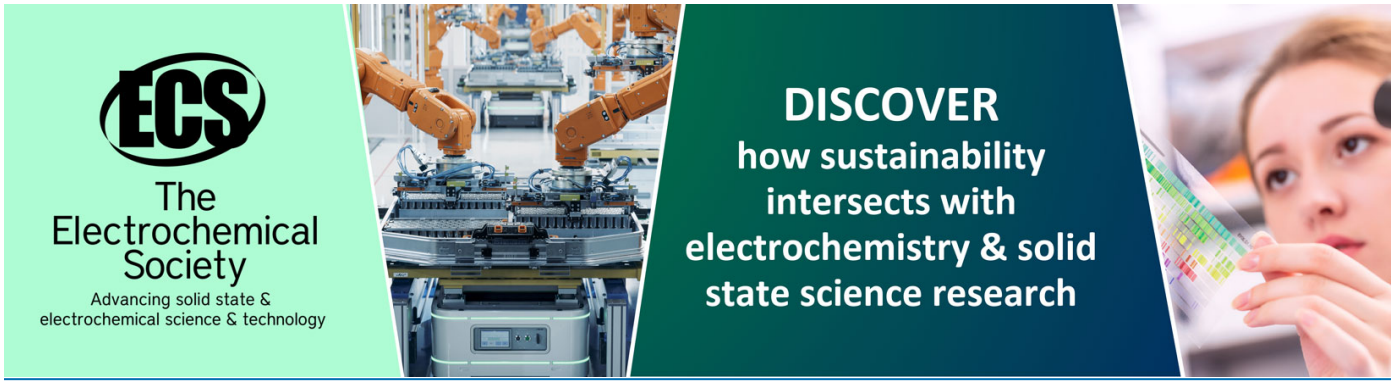
Reasoning about Object-Z formal specification with LTLC



IOP Conf. Series: Journal of Physics: Conf. Series **1176** (2019) 022017 doi:10.1088/1742-6596/1176/2/022017

Reasoning about Object-Z formal specification with LTLC

Haijun Huang[1] and Zhicheng Wen[1,*]

[1]Jiangxi University of Engineering, Xinyu, Jiangxi Province, 338000, China

*Corresponding author e-mail: zcwen@mail.shu.edu.cn

Abstract: It presents a method that linear temporal logic with clock (LTLC) is added to formal specification language Object-Z. LTLC Extending Object-Z is a modular formal specification language, which is the smallest extension for Object-Z. The complex real-time software formal specification can be described and verified easily by the extended Object-Z. Finally, an example is given to show that the formal specification of Object-Z can be reasoned and the correctness of the method is verified.

1. Introduction

Object-oriented formal specification is very suitable described by Object-Z[1]. Through the application of object-oriented technology[2-4], it can describe complex data and algorithms. There is only the abstract concept of termination and no concept of duration associated with operations. Therefore, a real-time reaction system is very difficult to be described by Object-Z. In order to describe the real-time specifications[5] about continuous time relationships or properties, this paper introduces a method that Object-Z is extended with Linear Temporal Logic with Clock(LTLC). The extended Object-Z using LTLC is the smallest extension of Object-Z syntax and semantics. For object-oriented real-time system modeling, it can provide an elegant symbol. In addition, the verification becomes convenient. Finally, this article uses an example to indicate that it can infer the Object-Z formal specification to verify that formal specification developed is correct.

2. Adding LTLC to Object-Z

2.1 Syntax for extended Object-Z

The syntax for extending Object-Z, we will use LTLC to introduce in this section. In general, symbolic logic formulas in the standard Object-Z are used to describe the invariants of classes and the predicates of operations. You can use LTLC formula to describe the invariants of the class and an operation predicates in the extended Object-Z.

*2.1.1 Structure for Object-Z class***.** An optional operation MAIN is introduced in the extended Object-Z class. There are several nondeterministic operations[6,7] in this class structure. In Figure 1, the structure is shown where the clock variable t_{now} can be used to initialize the class system. In an operation, it is only available for the other clock variables defined in a corresponding operation (explained in the next section).

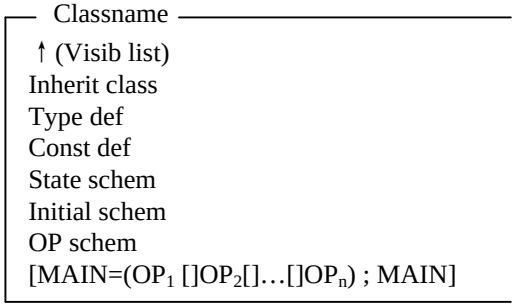


Fig 1 The extended Object-Z class

2.1.2 Structure for Object-Z operation. The operation predicate part of an extended Object-Z mode consists of some optional parts[8]. The operation mode structure is shown in Figure 2.

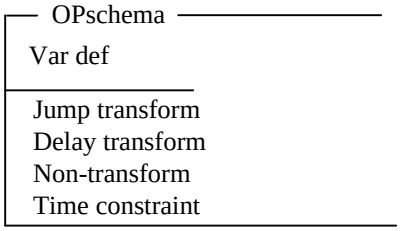


Fig 2 Extend Object-Z operation

Definition: The two corresponding clock variables are t_v and t'_v for any variable v.

2.2 Example

For example, there are three alternate signals of road traffic lights, red, yellow and green. Four indicator lights (green, yellow, red) and four states (green, green, yellow, red, yellow) are a simple example for the intersection indicator system. First, the green light lasts 60 seconds (green state). There is a central (green, yellow) transition state after the green light is switched to the red light. This will last 3 seconds. Then, red light for 50 seconds is held (red state). After that, the red light will switch to green light, but also shifted to the middle (red, yellow) which lasts for 4 seconds. These four signal states may change in sequence.

If the state variable is p in the state schema, it can be seen the state conversion defined, and it is clear that the corresponding clock variable is tp and t'p. This state variable p is a delay amount, and therefore corresponds to a delay conversion. It is shown for extended Object-Z style operation light switch in Fig. 3.

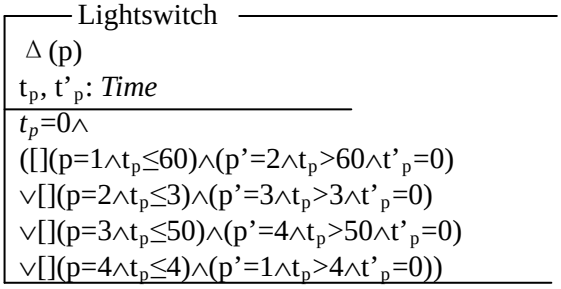


Fig 3 Operation lightswitch

It indicates for a predicate part that the clock variable tp starts first and it may convert from current state to the next state which will continue for a while. The integers 1, 2, 3 and 4 simply present the corresponding four states. For example, the notation of [](p=2×tp≤3)∧(p'=3×tp>3×t'p=0) means that the current state 2will last for 3 seconds, but the clock variable is reset last (t'p = 0).

3. A case study

To better illustrate the use of extended Object-Z to describe formal specifications, here is an example. The extension Object-Z specification is considered as follows: describing a formal system that includes two classes of console and buttons.

The feature of a button object can be described by the class button (shown in Fig. 4). The button contains the state variable recording whether the button is in the "0" or "1" position. First, the button object is in state 0. You just want to think the specific time mt_1 you can switch between cities of buttons. As a result, the state of "0 / 1" is changed. This operation is designated by a toggle operation. Therefore, button *on* is a toggle delay variable corresponding to two delay transformations.

The corresponding clock variable: t and t' are defined. The first line of the button class specification indicates that the interface between the button class and the environment object is limited to the execution and manipulation toggle of the initialization schema. Since toggle operations include only delay variables, it does not lead to time-out and is not required to be combined with restoration. The current time has two constraints every time ([](t_{now}> sometime, it must be time invariant) and the delay time mt_1 is smaller than the maximum delay class (updelay). The clock variable t_{now} is defined from any class.

| t_{now}: *Real-time[real-time part]*

Button_class

↑ (Initialization, Toggle)
| mt_1, sometimes, up_delay: *Tim*

on:{0,1}[optional]

[](t_{now}>sometimes)[optional]

Init

on=0

Toggle_op

Δ (on)
t_{on}, t'_{on} : *Time*

mt_1<up_delay∧t_{on} =0
([](on=0∧t_{on}≤mt_1)(on'=1∧t_{on}>mt_1∧t'_{on}=0)
[](on=1∧t_{on}≤mt_1)∧(on'=0∧t_{on}>mt_1∧t'_{on}=0))

Fig 4. Class Button

The Console class (Figure 5, Figure 6 and Figure 7) describes the function of the console object. The console consists of two buttons, each called busy and lazy. The intention is that busy buttons are toggled more frequently than delay buttons, but do not toggle more than the maximum number of buttons for hardware tolerance prediction. The Console class object contains the constant max that records the prediction error.

4. Reasoning formal specification

Formal verification includes model checking and theorem proving. In this section, we prove the extended Object-Z by theorem proving. You can infer the characteristics and behavior of the specification. The inference rules and methods of standard object Z reasoning are shown in [12]. The properties of this class are expressed as A:: d | Ψ ├ Φ. When the statements d and predicate are given, when the order of at least one predicate is true, the predicate is valid. That is to say, the predicate attribute is true. Variables and constants only define the predicate "Φ". It is accessible within the class, or is declared d.

Console

↑ (Init, ToggleOneButton, ShutDown)
| mt_2, mt_3, mt_4,mt_5: *Time*
| max: N

busyv, lazyv: Button
busy_c, lazy_c: N

[](busylazy)[](t_{now}>0)

Init

busy_c=0∧lazy_c=0
busyv.Init∧lazyv.Init[optional]

Operation: $SelectAndIncrement_0$(as shown in Fig 6.)
Operation: $ShutDown_0$(as shown in Fig7.)

SelectAndIncrement=
$SelectAndIncrement_0$ ⊿{t}**Restore**
ShutDown = $ShutDown_0$ ⊿{t}**Restore**
ToggleOneButton[]SelectAndIncrement b?.Toggle
MAIN=(ToggleOneButton[]ShutDown) ; MAIN

Fig 5. Class Console

$SelectAndIncrement_0$

Δ(busy_c, lazy_c)
b?: Button
$t_{b?}$, $t'_{b?}$, t_{lazy_c}, t'_{lazy_c}, t_{buzy_c}, t'_{buzy_c} :*Time*

busy_c<max$t_{b?}$=0t_{lazy_c}=0t_{buzy_c}'=0
((b?=lazy∧lazy_c<busy_c) ∧b?=busy)
([](b?=busy∧lazy_c'=lazy_c+1∧t_{lazy_c}>mt_3
$t_{b?}$>mt_2∧t'_{lazy_ctr}=0∧$t'_{b?}$ =0)
[](b?=busy∧busy_c'=busy_c+1
t_{buzy_c}>mt_4∧$t_{b?}$>mt_2∧t'_{buzy_c}=0∧$t'_{b?}$ =0))

Fig 6. Class $SelectAndIncrement_0$

$ShutDown_0$

Δ(busy_ctr)
t_{busy_ctr}, t'_{busy_ctr}: *Time*

t_{busy_ctr} =0
[](busy_ctr=maxbusy_ctr'=busy_ctr+1
t_{busy_ctr}>$mt_5$$t'_{busy_ctr}$=0)

Fig 7. Class Console

4.1 Reasoning the state variables

We may consider invariants, pre- and postconditions developed by Object-Z for LTLC. The symbol invariants(x_var) in the state pattern represent invariants. In the state mode of class Class, variable X represents the state variables. It has:

$Precondition_{OP}$ = ∀*x_var'*:X; *output_var*:OUTPUT•

OP(*x_var*, *x_var'*,*input_var*, *output_var*) ∧Invariant(*x_var'*)∧Invariant(*x_var*)

$Postcondition_{OP}$ = Invariant(*x_var'*) ∧OP(*x_var'*, *output_var*)

The postcondition of the operation keeps the predicate expression, containing poststate or output

variables. Precondition and postcondition state variables can be expressed synchronously in predicates.

Similarly, in real-time object-Z, efficient classes have invariants, and efficient operations have their pre- and postconditions.

(1) Invariant rule:

$Invariant_{Class}(x\_var,t_{now})= Invariant_{ilter}(t_{now,})\wedge Invariant_{Class}(x\_var)$

For the *Console* operation, we have:

$Invariant_{Console}(x\_var, t_{now})= (t_{now}>0)\wedge(busy\wedge lazy)$

(2) Precondition rule: $Pre_{OP}=Pre_{OP}(x\_var, x\_var', Tvariable, Tvariable')$

(3) Postcondition rule: $Post_{OP}=Invariant\wedge Post_{OP}(x\_var, x\_var', Tvariable, Tvariable')$

Where, *Tvariable* represents the time variable; PreOP (x_var, x_var ', T., T.') represents the preconditions, and can be calculated using this method. In normalization of object Z, InvarinatClass (t_{now}), PostOP (x_var, x_var ', t, t') and InvarinatClass(x_var, t) have similar meanings.

4.2 Reasoning the properties

For the class *Button* of the example above, we have:

Button:: Toggle ⊢
$([](on'=1\wedge t_{on}>mt_1\wedge t'_{on}=0)\wedge (on=0\wedge t_{on}\le mt_1)\vee$
$[] (on'=0\wedge t_{on}>mt_1\wedge t'_{on}=0)\wedge(on=1\wedge t_{on}\le mt_1))$
$\wedge t_{on} =0\wedge mt_1<updelay$

and

Button:: Toggle ⊢
$[] (on'=1\wedge t_{on}>mt_1\wedge t'_{on}=0)\wedge(on=0\wedge t_{on}\le mt_1)\vee$
$[] (on'=0\wedge t_{on}>mt_1\wedge t'_{on}=0)\wedge(on=1\wedge t_{on}\le mt_1)$

Then, we can use the theorem, it has:

Button:: Toggle ⊢
$[]((on'=1\wedge t_{on}>mt_1\wedge t'_{on}=0)\wedge(on=0\wedge t_{on}\le mt_1)\vee$
$(on'=0\wedge t_{on}>mt_1\wedge t'_{on}=0)\wedge(on=1\wedge t_{on}\le mt_1))$

and

Button:: Toggle ⊢
$(on'=1\wedge t_{on}>mt_1\wedge t'_{on}=0)\wedge(on=0\wedge t_{on}\le mt_1)\vee$
$(on'=0\wedge t_{on}>mt_1\wedge t'_{on}=0)\wedge(on=1\wedge t_{on}\le mt_1)$

That is, in the Toggle operation, $on=0\wedge t_{on}\le mt_1$ remains until $on'=1\wedge t_{on}>mt_1\wedge t'_{on}=0$ or $on=1\wedge t_{on}\le mt_1$ remains until $on'=0\wedge t_{on}>mt_1\wedge t'_{on}=0$. Symbol $t'_{on}=0$ indicates a reset of clock variables. It indicates that the switching state changes within 0/1 and the corresponding delay time is mt_1, which is consistent with the design requirements.

5. Conclusion

In this paper, we propose a method for extending LTLC to Object-Z. The extension with multithreaded module style language LTLC is the minimum extension of Object-Z, and it is useful to describe and verify formal specifications of complex system. Finally, use an example to instruct the Object-Z format specification to make sure it is correct.

Acknowledgements

Haijun Huang(1980.11-), male, Han, Nanchang city, Jiangxi Province, master's degree, associate professor, majoring in: communication engineering and electronic science. Corresponding author: Zhicheng Wen (1972.11-), male, Han, Dong'an county, Hunan province, Ph.D., professor, majoring in: network security, trusted software.

References

[1] Smith G, The Object-Z Specification Language, Advances in Formal Methods, Kluwer Academic Publishers, 2000.

[2] Smith G and Ian Hayes, An introduction to Real-Time Object-Z, Formal Aspects of Computing, 2002, 13:128-141
[3] Mahony, B.P. and Dong, J.S, Timed Communicating Object-Z, IEEE Transactions on Software Engineering, 2000, 26(2): 150-177
[4] Brendan Mahony and Jin Song Dong, Deep Semantic Links of TCSP and Object-Z: TCOZ Approach, Formal Aspects of Computing, BCS, 2002, 13:142-160
[5] P. BELLINI, R. MATTOLINI, and P. NESI, Temporal Logics for Real-Time System Specification, ACM Computing Surveys, Vol. 32, No.1, March 2000
[6] Liu Ming-xing, Ma Wu-bin, et al. Modeling and verification of services oriented cyber physical systems[J]. Journal of Computer Applications, 2014, 34(6): 1770-1773.
[7] Liu Yan, Zhang Xian, et al. Towards formal modeling and verification of pervasive computing systems[J]. Transactions on Computational Collective Intelligence XVI, 2014, 8070: 62-91.
[8] MA Li, LI Wei-kang, LIANG Chen, LI Ai-ping. Formal Modeling and Verification of Resource-oriented Internet of Things Systems[J]. Journal of Chinese Computer Systems, 2018, 39(1): 140-145.