

Self-Reconfigurable Control - Part II: Executing Based on Model Checking

He-xuan Hu^{1,2}

Agricultural and Animal Husbandry College of Tibet University¹

Lin-zhi, Tibet, P.R. China

College of Energy and Electrical Engineering

Hohai University²

Nanjing, Jiangsu Province, P.R. China

hexuan_hu@hhu.edu.cn

Abstract—This paper presents a formal framework for reconfigurable control, based on model checking. This framework first generates a flexible model (i.e., an execution structure) according to the diagnosis, then defines a temporal specification language to deal with the problems due to infinite execution cycles and non-determinism, and finally provides the algorithms that will automatically verify whether the updated model satisfies the desired specification. Our entire procedure of reconfigurable control is as follows: for a given series of observations and diagnoses, if there are faults in system, then a revised system model is constructed flexibly, according to the diagnoses, observations and available system descriptions. Next, model checking is applied to verify whether or not the current system model satisfies the desired objectives. This automatic reconfigurability calculation is one of the most important steps in our framework. If the given control objectives are achievable, the system will run according to the observations and current system model. Otherwise, the reconfiguration is considered as failed.

Keywords—STRIPS; Model checking; Automated Planning; Reconfigurable Control; Cause-effect

I. INTRODUCTION

In this paper, we continue to present the part II of reconfigurable control. Dynamic reconfigurable control is a field of fault tolerant control that has emerged over the past decade [1]. Reconfigurability — the possibility of ensuring system functions despite the occurrence of faults — is a key notion in many classes of reconfigurable systems [2], [3] and [4]. This problem has been addressed generally by many authors ([2], [4] and [5]) and from a functional point of view by Staroswiecki and Bayart [6]. However, the reconfigurability in the articles [2], [4] and [5] cannot be calculated qualitatively by the system itself.

In this paper, we propose a formal framework to automatically calculate reconfigurability qualitatively. This framework is divided into three parts: (1) a formal framework for modeling the system, already presented in its sister paper – part I; (2) a formal specification language for describing the desired goals, and 3) a verification method for establishing whether the system description satisfies the specification.

Using propositional logic formulae to formalize a system is not new in the field of diagnosis [7], [8], [9] and [10]. These propositional formulae include the control

commands and the sensor readings. In diagnosis, they are used to infer the current system conditions. But they cannot be used for reconfiguration because they are too simple; they only indicate that a system event has occurred, but do not provide information about the event's pre-conditions and effects. These pre-conditions and effects constitute the knowledge necessary to infer a sequence of control commands that will lead the system to the desired goals. In order to provide these pre-conditions and effects, we propose a framework based on STRIPS (Stanford Research Institute Problem Solver) [11] as a system modeling reference.

STRIPS can be used to find an operator configuration that will transform a given initial model into one that satisfies a specific goal condition. Moreover, STRIPS can easily be extended for use with the non-deterministic systems [12] and [13]. However, the STRIPS framework does put a limit on the description of temporal systems, in which the desired goals are defined as a fixed set of states, but the system is not always designed to terminate. For example, the regulation goal always maintains a water level at a set-point in tank. These temporal goals must be achieved at various states during the execution of a control command sequence, not just in the final states. Computation Tree Logic (CTL*) —used in [14], [15] and [16] — is introduced here as a formal specification language for describing the potentially cyclic execution sequences, such as maintaining a property, achieving a goal periodically or within a number of steps after the request was made, and achieving several goals in sequence.

In diagnosis, verification is a proof-based approach [17], in which the system description is a set of formulae Γ and the specification is another formula ζ . The verification method consists of trying to find a proof that $\Gamma \vdash \zeta$. Because this method requires a lot of guidance and expertise from the user [7], a model-based approach, called model checking [17], is introduced in this paper. The system is represented by a model M for an appropriate logic. The specification is represented by a formula ζ , and the verification method calculates whether the model M satisfies ζ ($M \models \zeta$). For finite models, this calculation is usually automatic.

As the mentioned above, an automated method can be provided for calculating reconfigurability without relying on pre-specified functional policies, such as function-

component graphs or descriptions of the relationships between the desired functions and the components needed to accomplish them [2], [6]. These graphs and descriptions constitute functional analysis methods, and they have powerful descriptive capabilities but poor qualitative computation capabilities.

Figure 1 shows the architecture of our method for reconfigurable control. For a given series of observations and diagnoses, if there are faults in system, then a revised system model is constructed flexibly, according to the diagnoses, observations and available system descriptions. Next, model checking is applied to verify whether or not the current system model satisfies the desired objectives. This automatic reconfigurability calculation is one of the most important steps in our framework. If the given control objectives are achievable, the system will run according to the observations and current system model. Otherwise, the reconfiguration is considered as failed.

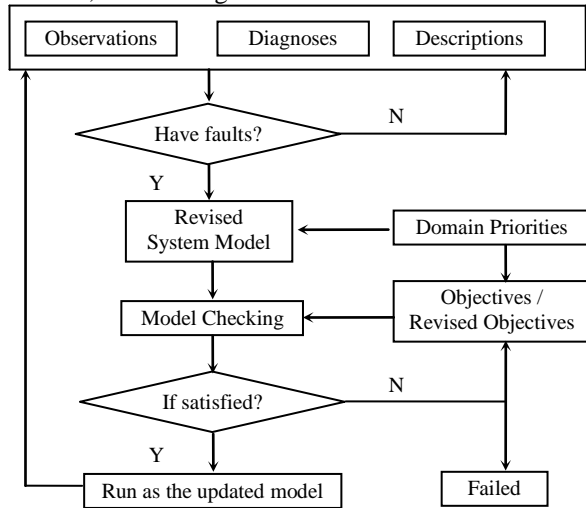


Figure 1. The reconfigurable control architecture

The rest of this paper is organized as follows. In section II, a benchmark example is used to illustrate the modeling notions presented in its sister paper – part I. The temporal logic used for describing extended goals is introduced in section III. The model checking method used to automatically calculate reconfigurability is presented in section IV. Section V summarizes the work done and discusses directions for future research.

II. THE BENCHMARK EXAMPLE

The chosen example comprises a level regulation process involving two identical connected tanks (Fig. 2). The inflow Q_p is provided by pump $P1$. The flow Q_v between the two tanks is controlled by valve $V1$, with $V2$ as a backup valve that should always be closed during normal behaviour. The connecting pipe is at a level of 30 cm (resp. 0 cm). The valve V_o , which is always open, is an outlet valve, located at the bottom of tank $T2$. In this example, it is assumed that all the valves are on/off valves, all the pipes have the same diameter, and the flow rate delivered by $P1$ is equal to the flow rate through $V1$ as the water level of tank $T1$ is 45 cm.

The model generated from Fig.1 is shown in Fig. 3. The fault scenario is that the valve $V1$ is blocked in the closed position. According to the system priorities and the system descriptions, the *Goal* ($L1$) will be changed to

25~30 cm and the control commands associated with valve $V1$, such as *Open*($V1$) and *Close*($V1$), will be removed and be made unavailable for generating the revised model. An illustration, shown in Fig. 4, explains the first step of the model generation.

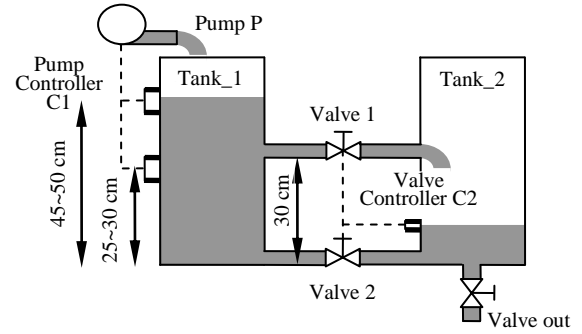


Figure 2. The two-tanks system

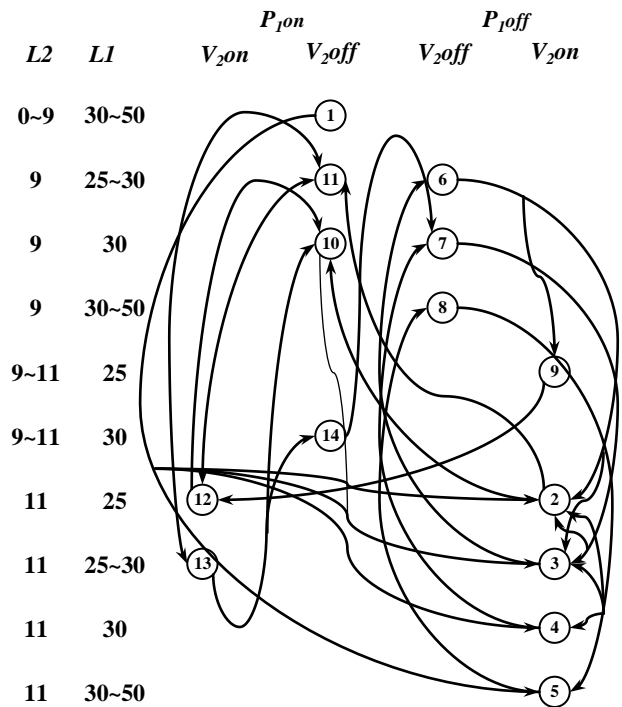


Figure 3. The model generated using the knowledge contained in Fig. 1

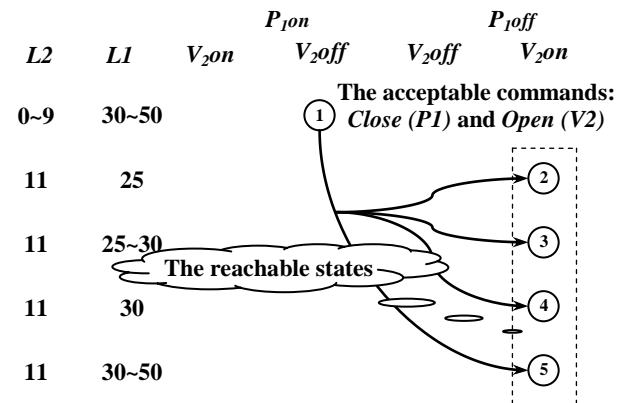


Figure 4. An illustration of the first step of model generation

The acceptable commands are *Close* ($P1$) and *Open* ($V2$). Applying the acceptable commands (i.e., *Close* ($P1$))

and *Open* ($V2$)), the reachable states from initial state 1 are states $2, 3, 4$ and 5 . The effects of these two commands are *Rise* ($L2$) and *Fall* ($L1$). *Rise* ($L2$) brings the level 2 to 11 cm and triggers other commands in a new state. *Fall* ($L1$) decreases level 1 to make it equal to the increase in level 2 triggered by *Rise* ($L2$). According to the current $L1$ and its decrease, the new level 1 could now possibly belong to four different zones (i.e., $25, 25\sim30, 30$ and $30\sim50$).

III. TEMPORALLY EXTENDED GOALS

As mentioned in the introduction, many systems are designed not to terminate. Temporal logic provides a formal framework for qualitatively describing how the truth values of desired goals can be changed over time. In temporal logic, time is discrete, and the present moment corresponds to the current state and next moment corresponds to the state following immediately after.

A. Definition of Computation Tree Logic (CTL*)

In accordance with the works of [14] and [15], we adopt the computation tree logic (CTL*) because it is able to take the non-determinism of the domain into account. The CTL* formulae are composed of ‘path quantifiers’ and ‘temporal operators’.

- Path quantifiers are used in a given state to specify that all or some of the paths starting at that state have certain properties. Here, a path is an infinite sequence of states. Two types of path quantifiers, ‘ A ’ and ‘ E ’, are possible:

- (1) ‘ A ’ is a universal path quantifier, meaning that certain properties hold true on all paths starting from a given state.

- (2) ‘ E ’ is an existential path quantifier, meaning that certain properties hold true on some paths starting from a given state.

- Temporal operators describe path properties. There are five basic types of operators:

- (1) ‘ X ’ (next time) requires that a property hold true in the path’s second state.

- (2) ‘ F ’ (eventually or in the future) is used to affirm that a property will hold true at some state on the path.

- (3) ‘ G ’ (always or globally) specifies that a property holds true at every state on the path.

- (4) ‘ U ’ (until) holds true if there is a state on the path in which the second property holds true, and if the first property holds true at every previous state on the path.

- (5) ‘ R ’ (release) requires that the second property holds true along the path, up to and including the first state where the first property holds, although the first property is not required to hold true in the future.

Definition 2 (CTL*): There are two types of formulae in CTL*: state formulae, which are true in a specific state, and path formulae, which are true along a specific path. Let AP be the set of atomic propositions. The goal language CTL* is defined by the following rules:

- (1) If $p \in AP$, then p is a state formula.
- (2) If f and g are state formulae, then $\neg f$, $f \wedge g$ and $f \vee g$ are state formulae.
- (3) If f is a state formula, then f is also a path formula.
- (4) If f is a path formula, then $E(f)$ and $A(f)$ are state formulae.
- (5) If f and g are path formulae, then $\neg f$, $f \wedge g$, $f \vee g$, $X(f)$, $F(f)$, $G(f)$, $f U g$ and $f R g$ are path formulae.

B. CTL* Semantics

The notion of the execution structure M [13] is introduced to specify the semantics of CTL*. An execution structure is the extension of the automaton — the model generated automatically in section III — created by adding a finite set of logical atoms L . It is also a quadruple (S, L, T, s_0) , in which the element A is replaced by L . Each state is labelled with a set of atomic propositions $L(s)$, which contains all atoms true in that state. This means that if p is an atomic proposition, then p is true at a state s if and only if p labels s (i.e., p is an element of $L(s)$). A path in M is an infinite state sequence, $\pi = s_0, s_1, \dots$ such that for every $i \geq 0$, $(s_i, s_{i+1}) \in T$.

In this semantic, π^i is used to denote the suffix of π starting at s_i . If f is a state formula, the notation $M, s \models f$ means that f holds true at state s in M . Similarly, if f is a path formula, the notation $M, \pi \models f$ means that f holds true along path π in M . The relation \models is defined inductively as follows (assuming that f_1 and f_2 are state formulae and g_1 and g_2 are path formulae):

- (1) $M, s \models p \Leftrightarrow p \in L(s)$.
- (2) $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1$.
- (3) $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$.
- (4) $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$.
- (5) $M, s \models E(g_1) \Leftrightarrow$ there is a path π from s , $M, \pi \models g_1$.
- (6) $M, s \models A(g_1) \Leftrightarrow$ for every path π from s , $M, \pi \models g_1$.
- (7) $M, \pi \models f_1 \Leftrightarrow s$ is the 1st state of π and $M, s \models f_1$.
- (8) $M, \pi \models X(g_1) \Leftrightarrow M, \pi^1 \models g_1$.
- (9) $M, \pi \models F(g_1) \Leftrightarrow$ there exists a $k \geq 0$ such that $M, \pi^k \models g_1$.
- (10) $M, \pi \models G(g_1) \Leftrightarrow$ for all $i \geq 0$, $M, \pi^i \models g_1$.
- (11) $M, \pi \models g_1 U g_2 \Leftrightarrow$ there exists a $k \geq 0$ such that $M, \pi^k \models g_2$ and for all $0 \leq j < k$, $M, \pi^j \models g_1$.
- (12) $M, \pi \models g_1 R g_2 \Leftrightarrow$ for all $j \geq 0$, if for every $i < j$, $M, \pi^i \not\models g_1$ then $M, \pi^j \models g_2$.

CTL* formulae can be built using the modal operators, AX / EX , AF / EF , AG / EG , AU / EU and AR / ER , where A and E are path quantifiers, and X and U are state quantifiers. With these operators, it is possible to define goals that specify the desired behaviours starting from a given state.

Goals expressed as CTL* formulae allow different classes of system requirements to be specified, for example:

- (1) reachability goals, such as $EF(g)$, which requires that the system may be able to reach desired states where f holds true and $AF(g)$, which requires that the system will be guaranteed to reach those desired states;
- (2) safety goals, such as $AG(\neg g)$, which means g must absolutely be avoided and $EG(\neg g)$, which means that an attempt must be made to avoid g ; and
- (3) maintainability goals, such as $AG(g)$, which means g must be maintained and $AF(AG(g))$, which means that the system will always reach some future state from which g can be permanently maintained.

C. CTL* Semantics in Non-deterministic Reconfiguration

As noted in section II, a particular form of uncertainty is used to model non-deterministic systems: control commands are modelled for different outcomes that cannot be predicted at the time of execution (i.e., it is impossible for the system to know *a priori* which of the different

possible outcomes will actually take place). However, there is a conceptual difficulty in non-deterministic reconfiguration because, from the conceptual point of view, different reconfiguration results may be obtained. For instance, a reconfiguration might guarantee that a goal be accomplished, or might just provide the possibility of success.

The temporal logic presented above allows the differences in these results to be described and defined. For example, $AF(g)$ -- a strong goal -- means that the reconfiguration guarantees the accomplishment of the desired goals, while $EF(g)$ -- a weak goal -- means that the reconfiguration only has a chance of success. As shown in figure 1, automatic reconfiguration is a complex procedure during which the goals can be changed at the moment that the fault report is received. For a desired property g , the temporal goal should be $AG(g)$ (i.e., g always holds true) in the normal operating mode. If there is a fault that causes g to deviate from its desired value, then the reconfiguration will correct this deviation and will keep g within its desired value range. But in a non-deterministic system, a control command sent by the reconfiguration procedure cannot be guaranteed to produce the desired effects (i.e., the original goal cannot be guaranteed). This situation can be described as $EF(AG(g))$, which means that g will eventually be accomplished at some future state from which g will be permanently maintained. However, this does not satisfy the requirements of some high security system. Thus, $EF(AG(g))$ must be changed into a strong solution, such as $AF(AG(g))$. If the execution structure satisfies the goal $AF(AG(g))$, then the reconfiguration will be successful in spite of non-determinism. All that remains to be done is to verify whether an execution structure satisfies the temporal goal, and section IV explains how this can be done.

Example 4.1 is a continuation of Fig. 3. The goal of the normal operating mode is to maintain the level of tank $T2$ between 9 cm and 11 cm and to try to maintain the level of tank $T1$ between 45 cm and 50 cm. This temporal goal can be described as $AG(9 \leq L2 \leq 11) \wedge EG(45 \leq L1 \leq 50)$. The fault scenario is that the valve $V1$ is blocked in the closed position, thus $L1$ will be below 9 cm and the goal of tank $T1$ will be changed into $EG(25 \leq L1 \leq 30)$ according to the system priorities. The goal of the reconfiguration can be expressed as $AF(AG(9 \leq L2 \leq 11) \wedge EG(25 \leq L1 \leq 30))$.

IV. THE MODEL CHECKING

A. The equivalent formulae in CTL*

Some equivalent relationships exist among the CTL* formulae. For example, there are ten basic operators -- AX / EX , AF / EF , AG / EG , AU / EU and AR / ER -- Each of these ten operators can be expressed in terms of the three operators EX , EG , and EU , as following:

- (1) $AX(f) = \neg EX(\neg f)$ (2) $EF(f) = E(True \ U \ f)$
- (3) $AG(f) = \neg EF(\neg f)$ (4) $AF(f) = \neg EG(\neg f)$
- (5) $A[f \ U \ g] \equiv \neg E[\neg g \ U (\neg f \wedge \neg g)] \wedge \neg EG(\neg g)$
- (6) $A[f \ R \ g] \equiv \neg E[\neg f \ U \neg g]$
- (7) $E[f \ R \ g] \equiv \neg A[\neg f \ U \neg g]$

B. The labelling algorithm

Let $M = (S, L, T, s_0)$ be an execution structure. Our task is to determine which states in S satisfy the goal formula ψ . The algorithm labels each state s with the set $label(s)$ of

the sub-formulae of ψ , which are all true in s . The algorithm need not handle every CTL* connective explicitly, since the connectives \neg , \wedge , \perp , EX , EG , and EU can represent all the other temporal connectives. First, the goal formula ψ is pre-treated, or in other words, ψ is written in terms of the connectives \neg , \wedge , \perp , EX , EG , and EU using the equivalences given above. Second, the states of M are labelled with the sub-formulae of ψ , starting with the smallest sub-formulae and working recursively towards ψ .

Initially, $label(s)$ is just $L(s)$. The algorithm then processes the sub-formulae with nested CTL* operators. When the algorithm terminates, the result is the set of states of the model that satisfy the formula. A case analysis is used below to describe how states are labelled.

If ψ is:

- (1) \perp : then no states are labelled with \perp ;
- (2) p : then label s with p if $p \in L(s)$;
- (3) $\psi_1 \wedge \psi_2$: label s with $\psi_1 \wedge \psi_2$ if s is already labelled both with ψ_1 and with ψ_2 ;
- (4) $\neg\psi_1$: label s with $\neg\psi_1$ if s is not already labelled with ψ_1 ;
- (5) $EG(\psi_1)$:
 — Label all the states with $EG(\psi_1)$;
 — If any state s is not labelled with ψ_1 , delete the label $EG(\psi_1)$;
 — Repeat: delete the label $EG(\psi_1)$ from any state if none of its successors is labelled with $EG(\psi_1)$; until there is no change.
- (6) $E[\psi_1 \ U \ \psi_2]$:
 — If any state s is labelled with ψ_2 , label it with $E[\psi_1 \ U \ \psi_2]$;
 — Repeat: label any state with $E[\psi_1 \ U \ \psi_2]$ if it is labelled with ψ_1 and at least one of its successors is labelled with $E[\psi_1 \ U \ \psi_2]$, until there is no change.
- (7) $EX(\psi_1)$: label any state with $EX(\psi_1)$ if one of its successors is labelled with ψ_1 .

As $AF(\psi_1)$ is often used in practice, the following algorithm to deal with it directly.

- (8) $AF(\psi_1)$:
 — If any state s is labelled with ψ_1 , label it with $AF(\psi_1)$.
 — Repeat: label any state with $AF(\psi_1)$ if all successor states are labelled with $AF(\psi_1)$, until there is no change.

These eight sub-algorithms can be combined to deal recursively with the different formula.

C. An application

Example 5.1 illustrates the model checking using the benchmark example described in section II. Fig. 3 shows the execution structure, which is the revised system model following the malfunctioning of valve $V1$. Each state is labelled with the atomic propositions that are true in the state.

The verified goal formula $AF(AG(9 \leq L2 \leq 11) \wedge EG(25 \leq L1 \leq 30))$ (here, f is used as the abbreviation for $9 \leq L2 \leq 11$, and g for $25 \leq L1 \leq 30$) is written in terms of the basic connectives (i.e., $\neg EG(EF(\neg f) \wedge \neg EG(g))$). First, the set of states that satisfy the atomic formulae are calculated, followed by those for the more complicated sub-formulae. Let $S(\psi)$ denote the set of all states labelled with the sub-formula ψ .

- (1) $S(\neg f) = \{1\}$

$$(2) S(g) = \{2, 3, 4, 6, 7, 9, 10, 11, 12, 13, 14\}$$

$$(3) S(EF(\neg f)) = \{1\}$$

In order to calculate $S(EG(g))$, first, the states of $S(g)$ are labelled with $EG(g)$ and then the label $EG(g)$ is deleted from any state if none of its successors is labelled with $EG(g)$. This deletion procedure is repeated until there is no change. Thus the calculation terminates with:

$$(4) S(EG(g)) = \{2, 3, 4, 6, 7, 9, 10, 11, 12, 13, 14\}$$

$$(5) S(\neg EG(g)) = \{1, 5, 8\}$$

$$(6) S(EF(\neg f) \wedge \neg EG(g)) = \{1\}$$

When computing $S(EG(EF(\neg f) \wedge \neg EG(g)))$, only state 1 is labelled with $EG(EF(\neg f) \wedge \neg EG(g))$ as was done in the last step (6). Clearly, state 1 has no successor labelled with $EG(EF(\neg f) \wedge \neg EG(g))$. So, the model verification continues:

$$(7) S(EG(EF(\neg f) \wedge \neg EG(g))) = \{\}$$

Finally, the converse of the transition relation is used to label all states in $S(EG(EF(\neg f) \wedge \neg EG(g)))$. Step (7) produces a result of \emptyset , thus implying that

$$(8) S(\neg EG(EF(\neg f) \wedge \neg EG(g))) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$$

This result is very strong because it means that the checked goal holds true along every path from any state in the execution structure. For example, since the initial state 1 is contained in this set, it can be concluded that, in this generated execution structure, the reconfigurable control started at that initial moment and is guaranteed to achieve the original goal $(AG(9 \leq L2 \leq 11) \wedge EG(25 \leq L1 \leq 30))$ at some later time.

V. CONCLUSION

In this paper, we proposed a formal framework for reconfigurable control, based on model checking. This framework first generates a flexible model (i.e., an execution structure) according to the diagnosis, then defines a temporal specification language to deal with the problems due to infinite execution cycles and non-determinism, and finally provides the algorithms that will automatically verify whether the updated model satisfies the desired specification. A benchmark example is used to illustrate the entire reconfiguration procedure. The results of this illustration show that our framework is able to express reconfiguration requirements very well and provides powerful qualitative computation capabilities.

In future research, it would be interesting to attempt to reduce the impact of the state explosion problem. There have been several noteworthy works [18], [19], [20], [21] and [22] which could inspire us to deal with this problem, including attempts to exploit abstractions, symmetries and compositionality.

ACKNOWLEDGMENT

This work is supported by "The Nature Science Foundation of Tibet: 13-38", "A Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions (Coastal Development Conservancy)".

REFERENCES

- [1] R. J. Patton, "Fault - Tolerant Control Systems: the 1997 Situation," IFAC Symposium on Fault Detection Supervision and Safety for Technical Processes, Vol. 3, Kingston Upon Hull, UK, 26-28 August 1997, pp. 1033-1054.
- [2] H.-X. Hu, A.-L. Gehin and M. Bayart, "Model Aggregation for Reconfigurable Control Based on Generic Component Model," in ICSSM'06, Troyes, France, 2006.
- [3] N. Eva Wu, K. Zhou and G. Salomon, "Control reconfigurability of linear time-invariant systems," Automatica, Vol. 36, Issue 11, November 2000, pp.1767-1771.
- [4] M. Staroswiecki and A.-L. Gehin, "Analysis of System Reconfigurability using Generic Component Models," in Control'98, Swansea, UK, 1998.
- [5] M. Blanke, M. Kinnaert, J. Lunze and M. Staroswiecki, Diagnosis and Fault-Tolerant Control, Springer, 2003.
- [6] M. Staroswiecki and M. Bayart, "Models and Languages for the Interoperability of Smart Instruments," Automatica, Vol. 32, no. 6, 1996, pp. 859-873.
- [7] I. Mozetic, "Hierarchical Model-based Diagnosis," In Readings in Model-based Diagnosis, Morgan Kaufmann, San Mateo, CA, 1992, pp. 354-372.
- [8] L. Chittaro and R. Ranon, "Hierarchical model-based diagnosis based on structural abstraction," Artificial Intelligence, Vol. 155, Issues 1-2, 2004, pp. 147-182.
- [9] R. Davis, "Diagnostic reasoning based on structure and behavior," Artificial Intelligence, Vol. 24, Issues 1-3, 1984, pp. 347-410.
- [10] R. Reiter, "A Theory of Diagnosis from First Principles," Artificial Intelligence, Vol. 32, Issues 1, 1987, pp. 57-95.
- [11] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," Artificial Intelligence, Vol. 2, Issues 3-4, Winter, 1971, pp. 189-208.
- [12] M. Ghallab, D. Nau and P. Traverso, Automated Planning: Theory and Practice, Morgan Kaufmann Publishers, 2004.
- [13] M. Pistore and P. Traverso, "Planning as model checking for Extended goals in Non-deterministic Domains," in Proc. IJCAI 2001, AAAI Press, 2001, pp. 479-486.
- [14] E. A. Emerson, "Temporal and Modal Logic," Handbook of theoretical computer science (vol. B): formal models and semantics, The MIT Press, 1991, pp. 995 - 1072.
- [15] E. M. Clarke, Jr. O. Grumberg, and D. A. Peled, Model Checking, The MIT Press, 2000.
- [16] F. Kabanza and S. Thiébaux, "Search Control in planning for Temporally Extended Goals," in Proc. of 15th International Conference on Automated Planning and Scheduling (ICAPS-05), 2005, pp. 130-139.
- [17] M. Huth and M. Ryan, Logic in Computer Science: Modeling and Reasoning about Systems (Second Edition), Cambridge University Press, 2004.
- [18] D. E. Long, "Model Checking, Abstraction, and Compositional Verification," PhD thesis, School of Computer Science, Carnegie Mellon University, July 1983.
- [19] D. R. Dams, "Abstract Interpretation and Partition Refinement for Model Checking," PhD thesis, Institute for Programming Research and Algorithmics. Eindhoven University of Technology, July, 1996.
- [20] E. M. Clarke, O. Grumberg, and D.E. Long, "Model Checking and Abstraction," ACM Transactions on Programming Languages and Systems, Vol. 16, no. 5, September 1994, pp. 1512-1542.
- [21] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic Model Checking: 1020 States and Beyond," Information and Computation (Special issue for the best papers from LICS'90), Vol. 98, no. 2, June, 1992, pp. 142-170.
- [22] A.-L. Gehin, H.-X. Hu, and M. Bayart, "A self-updating model for analysing system reconfigurability," Engineering Applications of Artificial Intelligence, Vol.25, Issue 1, 2012, pp.20-30.