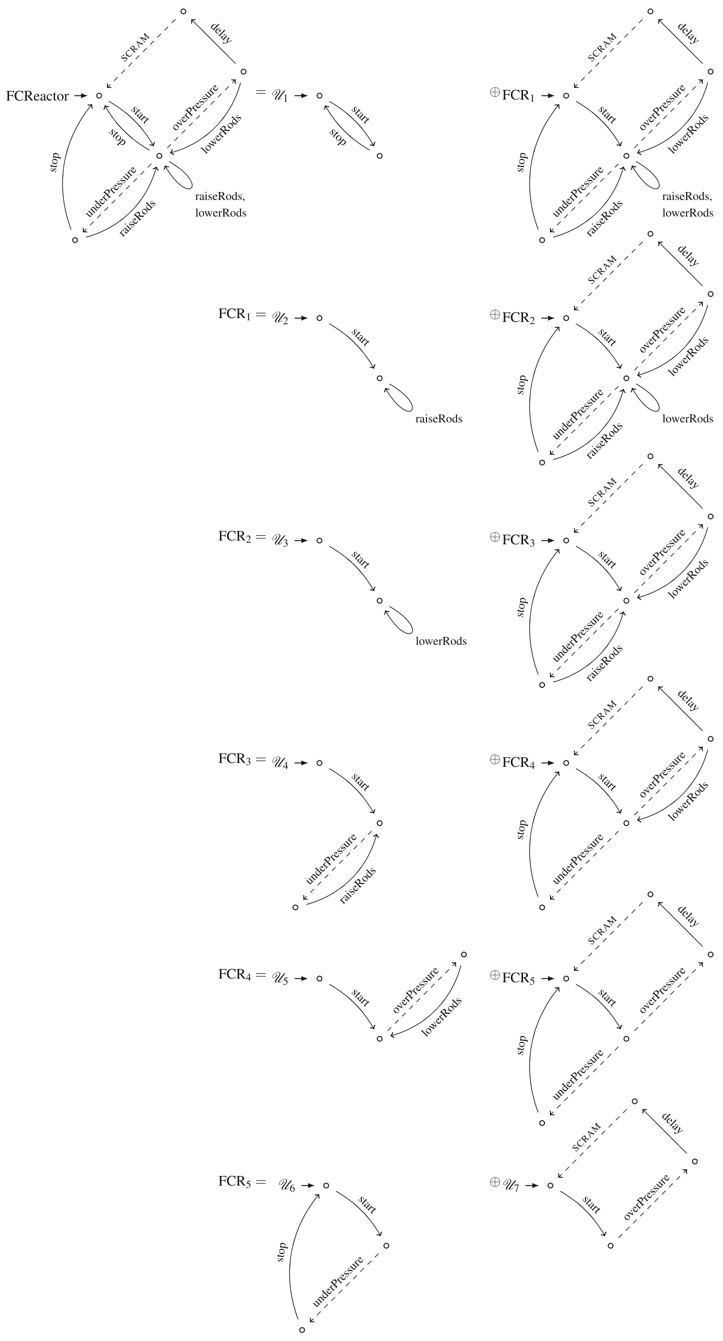

Fig. 11.9 Illustration of the decomposition algorithm, where basic HMI-LTSs are extracted one by one until there is nothing more to extract

and x' by construction in Algorithm 1. As “extract_unit” traverses that tree upwards from x and x', it cannot encounter undefined values of *pre*. The units extracted by the algorithm are subgraphs of the initial model. They contain a subset of the edges of that model and the corresponding subset of nodes.

Algorithm 1 Basic HMI-LTSs enumeration

Require: $\mathscr{M} = \langle S, \mathscr{L}^c, \mathscr{L}^o, s_0, \rightarrow \rangle$, a minimal deterministic HMI-LTS.
Ensure: $U = \{u_1, u_2, \ldots, u_n\}$, a minimal decomposition of $\mathscr{M}$ into basic HMI-LTSs.

```
 1: $U := \emptyset$                                          ▷ Initialise the decomposition with an empty set
 2: $E := \{(m, v, m') \in \rightarrow \mid m = s_0\}$                ▷ The set of edges yet to explore,
                                                              ▷ initialised with the edges starting at $s_0$
 3: $pre := [\bot, \ldots, \bot]$
 4: $pre[s_0] := \top$

 5: while $\exists(n, a, n') \in E \cap \rightarrow$ do
 6:     $E := E \setminus \{(n, a, n')\}$
 7:     if $pre[n'] \neq \bot$ then                                  ▷ Extract a tulip or a lasso
 8:         $(\mathscr{M}, u) :=$ extract_unit$(\mathscr{M}, pre, (n, a, n'))$
 9:         $U := U \cup \{u\}$
10:     else
11:         $pre[n'] := (n, a, n')$
12:         $E := E \cup \{(m, v, m') \in \rightarrow \mid m = n'\}$
13:         if $\deg(n', \rightarrow) = 1$ then                       ▷ Extract a single path
14:             $(\mathscr{M}, u) :=$ extract_unit$(\mathscr{M}, pre, (n, a, n'))$
15:             $U := U \cup \{u\}$
16:         end if
17:     end if
18: end while
19: return $U$
```

A decomposition is nonredundant if it does not contain two comparable elements. If it was the case, one of these elements could be further decomposed. The decomposition algorithm sketched in Fig. 11.9 always produces a nonredundant decomposition because each basic HMI-LTS contains actions that were not part of the previously removed basic elements, and that are removed with it. For example, the decomposition of the FCReactor system into $\{\mathscr{U}_1, \mathscr{U}_2, \mathscr{U}_3, \mathscr{U}_4, \mathscr{U}_5, \mathscr{U}_6, \mathscr{U}_7\}$ as shown in Fig. 11.9 is nonredundant.

A decomposition is minimal if no other decomposition of the same HMI-LTS contains fewer basic elements. The size of a minimal decomposition is called the *complexity* of an HMI-LTS. Minimal decompositions of the FCReactor system contain exactly seven elements, so the complexity of that model is 7.

We now know how to decompose an HMI-LTS into basic elements, and that decomposition gives us a measure of the complexity of that HMI-LTS.

Algorithm 2 Basic units extraction

```
Require: $\mathscr{M}$, a minimal deterministic HMI-LTS;
Require: $pre : S \to S \times (\mathscr{L}^c \cup \mathscr{L}^o) \times S : n \mapsto pre[n]$, mapping nodes to preceding edges;
Require: $(n, a, n')$ an edge to extract.
Ensure: $\mathscr{U}$, a basic HMI-LTS;
Ensure: $\mathscr{M}'$, an HMI-LTS strictly smaller than $\mathscr{M}$ and such that $u \oplus \mathscr{M}' = \mathscr{M}$.

  function EXTRACT_UNIT($\mathscr{M} = \langle S, \mathscr{L}^c, \mathscr{L}^o, s_0, \to\rangle, pre, (n, a, n')$)
      $(\to_{\mathscr{U}}) := \{(n, a, n')\}$
      $(\to) := (\to) \setminus \{(n, a, n')\}$

      for $x := n, n'$ do
          while $pre[x] \neq \top$ do
              $(x, v, x') := pre[x]$
              $(\to_{\mathscr{U}}) := (\to_{\mathscr{U}}) \cup \{(x, v, x')\}$
              if $\deg(x', \to) = 1$ then
                  $(\to) := (\to) \setminus \{(x, v, x')\}$
              end if
          end while
      end for

      $S_{\mathscr{U}} := \{n \in S \mid \exists v, x. (n, v, x) \in \to_{\mathscr{U}} \vee (x, v, n) \in \to_{\mathscr{U}}\}$
      $\mathscr{U} := \langle S_u, \mathscr{L}^c, \mathscr{L}^o, s_0, \to_{\mathscr{U}}\rangle$
      $S' := \{s_0\} \cup \{n \in S \mid \exists v, x. (n, v, x) \in \to \vee (x, v, n) \in \to\}$
      $\mathscr{M}' := \langle S', \mathscr{L}^c, \mathscr{L}^o, s_0, \to\rangle$
      return $(\mathscr{M}', u)$
  end function
```

11.5 How to Teach Full-Control

In this section, we show that it is possible to build a set of learning units such that each unit controls a given model, and such that all units can be combined into a full-control mental model.

The main idea is to decompose a full-control mental model of the system into basic subgraphs. It appears that basic subgraphs can be completed to form learning units that can control the system. This means that the completed basic subgraphs of a full-control mental model of a system form a set of independent, compatible mental models that can be merged to reproduce the behaviour of the full-control mental model.

Given two deterministic subgraphs $\mathscr{U}$ and $\mathscr{U}'$ of a deterministic HMI-LTS $\mathscr{M}$, we have the property that their merge $\mathscr{U} \oplus \mathscr{U}'$ is isomorphic to a subgraph of $\mathscr{M}$. This can be seen from the fact that $\oplus$ merges states that can be reached with the same traces, and that these states must correspond to exactly one state of $\mathscr{M}$, as $\mathscr{M}$ is deterministic.

Starting from a full-control mental model $\mathscr{M}$ of a system $\mathscr{S}$, we can decompose it into a set of basic HMI-LTSs. However, these basic HMI-LTSs do not necessarily control $\mathscr{S}$. To achieve this property, they need to be completed with respect to observations. This is sufficient because a mental model that controls a system must accept all the observations of that system but is allowed to ignore commands.

If we call $\rightarrow_S^o$ the transition relation of $\mathscr{S}$ restricted to observations, then any subgraph of a full-control mental model can be completed with $\rightarrow_S^o$ in order to control $\mathscr{S}$. Of course, only the connected component reachable from the initial state should be kept after the completion. In particular, any *basic* subgraph of a mental full-control mental model can be completed in order to control $\mathscr{S}$.

Definition 8 (*Observation Completion*)
Given an HMI-LTS $\mathscr{M} = \langle S, \mathscr{L}^c, \mathscr{L}^o, s_0, \rightarrow^c \cup \rightarrow^o \rangle$ and one subgraph $\mathscr{U} = \langle S_{\mathscr{U}}, \mathscr{L}^c, \mathscr{L}^o, s_0, \rightarrow_{\mathscr{U}} \rangle$ of $\mathscr{M}$, the *observation completion* of $\mathscr{U}$ is an HMI-LTS $\mathscr{U}'$ such that $\mathscr{U}'$ is the connected component of $\langle S, \mathscr{L}^c, \mathscr{L}^o, s_0, \rightarrow_{\mathscr{U}} \cup \rightarrow^o \rangle$ reachable from s_0.

Figure 11.10 shows the completion of the basic HMI-LTSs from Fig. 11.9. Each model is completed with reachable observations from FCReactor. The interpretation

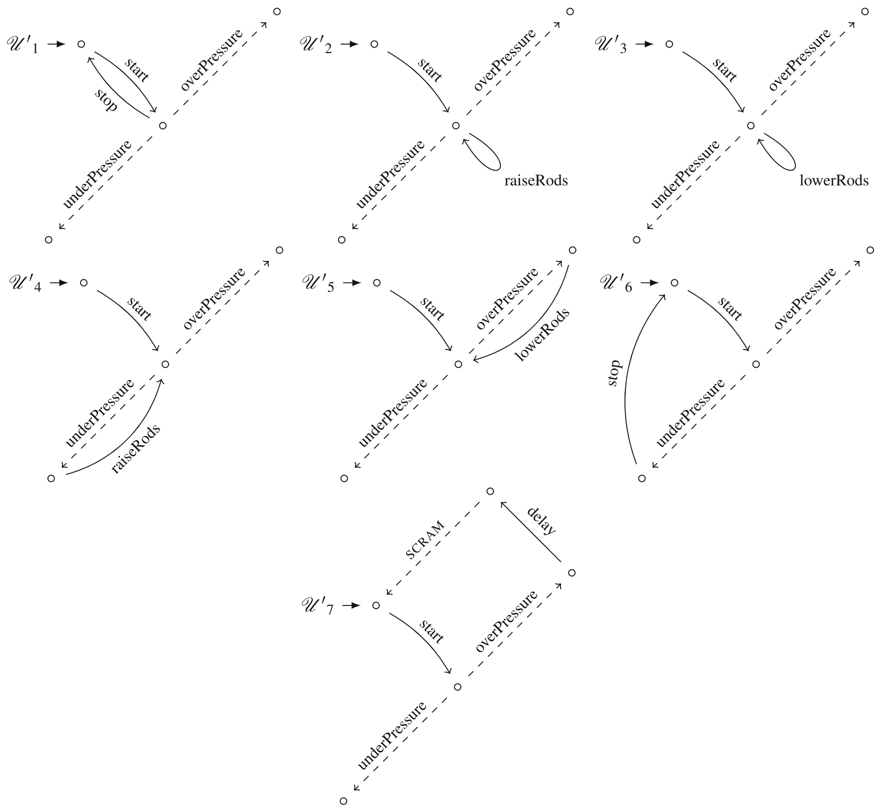

Fig. 11.10 The observation completion of the basic HMI-LTSs from Fig. 11.9 with respect to the only minimal full-control mental model of reactor, which is FCReactor

of $\mathscr{U}_1'$ is that the reactor can be turned on and off again, but that the operator needs to know that, when turned on, the reactor may emit warnings about the pressure. If a pressure warning event happens, the operator will not be surprised. She would however be unable to further operate the system. To unblock the situation, she could, for example, read the user manual to improve her knowledge of the system or ask a more experienced user.

The astute reader will have noticed that users need not to know about the delay transition. This is because the delay has been modelled as a command. If modelled as the observation that some time has elapsed, then no operator would be able to prevent the system to enter SCRAM. More advanced techniques need to be used to model passing time as a partially (un)controllable event on HMI-LTSs.

The observation completion of any subgraph of a full-control mental model $\mathscr{M}$ controls the intended system. Indeed, such a completed subgraph cannot prevent observations from occurring as the full-control mental model does not, and the completed graph has all the observations from the system. In particular, the observation completion of basic subgraphs of full-control mental models of a system $\mathscr{S}$ control that system $\mathscr{S}$. These elements also have the nice property of merging into completed subgraphs of $\mathscr{M}$ that themselves have control over $\mathscr{S}$.

Definition 9 (*Basic Learning Unit*)
Given a full-control mental model $\mathscr{M} = \langle S, \mathscr{L}^c, \mathscr{L}^o, s_0, \rightarrow \rangle$ where $\rightarrow = \rightarrow^c_{\mathscr{M}} \cup \rightarrow^o_{\mathscr{M}}$, a *basic learning unit* is a mental model $\mathscr{U} = \langle S_{\mathscr{U}}, \mathscr{L}^c, \mathscr{L}^o, s_0, \rightarrow_T \rangle$ where $\rightarrow_T$ is the connected component of $\rightarrow^o_{\mathscr{M}} \cup \rightarrow_b$ containing s_0 and $\langle S_{\mathscr{U}}, \mathscr{L}^c, \mathscr{L}^o, s_0, \rightarrow_b \rangle$ is a basic subgraph of $\mathscr{M}$.

With this definition, we can state that any full-control, deterministic (fc-deterministic) system is fully controlled by the merge of a set of basic learning units.

Theorem 1 (Decomposition) *Any finite fc-deterministic HMI-LTS $\mathscr{S}$ can be decomposed into a finite set $T = \{\mathscr{U}_1, \mathscr{U}_2, \dots \mathscr{U}_n\}$ of basic learning units such that*

- *each $\mathscr{U}_i$ controls $\mathscr{S}$;*
- *for each subset $I \subset \{1, 2, \dots n\}$ of indices, the partial merge $\bigoplus_{i \in I} \mathscr{U}_i$ of elements of T controls $\mathscr{S}$; and*
- *the complete merge $\bigoplus_{i=1}^{n} \mathscr{U}_i$ has full-control over $\mathscr{S}$.*

Proof By definition, any fc-deterministic HMI-LTS $\mathscr{S}$ has at least one minimal full-control mental model $\mathscr{M}$. We have shown that such a full-control mental model can be decomposed into a finite set of basic learning units that are completed basic subgraphs. Because they are completed subgraphs, these elements and any partial merge of these elements have control over $\mathscr{S}$. As the elements are the completion of the decomposition of $\mathscr{M}$ into basic HMI-LTS, their full merge will be exactly $\mathscr{M}$, and therefore fully controls $\mathscr{S}$. This proves that there exists a decomposition of $\mathscr{S}$ meeting the required properties.

The decomposition of an fc-deterministic system into a set of basic learning units may not be unique. Indeed, there may exist more than one minimal full-control mental model, and each full-control mental model may have multiple decompositions into basic learning units. Nevertheless, the minimal number of basic units required to decompose a mental model gives a measure of its complexity. We define the learning complexity of a system as the size of the smallest set of basic learning units that can be merged into a full-control mental model of that system. This metric measures the number of small learning units that an operator needs to learn before being able to control all the features of the system. This is different from the complexity of a system as defined at the end of Sect. 11.3 because the observation completion of two different basic units may turn to be the same basic learning unit. This metric is different from both the number of states and the number of transitions, which are the most common measures of complexity for transition systems.

11.6 Related Work

The idea of generating user manuals from formal specifications has been widely explored. Thimbleby and Ladkin (1995, 1996) provided a way to derive a complete description of system features by enumerating the sequences of actions to reach each state. Based on a formal description of a fax machine in the Prolog language, they generated a complete user manual describing how to perform every possible action. The fax machine is described as a tree of possible commands annotated with the information displayed in each node. A skeleton user manual is then generated by enumerating traces to each state of the system. The trace is separated into a sequence of actions and another sequence of observations. Finally, a technical author is required to turn the skeleton into a natural language. By comparison, our work gives hints on which elements should be described. Where Thimbleby and Ladkin describe one trace to each state, we propose to split the learning into stand-alone compatible units.

More recently, Delp et al. (2013) have implemented a system to generate a complete description of a system's formal model. Spichkova et al. (2014) presented a tool to maintain and update the technical documentation of a system based on its formal model. All of these authors have focussed their efforts in generating a complete description of the system. This work derives learning units from an ideal mental model and therefore tries to teach users a good mental model and not the full system itself.

Interestingly, user tasks model has been used by Kieras and Polson (1985) to compute the user complexity of a system. Each task is formally defined as a generalized transition network, and the complexity is measured by the number of states, the number of keystroke (their system's commands) and other direct metrics on the transition network. Further investigations are required to compare these approaches to the learning complexity defined here.

In this work, we used HMI-LTSs to model both systems and user mental models. This formalism is the result of research performed by Combéfis et al. (2011a) to detect and avoid surprises during interaction with a system. These tools are then used to define and verify the full-control property. Many authors have provided a formal description of systems and verified properties on it, but few model the mental model of users explicitly. Among them are Blandford et al. (2011) and Buth (2004). Buth checks the trace equivalence of a system and a mental model to ensure that both agree on the possible sequences of events. In contrast, the full-control property does not require the system to accept all the observations expected by the user. Buth's works can be considered an extension of Rushby (2002), where the user and the system are modelled separately but properties are verified on the single model of their interactions. The framework described by Bolton et al. (2008) uses a similar setup where erroneous mental models are derived from a correct one and combined with a model of the interface. The resulting model is then checked for errors that outline a vulnerability of the system to the simulated error. Finally, Campos et al. (2004) use mental models in the form of user tasks to check advanced properties on their systems.

The idea to generate mental models from system models has been explored separately and conjointly by Heymann and Degani (2007), Combéfis and Pecheur (2009). In both cases, the generation is constrained by a validity property to ensure that the mental model preserves desirable properties when interacting with the system and by a minimality property to ensure that the resulting models are efficient. Various properties can be checked on formal models in addition to full-control. Campos and Harrison (2008) define generic usability properties that could be used to generate better mental models.

The concept of mental model itself is not new. Carroll and Olson (1987) already proposed generalized transition networks as a formalism for mental models. They outlined the difference between prescriptive and descriptive mental models. Descriptive mental models represent actual users and can therefore only be checked against properties. Normative mental models are generated to verify these properties and can be used as a description of how users should behave. This is the kind of mental models that we need to use to describe what we want to teach to operators. Staggers and Norcio (1993) make a clear distinction between the conceptual model of the target system, the interface (or image) of that system, the actual mental model of the user and the scientist's conceptualization of that mental model. The power of HMI-LTSs is that we can use them to model the actual system restricted to its interface and the normative mental models of the users.

11.7 Conclusion

In Sect. 11.2, we described HMI-LTS and how they can be used to model interactive systems. We formally defined the control property and its full-control extension. These properties are defined on two HMI-LTSs representing a system interface and a descriptive mental model of a user. These properties are important because they are used to guide the generation of ideal, normative mental models for human operators.

Based on these tools, we then defined the merge operation that represents how a human augments its mental model by learning new mental models. We have provided some evidence that this operator is a natural way to formalize the learning process. We have also outlined the properties of the merge operator and the lattice structure it induces on HLI-LTSs.

Finally, we have shown how full-control mental models can be decomposed into basic learning units. These basic units have the desired properties of independence and minimality, and each has proper control over the full-control mental model and hence over the system. With this decomposition, we have defined a measure of the complexity of learning an interactive system.

Of course, there remains a lot of work to show how this theory relates to existing training material. For example, this works could be used to verify the modularity of system design by detecting irreducible large components. We could also investigate how the structure of existing user manual relates with sets of basic learning units, and how basic learning units can help generating such manuals. This theory opens the way towards formal analysis of training material.

References

Blandford A, Cauchi A, Curzon P, Eslambolchilar P, Furniss D, Gimblett A, Huang H, Lee P, Li Y, Masci P, Oladimeji P, Rajkomar A, Rukšėnas R, Thimbleby H (2011) Comparing actual practice and user manuals: a case study based on programmable infusion pumps. In: Proceedings of the 1st international workshop on engineering interactive computing systems for medecine and health care (EICS4Med 2011)

Bolton ML, Bass EJ, Siminiceanu RI (2008) Using formal methods to predict human error and system failures. In: Proceedings of the second international conference on applied human factors and ergonomics (AHFE 2008)

Buth B (2004) Analysing mode confusion: an approach using FDR2. In: Heisel M, Liggesmeyer P, Wittmann S (eds) Proceedings of the 23rd international conference on computer safety, reliability and security (SAFECOMP 2004). Lecture notes in computer science, vol 3219. Springer, pp 101–114

Campos JC, Harrison MD (2008) Systematic analysis of control panel interfaces using formal tools. In: Graham TCN, Palanque PA (eds) Proceedings of the 15th international workshop on design, specification and verification of interactive systems (DSV-IS 2008). Lecture notes in computer science, vol 5136. Springer, pp 72–85

Campos JC, Harrison MD, Loer K (2004) Verifying user interfaces behaviour with model checking. In: Augusto JC, Ultes-Nitsche U (eds) Proceedings of the 2nd international workshop on verification and validation of enterprise information systems (VVEIS 2004). INSTICC Press, pp 87–96

Carroll JM, Anderson NS, Olson JR et al (1987) Mental models in human-computer interaction: research issues about what the user of software knows, Number 12. National academies

Combéfis S (2013) A formal framework for the analysis of human-machine interactions. PhD thesis, Université catholique de Louvain

Combéfis S, Pecheur C (2009) A bisimulation-based approach to the analysis of human-computer interaction. In: Calvary G, Graham TCN, Gray P (eds) Proceedings of the ACM SIGCHI symposium on engineering interactive computing systems (EICS 2009). ACM, New York, NY, USA, pp 101–110

Combéfis S, Giannakopoulou D, Pecheur C, Feary M (2011a) Learning system abstractions for human operators. Proceedings of the 2011 international workshop on machine learning technologies in software engineering (MALETS 2011). New York, NY, USA. ACM, pp 3–10

Combéfis S, Giannakopoulou D, Pecheur C, Feary M (2011b) A formal framework for design and analysis of human-machine interaction. In: Proceedings of the 2011 IEEE international conference on systems, man, and cybernetics (SMC 2011). IEEE, pp 1801–1808

Delp C, Lam D, Fosse E, Lee C-Y (2013) Model based document and report generation for systems engineering. In: 2013 IEEE aerospace conference. IEEE, pp 1–11

Michael Heymann, Asaf Degani (2007) Formal analysis and automatic generation of user interfaces: approach, methodology, and an algorithm. Hum Factors: J Hum Factors Ergon Soc 49(2):311–330

Kieras David E, Polson Peter G (1985) An approach to the formal analysis of user complexity. Int J Man-Mach Stud 22(4):365–394

Palmer E (1995) "oops, it didn't arm."—a case study of two automation surprises. In: Jensen RS, Rakovan LA (eds) Proceedings of the 8th international symposium on aviation psychology (ISAP 1995), April 1995

Rushby J (2002) Using model checking to help discover mode confusions and other automation surprises. Reliab Eng Syst Saf 75(2):167–177

Sarter NB, Woods DD, Billings CE (1997) Automation surprises. In: Salvendy G (ed) Handbook of human factors and ergonomics, chapter 57. Wiley, pp 1926–1943

Spichkova M, Zhu X, Mou D (2014) Do we really need to write documentation for a system? CASE tool add-ons: generator+editor for a precise documentation. CoRR, 2014. http://arXiv.org/abs/1404.7265

Nancy Staggers, Norcio Anthony F (1993) Mental models: concepts for human-computer interaction research. Int J Man-Mach Stud 38(4):587–605

Thimbleby HW, Ladkin PB (1995) A proper explanation when you need one. In: Kirby MAR, Dix AJ, Finlay JM (eds) Proceedings of HCI '95 people and computers X, Huddersfield, August 1995. University Press, pp 107–118. ISBN 0-521-56729-7

Harold Thimbleby, Ladkin Peter B (1996) From logic to manuals. Softw Eng J 11(6):347–354

Tretmans J (2008) Model based testing with labelled transition systems. In: Hierons R, Bowen J, Harman M (eds) Formal methods and testing. Lecture notes in computer science, vol 4949. Springer, pp 1–38