Constraint Programming for Controller Synthesis

Gérard Verfaillie and Cédric Pralet

ONERA - The French Aerospace Lab, F-31055, Toulouse, France
{Gerard.Verfaillie,Cedric.Pralet}@onera.fr

Abstract. In this paper, we show how the problem of synthesis of a controller for a dynamic system that must satisfy some safety properties, possibly in a non deterministic and partially observable setting, can be modeled as a pure constraint satisfaction problem, by replacing the reachability property by a so-called weak reachability property. We show, first on a toy illustrative example, then on a real-world example of control of a satellite subsystem, how standard constraint programming tools can be used to model and solve the controller synthesis problem. Finally, we conclude with the strengths and weaknesses of the proposed approach.

1 The controller synthesis problem

1.1 An informal view

In this paper, we are interested in the closed loop control of dynamic systems (see Fig. 1). More precisely, we are interested in software controllers that are implemented on digital computers (most of the modern controllers) and that do not act continuously on the systems they control, but in a discrete way, by successive steps.

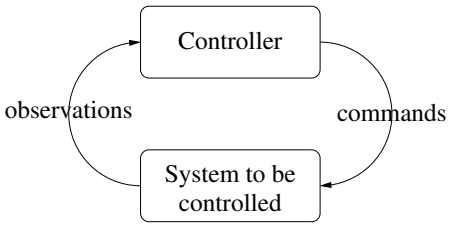

Fig. 1. Closed loop control of a dynamic system

At each step, the controller collects information on the system it controls (observations) and makes reactively a control decision (commands) as a function of the collected information. Observations may come from the system itself, from other systems with which the system interacts, or from human operators. They may contain synthetic information on past observations or commands (controller

memory). In the opposite direction, commands are sent to the system itself, to other systems, or to operators.

We assume that the system evolution that follows a command (system transition) is not deterministic and Markovian: several transitions are possible from the current state and command; the actual transition depends only on the current state and command; it does not depend on previous ones. However, we do not assume the existence of probability distributions on the set of possible transitions: only possible and impossible transitions are distinguished. Moreover, we do not assume that the actual state of the system is known at each step by the controller: only observations are available. We assume that the set of possible system states, the set of possible observations, and the set of possible commands are all discrete and finite.

We are interested in safety properties, that is in properties that must be satisfied by any transition of the controlled system. In such a setting, the controller synthesis problem consists in building off-line, before bringing the system into service, what is called a policy, that is a function which associates with each observation a command and which guarantees that, in spite of non deterministic transitions, every transition satisfies the safety properties.

Sect. 1 introduces the controller synthesis problem. Sect. 2 presents a CSP formulation of this problem, based on the notion of weak reachability. Sect. 3 shows how the proposed approach can be applied to the problem of control of a satellite subsystem. Sect. 4 concludes with the strengths and weaknesses of the proposed approach.

1.2 A formal definition

The controller synthesis problem can be formally defined as follows.

Problem data is:

- a finite sequence S of state variables;
- a sub-sequence $O \subseteq S$ of observable state variables;
- a finite sequence C of command variables;
- a set $I \subseteq \mathbf{d}(S)$ of possible initial states, assumed not to be empty;
- a set $T \subseteq \mathbf{d}(S) \times \mathbf{d}(C) \times \mathbf{d}(S)$ of possible transitions;
- a set $P \subseteq \mathbf{d}(S) \times \mathbf{d}(C) \times \mathbf{d}(S)$ of acceptable transitions.

Each state or command variable is assumed to have a finite domain of value[1]. Sets I, T, and P can be implicitly defined by finite sets of constraints[2].

A policy π is a partial function from $\mathbf{d}(O)$ to $\mathbf{d}(C)$. Let $df_\pi \subseteq \mathbf{d}(O)$ be the domain of definition of π.

[1] If x is a variable, $\mathbf{d}(x)$ denotes its domain. If X is a sequence of variables, $\mathbf{d}(X)$ denotes the Cartesian product of the domains of the variables in X. If X is a sequence of variables and Y a sub-sequence of X and if A is an assignment of X, $A_{\downarrow Y}$ denotes the assignment of Y (projection).

[2] We use the same notation for a set and its characteristic function: If S is a set and $SS \subseteq S$, for all $e \in S$, $SS(e)$ is true if and only if $e \in SS$.

Given a policy π, the set r_π of the states that are reachable from an initial state by following π can be defined as follows. If $r_{\pi,k}$ is the set of states that are reachable in less than k steps from an initial state, we have:

$$\forall s \in \mathbf{d}(S),\, r_{\pi,0}(s) = I(s) \tag{1}$$

$$\begin{aligned}&\forall k, 1 \leq k \leq |\mathbf{d}(S)| - 1,\, r_{\pi,k}(s) = r_{\pi,k-1}(s) \vee \\ &(\exists s' \in \mathbf{d}(S),\, r_{\pi,k-1}(s') \wedge df_\pi(s'_{\downarrow O}) \wedge T(s', \pi(s'_{\downarrow O}), s))\end{aligned} \tag{2}$$

$$r_\pi(s) = \max_{0 \leq k \leq |\mathbf{d}(S)|-1} r_{\pi,k}(s) \tag{3}$$

Eq. 1 expresses that a state is reachable in 0 step if and only if it is a possible initial state. Eq. 2 expresses that a state is reachable in less than k steps if and only if it is reachable in less than $k-1$ steps or if a transition is possible from a state that is is reachable in less than $k-1$ steps. Finally, Eq. 3 says that a state is reachable if and only if it is reachable in less than k steps, with $0 \leq k \leq |\mathbf{d}(S)| - 1$. Indeed, for any state $s \in S$, either it is not reachable, or it is in at most $|\mathbf{d}(S)| - 1$ steps. It must be stressed that, according to this definition, the set of reachable states depends on the chosen policy π.

Requirements on policy are the following:

$$\forall s \in \mathbf{d}(S),\, r_\pi(s) \rightarrow df_\pi(s_{\downarrow O}) \tag{4}$$

$$r_\pi(s) \rightarrow (\exists s' \in \mathbf{d}(S),\, T(s, \pi(s_{\downarrow O}), s')) \tag{5}$$

$$r_\pi(s) \rightarrow (\forall s' \in \mathbf{d}(S),\, T(s, \pi(s_{\downarrow O}), s') \rightarrow P(s, \pi(s_{\downarrow O}), s')) \tag{6}$$

Eq. 4 specifies that the policy shall be defined for all the reachable states. Eq. 5 specifies that the policy shall not lead to dead ends: for each reachable state, by following the policy, there is a possible transition. Finally, Eq. 6 enforces that the policy be "acceptable": for each reachable state, by following the policy, every possible transition is acceptable. It must be stressed that these requirements shall be satisfied, not on all the possible states, but only on those that are reachable from an initial state by following the chosen policy π.

A policy is valid if and only if it satisfies requirements 4, 5, and 6. The objective of controller synthesis is to produce a valid policy or to prove that such a policy does not exist.

1.3 A toy example

As an example, let us consider the toy example used in [16]. We consider a robot that is able to move on the grid of Fig. 2 where walls are shown in bold. Initially,

the robot is on one of the places of x-coordinate 2. At each step, it is on a place p that it does not know directly. It only observes the walls immediately around p. At each step, it moves north, south, east, or west. It cannot stay where it is. Because of the presence of walls, some moves are not feasible. The robot shall avoid the place marked X $(x = 3, y = 1)$ that is considered to be dangerous.

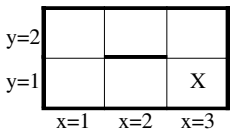

Fig. 2. Robot control problem on a grid

To model this problem, we consider six state variables: $S = \{x, y, w_N, w_S, w_E, w_W\}$. x and y represent the robot position, with $\mathbf{d}(x) = [1..3]$ and $\mathbf{d}(y) = [1..2]$. w_N, w_S, w_E, and w_W are Boolean variables that represent the presence or the absence of a wall at north, south, east, and west of the current place. Only state variables w_N, w_S, w_E, and w_W are observable by the robot: $O = \{w_N, w_S, w_E, w_W\}$. We consider only one command variable m which represents the robot move, with $\mathbf{d}(m) = \{m_N, m_S, m_E, m_W\}$ (four possible moves).

The set I of the possible initial states is defined by the following five unary constraints: $x = 2$, w_N, w_S, $\neg w_E$, and $\neg w_W$.

The set T of the possible transitions is defined by the following constraints:

$$w_N \rightarrow (m \neq m_N),\ w_S \rightarrow (m \neq m_S),\ w_E \rightarrow (m \neq m_E),\ w_W \rightarrow (m \neq m_W)$$

$$x' = x + (m = m_E) - (m = m_W),\ y' = y + (m = m_N) - (m = m_S)$$

$$w'_N = (y' = 2 \vee (y' = 1 \wedge x' = 2)),\ w'_S = (y' = 1 \vee (y' = 2 \wedge x' = 2))$$
$$w'_E = (x' = 3),\ w'_W = (x' = 1)$$

The first four constraints (first line) express the feasible moves, taking into account the possible presence of walls. The following two (second line) express the possible transitions which, in this case, are all deterministic. For each state variable z, z' represents its value at the next step. We assume that a constraint returns value 1 if it is satisfied and value 0 otherwise. The last four constraints (last two lines) result from the available knowledge of the grid topology.

The set P of the acceptable transitions is defined by the binary constraint $\neg((x' = 3) \wedge (y' = 1))$.

1.4 Existing methods

The method that is by far the most used to build a controller consists in specifying it using any general or specific purpose programming language, for example

one of the family of the synchronous languages [9]. Once the controller programmed, its properties can be checked, either experimentally by simulation, or formally by using proof or model-checking tools [4].

Controller synthesis is an alternative approach which aims at building a controller automatically from the properties of the system to be controlled and from the requirements on the controlled system. The resulting controller is valid by construction and no further check is theoretically necessary on it.

The first works on controller synthesis are based on automata and language theory [18]. Then, many works use automata to model the physical system and temporal logics to specify requirements [14] (see for example the ANZU et RATSY tools [11,19], which both assume complete state observability). The MBP tool (Model-Based Planner) uses symbolic model-checking techniques, based on BDDs (Binary Decision Diagrams) to synthesize controllers that guarantee a goal to be reached in spite of non determinism and partial observability [2,13]. Generic search algorithms for the synthesis of finite memory controllers are proposed in [3,16], with the same assumptions of non determinism and partial observability (but with some restrictions in [3]).

The proximity between the controller synthesis problem and (PO)MDP ((Partially Observable) Markov Decision Processes [17]) must be emphasized. Dynamic programming algorithms are the most used to solve (PO)MDP. The first difference between both problems is that, in (PO)MDP, conditional probability distributions on the states resulting from a transition and on the observations resulting from a state are assumed to be available. The second one is that requirements take in (PO)MDP the form of an additive global criterion to be optimized.

The difference between the controller synthesis problem and the planning problem in Artificial Intelligence [8] must be emphasized too. In planning, we are interested in reachability properties: a goal state must be reached and the control is assumed to be stopped once the goal reached. In controller synthesis, we are interested in safety properties which must be satisfied along the whole system trajectory and the control is assumed to never stop.

2 Formulation as a constraint satisfaction problem

2.1 First formulation

Let us associate with each observation $o \in \mathbf{d}(O)$ a variable $\pi(o)$ of domain $\mathbf{d}(C) \cup \{\perp\}$ which represents the command to be applied when o is observed. Value $\perp$ represents the absence of command. Let us associate with each state $s \in \mathbf{d}(S)$ the following variables:

- a Boolean variable $r_\pi(s)$ which represents the fact that s is reachable or not;
- for each $k, 0 \leq k \leq |\mathbf{d}(S)| - 1$, a Boolean variable $r_{\pi,k}(s)$ which represents the fact that s is reachable or not in less than k steps.

If we replace $df_\pi(o)$ by $\pi(o) \neq \bot$, Eqs. 1 to 6 define a constraint satisfaction problem P (CSP [20]) which models exactly the controller synthesis problem. Unfortunately, the number of variables of P is prohibitive, mainly due to variables $r_{\pi,k}(s)$: for each $s \in \mathbf{d}(S)$, we have $|\mathbf{d}(S)|$ such variables. As a result, the number of variables $r_{\pi,k}(s)$ is equal to $|\mathbf{d}(S)|^2$. To get round this difficulty, we are going to use a relaxation of the reachability property we will refer to as weak reachability.

2.2 Reachability and weak reachability

We use the following definition of weak reachability: a relation wr_π is a weak reachability relation associated with a policy π if and only if it satisfies the following two equations:

$$\forall s \in \mathbf{d}(S), \quad I(s) \rightarrow wr_\pi(s) \tag{7}$$

$$\forall s, s' \in \mathbf{d}(S), (wr_\pi(s) \wedge df_\pi(s_{\downarrow O}) \wedge T(s, \pi(s_{\downarrow O}), s')) \rightarrow wr_\pi(s') \tag{8}$$

Eq. 7 expresses that, if a state is a possible initial state, it is weakly reachable. Eq. 7 expresses that, if a state s is weakly reachable and if a transition is possible from state s to another state s', state s' is weakly reachable too. From this definition of weak reachability, the following four properties can be established.

Property 1 *Let π be a policy. The associated reachability relation r_π is unique. On the contrary, several associated weak reachability relations wr_π may exist.*

The reachability relation is unique because it is defined by Eqs. 1 to 3 which are all equality equations. The possible existence of several weak reachability relations is shown in the example of Fig 3 where the reachability graph associated with a policy is displayed. In this graph, nodes represent states and arcs represent possible transitions.

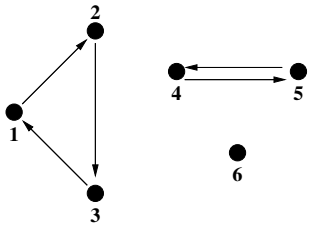

Fig. 3. Example of reachability graph associated with a policy

On this example, if 1 is the only possible initial state, reachability relation r is defined by the set of states $\{1, 2, 3\}$. However, the four relations defined by the set of states $\{1, 2, 3\}$, $\{1, 2, 3, 4, 5\}$, $\{1, 2, 3, 6\}$, and $\{1, 2, 3, 4, 5, 6\}$ all satisfy Eqs. 7 and 8 and thus are all weak reachability relations.

Property 2 *Let π be a policy, r_π be the associated reachability relation, and wr_π be any associated weak reachability relation. r_π is a subset of wr_π.*

To establish this property, let us prove that $\forall s' \in \mathbf{d}(S), r_\pi(s') \rightarrow wr_\pi(s')$.

If $r_\pi(s')$, there exists k, $0 \leq k \leq |\mathbf{d}(S)| - 1$ such that $r_{\pi,k}(s')$ (according to Eq. 3). Let us prove by recurrence on k that $\forall k \geq 0, \forall s' \in \mathbf{d}(S), r_{\pi,k}(s') \rightarrow wr_\pi(s')$.

For $k = 0$, if $r_{\pi,0}(s')$, we have $I(s')$ (according to Eq. 1) and thus $wr_\pi(s')$ (according to Eq. 7).

For $k > 0$, let us assume that $\forall s' \in \mathbf{d}(S), r_{\pi,k-1}(s') \rightarrow wr_\pi(s')$. If $r_{\pi,k}(s')$, we have (according to Eq. 2), either $r_{\pi,k-1}(s')$ and thus $wr_\pi(s')$ (according to the recurrence assumption), or $\exists s \in \mathbf{d}(S)\,,\ r_{\pi,k-1}(s) \wedge df_\pi(s_{\downarrow O}) \wedge T(s, \pi(s_{\downarrow O}), s')$ and thus $\exists s \in \mathbf{d}(S)\,,\ wr_\pi(s) \wedge df_\pi(s_{\downarrow O}) \wedge T(s, \pi(s_{\downarrow O}), s')$ (still according to the recurrence assumption) and finally $wr_\pi(s')$ (according to Eq. 8). From that, we can deduce that $\forall s' \in \mathbf{d}(S), r_{\pi,k}(s') \rightarrow wr_\pi(s')$.

As a consequence, $\forall s' \in \mathbf{d}(S), r_\pi(s') \rightarrow wr_\pi(s')$.

Property 3 *Let π be a policy. The associated reachability relation r_π is an associated weak reachability relation.*

To establish it, it suffices to prove that r_π satisfies Eqs. 7 and 8 which define weak reachability. r_π satisfies them due to the definition of reachability (Eqs. 1 to 3).

Property 4 *Let π be a policy. The associated reachability relation r_π is the unique smallest weak reachability relation associated with π (smallest with regard to inclusion and thus to cardinality).*

This property is the immediate consequence of properties 1, 2, and 3.

2.3 Second formulation

By using the notion of weak reachability, we propose the following formulation of the controller synthesis problem.

This formulation uses two sets of variables:

- for each observation $o \in \mathbf{d}(O)$, a variable $\pi(o)$ of domain $\mathbf{d}(C) \cup \{\perp\}$ which represents the command to be applied when o is observed;
- for each state $s \in \mathbf{d}(S)$, a Boolean variable $wr_\pi(s)$ which represents the fact that s is weakly reachable or not.

The constraints to be satisfied are defined by the following equations:

$$\forall s \in \mathbf{d}(S), \quad I(s) \rightarrow wr_\pi(s) \tag{9}$$

$$\forall s, s' \in \mathbf{d}(S), (wr_\pi(s) \wedge T(s, \pi(s_{\downarrow O}), s')) \rightarrow wr_\pi(s') \tag{10}$$

$$\forall s \in \mathbf{d}(S),\, wr_\pi(s) \rightarrow (\pi(s_{\downarrow O}) \neq \bot) \tag{11}$$

$$wr_\pi(s) \rightarrow (\exists s' \in \mathbf{d}(S),\, T(s, \pi(s_{\downarrow O}), s')) \tag{12}$$

$$wr_\pi(s) \rightarrow (\forall s' \in \mathbf{d}(S),\, T(s, \pi(s_{\downarrow O}), s') \rightarrow P(s, \pi(s_{\downarrow O}), s')) \tag{13}$$

Eqs. 9 and 10 are copies of Eqs. 7 and 8 which define weak reachability. Eqs. 11 to 13 are copies of Eqs. 4 to 6 which define requirements on policy, where r_π is replaced by wr_π. In Eq. 11, $df_\pi(o)$ is replaced by the equivalent formulation $(\pi(o) \neq \bot)$. Eq. 10 is finally simplified to take into account Eq. 11.

Whereas the previous CSP P (see Sect. 2.1) involved $|\mathbf{d}(O)|+|\mathbf{d}(S)|+|\mathbf{d}(S)|^2$ variables, this CSP P' involves only $|\mathbf{d}(O)| + |\mathbf{d}(S)|$ variables.

We are going to show that, as P does, P' models exactly the controller synthesis problem, that is that solving P' in order to solve the controller synthesis problem is correct, complete, and possibly optimal. Correctness means that, if P' has a solution Sol, the projection of Sol on policy variables π is a solution of the controller synthesis problem. Completeness means that, if the controller synthesis problem has a solution, P' has a solution too. Optimality means that the produced policy is defined only for reachable observations (associated with at least one reachable state).

2.4 Correctness, completeness, and optimality

Correctness results from Prop. 2. Let us indeed assume that P' has a solution, made of a policy π and of a relation wr_π. According to Eqs. 9 and 10, relation wr_π is a weak reachability relation. Let r_π be the reachability relation associated with π. According to Prop. 2, we have : $\forall s \in \mathbf{d}(S), r_\pi(s) \rightarrow wr_\pi(s)$. Hence, relation r_π satisfies the requirements associated with Eqs. 11 to 13. As a consequence, π is a solution of the controller synthesis problem.

As for completeness, it results from Prop. 3. Let us indeed assume that the controller synthesis problem has a solution, made of a policy π and of the associated reachability relation r_π. According to Prop. 3, r_π is a weak reachability relation. Hence, π and r_π make up together a solution of P'. P' is thus consistent and a complete constraint solver is able to produce a solution.

It remains that, if P' is consistent, the produced policy π may be defined for unreachable observations (associated with no reachable states). Practically, there is no issue because these observations will never be reached by following π. However if, for the sake of readability or compactness, we prefer a policy π that is defined only for reachable observations, it suffices to replace $wr_\pi(s) \rightarrow (\pi(s_{\downarrow O}) \neq \bot)$ by $wr_\pi(s) \leftrightarrow (\pi(s_{\downarrow O}) \neq \bot)$ in Eq. 11 and to transform the resulting constraint satisfaction problem into a constraint optimization problem where the criterion to be minimized is the number of weakly reachable states ($\sum_{s \in \mathbf{d}(S)} wr_\pi(s)$). If we get an optimal solution with optimality proof, we have, according to Prop. 4,

the guarantee that the produced relation wr_π is the reachability relation r_π and that the produced policy π is defined only for reachable observations.

2.5 OPL model

We used the OPL language [10] to express the constraint satisfaction (optimization) model associated with any given controller synthesis problem. Any other constraint programming language could have been used. These models are optimized in order to limit as much as possible the number of resulting CSP constraints. For the moment, these models are built manually, what is a potential source of errors. The automatic construction of an OPL model from the sets of variables S, O, and C and from the sets of constraints that define relations I, T, and P could be however considered.

On the toy example of Sect. 1.3, this model generates 112 variables and 980 constraints. It is solved by the CP Optimizer tool associated with OPL in less than 1/100 second. The policy produced is the following one:

$$w_N \wedge w_S \wedge \neg w_E \wedge \neg w_W : \ m = m_W$$
$$w_N \wedge \neg w_S \wedge \neg w_E \wedge w_W : \ m = m_S$$
$$\neg w_N \wedge w_S \wedge \neg w_E \wedge w_W : \ m = m_N$$

Each line is associated with a weakly reachable observation. On each line, the observation appears before the colon and the associated command after. For example, the first line specifies that, if walls are observed at north and south and not at east and west, the robot shall move west. This policy consists in moving west and then in alternating north and south moves. In this case, the policy produced without optimization (no optimization criterion) is luckily "optimal": it is defined for only reachable observations.

3 A real-world example

To validate the proposed approach, we made use of it on a real-world problem of control of a satellite subsystem, previously introduced in [15].

The context is an Earth watching satellite whose mission is to detect and to observe hot spots at the Earth surface, due to forest fires or volcanic eruptions [5]. It is equipped with a wide swath detection instrument and with a narrow swath observation one. In case of detection of a hot spot, an alarm must be sent to the ground using a geostationary relay satellite and observations of the ground area must be performed. Observation data is then downloaded towards ground stations during visibility windows. In such a context, we are interested in an equipment referred to as DSP (Digital Signal Processor) in charge of the analysis of the images produced by the detection instrument and of the detection of hot spots in these images. The DSP is made of three elements:

– an analyser in charge of image analysis itself;

– a circuit which supplies the analyser with the necessary current;
– a switch which allows the circuit to be open or not and thus tension to be present or not at the ends of the circuit.

The DSP control module receives ON or OFF requests from the detection control module. At each step, it produces several outputs (see Fig. 4):

– a command to the DSP switch;
– signals towards the detection control module, giving information about the correct or incorrect behavior of the DSP;
– a signal towards the alarm control module, giving information about the detection or not of hot spots.

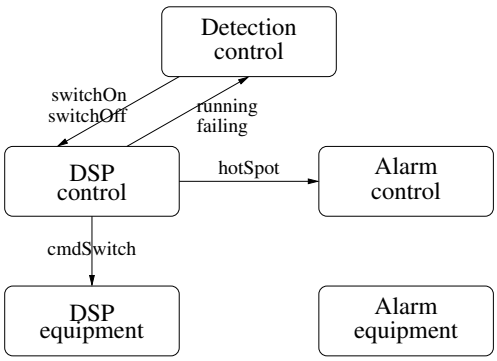

Fig. 4. Inputs and outputs of the module in charge of controlling the DSP

Each of the three elements that make the DSP may fail. When the analyser does not fail and receives current, hot spot detection is assumed to run correctly. When the analyser fails or does not receive current, hot spot detection does not run. When the circuit fails, the analyser does not receive current. When the switch fails, tension in the circuit may be inconsistent with the switch command.

Informally speaking, the highest level safety properties that must be satisfied at each step are the following: when the DSP is ON and no element fails, detection shall be correct (hot spot detection signal in case of hot spot; no detection signal, otherwise); when the DSP is OFF, it shall detect nothing.

To model this problem, we use the following state variables (set S):

– $switchOn \in \{0, 1\}$: presence or not of an ON request: 1 for presence;
– $switchOff \in \{0, 1\}$: presence or not of an OFF request: 1 for presence;
– $switched \in \{0, 1\}$: last ON or OFF request received by the controller (controller memory): 1 for ON and 0 for OFF;
– $tension \in \{0, 1\}$: presence or not of tension at the ends of the circuit: 1 for presence;

- *current* $\in \{0,1\}$: presence or not of current in the circuit: 1 for presence;
- *faultAnalyser* $\in \{0,1\}$: analyser failure or not: 1 for failure;
- *faultCircuit* $\in \{0,1\}$: circuit failure or not: 1 for failure;
- *faultSwitch* $\in \{0,1\}$: switch failure or not: 1 for failure;
- *inputIm* $\in \{NOIM, NORM, HOT\}$: type of the analyser input image: $NOIM$ in case of absence of image, $NORM$ in case of an image without hot spot, HOT in case of an image with hot spot; to control the DSP, we are not interested in the precise content of images, for example, the geographical position of hot spots; we are only interested in the presence or not of hot spots;
- *resultAnal* $\in \{NOIM, NORM, HOT\}$: result of analysis.

Among these variables, only variables *switchOn*, *switchOff*, *switched*, *tension*, *current*, and *resultAnal* are observable by the controller (set O). Failures, as well as the type of the analyser input image, are not known by the controller.

We use the following command variables (set C):

- *cmdSwitch* $\in \{0,1\}$: command to the switch: 1 for ON and 0 for OFF;
- *cmdMemory* $\in \{0,1\}$: updating of the controller memory: 1 for ON and 0 for OFF;
- *running* $\in \{0,1\}$: information about the correct behaviour of the DSP: 1 for correct;
- *failing* $\in \{0,1\}$: information about the incorrect behaviour of the DSP: 1 for incorrect;
- *hotSpot* $\in \{0,1\}$: information about hot spot detection: 1 for detection.

To structure model writing (relations I, T, and P), it may be useful to build the graph that represents dependencies between state and command variables. Fig. 5 shows this graph in a modeling framework that is close to the graphical language associated with the SCADE tool [7]. Boxes labelled with "pre" allow access to the previous value of a state variable to be represented. We can see for example that the current tension depends on the previous one, on the command to the DSP switch, and on the switch state (failure or not).

Possible initial states (relation I) are defined by the following constraints on state variables: $\neg$*switchOn*, $\neg$*switchOff*, $\neg$*switched*, $\neg$*current*, $\neg$*tension*, (*resultAnal* $= NOIM$), (*inputIm* $= NOIM$). Failures are thus possible in the initial state.

Possible transitions (relation T) are defined by the following constraints on state and command variables:

Constraints on higher level modules:

$\neg$(*switchOn* $\wedge$ *switchOff*)

Constraints on controller memory:

switched' $=$ *cmdMemory*

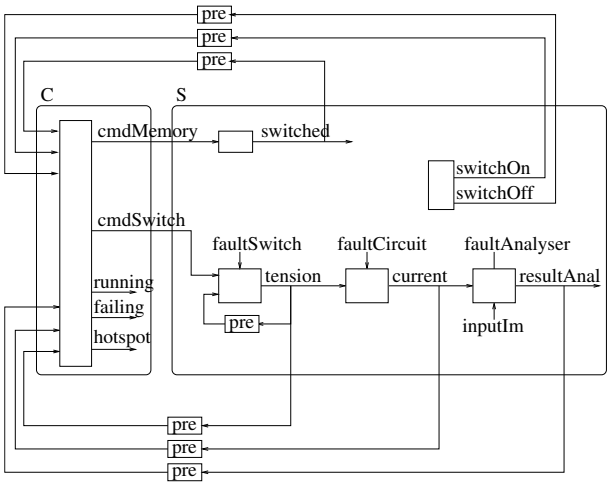

Fig. 5. Dependency graph between state and command variables

Constraints on switch:
$\neg tension \wedge \neg cmdSwitch \rightarrow \neg tension'$
$tension \wedge cmdSwitch \rightarrow tension'$
$\neg faultSwitch \rightarrow (tension' = cmdSwitch)$
Constraints on circuit:
$current = (tension \wedge \neg faultCircuit)$
Constraints on analyser:
$(\neg current \vee faultAnalyser) \rightarrow (resultAnal = NOIM)$
$(current \wedge \neg faultAnalyser) \rightarrow (resultAnal = inputIm)$

Let us recall that, for each state variable x, x' represents its value at the next step. Constraints on higher level modules express that an ON request and an OFF request cannot be emitted simultaneously. Constraints on controller memory express that the controller memory is always correctly updated. Constraints on the switch express that, when the switch does not fail, tension is consistent with the command to the switch. Those on the circuit express that there is current if and only if there is tension and no circuit failure. Finally, those on the analyser express that, when the analyser does not fail, the result of analysis is consistent with the type of the input image.

Acceptable transitions (set P) are defined by the following constraints on state and command variables:

Requirements on controller memory updating:

$switchOn \rightarrow cmdMemory$

$switchOff \rightarrow \neg cmdMemory$

$(\neg switchOn \wedge \neg switchOff) \rightarrow (cmdMemory = switched)$

Requirements on hot spot detection:

$hotSpot = (running \wedge (resultAnal = HOT))$

Requirements on failure detection:

$running = (switched \wedge (resultAnal \neq NOIM))$

$failing = (switched \wedge (resultAnal = NOIM)) \vee$
$(tension \neq switched)) \vee (current \neq tension))$

$\neg(running \wedge failing)$

$switched \rightarrow (running \vee failing)$

Highest level requirements:

$((inputIm = HOT) \wedge running) \rightarrow hotSpot$

$hotSpot \rightarrow (inputIm = HOT)$

$(\neg faultSwitch \wedge \neg faultCircuit' \wedge \neg faultAnalyser' \wedge switched')$
$\rightarrow (resultAnal' = inputIm')$

$(\neg faultSwitch \wedge \neg switched') \rightarrow (resultAnal' = NOIM)$

For example, constraints on hot spot detection enforce that a detection signal is emitted if and only if the DSP runs correctly and a hot spot is detected in the input image. Constraints on failure detection enforce that the DSP controller cannot say simultaneously that the DSP is running correctly and that it is failing and that, when the DSP is ON, the DSP controller must say whether or not the DSP runs correctly. Highest level requirements enforce that, when a hot spot is detected in the input image and the DSP runs correctly, a detection signal is emitted. Conversely, they enforce that, when a detection signal is emitted, a hot spot is present in the input image. Moreover, they enforce that, when no failure occurs and the DSP is ON, the result of analysis is consistent with the type of the input image. On the contrary, they enforce that, when the DSP is OFF and the switch does not fail, there is no analysis result.

The associated OPL model generates 2356 variables and 512175 constraints (information given by the OPL interface). It is solved by CP Optimizer in 3.39 seconds seconds thanks to initial constraint propagation which assigns 2338 variables (results obtained on a Ultra 45 SUN workstation running under Unix and using a 1.6 GHz processor and 1 GB of RAM). As with the toy example of Sect. 1.3, the policy produced without optimization is luckily "optimal": it is defined for only reachable observations.

4 Strengths and weaknesses of the proposed approach

In this paper, we proposed a formulation of the problem of synthesis of a controller for a dynamic system that must satisfy some safety properties as a pure constraint satisfaction (optimization) problem. As far as we know, this the first time that such a formulation has been proposed. The constraint-based approach proposed in [12] is correct, but incomplete: it may fail to find a valid policy, even if such a policy exists. Beyond classical planning and scheduling applications of constraint programming, this opens a large new domain of application of constraint programming techniques to the control of discrete event dynamic systems. It must be moreover stressed that the proposed formulation can be used either to synthesize or to check controllers: checking is simpler than synthesis because the policy is known in case of checking, whereas it is unknown in case of synthesis. It must be also stressed that, using the same approach, ILP (Integer Linear Programming) or SAT (Boolean Satisfiability) modeling frameworks can be used instead of CP, depending on the nature of variables, domains, and constraints.

The proposed approach can be compared with the formulation of the problem of synthesis of an optimal policy for an MDP (Markov Decision Process [17]) as a pure linear programming problem [6]. Indeed, in this formulation, a variable is associated with each state s, representing the optimal gain it is possible to get from s. Bellman optimality equations [1] take the form of linear constraints: one constraint per state-command pair. The criterion to be optimized is the sum of the variables. However, reachability is not taken into account: all states are assumed to be reachable at any step. The approach we proposed can be seen as the "logical" counterpart of this MDP solving approach.

The main advantage of the proposed approach is that it allows us to rely entirely on existing generic constraint programming tools for solving. When constraint programming models will be automatically built from the problem definition (sets of variables S, O, and C, and sets of constraints that define relations I, T, and P), only problem definition shall be changed from a dynamic system to another.

Its main drawback is the huge number of resulting CSP variables and constraints: at least one variable per state and one constraint per pair of states. Because the number of possible states is an exponential function of the number of state variables (not to be mistaken for CSP variables), this approach can be used only on problems that involve a small number of state variables: in the order of ten as in the DSP example of Sect. 3.

The number of CSP variables and constraints has an impact on efficiency: as an example, on the DSP example of Sect. 3, whereas CP Optimizer takes 3.39 seconds to produce a valid policy, the Dyncode tool [16], which implements search algorithms dedicated to controller synthesis, takes only 0.12 second.

As a consequence, other approaches, still based on constraint programming, but less consuming in terms of variables and constraints, should be explored.

Acknowledgments

This work has been performed in the context of the French CNES-ONERA AGATA project (Autonomy Generic Architecture: Tests and Applications; see `http://www.agata.fr`) whose aim is to develop techniques allowing space system autonomy to be improved. We would like to thank Michel Lemaître for his valuable contribution.

References

1. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
2. Bertoli, P., Cimatti, A., Roveri, M., Traverso, P.: Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In: Proc. of IJCAI-01. pp. 473–478 (2001)
3. Bonet, B., Palacios, H., Geffner, H.: Automatic Derivation of Memoryless Policies and Finite-State Controllers Using Classical Planners. In: Proc. of ICAPS-09 (2009)
4. Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)
5. Damiani, S., Verfaillie, G., Charmeau, M.C.: An Earth Watching Satellite Constellation: How to Manage a Team of Watching Agents with Limited Communications. In: Proc. of AAMAS-05. pp. 455–462 (2005)
6. Dantzig, G.: Linear Programming and Extensions. Princeton University Press (1963)
7. Esterel Technologies: SCADE, http://www.esterel-technologies.com/
8. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann (2004)
9. Halbwachs, N.: Synchronous Programming of Reactive Systems. Kluwer (1993)
10. IBM ILOG: Cplex optimization studio, http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/
11. Jobstmann, B., Galler, S., Weiglhofer, M., Bloem, R.: Anzu: A Tool for Property Synthesis. In: Proc. of CAV-07. pp. 258–262 (2007)
12. Lemaître, M., Verfaillie, G., Pralet, C., Infantes, G.: Synthèse de contrôleur simplement valide dans le cadre de la programmation par contraintes. In: Actes de JFPDA-10 (2010)
13. MBP Team: Model Based Planner, http://sra.itc.it/tools/mbp/
14. Pnueli, A., Rosner, R.: On the Synthesis of a Reactive Module. In: Proc. of POPL-89. pp. 179–190 (1989)
15. Pralet, C., Lemaître, M., Verfaillie, G., Infantes, G.: Synthesizing Controllers for Autonomous Systems: A Constraint-based Approach. In: Proc. of iSAIRAS-10 (2010)
16. Pralet, C., Verfaillie, G., Lemaître, M., Infantes, G.: Constraint-based Controller Synthesis in Non-deterministic and Partially Observable Domains. In: Proc. of ECAI-10. pp. 681–686 (2010)
17. Puterman, M.: Markov Decision Processes, Discrete Stochastic Dynamic Programming. John Wiley & Sons (1994)
18. Ramadge, P., Wonham, W.: The Control of Discrete Event Systems. Proc. of the IEEE 77(1), 81–98 (1989)
19. RATSY Team: RATSY: Requirements Analysis Tool with Synthesis, http://rat.fbk.eu/ratsy/
20. Rossi, F., Beek, P.V., Walsh, T. (eds.): Handbook of Constraint Programming. Elsevier (2006)