Solving the Sabotage Game is PSPACE-hard

Christof Löding and Philipp Rohde

Lehrstuhl für Informatik VII, RWTH Aachen
{loeding,rohde}@informatik.rwth-aachen.de

Abstract. We consider the sabotage game as presented by van Benthem. In this game one player moves along the edges of a finite multi-graph and the other player takes out a link after each step. One can consider usual algorithmic tasks like reachability, Hamilton path, or complete search as winning conditions for this game. As the game definitely ends after at most the number of edges steps, it is easy to see that solving the sabotage game for the mentioned tasks takes at most PSPACE in the size of the graph. In this paper we establish the PSPACE-hardness of this problem. Furthermore, we introduce a modal logic over changing models to express tasks corresponding to the sabotage games and we show that model checking this logic is PSPACE-complete.

1 Introduction

In some fields of computer science, especially the controlling of reactive systems, an interesting sort of tasks arises, which consider temporal changes of a systems itself. In contrast to the usual tasks over reactive systems, where movements *within* a system are considered, an additional process affects: the dynamic change of the system itself. Hence we have two different processes: a *local* movement within the system and a *global* change of the system. Consider, for example, a network where connections or servers may break down. Some natural questions arise for such a system: is it possible – regardless of the removed connections – to interchange information between two designated servers? Is there a protocol which guarantees that the destination can be reached? Another example for a task of this kind was recently given by van Benthem [1], which can be described as the *real Travelling Salesman Problem*: is it possible to find your way between two cities within a railway network where a malevolent demon starts cancelling connections?

As usual one can model such kind of reactive system as a two-person game, where one player tries to achieve a certain goal given by a winning condition and the other player tries to prevent this. As winning conditions one can consider algorithmic tasks over graphs as, e.g., reachability, Hamilton path, or complete search. Determining the winner of these games gives us the answers for our original tasks.

In this paper we show that solving *sabotage games* where one player (the *Runner*) moves along edges in a multi-graph and the other player (the *Blocker*) removes an edge in each round is PSPACE-hard for the three mentioned winning conditions. The main aspect of the sabotage game is that the *Runner* can only act *locally* by moving one step further from his actual position whereas the *Blocker* has the possibility to behave *globally* on the arena of the game. So the sabotage game is in fact a match between a *local* and a *global* player. This distinguishes the sabotage game from the classical games that are studied in combinatorial game theory (see [2] for an overview).

In Sect. 2 we introduce the basic notions of the sabotage game. In Sect. 3 we show the PSPACE-hardness for the sabotage game with the reachability condition on undirected graphs by giving a polynomial time reduction from the PSPACE-complete problem of *Quantified Boolean Formulas* to these games. In Sect. 4 we give polynomial time reductions from sabotage games with reachability condition to the other winning conditions. In the last section we introduce the extension SML of modal logic over transitions systems which captures the concept of removing edges, i.e., SML is a modal logic over *changing models*. We give the syntax and the semantics of SML and provide a translation to first order logic. By applying the results of the first part we will show that model checking for this logic is PSPACE-complete.

We would like to thank Johan van Benthem and Peter van Emde Boas for several ideas and comments on the topic.

2 The Sabotage Game

In this section we give the definition of the sabotage game and we repeat three algorithmic tasks over graphs which can be considered as winning conditions for this game.

A *multi-graph* is a pair (V, e) where V is a non-empty, finite set of vertices and $e : V \times V \to \mathbb{N}$ is an edge multiplicity function, i.e., $e(u, v)$ denotes the number of edges between the vertices u and v. $e(u, v) = 0$ means that u and v are not connected. In case of an undirected graph we have in addition $e(u, v) = e(v, u)$ for all $u, v \in V$. A *single-graph* is given by a multiplicity function with $e(u, v) \leq 1$ for all vertices $u, v \in V$. The size of a multi-graph (V, e) is given by $|V| + |E|$, where we set $|E| := \sum_{u,v \in V} e(u, v)$ for directed graphs and $|E| := \frac{1}{2} \sum_{u,v \in V} e(u, v)$ for undirected graphs.

Let (V, e_0) be a multi-graph and $v_0 \in V$ be an initial vertex. The two-person sabotage game is played as follows: initially the game arena is $\mathcal{A}_0 = (V, e_0, v_0)$. The two players, which we call *Runner* and *Blocker*, move alternatingly, where the *Runner* starts his run from vertex v_0. At the start of round n the *Runner* moves one step further along an existing edge of the graph, i.e., if v_n is his actual position, he chooses a $v_{n+1} \in V$ with $e_n(v_n, v_{n+1}) > 0$ and moves to v_{n+1}. Afterwards the *Blocker* removes one edge of the graph, i.e., he chooses two vertices u and v somewhere in the graph with $e_n(u, v) > 0$. In the directed case we define $e_{n+1}(u, v) := e_n(u, v) - 1$ and $e_{n+1}(\cdot, \cdot) := e_n(\cdot, \cdot)$ otherwise. In the undirected case we let $e_{n+1}(u, v) := e_{n+1}(v, u) := e_n(u, v) - 1$. The multi-graph $\mathcal{A}_{n+1} = (V, e_{n+1}, v_{n+1})$ becomes the arena for the next round. The game ends, if either the *Runner* cannot make a move, i.e., there is no link starting from his actual position or if the winning condition is fulfilled.

As winning conditions for the sabotage game on an undirected or directed graph one can consider the usual tasks over graphs, for example:

1. *Reachability*: the *Runner* wins iff he can reach a given vertex (which we call the *goal*)
2. *Hamilton Path* or *Travelling Salesman*: the *Runner* wins iff he can move along a Hamilton Path, i.e., he visits each vertex exactly once
3. *Complete Search*: the *Runner* wins iff he can visit each vertex (possibly more than once)

It is easy to see that for the reachability game with one single goal the use of multi-graphs is crucial, but we can bound the multiplicity uniformly by two or, if we allow a second goal vertex, we even can transform every multi-graph game into a single-graph game:

Lemma 1. *Let G be a sabotage game with reachability condition on a multi-graph arena $\mathcal{A}$. Then there are games G', G'' on arenas $\mathcal{A}', \mathcal{A}''$ with a size polynomial in the size of $\mathcal{A}$ such that the* Runner *wins G iff he wins G', resp. G'', and $\mathcal{A}'$ is a single-graph with two goals and $\mathcal{A}''$ is a multi-graph with one goal and only single or double edges where the double edges occur only connected with the goal.*

Proof. We only sketch the proof for directed graphs. To obtain $\mathcal{A}'$ one adds a new goal and replaces each edge between vertices u and v with multiplicity $k > 0$ by the construction depicted in Fig. 1 (with k new vertices). We actually need a new goal if v is the original goal. The arena $\mathcal{A}''$ is constructed similarly: if v is not the original goal we apply the same construction (Fig. 1), but reusing the existing goal instead of adding a new one. If v is the goal then we add double edges from the new vertices to v (see Fig. 2). Note that *Blocker* does not gain additional moves because all new vertices are directly connected to the goal. $\square$

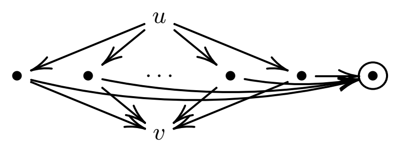


Fig. 1. Replacement for $\mathcal{A}'$

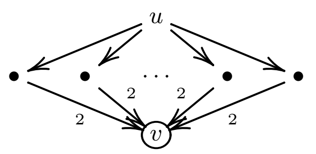


Fig. 2. Replacement for $\mathcal{A}''$

Since edges are only deleted but not added during the play the following fact is easy to see:

Lemma 2. *If the* Runner *has a winning strategy in the sabotage game with reachability condition then he can win without visiting any vertex twice.*

In the sequel we will introduce several game arenas where we use edges with a multiplicity 'high enough' to ensure that the *Blocker* cannot win the game by reducing these edges. In figures these edges are represented by a curly link $\bullet \sim\!\sim\!\sim \bullet$. For the moment we can consider these links to be 'unremovable'. Due to the previous lemma we have: if the *Runner* can win the reachability game at all, then he can do so within at most $|V|-1$ rounds. Hence we can set the multiplicity of the 'unremovable' edges to $|V|-1$. To bound the multiplicity of edges uniformly one can apply Lemma 1.

3 PSPACE-Hardness for Sabotage Reachability Games

In this section we prove that the PSPACE-complete problem of *Quantified Boolean Formulas* (cf. [3]), QBF for short, can be reduced by a polynomial time reduction to sabotage games on undirected graphs with the reachability condition.

Let $\varphi \equiv \exists x_1 \forall x_2 \exists x_3 \dots Q x_n \psi$ be an instance of QBF, where Q is $\exists$ for n odd and $\forall$ otherwise and ψ is a quantifier-free Boolean formula in conjunctive normal form. We will construct an undirected game arena for a sabotage game G_φ with a reachability condition such that the *Runner* has a winning strategy in the game iff the formula φ is satisfiable.

A reduction like the classical one from QBF to the *Geography Game* (cf. [3]) does not work here, since the *Blocker* may destroy connections in a part of the graph which should be visited only later in the game. This could be solved by blowing up the distances, but this approach results in an arena with a size exponential in the size n of φ. So we have to restrict the liberty of the *Blocker* in a more sophisticated way, i.e., to force him removing edges only 'locally'.

The game arena G_φ consists of two parts: a chain of n gadgets where first the *Runner* chooses an assignment for x_1, then the *Blocker* chooses an assignment for x_2 before the *Runner* chooses an assignment for x_3 and so on. The second part gives the *Blocker* the possibility to select one of the clauses of ψ. The *Runner* must certify that this clause is indeed satisfied by the chosen assignment: he can reach the goal vertex and win the game iff at least one literal in the clause is true under the assignment.

Figure 5 shows an example of the sabotage game G_φ for the formula $\varphi \equiv \exists x_1 \forall x_2 \exists x_3 \, (c_1 \wedge c_2 \wedge c_3 \wedge c_4)$ where we assume that each clause consists of exactly three literals. In the following we describe in detail the several components of G_φ and their arrangement. The main step of the construction is to take care about the opportunity of the *Blocker* to remove edges *somewhere* in the graph.

The $\exists$-gadget. The gadget where the *Runner* chooses an assignment for the x_i with i odd is displayed in Fig. 3. We are assuming that the run reaches this gadget at vertex A at the first time. Vertex B is intended to be the exit. In the complete construction there are also edges from X_i, resp. $\overline{X_i}$ leading to the last gadget of the graph, represented as dotted lines labelled by *back*. We will see later that taking these edges as a shortcut, *starting from* the $\exists$-gadget directly

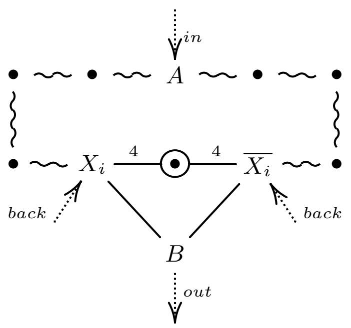


Fig. 3. $\exists$-gadget for x_i with i odd

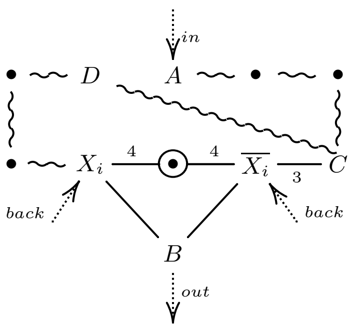


Fig. 4. $\forall$-gadget for x_i with i even

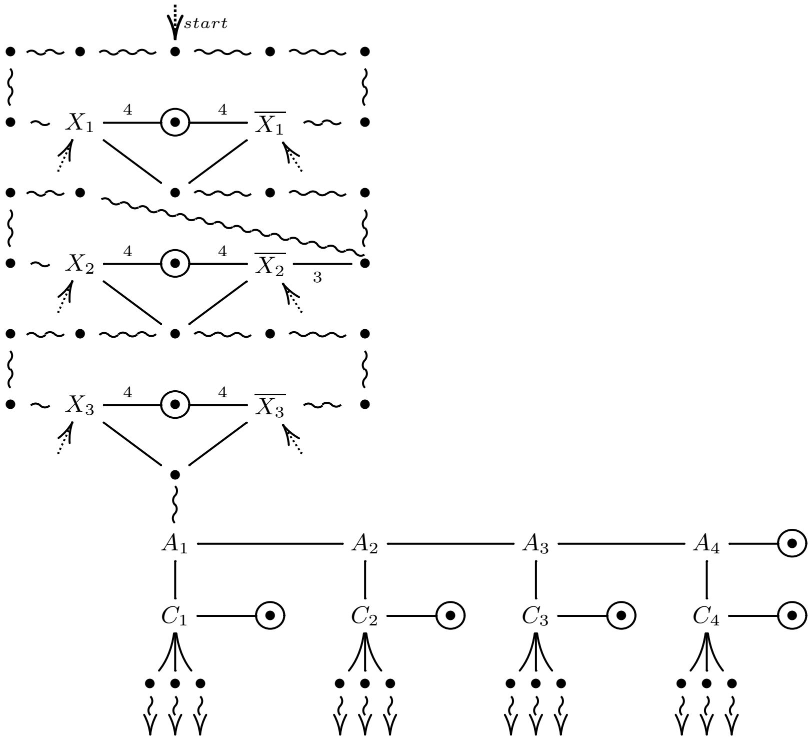
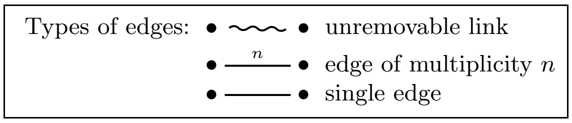


Fig. 5. The arena for $\exists x_1 \forall x_2 \exists x_3\ (c_1 \wedge c_2 \wedge c_3 \wedge c_4)$



to the last gadget is useless for the *Runner*. The only meaningful direction is *coming from* the last gadget back to the ∃-gadget. So we temporary assume that the *Runner* does not take these edges. In the sequel we further assume, due to Lemma 2, that the *Runner* does not move backwards.

The *Runner* makes his choice simply by moving from A either to the left or to the right. Thereby he moves either towards X_i if he wants x_i to be `false` or towards $\overline{X_i}$ if he wants x_i to be `true`. We consider only the first case. The *Blocker* has exactly four steps to remove all the links between X_i and the goal before the *Runner* reaches this vertex. On the other hand the *Blocker* cannot remove edges somewhere else in the graph without loosing the game. Why we use four steps here will be clarified later on. If the *Runner* has reached X_i and he moves towards B then the *Blocker* has to delete the edge between B and $\overline{X_i}$ since otherwise the *Runner* can reach the goal on this way (there are still four edges left between $\overline{X_i}$ and the goal).

The ∀-gadget. The gadget where the *Blocker* chooses an assignment for the x_i with i even is a little bit more sophisticated. Figure 4 shows the construction.

If the *Blocker* wants x_i to be `false` he tries to lead the *Runner* towards X_i. In this case he simply removes the three edges between C and $\overline{X_i}$ during the first three steps. Then the *Runner* has to move across D and in the meantime the *Blocker* deletes the four edges between X_i and the goal to ensure that the *Runner* cannot win directly. As above he removes in the last step the link between B and $\overline{X_i}$ to prevent a premature end of the game.

If the *Blocker* wants to assign `true` to x_i he should lead the *Runner* towards $\overline{X_i}$. To achieve this aim he removes three of the four links between $\overline{X_i}$ and the goal before the *Runner* reaches C. Nevertheless the *Runner* has the free choice at vertex C whether he moves towards X_i or towards $\overline{X_i}$, i.e., the *Blocker cannot guarantee* that the run goes across $\overline{X_i}$. But let us consider the two possible cases: first we assume that the *Runner* moves as intended and uses an edge between C and $\overline{X_i}$. In this round the *Blocker* removes the last link from $\overline{X_i}$ to the goal. Then the *Runner* moves to B and the *Blocker* deletes the edge from B to X_i.

Now assume that the *Runner* 'misbehaves' and moves from C to D and further towards X_i. Then the *Blocker* first removes the four edges between X_i and the goal. When the *Runner* now moves from X_i to B the *Blocker* has to take care that the *Runner* cannot reach the goal via the link between B and $\overline{X_i}$ (there is still one edge left from $\overline{X_i}$ to the goal). For that he can delete the last link between $\overline{X_i}$ and the goal and isolate the goal completely within this gadget.

The verification gadget. The last component of the arena is a gadget where the *Blocker* can choose one of the clauses of the formula ψ. Before we give the representation of this gadget let us explain the idea. If the *Blocker* chooses the clause c then the *Runner* can select for his part one literal x_i of c. There is an edge back to the ∃-gadget if i is odd or to the ∀-gadget if i is even, videlicet to X_i if x_i is positive in c, resp. to $\overline{X_i}$ if x_i is negative in c. So if the chosen assignment satisfies ψ, then for all clauses of ψ there is at least one literal which is `true`. Since the path through the assignment gadgets visits the opposite truth values this means that there is at least one edge back to an X_i, resp. $\overline{X_i}$, which itself is connected to the goal by an edge with a *multiplicity of four* (assuming that the *Runner* did not misbehave in the $\forall$-gadget). Therefore the *Runner* can reach the goal and wins the game.

For the converse if the chosen assignment does not satisfy ψ, then there is a clause c in ψ such that every literal in c is assigned `false`. If the *Blocker* chooses this clause c then every edge back to the assignment gadgets ends in an X_i, resp. $\overline{X_i}$, which is *unconnected* to the goal. If we show that there is no other way to reach the goal this means that the *Runner* looses the game.

But we have to be very careful neither to allow any shortcuts for the *Runner* nor to give the *Blocker* to much liberty. Figure 5 contains the verification gadget for $\psi \equiv c_1 \wedge c_2 \wedge c_3 \wedge c_4$ where each clause c_i has exactly three literals. The curly edges at the bottom of the gadget lead back to the corresponding literals of each clause.

The *Blocker* chooses the clause c_k by first removing the edges from A_j to C_j for $j < k$ one after the other. Then he cuts the link between A_k and A_{k+1}, resp. between A_k and the goal if c_k is the last clause. By Lemma 2 it is useless for the *Runner* to go back, thus he can only follow the given path to C_k. If he reaches this vertex the *Blocker* must remove the link from C_k to the goal to prevent the win for the opponent. In the next step the *Runner* selects a literal x_i, resp. $\neg x_i$ in c_k, moves towards the corresponding vertex and afterwards along the curly edge back to the assignment gadgets as described above. At this point the *Blocker* has exactly *two* moves left, i.e., he is allowed to remove two edges somewhere in the graph. But we have: if the 'right' assignment for this literal has been chosen then there are exactly *four* edges left connecting the corresponding vertex and the goal. So the *Blocker* has not the opportunity to isolate the goal and the *Runner* wins the game. Otherwise, if the 'wrong' assignment has been chosen then there is no link from X_i, resp. $\overline{X_i}$ to the goal left. Any continuation which the *Runner* could take either leads him back to an already visited vertex (which is a loss by Lemma 2) or, by taking another *back*-edge in the 'wrong' direction, to another vertex in the verification gadget. We handle the latter case in general: if the *Runner* uses a shortcut starting from a literal vertex and moves directly to the bottom of the verification gadget then the *Blocker* can prevent the continuation of the run by removing the corresponding single edge between the clause vertex C_k and the vertex beneath and the *Runner* has to move back. So the *Runner* wins the game if and only if he wins it without using any shortcut.

If the *Runner* reaches a vertex A_k and the *Blocker* removes either the edge between A_k and C_k *or* the one between C_k and the goal *or* one of the edges leading to the vertices beneath C_k (one for each literal in c_k) then the *Runner* moves towards A_{k+1}, resp. towards the goal if c_k is the last clause. The *Runner* has to do so since, in the latter two cases, entering the 'damaged' area around C_k could be a disadvantage for him.

Finally we consider the case that the *Blocker* removes an edge somewhere else in the graph instead. This behaviour is only reasonable if the chosen assignment satisfies ψ. So consider the round when the *Runner* reaches for the first time an A_k such that the edges from A_k to A_{k+1}, resp. the goal, *as well as* all edges connected to C_k are still left. If c_k is the last clause then the *Runner* just reaches the goal and wins the game. Otherwise he moves to C_k, chooses an appropriate literal x_i, resp. $\neg x_i$ such that at least three edges from the corresponding vertex are still left (at least one literal of this kind exists in each clause). Since A_k is the first vertex with this property the *Blocker* has gained only *one* additional move, so nevertheless it remains at least *one* edge from the vertex X_i, resp. $\overline{X_i}$ to the goal. So if the *Runner* can chose a satisfying assignment at all then the *Blocker* cannot prevent the win for the *Runner* by this behaviour. This explains the multiplicity of four within the assignment gadgets.

This completes the construction of the game G_φ. Obviously, this construction can be done in polynomial time. Therefore, we obtain the following results.

Lemma 3. *The* Runner *has a winning strategy in the sabotage game G_φ iff φ is satisfiable.*

Theorem 4. *There is a polynomial time reduction from QBF to sabotage games with reachability winning condition on undirected graphs. In particular solving these games is PSPACE-hard.*

Since each edge of the game G_φ has an 'intended direction', it is straight forward to check that a similar construction works for directed graphs as well. The construction can also be adapted to prove the PSPACE-hardness of other variants of the game, e.g., if the *Blocker* is allowed to remove up to n edges in each round for a fixed number n or if the *Blocker* removes vertices instead of edges. For the details we refer the reader to [4].

4 The Remaining Winning Conditions

In this section we give polynomial time reductions from sabotage games with reachability condition to the ones with complete search condition and with Hamilton path condition. We only consider games on undirected graphs.

Let G be a sabotage game on an undirected arena $\mathcal{A} = (V, e, v_0)$ with the reachability condition. We present an arena $\mathcal{B}$ such that the *Runner* wins G iff he wins the game G' on $\mathcal{B}$ with the complete search condition iff he wins the game G'' on $\mathcal{B}$ with the Hamilton path condition.

To obtain $\mathcal{B}$ we add several vertices to $\mathcal{A}$: let $m := |V| - 2$ and let $v_1, \ldots, v_m$ be an enumeration of all vertices in $\mathcal{A}$ except the initial vertex and the goal. We add a sequence $P_1, \ldots, P_m$ of new vertices to $\mathcal{A}$ together with several chains of new vertices such that each chain has length $\max\{|V|, |E|\}$ and their nodes are linked among each other by 'unremovable' edges. We add these chains from P_i as well as from P_{i+1} to vertex v_i for $i < m$ and one chain from P_m to vertex v_m. Furthermore we add for $i < m$ shortcuts from the last vertices in the chains between P_i and v_i to the last vertices in the chains between P_{i+1} and v_i to give the *Runner* the possibility to skip the visitation of v_i. Additionally there is one link with multiplicity $|V|$ from P_1 to the goal in $\mathcal{A}$, see Fig. 6.

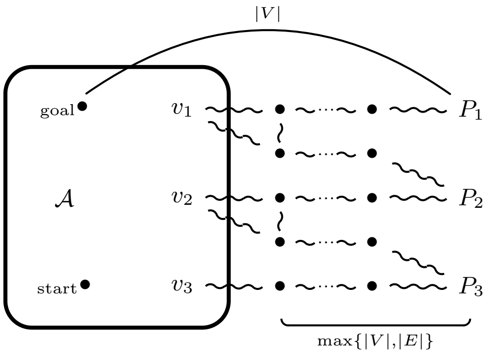


Fig. 6. Game arena $\mathcal{B}$

If the *Runner* can reach the goal in the original game G then by Lemma 2 he can do so within at most $|V| - 1$ steps. In this case there is at least one link to P_1 which he uses to reach P_1. He follows the chain to v_1. If he had already visited v_1 on his way to the goal he uses the shortcut at the last vertex in the chain, otherwise he visits v_1. Afterwards he moves to P_2 using the next chain. Continuing like this he reaches P_m and moves towards the last vertex v_m. If he had already visited v_m he just stops one vertex before. Otherwise he stops at v_m. Moving this way he visits each vertex of $\mathcal{B}$ exactly once and wins both games G' and G''.

For the converse: if the *Runner* cannot reach the goal in G then he cannot do so in the games G' and G'' as well. If he tries to use a shortcut via some P_i the *Blocker* has enough time on the way to P_i to cut all the links between the goal and P_1. On *Runner's* way back from some P_j to a vertex in $\mathcal{A}$ he is able to remove all edges in the original game arena $\mathcal{A}$ to isolate the goal completely. Thus the *Runner* looses both games G' and G'' on $\mathcal{B}$. So we have:

Theorem 5. *There is a polynomial time reduction from sabotage games with reachability condition to sabotage games with complete search condition, resp., with Hamilton path condition. In particular solving these games is PSPACE-hard.*

5 A Sabotage Modal Logic

In [1] van Benthem considered a 'sabotage modal logic', i.e., a modal logic over *changing models* to express tasks corresponding to sabotage games. He introduced a cross-model modality referring to submodels from which objects have been removed. In this section we will give a formal definition of a sabotage modal logic with a 'transition-deleting' modality and we will show how to apply the results of the previous sections to determine the complexity of uniform model checking for this logic. To realise the use of multi-graphs we will interpret the logic over edge-labelled transition systems. By applying Lemma 1 the complexity results for the reachability game can be obtained for multi-graphs with a uniformly bounded multiplicity. Hence we can do with a finite alphabet Σ.

Definition 6. *Let p be an unary predicate symbol and $a \in \Sigma$. Formulae of the* sabotage modal logic SML *over transition systems are defined by*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \Diamond_a \varphi \mid \diamondminus_a \varphi$$

The dual modality $\Box_a$ and the label-free versions $\Diamond$, $\Box$ are defined as usual. The modalities $\boxminus_a$, $\diamondminus$ and $\boxminus$ are defined analogously.

Let $\mathcal{T} = (S, \{R_a \mid a \in \Sigma\}, L)$ be a transition system. For $t, t' \in S$ and $a \in \Sigma$ we define the submodel $\mathcal{T}^a_{(t,t')} := (S, \{R_b \mid b \in \Sigma \setminus \{a\}\} \cup \{R_a \setminus \{(t,t')\}\}, L)$. For a given state $s \in S$ the semantics of SML is defined as for usual modal logic together with

$$(\mathcal{T}, s) \models \diamondminus_a \varphi \text{ iff there is } (t, t') \in R_a \text{ such that } (\mathcal{T}^a_{(t,t')}, s) \models \varphi$$

For a transition system $\mathcal{T}$ let $\hat{\mathcal{T}}$ be the corresponding FO-structure. Similar to the usual modal logic one can translate the logic SML into first order logic. Since FO-model checking is in PSPACE we obtain (see [4] for a proof):

Theorem 7. *For every* SML*-formula φ there is an effectively constructible* FO*-formula $\hat{\varphi}(x)$ such that for every transition system $\mathcal{M}$ and state s of $\mathcal{M}$ one has $(\mathcal{T}, s) \models_{\mathrm{SML}} \varphi$ iff $\hat{\mathcal{T}} \models_{\mathrm{FO}} \hat{\varphi}[s]$. The size of $\hat{\varphi}(x)$ is polynomial in the size of φ. In particular, SML-model checking is in PSPACE.*

We can express the winning of the *Runner* in the sabotage game G on directed graphs with the reachability condition by an SML-formula. For that we consider the game arena as a transition system $\mathcal{T}(G)$ such that the multiplicity of edges is captured by the edge labelling and such that the goal vertex of the game is viewed as the only state with predicate p. We inductively define the SML-formula γ_n by $\gamma_0 := p$ and $\gamma_{i+1} := (\Diamond \boxminus \gamma_i) \vee p$. Then we obtain the following lemma (see [4] for a proof) and in combination with Theorem 4 the PSPACE-completeness of SML model checking.

Lemma 8. *The* Runner *has a winning strategy from vertex s in the sabotage game G iff $(\mathcal{T}(G), s) \models \gamma_n$ where n is the number of edges of the game arena.*

Theorem 9. *Model checking for the sabotage logic SML is PSPACE-complete.*

References

1. van Benthem, J.: An essay on sabotage and obstruction. In Hutter, D., Werner, S., eds.: Festschrift in Honour of Prof. Jörg Siekmann. LNAI. Springer (2002)
2. Demaine, E.D.: Playing games with algorithms: Algorithmic combinatorial game theory. In: Proceedings of MFCS 2001. Volume 2136 of LNCS., Springer (2001) 18–32
3. Papadimitriou, C.H.: Computational Complexity. Addison–Wesley (1994)
4. Löding, C., Rohde, P.: Solving the sabotage game is PSPACE-hard. Technical Report AIB-05-2003, RWTH Aachen (2003)