Reactive graphs in action

(extended version) ⋆

David Tinoco[1], Alexandre Madeira[1],
Manuel A. Martins[1], and José Proença[2]

[1] CIDMA, Dep. Mathematics, University of Aveiro, Aveiro, Portugal
[2] CISTER, Faculty of Sciences, University of Porto, Porto, Portugal

Abstract. *Reactive graphs* are transition structures whereas edges become active and inactive during its evolution, that were introduced by Dov Gabbay from a mathematical's perspective. This paper presents Marge (`https://fm-dcc.github.io/MARGe`), a web-based tool to visualise and analyse reactive graphs enriched with labels. Marge animates the operational semantics of reactive graphs and offers different graphical views to provide insights over concrete systems. We motivate the applicability of reactive graphs for adaptive systems and for featured transition systems, using Marge to tighten the gap between the existing theoretical models and their usage to analyse concrete systems.

1 Introduction and motivation

A reactive graph is a transition structure that updates its transitions along its execution. This concept has been introduced by Dov Gabbay in [12]. It generalizes the static notion of a graph by incorporating high-order edges that capture updates on the accessibility relations. The notion of reactivity for these structures is not coined only in the standard sense of Harel and Pnueli [16], as systems that react to their environment and are not meant to terminate, but as systems whose accessibility relation is a result of the transformations induced by the transitions executed so far.

Let us consider the model of a simple vending machine in Fig. 1 to motivate reactive graphs. Edges in reactive graphs can be active or inactive, and only transitions involving active edges can be executed. All edges in our vending

⋆ This work extends a FACS 2024 publication [21] and includes minor corrections.

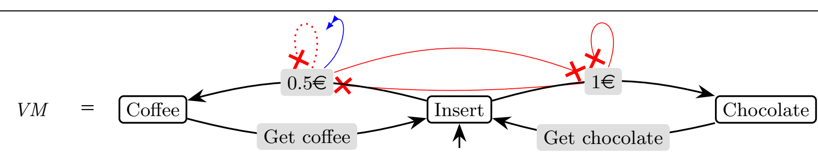

Fig. 1: A reactive graph of a vending machine offering two different products

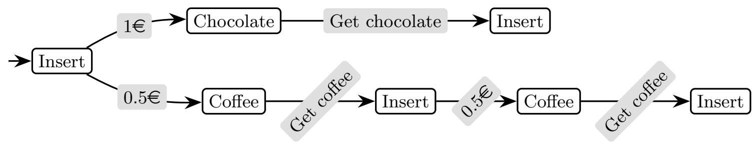

Fig. 2: The LTS of the vending machine in Fig. 1

machine are active, except one with a dotted line. Executing a transition triggers an update on the set of active and inactive edges.

The machine *VM* in Fig. 1 can receive from the user at most 1€. The arrows between states represent *ground edges*, which are labelled with actions; the others arrows represent *hyper edges*, i.e., edges that can activate (↠) and deactivate (⇸) edges. When the action 1€ is performed some edges are deactivated and the machine goes to the *Chocolate* state. More specifically, both edges labelled by 0.5€ and 1€ are deactivated. Executing instead the edge labelled by 0.5€ would enable a deactivating edge, and executing it a second time would deactivate it.

A solid supporting theory for these models, including proposals of specification logics, has been studied in the last years (e.g. [14,15]) and was summed up in the book [13]. The main advantage of reactive graphs with respect to traditional labelled transition systems (LTS) is the compact representation of dynamic systems. Gabbay showed the encoded LTS of a given reactive system can have an exponentially larger number of states [13, Prop. 8.8]. For example, our vending machine can be expressed with an LTS with seven states, depicted in Fig. 2. In a larger example borrowed from Cordy et al. [8, Fig. 1], presented in Appendix A, the encoded LTS has 7x more states and 4.6x more edges (including hyper edges). Furthermore, we believe that many examples become easier to understand and to maintain with reactive graphs than with traditional LTS.

Reconfigurable systems and variability. Not all scenarios and application domains can exploit the advantages of modelling by reactive graphs. The benefits of compactness and being more intuitive are more evident when analysing **reconfigurable systems**. These are systems that operate in different modes of execution, e.g., an operating systems that supports users with different permissions or different performance modes (e.g. an ocean exploring robot that adjusts its behaviour based on the distance of its base, its energy levels, and its environmental conditions).

Work on *software product lines* (SPL) focuses mainly on **configurable systems**, i.e., how to develop, maintain, and reason over families of software that share many commonalities. Feature transition systems [3,8] are structures often used to model such systems, which are enriched with annotations over features, allowing an initial configuration of the variant to select which transitions are active. A branch of SPL focuses on *dynamic SPL* [7] addressing *reconfigurable and self-adaptive systems*, in which the configuration can change over time, which relates to our reactive graphs.

This paper introduces Marge: an open-source web-based prototype tool designed for the visualisation and analysis of labelled reactive graphs. It includes several examples, both from the literature on reactive graphs [13] and from work on dynamic SPL [8]. The goal of Marge is to help increasing the adoption of reactive graphs, providing insights over the capabilities and challenges of modelling reconfigurable systems with reactive graphs, and to expose these to different domains to increase it applicability. Currently Marge does not aim at contributing directly to the community on adaptative SPLs, since it still misses a more user-friendly specification language and mechanisms to support larger systems. Marge provides support to: (1) *visualise* reactive graphs, (2) *animate* its operational semantics, (3) *explore* the full state-space of its underlying LTS, (4) *verify* properties such as deadlocks and conflicts, and (5) *compare* reactive graphs using an observational equivalence.

2 Multi-actions reactive graphs

A *multi-action reactive graph*, or simply *reactive graph*, is a labelled transition system with transitions enriched with a reaction that activates and deactivates transitions, defined formally below.

Definition 1 (Reactive graph). *A* Multi-Action Reactive Graph *is a tuple* $M = (W, Act, E, \rightarrow, \twoheadrightarrow, \multimap, \bar{\cdot}, w_0, \alpha_0)$ *where:*

- $W \neq \varnothing$ *is the* set of states*;* Act *is the* set of actions*;* E *is the* set of edges*;*
- $\rightarrow\ \subseteq W \times Act \times W$ *is the set of ground edges;* $\twoheadrightarrow\ \subseteq E \times E$ *is the set of activating edges;* $\multimap\ \subseteq E \times E$ *is the set of deactivating edges;* $\bar{\cdot} : E \longrightarrow (\rightarrow \cup \twoheadrightarrow \cup \multimap)$ *is an injective function that maps edges in* E *to their internal details;*
- $w_0 \in W$ *is the* initial state*;* $\alpha_0 \subseteq E$ *is the set of* initially active edges.

Notation. Recall the vending machine in Fig. 1. When formalising this as a reactive graph we say that: (i) *Coffee* belongs to W (among others), (ii) 0.5€ is an action in Act, $\langle Insert, 0.5€, Coffee\rangle$ is a ground edge in $\rightarrow$, (iii) $\langle e^{\dagger}, e^{\dagger}\rangle$ is a deactivating edge in $\multimap$ where $\overline{e^{\dagger}} = \langle Insert, 0.5€, Coffee\rangle$, (iv) $w_0 = Insert$, and (v) α_0 is the set of edges in E without the deactivating edge $\langle e^{\dagger}, e^{\dagger}\rangle$.

A reactive graph has an initial state w_0 and an initial set of active edges α_0. Evolving a reactive graph means transitioning to a new state, connected by an enabled ground edge from w_0, and updating the set of active edges. We start by formalising the set of activate and deactivate edges by another given edge, and then formalise the evolution of reactive graphs.

Definition 2 (Activation and deactivation). *Given a reactive graph* M*, an edge* $e \in E_M$ *and a set of active edges* $\alpha \subseteq E_M$*, we define the set of edges* *activated by* e *(resp. deactivated by* e*), written* $\mathsf{on}(e,\alpha)$ *(resp.* $\mathsf{off}(e,\alpha)$*) as follows.*

$$\begin{aligned}
\mathsf{from}(e_s) &= \{e \mid \exists e_t \cdot \overline{e} = (e_s, e_t)\} \\
\mathsf{from}^*(e,\alpha) &= \bigcup_{r \in (\mathsf{from}(e) \cap \alpha)} \mathsf{from}^*(r, \alpha \backslash \{e\}) \cup \{r\} \\
\mathsf{on}(e,\alpha) &= \{e_t \mid e_{trg} \in \mathsf{from}^*(e,\alpha) \wedge \exists e_s \cdot \overline{e_{trg}} = (e_s, e_t) \in \twoheadrightarrow\!\!\!\!\rightarrow\} \\
\mathsf{off}(e,\alpha) &= \{e_t \mid e_{trg} \in \mathsf{from}^*(e,\alpha) \wedge \exists e_s \cdot \overline{e_{trg}} = (e_s, e_t) \in \multimap\}
\end{aligned}$$

Intuitively $\mathsf{from}(e_s)$ returns the hyper edges that start from e_s, from^* keeps traversing from to collect all (active) hyper edges triggered from a single edge, and $\mathsf{on}(e,\alpha)$ (resp. $\mathsf{off}(e,\alpha)$) collect all the targets triggered from e by an activating (resp. deactivating) edge. For example, in the vending machine in Fig. 1 we have that $\mathsf{off}(e_1, E) = \{e_1, e_2\}$, where $\overline{e_1} = \langle \mathit{Insert}, 1€, \mathit{Chocolate}\rangle$ and $\overline{e_2} = \langle \mathit{Insert}, 0.5€, \mathit{Coffee}\rangle$. This means that executing the edge from *Insert* to *Chocolate* triggers the deactivating edges e_1 and e_2. Using this notion of (de)activation, the evolution of a reactive graph is formalised below.

Definition 3 (Semantics). *The semantics of a reactive graph M is given by the evolution of a configuration $\langle w, \alpha\rangle$ of a state $w \in W$ and active edges $\alpha \subseteq E$, starting from the initial configuration $\langle w_0, \alpha_0\rangle$, given by the rule below.*

$$\frac{\exists e \in \alpha \ \cdot \ \overline{e} = \langle w, a, w'\rangle \ \wedge \ \alpha' = \big(\alpha \cup \mathsf{on}(e,\alpha)\big)\backslash \mathsf{off}(e,\alpha)}{\langle w, \alpha\rangle \xrightarrow{a}_M \langle w', \alpha'\rangle}$$

Using the semantics above to our reactive vending machine from Fig. 1 we obtain the LTS depicted in Fig. 2. This semantics differs from Gabbay's semantics by atomically collecting all activate edges before applying their (de)activations effects, instead of activating and deactivating edges during the traversal of triggered edges. It also introduces a bias: whenever an edge is both activated and deactivated in a step, deactivation takes precedence. However this may not be intended, which we will address in the next section over contradictory effects.

Relevant properties of reactive graph. As seen in Definition 3, the behaviour of a reactive graph $M = (W, \mathit{Act}, E, \rightarrow, \twoheadrightarrow\!\!\!\!\rightarrow, \multimap, \dot{\neg}, w_0, \alpha_0)$ from a configuration $\langle w, \alpha\rangle$ can be represented by the LTS induced by relation $\rightarrow_M = \bigcup\{\xrightarrow{a}_M \mid a \in \mathit{Act}\}$. Many standard properties of reactive graphs can be defined over the LTS induced by the semantics of reactive graph, namely:

Deadlocks. A deadlock is a state from which there is no transition (in our case an *active transition*), often undesirable. In reactive graphs we can also search for deadlocks by traversing the induced LTSs from w_0 while searching for states without outgoing transitions.

Unreachable states. An unreachable state, also undesirable in many systems, is a state that cannot be reached from the initial configuration.

Observational equivalence. As in standard LTS, two configurations are said to be equivalent if they behave in the same way. One way of defining such kind of equivalences is by means of their induced LTS: two configurations are behavioural equivalent if their induced LTS are bisimilar.

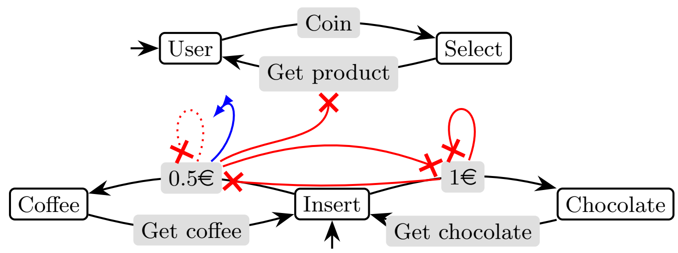

Fig. 3: An example of two reactive graphs and one intrusive edge

---

Other properties that can be analysed directly over reactive graphs include:

Contradictory effects. A contradictory effect is when a step triggers both the activation and deactivation of the same edge. The semantics in Definition 3 gives priority to disabling, but often these situations are the result of bad design decisions that should be avoided, and can be signalled as warnings.

Unreachable transitions. Similarly to unreachable states, (hyper) edges that cannot be fired are usually undesirable or a result of a bad understanding of a system. Hence it is a property that can also be investigated directly over reactive graphs.

Products on reactive graphs. Synchronous and asynchronous products of RG can be defined in the standard way. This section discusses a new product, called *intrusive product*. It allows the connections between two RGs, i.e., the execution of an action in a given machine can interfere with the activation/deactivation actions of the other machine, and vice-versa.

Definition 4. *Given two multi-action reactive graphs M_1, M_2, and $\Gamma^{\oplus}, \Gamma^{\ominus} \subseteq E_1 \times E_2 \cup E_2 \times E_1$ is the set of intrusive edges between M_1 and M_2. The effects produced by $e \in E_{M_i}$ in M_i is given for the set follow:*

$$\alpha_i(\Gamma^{\oplus}, \Gamma^{\ominus}, e) = \big(\alpha_i \cup \mathsf{on}(e, \alpha_i) \cup \Gamma^{\oplus}(e)\big) \backslash \big(\mathsf{off}(e, \alpha_i) \cup \Gamma^{\ominus}(e)\big)$$

Figure 3 illustrates an intrusive product of $Usr \mathbin{/\!/}_{\varnothing, \Gamma^{\ominus}} VM$, where Usr is the upper RG and $\Gamma^{\ominus} = \{\langle\langle Insert, \text{0.5€}, Coffee\rangle, \langle User, \text{Get product}, Select\rangle\rangle\}$.

Formally the asynchronous product ($\mathbin{/\!/}_{\Gamma^{\oplus}, \Gamma^{\ominus}}$) is defined by the rules below.

$$\frac{\exists\ e \in \alpha_1 \cdot \overline{e} = s_1 \xrightarrow{a} s_1' \wedge \alpha_1' = \big(\alpha_1 \cup \mathsf{on}(e, \alpha_1)\big) \backslash \mathsf{off}(e, \alpha_1) \wedge \alpha_2' = \alpha_2(\Gamma^{\oplus}, \Gamma^{\ominus}, e)}{\langle s_1, \alpha_1\rangle \mathbin{/\!/}_{\Gamma^{\oplus}, \Gamma^{\ominus}} \langle s_2, \alpha_2\rangle \xrightarrow{a} \langle s_1', \alpha_1'\rangle \mathbin{/\!/}_{\Gamma^{\oplus}, \Gamma^{\ominus}}, \langle s_2, \alpha_2'\rangle}$$

$$\frac{\exists\ e \in \alpha_2 \cdot \overline{e} = s_2 \xrightarrow{a} s_2' \wedge \alpha_2' = \big(\alpha_2 \cup \mathsf{on}(e, \alpha_2)\big) \backslash \mathsf{off}(e, \alpha_2) \wedge \alpha_1' = \alpha_1(\Gamma^{\oplus}, \Gamma^{\ominus}, e)}{\langle s_1, \alpha_1\rangle \mathbin{/\!/}_{\Gamma^{\oplus}, \Gamma^{\ominus}} \langle s_2, \alpha_2\rangle \xrightarrow{a} \langle s_1, \alpha_1'\rangle \mathbin{/\!/}_{\Gamma^{\oplus}, \Gamma^{\ominus}} \langle s_2', \alpha_2'\rangle}$$

The synchronous product ($\between_{\Gamma^{\oplus}, \Gamma^{\ominus}}$) is based on shared actions, defined below.

$$\frac{\begin{array}{l} \exists\ e_1 \in \alpha_1 \cdot \overline{e_1} = s_1 \xrightarrow{a} s_1' \\ \exists\ e_2 \in \alpha_2 \cdot \overline{e_2} = s_2 \xrightarrow{a} s_2' \end{array} \wedge \begin{array}{l} \alpha_1' = \big(\alpha_1 \cup \mathsf{on}(e_1, \alpha_1) \cup \Gamma^{\oplus}(e_1)\big) \backslash \big(\mathsf{off}(e_1, \alpha_1) \cup \Gamma^{\ominus}(e_1)\big) \\ \alpha_2' = \big(\alpha_2 \cup \mathsf{on}(e_2, \alpha_2) \cup \Gamma^{\oplus}(e_2)\big) \backslash \big(\mathsf{off}(e_2, \alpha_2) \cup \Gamma^{\ominus}(e_2)\big) \end{array}}{\langle s_1, \alpha_1\rangle \between_{\Gamma^{\oplus}, \Gamma^{\ominus}} \langle s_2, \alpha_2\rangle \xrightarrow{a} \langle s_1', \alpha_1'\rangle \between_{\Gamma^{\oplus}, \Gamma^{\ominus}} \langle s_2', \alpha_2'\rangle}$$

This product supports the modelling, e.g., of self-adaptive systems with a layer that manages a given system and the actual system being adapted [17].

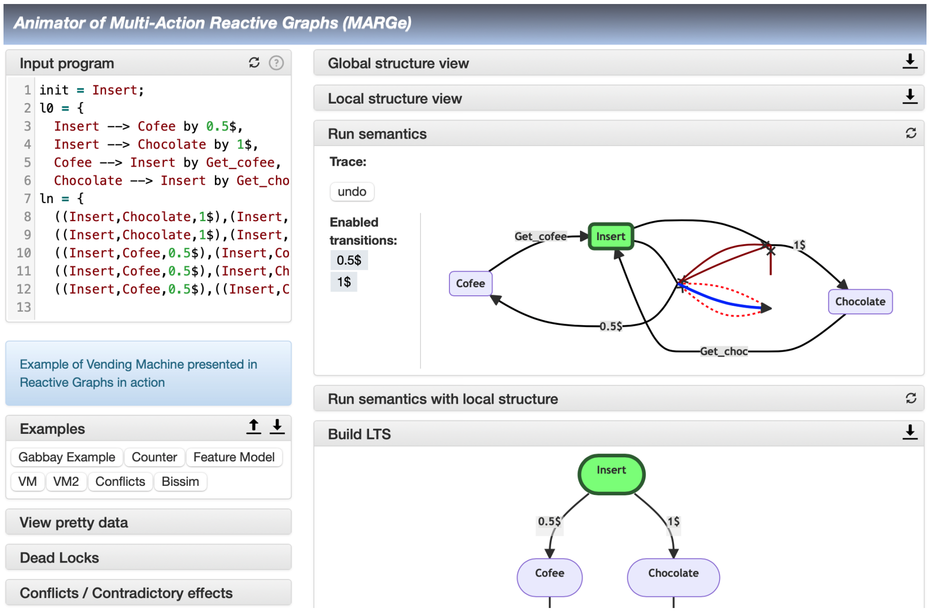

Fig. 4: Screenshot of the web interface of the Marge tool

---

3 The Marge tool

This section briefly presents the Marge tool and its features. A screenshot of its web interface can be found in Fig. 4 with the vending machine example from Fig. 1. Marge is open-source and developed in *Scala*, using the *Mermaid* library[3] to produces graphical representations of the reactive graph and its semantics. The tool is compiled to *JavaScript* that is used to build an interactive web page, using the CAOS library [18] which includes support to animate operational semantics and compare semantics.

Using the tool. The tool can be used at `https://fm-dcc.github.io/MARGe`, illustrated in Fig. 4 with our vending machine example. The model is introduced using a textual description in the "Input program" widget (top left). It can also be found in the list of examples (middle left). The remaining widgets provide our analysis and visualisations, and can be either collapsed (as the "Global structure view") or expanded (as the "Run semanitcs").

Available widgets. Some of the available widgets are described below.

- Input program – uses a textual notation, mimicking the mathematical structures, not yet optimised to be compact and maintainable.
- Global structure view – shows the graphical representation of a reactive graph. A simplified version without hyper edges and deactivated edges is depicted in the widget "local structure view".

[3] Mermaid is popular markup language for diagrams, cf. `https://mermaid.js.org`

– Run semantics – allows the user to simulate the reactive graph by selecting, at each step, an active transition that should be taken. After selecting this transition the graph is updated, including the active edges and the current state.
– Generated LTS – displays the underlying LTS by expanding all possible actions of the the reactive graph (up to a fixed bound).
– Number of states and edges – presents the number of states and edges of both the reactive graph and its encoded LTS. E.g., our vending machine as a similar number of states and edges. But a variation (available online) with a limited stock (instead of limited money) uses 4 states, 5 ground edges, and 3 hyper edges, against 19 states and 20 edges in the encoded LTS.
– Find strong bisimulation – checks if two reactive graphs separated by '$\sim$' in the input program are equivalent (i.e., bisimilar), providing either a bisimulation or an an explanation for not finding one.
– Conflicts/Contradictory effects – finds conflicts when they exist, i.e. traces until a transition that simultaneously tries to activate and deactivate.
– DeadLocks – checks the existence of deadlocks in the behaviour of a given reactive graph.
– Products – this a set of widgets, presenting the different types of product introduced in this paper.

4 Conclusions and future work

Reactive graphs can provide a compact and insightful representation of a variety of reconfigurable scenarios, e.g., in the context of communication protocols [11] and in a biological setting [20]. They are also closely related to van Benthem's game models with adversarial agents [4] and to Areces et al.'s logic with reconfiguring modal operators [1]. Extensions to reactive graphs with the paraconsistency paradigm have also been recently proposed [9]. Most work on reactive graphs is theoretical, and a small effort has been done to provide tool support and automatization of results. This paper presents the tool Marge with basic editor and exploration mechanisms of reactive graphs, including specific analysis such as a search for contradictory effects. As future work, we intent to improve the usability of Marge (e.g., improving the input language), extend it to support fuzzy extensions (to measure, e.g., costs and rewards from applying reconfigurations [20,6]), and to integrate a model checker according to a suitable logic. The latter would be similar to how we integrated the mCRL2 toolset [5] to analyse connectors [19] and team automata [2], based on a predecessor of CAOS [10].

Acknowledgments This work is supported by the FCT, the Portuguese funding agency for Science and Technology, with the projects UIDB/04106/2020 (https://doi.org/10.54499/UIDB/04106/2020), UIDP/04106/2020 (https://doi.org/10.54499/UIDP/04106/2020), and PTDC/CCI-COM/4280/2021. It was also supported by the CISTER Research Unit (UIDP/UIDB/04234/2020), financed by National Funds through FCT/MCTES and by project Ibex (ref. PTDC/CCI-COM/4280/2021) financed by national funds through FCT.

References

1. Areces, C., Fervari, R., Hoffmann, G.: Relation-changing modal operators. Log. J. IGPL **23**(4), 601–627 (2015). https://doi.org/10.1093/JIGPAL/JZV020
2. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Can we communicate? using dynamic logic to verify team automata. In: Chechik, M., Katoen, J.P., Leucker, M. (eds.) Proceedings of the 25th International Symposium on Formal Methods (FM 2023). Lecture Notes in Computer Science, vol. 14000. Springer (2023). https://doi.org/10.1007/978-3-031-27481-7_9
3. ter Beek, M.H., Damiani, F., Lienhardt, M., Mazzanti, F., Paolini, L.: Efficient static analysis and verification of featured transition systems. Empirical Software Engineering **27**(1), 10 (2021). https://doi.org/10.1007/s10664-020-09930-8
4. van Benthem, J.: An essay on sabotage and obstruction. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 2605, pp. 268–276. Springer (2005). https://doi.org/10.1007/978-3-540-32254-2_16
5. Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C.: The mCRL2 Toolset for Analysing Concurrent Systems. In: Vojnar, T., Zhang, L. (eds.) TACAS. LNCS, vol. 11428, pp. 21–39. Springer (2019). https://doi.org/10.1007/978-3-030-17465-1_2
6. Campos, S., Santiago, R.H.N., Martins, M.A., Figueiredo, D.: Introduction to reversal fuzzy switch graph. Sci. Comput. Program. **216**, 102776 (2022). https://doi.org/10.1016/J.SCICO.2022.102776
7. Capilla, R., Bosch, J., Trinidad, P., Ruiz-Cortés, A., Hinchey, M.: An overview of dynamic software product line architectures and techniques: Observations from research and industry. Journal of Systems and Software **91**, 3–23 (2014). https://doi.org/https://doi.org/10.1016/j.jss.2013.12.038
8. Cordy, M., Classen, A., Heymans, P., Legay, A., Schobbens, P.Y.: Model checking adaptive software with featured transition systems. Assurances for Self-Adaptive Systems: Principles, Models, and Techniques pp. 1–29 (2013)
9. Costa, D., Figueiredo, D., Martins, M.A.: Relation-changing models meet paraconsistency. J. Log. Algebraic Methods Program. **133**, 100870 (2023). https://doi.org/10.1016/J.JLAMP.2023.100870
10. Cruz, R., Proença, J.: Reolive: Analysing connectors in your browser. In: Mazzara, M., Ober, I., Salaün, G. (eds.) Software Technologies: Applications and Foundations - STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11176, pp. 336–350. Springer (2018). https://doi.org/10.1007/978-3-030-04771-9_25
11. Figueiredo, D., Martins, M.A., Barbosa, L.S.: A note on reactive transitions and reo connectors. In: de Boer, F.S., Bonsangue, M.M., Rutten, J. (eds.) It's All About Coordination - Essays to Celebrate the Lifelong Scientific Achievements of Farhad Arbab. Lecture Notes in Computer Science, vol. 10865, pp. 57–67. Springer (2018). https://doi.org/10.1007/978-3-319-90089-6_4
12. Gabbay, D.M.: Reactive kripke models and contrary to duty obligations. part a: Semantics. Journal of Applied Logic **11**(1), 103–136 (2013). https://doi.org/https://doi.org/10.1016/j.jal.2012.08.001
13. Gabbay, D.M.: Reactive Kripke Semantics. Cognitive Technologies, Springer (2013). https://doi.org/10.1007/978-3-642-41389-6

14. Gabbay, D.M., Marcelino, S.: Modal logics of reactive frames. Stud Logica **93**(2-3), 405–446 (2009). https://doi.org/10.1007/S11225-009-9214-1
15. Gabbay, D.M., Marcelino, S.: Global view on reactivity: switch graphs and their logics. Ann. Math. Artif. Intell. **66**(1-4), 131–162 (2012). https://doi.org/10.1007/S10472-012-9316-8
16. Harel, D., Pnueli, A.: On the development of reactive systems. In: Apt, K.R. (ed.) Logics and Models of Concurrent Systems. pp. 477–498. Springer Berlin Heidelberg, Berlin, Heidelberg (1985)
17. Päßler, J., ter Beek, M.H., Damiani, F., Tarifa, S.L.T., Johnsen, E.B.: Formal modelling and analysis of a self-adaptive robotic system. In: Herber, P., Wijs, A. (eds.) iFM 2023 - 18th International Conference, iFM 2023, Leiden, The Netherlands, November 13-15, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14300, pp. 343–363. Springer (2023). https://doi.org/10.1007/978-3-031-47705-8_18, `https://doi.org/10.1007/978-3-031-47705-8_18`
18. Proença, J., Edixhoven, L.: Caos: A reusable scala web animator of operational semantics. In: Jongmans, S.S., Lopes, A. (eds.) Coordination Models and Languages - 25th IFIP WG 6.1 International Conference, COORDINATION 2023, Held as Part of the 18th International Federated Conference on Distributed Computing Techniques, DisCoTec 2023, Portugal, June 19-23, 2023, Proceedings. Lecture Notes in Computer Science, vol. 13908, pp. 163–171. Springer (2023). https://doi.org/10.1007/978-3-031-35361-1_9
19. Proença, J., Madeira, A.: Taming hierarchical connectors. In: Hojjat, H., Massink, M. (eds.) Fundamentals of Software Engineering - 8th International Conference, FSEN 2019, Tehran, Iran, May 1-3, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11761, pp. 186–193. Springer (2019). https://doi.org/10.1007/978-3-030-31517-7_13
20. Santiago, R.H.N., Martins, M.A., Figueiredo, D.: Introducing fuzzy reactive graphs: a simple application on biology. Soft Comput. **25**(9), 6759–6774 (2021). https://doi.org/10.1007/S00500-020-05353-1
21. Tinoco, D., Madeira, A., Martins, M.A., Proença, J.: Reactive graphs in action (extended version). In: Marmsoler, D., Sun, M. (eds.) Formal Aspects of Component Software - 20th International Conference, FACS 2024, Milan, Italy, September 9-10, 2024, Revised Selected Papers. Lecture Notes in Computer Science, vol. 15189, pp. 97–105. Springer (2024). https://doi.org/10.1007/978-3-031-71261-6_6

A Modelling of a larger example in Marge

This sections presents an example describing a feature transitions system (FTS), borrowed from Cordy et al. [8, Fig. 1]. As in the original example, it uses a set of features that can be selected in an initial setup step, triggering the activation and deactivation of edges. Our reactive graph version of this FTS is depicted in Fig. 5. Variations of this model for *reconfigurable* FTS can also be made, where the feature selection can be modified outside the setup step.

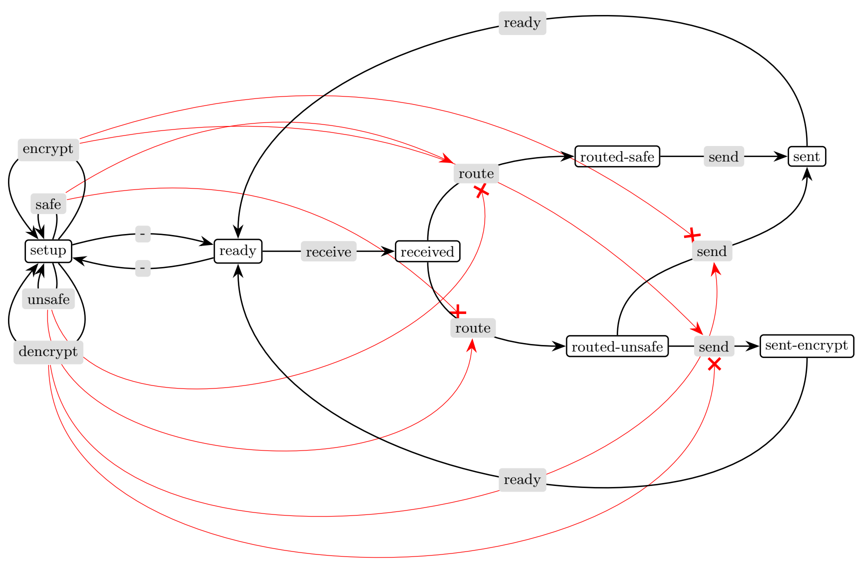

Fig. 5: Example adapted from [8]

A screenshot of Marge analysing this reactive graph is presented in Fig. 6.

In Fig. 7 we can see the explosion of states and the how reactive graphs can compactly represent this systems. The reactive graph has 7 states, 14 ground edges, and 8 hyper edges, while the encoded LTS has 51 states and 101 edges. One can confirm that this system has no deadlocks and no contradictory effects, using the widgets depicted in Fig. 8. This means that is always possible go to another state, regardless of the current state, and there is no transition that can trigger contradictory effects.

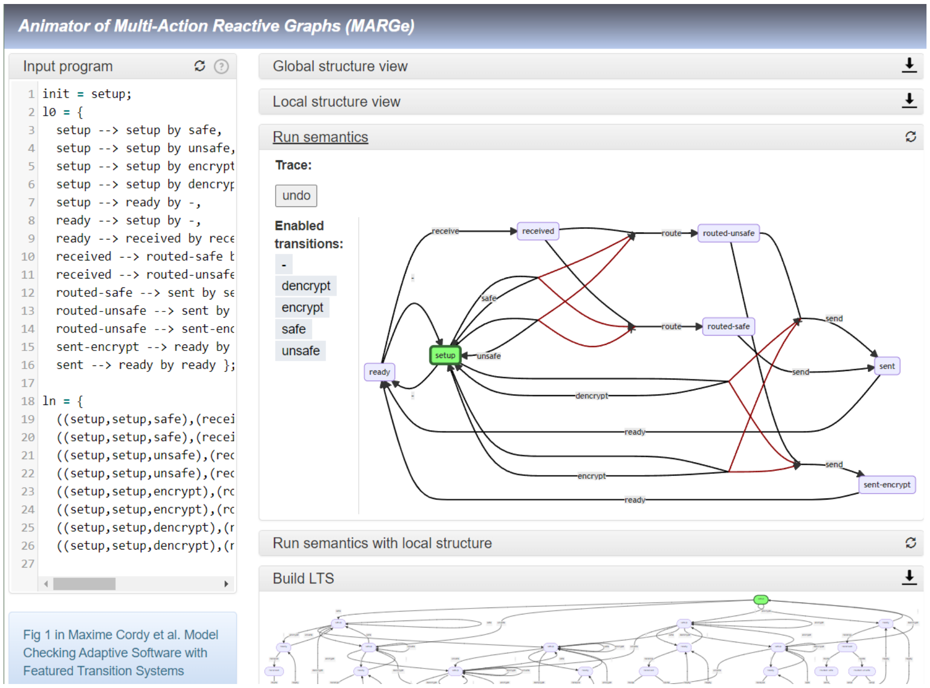

Fig. 6: Screenshot of Marge modelling a FTS

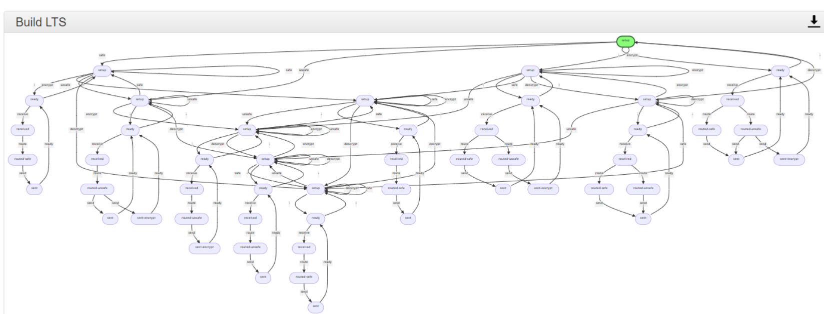

Fig. 7: Screenshot of the Build LTS widget with the LTS of the FTS example

Fig. 8: Screenshot of the widgets that search for deadlocks and contradictory effects