Statecharts and Ladder Logic Diagrams in Control of Industrial Processes

Kristijan-Igor Kopčok,

Dipl. Ing. ✉ Dept. of Automatic Control Systems, Faculty of Electrical Engineering and Information Technology, Slovak University of Technology Ilkovičova 3, 812 19 Bratislava, Slovakia 💻kopcok@kasr.elf.stuba.sk

1. Abstract

The logic controller is a supervisory system which controls parallel and synchronized sequences of elementary operations in the controlled system. Even though logic controllers are very important in the process of control in the industry, there is not yet a standard integrated tool, which is sufficiently powerful, versatile and simple to use, and with which it is possible to carry out formal analysis of correctness. In this paper we will present statecharts as tool for modeling and control design of industrial processes, their analysis method and their transformation into ladder logic i.e. the language of programmable logic controllers.

Keywords: statecharts, reachability tree, ladder logic, programmable logic controler.

2. Statecharts

Statecharts are extension of classical state machines. Their main advantage is the fact that they enable more simple specification and design of discrete event systems than other tools like for example Petri nets. Statecharts are modular and expressive i.e. it is possible to describe complex behavior of the system with relatively simple statechart. Formal definition of statecharts is given in Definition 1.

Definition 1: Statechart M is a 8-tuple $M = (Q, I, EV, S, O, \lambda, \xi, \alpha)$ where

Q is the global set of states;

I is teh set of transitions (names of transitions);

EV is the set of events to which statechart should react;

$S \in Q$ is the default input state;

O is the set of generated operations or events;

$\lambda: Q \times I \times EV \rightarrow Q \times O$ is the next state function;

$\xi: Q \rightarrow Q$ is the XOR hierarchical state function;

$\alpha: Q \rightarrow Q$ is the AND hierarchical state function;

It is an oriented chart in which nodes are called states and arcs are called transitions. Example of the simple statechart with it's basic elements is shown in Fig. 1. In this figure it can be seen that states (at any level of the hierarchical structure) are represented with round squares (Fig. 1. states A, B, C) and transitions are represented with oriented arcs. In general, arc can start and end at any level of the hierarchical structure. Every arc is labeled with event e.g. with event's name (Fig. 1. α, β, δ) or with it's name and the condition in parentheses. (Fig. 1. $\gamma(P)$). System can be in state A, B or C. This

depends from occurrence of events α, β, δ. For example, if system is in state B and event α occurs system will change it state to A.

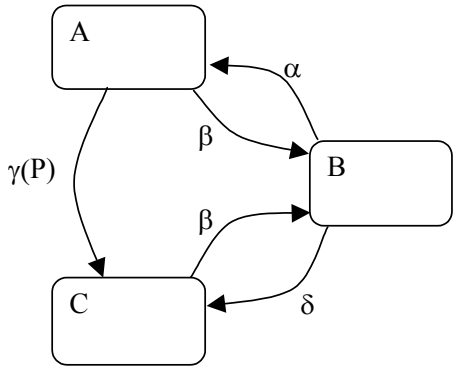

Fig. 1. Example of statechart

Two types of hierarchy e.g. decomposition are considered to make statechart:

- XOR decomposition
- AND decomposition

2.1. XOR decomposition

This decomposition means that a state at one level of the hierarchy corresponds to a state in the previous level in the same hierarchy. We can see example of XOR decomposition in Fig. 2. it can bee seen in this figure that event β transfers the system into the state B weather it is in the state A or state C at the moment of event occurrence. Considering this fact we can cluster states A and C into a new super-state D and replace two arcs β by one, as it is in Fig. 2. State D is the exclusive OR of A and C i.e. to be in state D system must be either in state A or in C, and not in both.

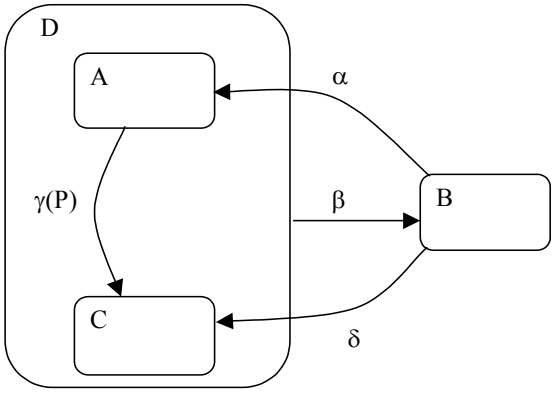

Fig. 2. . XOR refinement e.g. superstate

Another interesting statecharts elements are **default state** and **history** or **H block**. Default state is the solution for the problem of the initial state activation. It determinates which state will be active when system enters into superstate unless otherwise specified. Graphical representation of default state is as an oriented arc with a black dot and it is given Fig. 3.

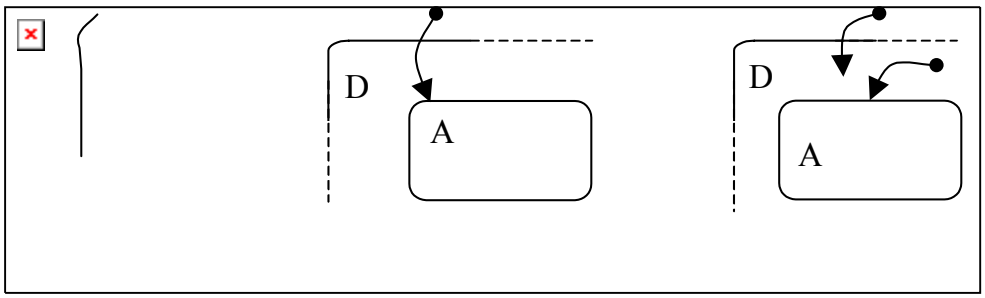

Fig. 3. Graphical representation of default state

History block allows entering a group of states by the system's history i.e. it will enter the state most recently visited.. History is applied only on the level in which it appears. Graphical representation of this block is by H in a circle as it is shown in Fig. 4. According to this figure case (a), when event α occurs system enters into superstate B. As the states C and D are components of this superstate and there is a history block, system will enter into state that was active for the last time of superstate B activation or into state C if it enters into B for the first time.

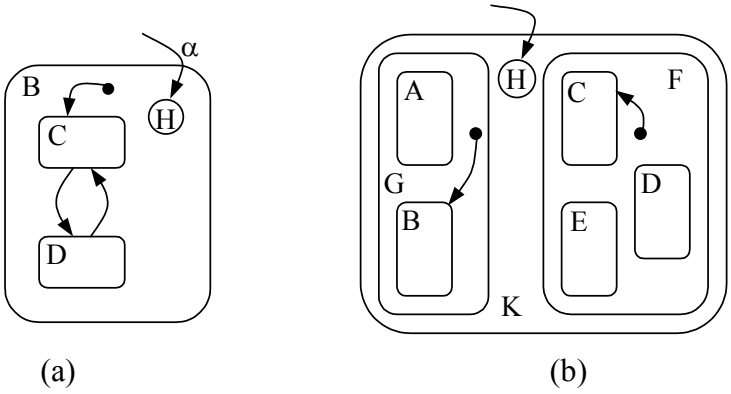

Fig. 5. Examples of H block

2.2. AND decomposition

This decomposition means that a state at one level corresponds to a fixed number of parallel states in the next level of the same hierarchy. Graphical representation of such state is the physical splitting of a box into components using dashed lines.

In Fig. 6. we can see a state Y consisting of AND components A and D, with the property that being in Y entails being in some combination B or C with E, F or G. We say that Y is the orthogonal product of A and D. The components A and D are no different conceptually from any other superstates.

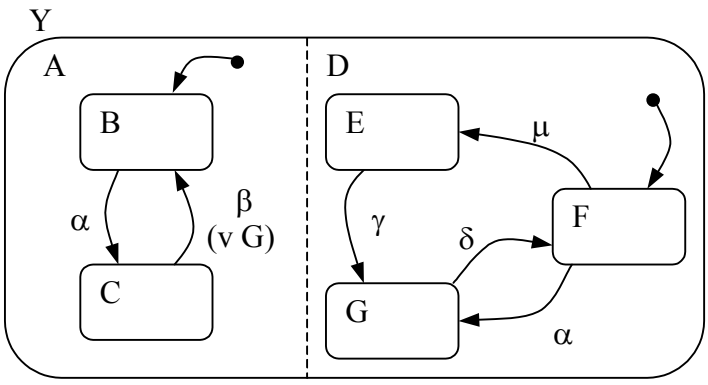

Fig.6.. AND decomposition or orthogonality

The same system but without AND decomposition is shown in Fig. 7. We can see that Fig. 7. contains 6 states because there are three and two states in components shown in Fig. 6. In case of two components with one thousand states each the result would be one million states.

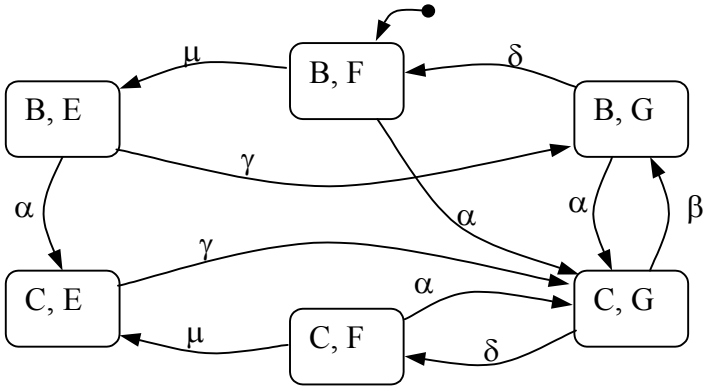

Fig. 7.. Statechart in Fig. 6. without AND decomposition

2.3. Temporal logic in statecharts

We can use temporal logic in conditions that are specifying system behavior in the past. For simple cases H block can solve this problem but for more complex problems we must use temporal logic. For example if we add following expression of temporal logic as a condition to a transition

$(\neg(\text{in A}) \wedge \neg(\text{in B}))$ since (D)

means that the state to which such transition leads will be activated only if sometimes in the past state D was activated and states A and B were never activated.

2.4. Actions and activities

Statecharts represents control part of system that is in charge of generating decisions and in that way influence behavior of the system. This property is secured through actions and activities. The action is split-second happening that take ideally zero time. An activity always takes a nonzero amount of time.

If we want a transition labeled a in one component of statechart to trigger another transition in an orthogonal component, without necessarily having any immediate external effects, we can simply label firs α/S and the second with S. Upon sensing a the first transition will be taken and the action S will be carried out, generating S as an event that can be sensed elsewhere. In the other component the S transition will be taken instantaneously. In Fig. 8. we can see examples of actions and activities.

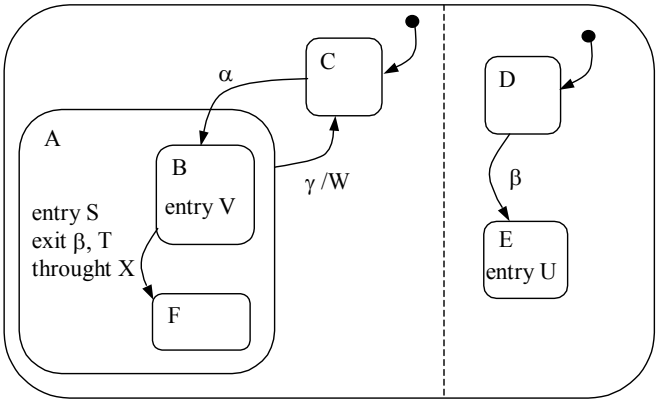

Fig. 8. Example of actions and activities

3. Reachability tree for statecharts

The construction if the reachability tree considers every possible computation sequence for the statechart. At each step we suppose that the event expression evaluates to true thus leading to a new configuration. Conditions are included in the event expressions and are also considered when firing a transition. Fig 9. Illustrates a statechart specifying the problem of two processes sharing two resources. Two processes run concurrently and this is modeled by the AND state P1×P2. P1 and P2 are XOR states as they may be in only one of their three basic states. In Fig. 10. we can see the reachability tree for statechart shown in Fig. 9.

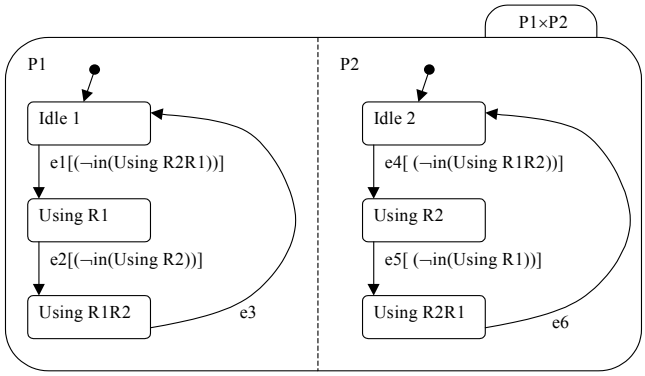

Fig. 9. Example of two processes and two resources

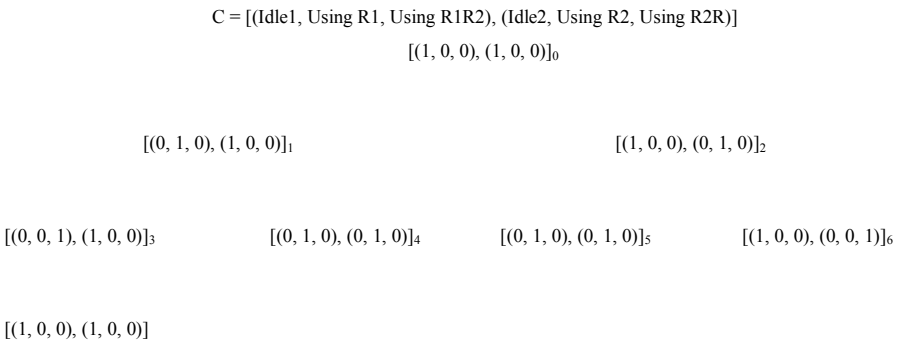

Fig. 10. Reachability tree for statechart form Fig. 9.

3.1. Dynamic properties verified through the reachability tree

Valid sequence of events: A sequence of event expressions is valid if each event may cause a transition to fire yielding a configuration change.

Reachability: A state configuration C_k is reachable from another state configuration C_i if there is at least one valid sequence of events s_1, s_2, ..., s_n and a set of intermediate state configurations $\{C_i, C_{i+1}, ..., C_{i+n}\}$ such that $C_i \xrightarrow{S_1} C_{i+1} \xrightarrow{S_2} C_{i+2} \xrightarrow{S_n} C_{i+n} = C_k$. The state configuration C_i may be the statechart's initial state configuration or any other configuration.

Reinitiability: A statechart is reinitiable when for each state configuration C_i reached from initial configuration C_0, there is a sequence of events that leads back to C_0.

Deadlock: A statechart has a deadlock if its execution may reach a state configuration in which progress can not be made, because no transition is able to fire.

Usage of transitions: A transition is used if it appears in at least one execution path of its statechart.

4. Transformation of statecharts to Ladder logic

After creating and analysing a statechart model we can perform its transformation to ladder logic. Simple example of this transformation is given in Fig. 11.

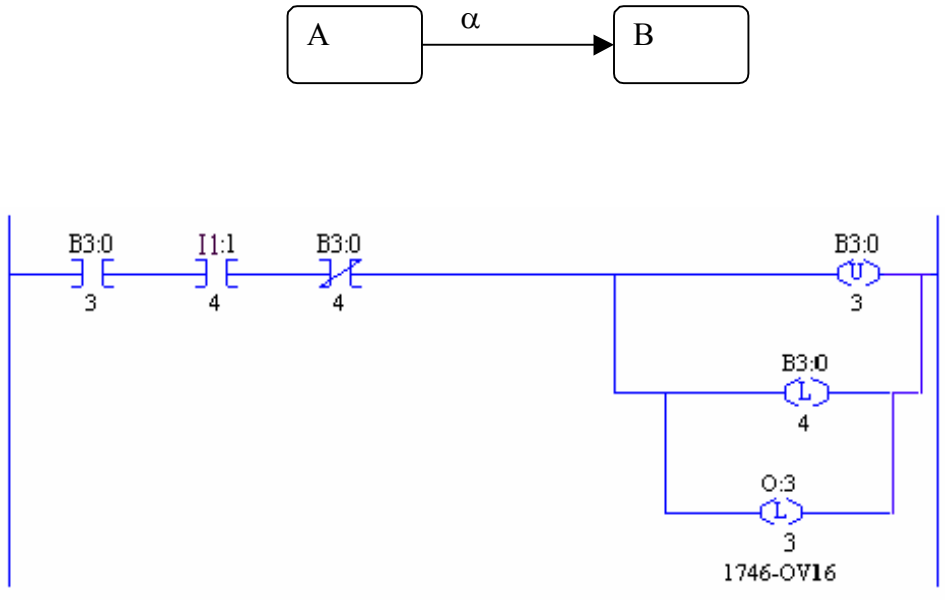

Fig. 11. Transformation of statecharts to ladder logic

In the ladder logic we can divide every rung of the ladder into two parts. Left part of ladder contains testing conditions and the right part of ladder contains actions and activities. If the left part of ladder is evaluated as true than the right part of ladder is executed.

We will explain transformation using example shown in Fig. 11. Statechart that we will transform has two states A and B and one transition activated with event α. In the ladder logic we will use internal binary variabiles to indicate the state of the statechart states and physical inputs will test the occurrence of the external events. In this example state A will be represented with variable B3:0/3, state B will be represented with variable B3:0/4 and event α will be represented with input signal I1:1/4.

In the left part of ladder we are testing wether state A and event α are active and state B is not active. If this condition is fulfill State A will be deactivated and state B will be activated.

5. References

[1] Harel, D.: Statecharts: A Visual Formalism for Complex System. Science of Computer Programing, Vol 8, pp. 231-274, 1987.

[2] Berio, G., Vernadat, F. B.: Formal Foundation for a Process/Resource Approach in Manufacturing Systems Behavior Modelling. 14th Triennal World Congress of IFAC, Bejing, P. R. China, 1999.

[3] Masiero, P. C., Maldonado, J. C., Boaventura, I. G. : A Reachability Tree for Statecharts and Analysis of Some Properties. Information and Software Technology, Vol 10, pp. 615-624, 1994.