Modal Scenarios as Automata

David Harel and Amir Kantor

Faculty of Mathematics and Computer Science,
Weizmann Institute of Science, Rehovot, Israel
{dharel,amir.kantor}@weizmann.ac.il

Abstract. Scenario-based modeling in *live sequence charts* (LSC) involves specifying multi-modal inter-object scenarios, in which events can be mandatory (hot) or possible (cold). In translating LSCs into automata over infinite words, an intermediate step constructs a kind of transition system that we call a *modal state structure* (MSS). Here we present MSSs as abstract forms of modal scenarios (with both mandatory, possible and forbidden behavior), which may encode more general patterns than those inherent in LSC, such as loops, alternatives and breaks. MSSs are essentially automata, in which the notion of temperature is adopted from LSCs, replacing traditional acceptance conditions.

Keywords: live sequence charts, scenario, multi-modal, automata, modal state structures.

Dedicated to Prof. Yaacov Choueka: scholar, teacher, researcher — all of the very highest quality — and a true inspiration too.

1 Introduction

Scenario-based specification and programming is an approach to the modeling of reactive systems via inter-object interactions [5,14]. The components of the system interact with each other and with the system's environment (including the user). In this approach the inter-object interactions *define* the behavior of the system. This is in contrast to the more traditional intra-object approach (e.g. statecharts [10,11]), where a reactive system is defined through the behavior of each of its components. The scenario-based approach considers what happens between the components and the environment, with less emphasis put on the separate behavior of each object.

Live sequence charts (LSC) [5,14] is a primary example of a visual scenario-based formalism, which may be used for the specification and the programming of reactive systems. The language extends classical message sequence charts (MSC) [15], mainly by being *multi-modal*, i.e., distinguishing between behaviors that *may* happen in the system (cold) and those that *must* happen (hot). LSCs can also naturally express a variety of flavors of behavior and constraints, such as forbidden scenarios.

N. Dershowitz and E. Nissan (Eds.): Choueka Festschrift, Part I, LNCS 8001, pp. 156–167, 2014.

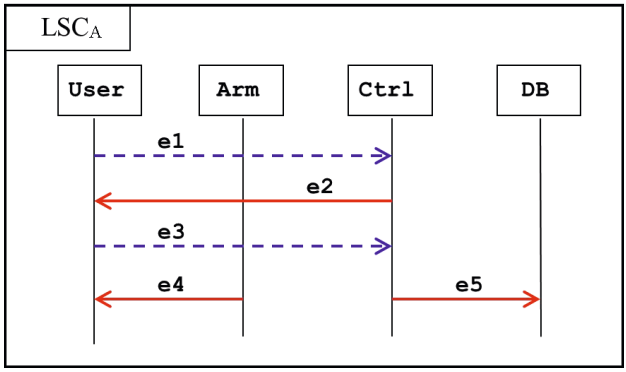

Fig. 1. An example: LSC_A

We consider here the UML2-compliant variant of the LSC language, defined in [13], which is slightly generalized and more uniform than the original. A chart contains objects arranged in vertical lifelines and the messages (events) that they send and receive drawn as horizontal arrows. Messages are either hot or cold (mandatory or possible, respectively). Since we are interested here purely in the inter-object dynamics of scenarios, we disregard conditions that constrain the state of the system. We do, however, allow trivial True/False conditions (either hot or cold), in order to specify synchronization points (using the condition True) and anti-scenarios (using hot False conditions).

The semantics of LSC may be presented via a translation into Büchi automata [16,13]. As an intermediate step, a process of 'unwinding' the chart results in a structure that is essentially a kind of modal transition system. This transition system, which we call a *modal state structure* (MSS), captures the modal scenario encoded in the chart.

In Sect. 2, we present an example of a live sequence chart and its underlying MSS. Section 3 provides a definition of modal state structures, and presents their semantics via Büchi automata. Section 4 extends the interpretation of MSS to express universality and a similar notion of iteration. A few patterns of scenarios that can be encoded naturally in MSS are considered in Sect. 5. Section 6 concludes with some shortcomings of MSS and future work.

2 MSS Underlying LSC

LSC_A, which appears in Fig. 1, is an example of a universal live sequence chart. It may correspond to an interaction of purchasing a product in a vending machine, and it captures the following modal scenario. If and when the `User` sends $\mathtt{e}_1$ to `Ctrl`, the latter must respond with $\mathtt{e}_2$. Then, if `User` sends $\mathtt{e}_3$ to `Ctrl`, the mechanical `Arm` must respond with $\mathtt{e}_4$ and `Ctrl` must update the database by sending $\mathtt{e}_5$. There is no order dependency of $\mathtt{e}_4$ and $\mathtt{e}_5$.

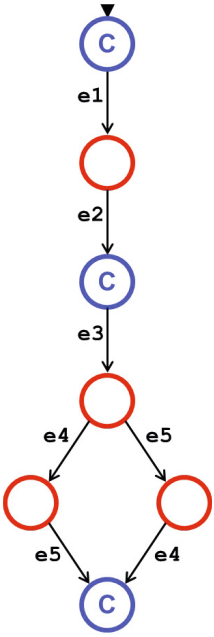

Fig. 2. MSS_A underlying LSC_A

In order to present trace-based semantics for LSCs, they may be translated into Büchi automata [16,13]. This involves 'unwinding' [16] each chart into states that correspond to cuts in the chart (which record progress along lifelines). This results in what we shall call a modal state structure (MSS), which captures the modal scenario encoded in the chart. MSS_A, which corresponds to LSC_A, is depicted in Fig. 2. Each state of MSS_A corresponds to a cut in LSC_A, and the initial state (drawn with a small incoming black triangle) is the minimal cut of LSC_A. The temperature of cuts, either cold or hot, is recorded by temperature of states (cold states are drawn marked with the letter 'C'). Outgoing transitions of each state correspond to enabled events in the corresponding LSC_A cut, each of which results in a change of cut. We regard an event as holding information about its source and target, e.g. $event = source!message?target$, so that this information is not lost in MSS_A.

In addition to the above, we also designate a set R of restricted events of MSS_A. These are not allowed to occur 'out of order' in MSS_A (i.e., when they are not enabled), and cause a violation if they do (see Sect. 3). There are few possible interpretations of LSCs with respect to this set of restricted events [16,14]. We may interpret LSC_A in the *strict* sense, meaning that the restricted events are exactly the events appearing in the chart, i.e., $R = \{\mathtt{e}_1, \mathtt{e}_2, \ldots, \mathtt{e}_5\}$. When one of these events occurs out of order, it causes a violation. Other events may occur without causing such violation. Other interpretations, namely *immediate* and *tolerant* (or *weak*), are discussed in Sect. 3.

3 MSS and Büchi Automata

A *modal state structure* (MSS) $\mathcal{M}$ is defined by $\mathcal{M} = \langle \Sigma, S, s_0, \rightarrow, C, R \rangle$, where Σ is a finite alphabet of events, S is a finite set of states, $s_0 \in S$ is the initial state, $\rightarrow \subseteq S \times \Sigma \times S$ is a labeled transition relation, and $C \subseteq S$ are designated as *cold* states. All other states are *hot*. $R \subseteq \Sigma$ is a set of *restricted* events, which are roughly events that cause a *violation* if they appear 'out of order'.

For $s, s' \in S$ and $e \in \Sigma$, we write $s \xrightarrow{e} s'$ iff $\langle s, e, s' \rangle \in \rightarrow$. Moreover, for $s \in S$ and $e \in \Sigma$, $s \xrightarrow{e}$ denotes that there exists $s' \in S$ such that $s \xrightarrow{e} s'$.

Let $s \in S$ be any state. The set of *enabled* events in s, i.e., events on some transition outgoing from s, is defined by $\mathrm{E}(s) = \{e \in \Sigma : s \xrightarrow{e}\}$. We say that s is *dead-end* iff $\mathrm{E}(s) = \emptyset$. All other events, namely *disabled* events $\Sigma \setminus \mathrm{E}(s)$, are partitioned into two sets: *violating* events $\mathrm{V}(s)$ and *indifferent* events $\mathrm{I}(s)$. $\mathrm{V}(s)$ is the set of disabled events that are also restricted; or, in the case of a dead-end state, all disabled events. These are the events that cause a violation (either cold or hot). The rest of the disabled events do not cause a violation, and are indifferent in state s. More formally we define

$$\mathrm{V}(s) = \begin{cases} R \setminus \mathrm{E}(s) & \text{if } \mathrm{E}(s) \neq \emptyset \\ \Sigma & \text{if } \mathrm{E}(s) = \emptyset \end{cases}$$

$$\mathrm{I}(s) = \begin{cases} (\Sigma \setminus R) \setminus \mathrm{E}(s) & \text{if } \mathrm{E}(s) \neq \emptyset \\ \emptyset & \text{if } \mathrm{E}(s) = \emptyset \end{cases} .$$

Figure 2 is an example of an MSS, obtained from the LSC of Fig. 1.

$\mathcal{M}$ is interpreted as following. An infinite word $\alpha \in \Sigma^\omega$ denotes a chain of events, and may designate a trace of an infinite execution. It is accepted by $\mathcal{M}$ (i.e., it complies with $\mathcal{M}$) iff there is an accepting run of $\mathcal{M}$ on α. The *language of* $\mathcal{M}$, denoted $\mathcal{L}(\mathcal{M})$, is the set of $\alpha \in \Sigma^\omega$ that are accepted by $\mathcal{M}$. A cold state is stable, while a hot state is unstable and carries a commitment to arrive later at a cold state. Consider an event e occurring when a run of $\mathcal{M}$ is in some state s (initially in s_0). If e is enabled in s, it must lead to some $s' \in S$ such that $s \xrightarrow{e} s'$. If e is indifferent in s (i.e., $e \in \mathrm{I}(s)$) we must stay at state s. And if, however, e is violating in s (i.e., $e \in \mathrm{V}(s)$), a violation occurs. Such a violation is either hot or cold according to the temperature of s. A hot violation is regarded as invalid (since a hot state is unstable and carries a commitment to eventually reach a cold state), yielding a rejection of the run. On the other hand, a cold violation (also called *completion*) yields acceptance of the run. If no violation occurs during the run it is accepting iff every hot state is followed by a cold state; i.e., cold states occur infinitely often.

In order to present the semantics of $\mathcal{M}$, we translate it into a Büchi automaton over infinite words [21,4], which we denote by $\mathfrak{B}(\mathcal{M})$; see Fig. 3. Specifically, we let $\mathfrak{B}(\mathcal{M}) = \langle \Sigma, \overline{S}, s_0, \Delta, F \rangle$, where Σ is the alphabet of $\mathfrak{B}(\mathcal{M})$, $\overline{S} := S \dot{\cup} \{\top\}$ are

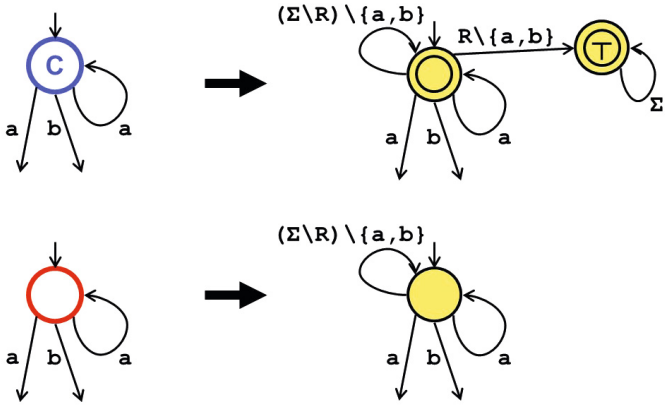

Fig. 3. Translation of an MSS into a Büchi automaton

its states, and s_0 is the initial state. $F := C \cup \{\top\}$ is the set of accepting states of $\mathfrak{B}(\mathcal{M})$, and its transition relation Δ is given by

$$\begin{aligned}\Delta := \ \rightarrow \ &\cup \ \{\, \langle s, e, s\rangle : s \in S,\ e \in \mathrm{I}(s) \,\} \\ &\cup \ \{\, \langle s, e, \top\rangle : s \in C,\ e \in \mathrm{V}(s) \,\} \\ &\cup \ \{\, \langle \top, e, \top\rangle : e \in \Sigma \,\} \ .\end{aligned}$$

The language of $\mathcal{M}$, denoted by $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathfrak{B}(\mathcal{M})) \subseteq \Sigma^\omega$, is taken to be the language accepted by the automaton $\mathfrak{B}(\mathcal{M})$; i.e., consisting of the infinite words $\alpha \in \Sigma^\omega$ on which there exists an accepting run of $\mathfrak{B}(\mathcal{M})$. From this translation we see that runs that enter a cold (hot) dead-end state s are accepted (rejected). So these simple constructs are essentially 'accept' and 'reject', respectively. The result of translating MSS_A (originating from LSC_A) is depicted in Fig. 4.

The difference between MSS and Büchi automata lies in the semantics of disabled events. In MSS, if a disabled event occurs at state s, it is either indifferent or violating, and in the latter case it yields either acceptance or rejection of the run, according to the temperature of s. In contrast, in Büchi automata disabled events are always rejecting in both accepting and non-accepting states. We believe that the duality of hot and cold states in MSS, and the existence of indifferent events, is natural for specifying modal scenarios.

Different values of the set R of restricted events yield different semantical variants. When $R = \Sigma$, we term the MSS *immediate*, and any disabled event is violating, so transitions are to be taken immediately (if taken at all). If R is exactly the set of events appearing in the transition relation $\rightarrow$, the MSS is termed *strict* [16,14]. When $R = \emptyset$, no events are restricted, and the MSS is termed *tolerant*, or *weak* [16,14]. In this case, all disabled events are indifferent (unless the state is dead-end), and leave the scenario in the same state.

Given a Büchi automaton $\mathfrak{B}$ over alphabet Σ, we can translate it into an MSS $\mathcal{M}_\mathfrak{B}$ over Σ, in which the set R of restricted events is arbitrary. In this reverse translation we turn every accepting (resp. non-accepting) state q of $\mathfrak{B}$ into a

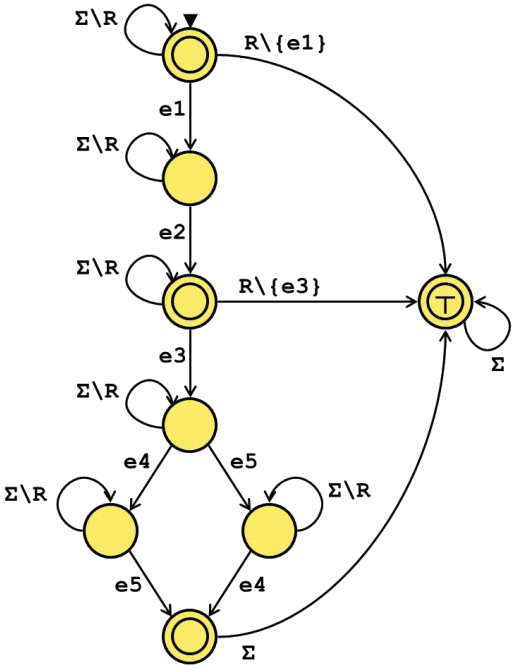

Fig. 4. Büchi automaton obtained from MSS_A

cold (resp. hot) state, and add transitions $\Sigma \setminus \mathrm{E}(q)$ leading to a 'reject' state (an additional hot state with no outgoing transitions). This translation is depicted in Fig. 5. Note that $\mathcal{M}_{\mathfrak{B}}$ has no disabled events — all events at any state (except for the 'reject' state) are enabled — so $\mathcal{M}_{\mathfrak{B}}$ acts just like a Büchi automaton. From this we see that given a finite alphabet Σ and any set of restricted events $R \subseteq \Sigma$, MSSs over Σ and R are just as expressive as Büchi automata (i.e., they yield the ω-regular languages), and translations are simple. Immediate, strict and weak semantics are all equivalent in this general context of MSS.

4 Universality and Iteration

The semantics suggested for MSS is *initial*, in the sense that the prescribed behavior is checked to hold starting from the beginning of a trace (cf. [3,17]). The language defined above corresponds to this initial interpretation of $\mathcal{M}$, and is now denoted $\mathcal{L}^{\mathrm{init}}(\mathcal{M}) = \mathcal{L}(\mathcal{M})$. However, scenarios prescribed by universal LSCs are usually interpreted as invariants. This means that in order for a trace $\alpha \in \Sigma^\omega$ to be accepted, it must conform to the scenario from *any* point on.

This *universal* semantics may be defined for MSS as follows. Denote by α/i the infinite word obtained from α with the first i letters truncated ($i \in \omega$). The universal language of MSS $\mathcal{M}$ is defined by

$$\mathcal{L}^{\mathrm{univ}}(\mathcal{M}) = \{\, \alpha \in \Sigma^\omega : \forall i \in \omega,\, \alpha/i \in \mathcal{L}^{\mathrm{init}}(\mathcal{M}) \,\} \ .$$

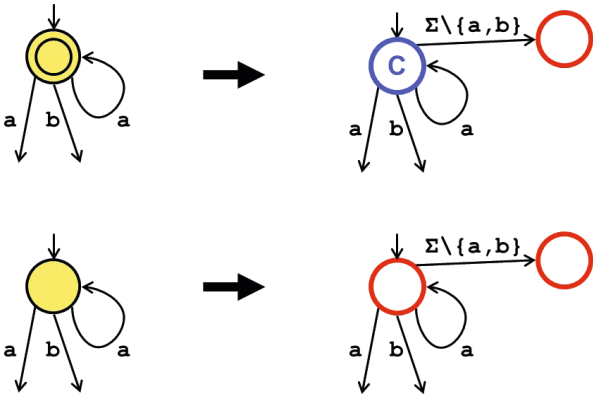

Fig. 5. Translation of a Büchi automaton into an MSS

Alternatively, we may construct an alternating Büchi automaton [20,18] from $\mathcal{M}$, denoted $\mathfrak{B}^{\text{univ}}(\mathcal{M})$, which yields this universal language directly (cf. [13]). It is obtained from $\mathfrak{B}(\mathcal{M})$ by adding to each transition leaving the initial state a conjunction with a transition to the initial state. A translation of this alternating automaton into a non-alternating one (and thus also into an MSS with initial semantics) is possible, but prolix [7,20].

A possible alternative to universality is an *iterative* semantics for MSS (cf. [3]). In this interpretation, similarly to initial semantics, we need only follow a single copy of the MSS at any point during the trace (except for non-deterministic transitions). However, if the scenario is completed (i.e., a cold violation occurs) the MSS starts over from its initial state, and needs to hold again from this point on. The event that caused the completion is not skipped; it is reconsidered at the initial state. Iterative semantics is more permissive than universal, as the scenario is checked to hold only from certain points on (the beginning, and points of completion). It is, however, more restrictive than initial semantics. More formally, $\mathcal{L}^{\text{univ}}(\mathcal{M}) \subseteq \mathcal{L}^{\text{iter}}(\mathcal{M}) \subseteq \mathcal{L}^{\text{init}}(\mathcal{M})$. When following a trace from the beginning, it may reside in any of the states of $\mathcal{M}$. In contrast to universality, there are no 'overlapping' configurations that correspond to different starting points of $\mathcal{M}$ along the trace. This suggests that iterative semantics may be easier to grasp and use in certain contexts. Moreover, in the context of LSC play-out [14] for instance, iterative semantics may yield more efficient executions, since only one copy of LSC is needed.

We present iterative semantics of $\mathcal{M}$ by translation into a Büchi automaton, denoted $\mathfrak{B}^{\text{iter}}(\mathcal{M})$; see Fig. 6. The automaton is defined by $\mathfrak{B}^{\text{iter}}(\mathcal{M}) = \langle \Sigma, S, s_0, \Delta, F \rangle$, where Σ is the alphabet of $\mathfrak{B}^{\text{iter}}(\mathcal{M})$, S are its states, and s_0 is the initial state. The set of accepting states is $F := C$, and the transition relation Δ is given by

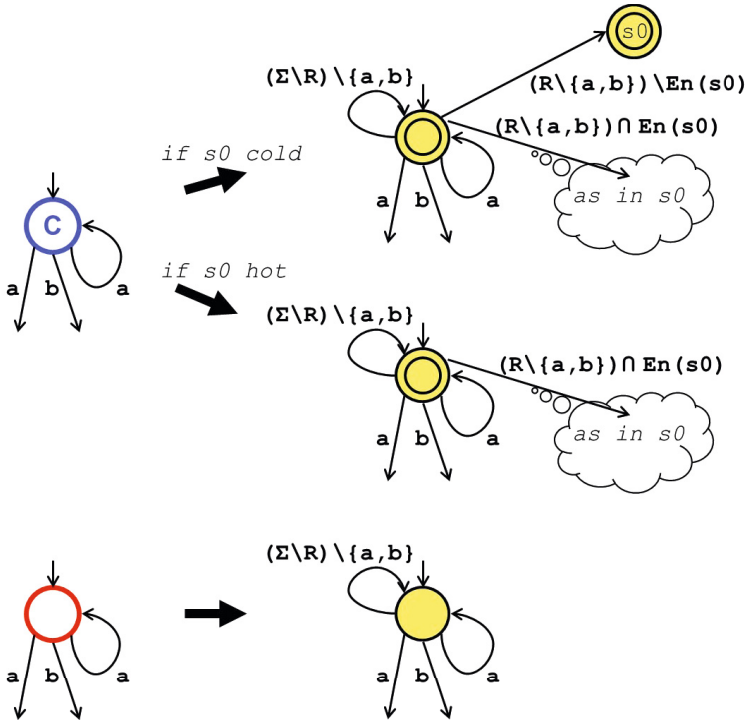

Fig. 6. Translation of an MSS with iterative interpretation into a Büchi automaton

$$\begin{aligned}\Delta := \; \rightarrow \; & \cup \; \{\, \langle s, e, s\rangle : s \in S,\, e \in \mathrm{I}(s) \,\} \\ & \cup \; \{\, \langle s, e, q\rangle : s \in C,\, e \in \mathrm{V}(s) \cap \mathrm{E}(s_0),\, q \in S,\, s_0 \xrightarrow{e} q \,\} \\ & \cup \; \{\, \langle s, e, s_0\rangle : s \in C,\, e \in \mathrm{V}(s) \cap \mathrm{I}(s_0) \,\} \\ & \cup \; \{\, \langle s, e, s_0\rangle : s \in C,\, e \in \mathrm{V}(s) \cap \mathrm{V}(s_0),\, s_0 \in C \,\}\,^{1} \;\; .\end{aligned}$$

For M we define the iterative language $\mathcal{L}^{\mathrm{iter}}(\mathcal{M}) = \mathcal{L}(\mathfrak{B}^{\mathrm{iter}}(\mathcal{M})) \subseteq \Sigma^{\omega}$, to be the language accepted by the automaton $\mathfrak{B}^{\mathrm{iter}}(\mathcal{M})$. From the definition of $\mathfrak{B}^{\mathrm{iter}}(\mathcal{M})$ we see that a hot dead-end state s is essentially 'reject', as all runs that enter it are rejected. Regarding a cold dead-end state s, it is equivalent in $\mathfrak{B}^{\mathrm{iter}}(\mathcal{M})$ to the initial state (except possibly being accepting, as s_0 may be hot). This essentially means "accept and continue with the next iteration". The result of translating MSS_A with the iterative interpretation into a Büchi automaton is depicted in Fig. 7.

Any Büchi automaton may be translated into an MSS with iterative interpretation (in which the set R of restricted events is arbitrary). Actually, the same translation into initial MSS works here too. From this we see that given

[1] This part of Δ is empty if s_0 is hot.

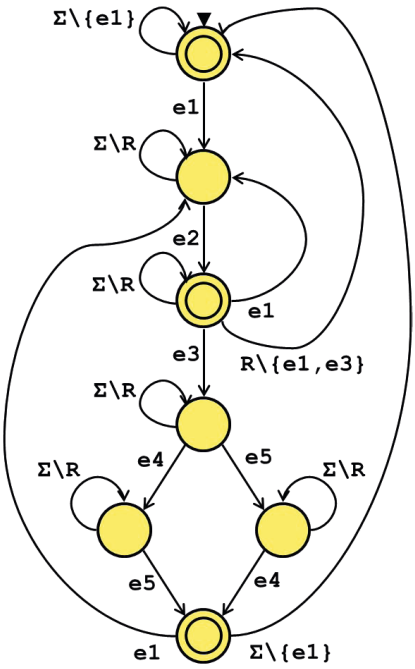

Fig. 7. Büchi automaton obtained from MSS_A with iterative interpretation

a finite alphabet Σ and $R \subseteq \Sigma$ any set of restricted events, iterative MSS over Σ and R are just as expressive as Büchi automata over Σ; i.e., they yield the ω-regular languages. Immediate, strict and weak semantics are all equivalent in this context as well.

5 Patterns in MSS

Extended patterns are available in LSC through advanced constructs [13,14], which extend the syntax and the basic partial-order semantics of LSC. MSS allow the abstract specification of such patterns with only few primitive notions. Note that in the context of MSS we do not consider conditions or guards that constrain the state of the system.

Alternatives and *breaks* in the progress of modal scenarios are inherent in MSS through a multitude of transitions originating from a state, including non-deterministic ones. Unbounded *loops* are also easy to specify. Figure 8 shows a loop and a break escaping from it.

6 Shortcomings of MSS and Future Work

Bounded loops are not explicitly supported in MSS. They may be modeled by unraveling them into finite segments of the scenario. Additionally, in MSS one

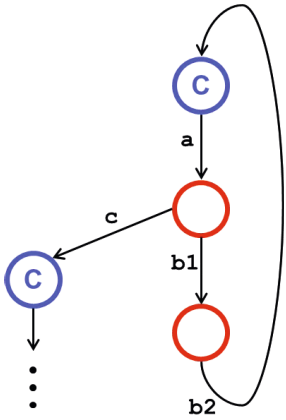

Fig. 8. Loop and break

needs to specify the transitions of each state separately. In many situations, however, taking a single transition should cause only a *partial* change in the required behavior. Consider for instance the LSC of Fig. 1 after e_3 occurs. There are two events that must take place — e_4 and e_5. After e_4 occurs, e_5 is still required to occur, but e_4 is not. The required behavior changed, but only partially. In MSS this needs to be specified accordingly in each state separately; see Fig. 2. This independence of states in MSS yields additional expressive power, but it affects the ease of use of this formalism in such cases.

In Petri nets [24,22] transitions cause only partial, local, change in the global state. This principle traditionally suggests interpretations like concurrency and distribution. In another piece of work [12], we consider a multi-modal extension of Petri nets as a flexible and expressive means to specifying modal scenarios. We introduce modalities into labeled Petri nets by assigning each transition a temperature, either hot or cold. Now, a marking (i.e., the global state of the net) is hot (unstable) if there is an enabled hot transition, and otherwise it is cold (stable). Into this new setting we then incorporate other notions related to temperature, such as hot and cold violations. Moreover, a unification principle is introduced, through which firings of few enabled transitions, which are labeled with the same event, are identified.

The definitions we presented for the semantics of MSS do not relate specifically to an interaction between an open system and its environment. In an open system, the system and its environment are regarded as adversaries (see [23,9]). This issue is treated in the context of LSC through the definition of its operational semantics (namely, play-out [14] and its extensions). Another treatment appears in [2,1]; it involves not only distinguishing system events from environment events, but also expresses the semantics of LSC by dividing it into safety and liveness. Extending the semantics of MSS to relate explicitly to open-systems in a general and natural way is an interesting topic for future research. In this context, one may discuss the execution of such abstract scenarios.

We note that MSS should not be confused with modal transition systems (MTS) [19], which carry a different kind of modality; essentially that of being partially defined. The latter is related to incomplete information about the transitions of the system during modeling phases [6], and to the analysis of properties [8]. Roughly, some of the transitions are marked as provisional, and may be retained or removed in a sequence of refinements that continues until no provisional transitions remain. In MSS, in contrast, modality is attached to states, and its interpretation is of a different nature.

Acknowledgements. The authors wish to thank Shahar Maoz and Itai Segall for many helpful discussions. The research was supported in part by The John von Neumann Minerva Center for the Development of Reactive Systems at the Weizmann Institute of Science, and by a Grant from the G.I.F., the German-Israeli Foundation for Scientific Research and Development. The research was also supported in part by an Advanced Research Grant to the first-listed author, from the European Research Council (ERC), under the European Community's Seventh Framework Programme (FP7/2007-2013).

References

1. Bontemps, Y.: Relating Inter-Agent and Intra-Agent Specifications (The Case of Live Sequence Charts). Ph.D. thesis, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique (University of Namur, Computer Science Dept) (April 2005)
2. Bontemps, Y., Heymans, P., Schobbens, P.Y.: From live sequence charts to state machines and back: A guided tour. IEEE Trans. Software Eng. 31(12), 999–1014 (2005)
3. Brill, M., Damm, W., Klose, J., Westphal, B., Wittke, H.: Live Sequence Charts. In: Ehrig, H., Damm, W., Desel, J., Große-Rhode, M., Reif, W., Schnieder, E., Westkämper, E. (eds.) INT 2004. LNCS, vol. 3147, pp. 374–399. Springer, Heidelberg (2004)
4. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proc. Internat. Congress on Logic Methodology and Philosophy of Science 1960, pp. 1–12. Stanford Univ. Press, Stanford (1962)
5. Damm, W., Harel, D.: LSCs: Breathing Life into Message Sequence Charts. J. on Formal Methods in System Design 19(1), 45–80 (2001); preliminary version in Ciancarini, P., Fantechi, A., Gorrieri, R. (eds.): Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS 1999), pp. 293–312. Kluwer Academic Publishers (1999)
6. D'Ippolito, N., Fischbein, D., Chechik, M., Uchitel, S.: MTSA: The Modal Transition System Analyser. In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), pp. 475–476. IEEE Computer Society, Washington, DC (2008)
7. Drusinsky, D., Harel, D.: On the power of bounded concurrency I: finite automata. J. ACM 41(3), 517–539 (1994)
8. Godefroid, P., Huth, M., Jagadeesan, R.: Abstraction-Based Model Checking Using Modal Transition Systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 426–440. Springer, Heidelberg (2001)

9. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
10. Harel, D.: Statecharts: A visual formalism for complex systems. Sci. Comput. Program. 8(3), 231–274 (1987)
11. Harel, D., Gery, E.: Executable object modeling with statecharts. IEEE Computer 30(7), 31–42 (1997)
12. Harel, D., Kantor, A.: Multi-modal scenarios revisited: A net-based representation. Theoretical Computer Science 429, 118–127 (2012)
13. Harel, D., Maoz, S.: Assert and Negate Revisited: Modal Semantics for UML Sequence Diagrams. Software and Systems Modeling (SoSyM) 7(2), 237–252 (2008), Preliminary version appeared in Proc. 5th Int. Workshop on Scenarios and State-Machines (SCESM 2006) at the 28th Int. Conf. on Soft. Eng (ICSE 2006), May 13-19, pp. 237–252. ACM Press (2008)
14. Harel, D., Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Springer (2003)
15. ITU: International Telecommunication Union Recommendation Z.120: Message Sequence Charts. Tech. rep. (1996)
16. Klose, J., Klose, H.: An Automata Based Interpretation of Live Sequence Charts. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 512–527. Springer, Heidelberg (2001)
17. Kugler, H., Harel, D., Pnueli, A., Lu, Y., Bontemps, Y.: Temporal Logic for Scenario-Based Specifications. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 445–460. Springer, Heidelberg (2005)
18. Kupferman, O., Vardi, M.Y.: Weak alternating automata are not that weak. ACM Trans. Comput. Logic 2(3), 408–429 (2001)
19. Larsen, K.G., Thomsen, B.: A modal process logic. In: Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS 1988), pp. 203–210. IEEE Computer Society Press (1988)
20. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. Theor. Comput. Sci. 32(3), 321–330 (1984)
21. Mukund, M.: Finite-state automata on infinite inputs. In: The 6th National Seminar on Theoretical Computer Science, Banasthali, Rajasthan, India (August 1996)
22. Peterson, J.L.: Petri nets. ACM Comput. Surv. 9(3), 223–252 (1977)
23. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1989), pp. 179–190. ACM, New York (1989)
24. Reisig, W.: Petri nets: an introduction. Springer, New York (1985)