International Journal of Industrial Engineering and Management (IJIEM), Vol.2 No 4, 2011, pp. 139-143
Available online at http:// www.iim.ftn.uns.ac.rs/ijiem_journal.php
ISSN 2217-2661

UDK: 519.6:004 **Research paper**

Automatic Verification of Inheritance

Aleksandar Kupusinac
Ph.D., Assistant Professor, Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad, Republic of Serbia, sasak@uns.ac.rs

Dušan Malbaški
Ph.D., Full Professor, Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad, Republic of Serbia, malbaski@uns.ac.rs

Received (27.05.2011.); Revised (17.09.2011.); Accepted (22.10.2011.)

Abstract

Implementation of inheritance in programming languages as they are does not impose any restrictions regarding the relation between the superclass and its subclass. This means that in principle it is almost possible to derive any class from any other. On the other hand, from the theoretical point of view the inheritance models a specialization relationship, meaning that logically the subclass must be a specialization of the superclass. Breaking this rule causes serious problems especially with polymorphic replacement. This fact is a motivation to introduce the concept of correct inheritance and to establish the appropriate conditions that must be satisfied by the subclass. In the present paper, we discuss such formal conditions and develop a formal recursive algorithm for automatic verification of inheritance. The solution is based solely on the first-order predicate logic.

Key words: *formal methods, inheritance, object-oriented program verification, program correctness*

1. INTRODUCTION

Programming languages as they are do not impose any restrictions on the real relationship between the classes that are connected by inheritance. By using *private* inheritance in C++ [5] or by combining the inheritance mechanism with interface implementation in Java [6, 15] it is technically possible to derive any class from any other. On the other hand, overriding a method inevitably influences the invariant and may cause problems with polymorphic replacement. For instance, let the predicate:

$$IS_K \equiv (this = null) \vee [(this \neq null) \wedge (x \geq 0)],$$

be the strong invariant of some class K, and the predicate:

$$IS_L \equiv (this = null) \vee [(this \neq null) \wedge (x \leq 0)],$$

the strong invariant of its subclass L in which some method m is overridden:

```
public class K {
    protected int x;
    public K() {x=0;}
    public void m() {x++;}
}

public class L extends K {
    public void m() {x--;}
}
```

If an object `obL` from the subclass is in a valid state in which IS_L holds but not IS_K and if the polymorphic assignment `obK=obL` is executed, then the object `obK` from the superclass will be left in an invalid state. In that way, the object `obK` will disturb the correctness of the entire program. We must stress that such situations are possible and can be implemented in any object-oriented programming language without restrictions imposed by the language itself.

The solution has to be found somewhere else, and namely in the Substitution Law [13] i.e. in *subtyping* [2, 11]. Both rules prescribe that the polymorphic replacement of the superclass instance by the subclass instance must not in any way change the behavior of the client software. Meyer [14] states that the overridden method may weaken the precondition and/or strengthen the postcondition of its original.

This paper describes a recursive algorithm that verifies the correctness of inheritance, meaning that it checks if methods break the global relationship between the class and its subclass.

We start with an obvious conclusion that the creation of a class by inheritance means that it consists of the content taken from the superclass with modification (e.g. overriding) or without modification (i.e. inheritance in a narrow sense, as defined in [1]) as well as the additional part defined in the subclass. We will restrict our considerations only on the single superclass leaving the multiple inheritance for the future work.

2. BASIC CONCEPTS

Definition 1. (Abstract state space) *The abstract space of the class K is a nonempty finite set of abstract states U_K that is assigned to the class K.*

The set of all fields of the class K is denoted by Φ_K. Assuming that there is at least one field in the class it follows that $\Phi_K \neq \emptyset$, so $U_K \neq \emptyset$. With every abstract state from the set U_K we associate a field vector $\overline{\varphi} = (\varphi_1,...,\varphi_k)$. The abstract state space U_K we divide into two disjoint parts, namely the sets of *valid* states V_K and *invalid* states N_K. The set of valid states V_K is assumed to be nonempty. Null-state o is a valid (quasi) state that an object occupies before its construction or after its destruction, i.e. when $this = null$, where $this$ stands for the representative object. The set of all methods in the class K is denoted by M_K provided that $M_K \neq \emptyset$.

Definition 2. (Invariant) *An invariant in the class K is every predicate I_K defined over its abstract state space U_K that is true in every valid state.*

Definition 3. (Strong invariant) *Strong invariant in the class K is any predicate IS_K defined over U_K with the properties:*

1) IS_K *is an invariant in the class* K*,*

2) If I *is an invariant in the class* K *then*

$$\forall s \in U_K,\ IS_K(s) \Rightarrow I(s).$$

Note that the invariant must be true in all valid states, whilst in invalid states it may or may not. In the class K a strong invariant always exists and it is unique up to the level of equivalency. Strong invariant completely defines the set of valid states, i.e.

$$V_K = \{s \mid s \in U_K,\ IS_K(s)\}.$$

Hoare triples [8, 9] are two special formulas of the first-order predicate calculus [10] defined over the abstract state space U_K. We introduce dynamic form of the total correctness formula (*DTCF*) in the object-oriented environment:

$$\forall s[P(s) \Rightarrow \exists s' m(s,s') \wedge \forall s'(m(s,s') \Rightarrow \hat{Q}(s,s'))],$$

shortly denoted by $\{P\}m\{\hat{Q}\}$, where the postcondition $\hat{Q}(s,s')$ is now dynamic [12], meaning that it is a function of initial and final states, thus representing the transition function. In order to avoid confusion we will denote dynamic postconditions by the sign " ^ ". The structures of static and dynamic total correctness formulas are the same. The predicate $\hat{Q}(s,s')$ can be formally replaced by the predicate $R(s')$ such that

$$\forall s \forall s' \hat{Q}(s,s') \Leftrightarrow R(s'),$$

so the mathematical apparatus developed in the scope of static postconditions may be used for dynamic without any restrictions, e.g. all general laws of Hoare logic hold in dynamic environment as well, such as the laws of consequence, conjunction, disjunction etc. The interpretation of formula $\{P\}m\{\hat{Q}\}$ is the following – if the predicate P is true in some state then the method m terminates from that state and the transition will occur that satisfies the dynamic predicate $\hat{Q}$. The set of all states from which a method terminates is described by a special predicate called *guard* [3, 4, 7].

Definition 4. (Guard) *The guard of the method m denoted by $\Gamma(m)$ is a predicate with the following properties:*

1.) $\{\Gamma(m)\}m\{\top\}$*, i.e. if the precondition* $\Gamma(m)$ *is satisfied than the method* m *must terminate,*

2.) $\{P\}m\{\top\} \Rightarrow \forall s(P(s) \Rightarrow \Gamma(m)(s))$.

Definition 5. (Strongest dynamic postcondition) *The strongest dynamic postcondition of the method m with respect to the precondition P is predicate $s\hat{d}p(m,P)$ if:*

1) $\{P\}m\{s\hat{d}p(m,P)\}$,

2) $\{P\}m\{\hat{Q}\} \Rightarrow \forall s \forall s'(s\hat{d}p(m,P)(s,s') \Rightarrow \hat{Q}(s,s'))$.

The most important case is the strongest dynamic postcondition with the guard Γ as the precondition. Let m be a method of the class K. The dynamic predicate $s\hat{d}p(m,\Gamma(m))$ determines all valid transitions that can be accomplished by m and only them.

3. CORRECT INHERITANCE

Let the class L be a subclass of the class K. Let IS_K be a strong invariant of the class K over the fields $\overline{f} = (f_1,..., f_n)$. Let the state space of L be defined over the fields $\overline{\varphi} = (f_1,..., f_n, g_1,..., g_m)$, and $\overline{\varphi} \setminus \overline{f}$ denotes the restriction of the vector $\overline{\varphi}$ on $\overline{f}$. Let $\overline{\varphi}$ be a state, which satisfies $IS_L(\overline{\varphi} \setminus \overline{f})$ and does not satisfy $IS_K(\overline{f})$. If the instance `obL` of the class L is in this state and if the polymorphic assignment `obK=obL` is executed, where `obK` is an instance of the class K, then `obK` will enter an invalid state. This implies

$$IS_L(\overline{\varphi} \setminus \overline{f}) \Rightarrow IS_K(\overline{f}).$$

Further, owing to the fact that every valid state of class L is described by exactly one n-tuple $(f_1,..., f_n)$, it follows that $IS_L(\overline{\varphi}) \Rightarrow IS_L(\overline{\varphi} \setminus \overline{f})$, implying

$$IS_L(\overline{\varphi}) \Rightarrow IS_K(\overline{f}).$$

Definition 6. (Correct inheritance) *Let the class L be a subclass of the class K and let IS_L and IS_K be their strong invariants. We say that inheritance is correct iff*

$$IS_L \Rightarrow IS_K .$$

The Definition 6. describes the global disposition of a subclass to its superclass and if it holds then polymorphic assignment obK=obL will not disturb correctness.

4. VERIFICATION OF INHERITANCE

Let the class L be a subclass of the class K. We divide the set of subclass L methods into disjunctive subsets: M_L^C with constructors, M_L^I with inherited methods, M_L^O of overridden methods and M_L^A with added methods. Inherited methods may invoke the methods that are overridden and thus indirectly change the behavior of an instance. Let the union of M_L^I, M_L^O and M_L^A be denoted by M_L^{IOA}.

The class L constructors create objects and put them into initial states. Let the predicate IS_L^0 describes all the initial states of class L objects and only them.

Now, consider the set $M_L^{IOA} = \{m_1,...,m_n\}$. Let us form the following recursive sequence of predicates:

Algorithm 1.

$$Z_0 \equiv IS_L^0$$
$$Z_1 \equiv Z_0 \vee s\hat{d}p(m_1, Z_0) \vee ... \vee s\hat{d}p(m_n, Z_0)$$
$$Z_2 \equiv Z_1 \vee s\hat{d}p(m_1, Z_1) \vee ... \vee s\hat{d}p(m_n, Z_1)$$
...

Let $Z = \{Z_0, Z_1,...\}$ be a set of the predicates described above. Apparently, the set Z is a complete chain since every subset of Z has both lower and upper bounds. Define the function $\sigma : Z \to Z$ in the following way:

$$\sigma(z) \equiv z \vee s\hat{d}p(m_1, z) \vee ... \vee s\hat{d}p(m_n, z),\ z \in Z .$$

The function σ is monotonic on the complete chain Z, i.e.

$$(z \Rightarrow y) \Rightarrow (\sigma(z) \Rightarrow \sigma(y)),\ z, y \in Z ,$$

so by Tarski theorem [16] it has a fixpoint Z_{fix} on the set Z for which

$$Z_{fix} \equiv Z_j,\ j \geq fix .$$

Theorem 1. (Correct inheritance) *Let the class L be a subclass of the class K. The inheritance is correct iff*

$$O \vee Z_{fix} \Rightarrow IS_K ,$$

where O stands for null-predicate $this = null$.

Proof. *The subclass L constructors create objects and put them into initial states described by Z_0. Consider an object of the subclass L that is in the initial state. The predicate $s\hat{d}p(m, Z_0)$ describes all states that this object can reach by invoking the method $m \in M_L^{IOA}$. Then, the disjunction:*

$$s\hat{d}p(m_1, Z_0) \vee ... \vee s\hat{d}p(m_n, Z_0),$$

describes all states that this object can reach by invoking all methods from set M_L^{IOA}. The next step is:

$$Z_1 \equiv Z_0 \vee s\hat{d}p(m_1, Z_0) \vee ... \vee s\hat{d}p(m_n, Z_0)$$

and we obtain predicate Z_1 that includes Z_0 states and states reachable from Z_0. We recursively repeat this step and obtain fixpoint Z_{fix} that describes all states reachable by subclass objects. Based on that, we conclude that predicate $O \vee Z_{fix}$ is strong invariant in the subclass L. By the Definition 6. and by

$$O \vee Z_{fix} \Rightarrow IS_K ,$$

the inheritance is correct.

Q.E.D.

4.1 Example 1

Let us consider the following superclass K and its subclass L:

```
public class K {
   protected int x;
   public K() {x=0;}
   public void m() {
      if(x>=6) throw RuntimeException();
      x++;
   }
}
public class L extends K {
   public void m() {
      if(x<=-6) throw RuntimeException();
      x--;
   }
}
```

The predicate

$$IS_K \equiv (this = null) \vee [(this \neq null) \wedge (0 \leq x \leq 6)],$$

is a strong invariant of the superclass K. The superclass constructor leaves the object in the initial state that satisfies

$$IS_L^0 \equiv (this \neq null) \wedge (x = 0) .$$

For the overridden method m, we create the recursive sequence of predicates:

$$Z_0 \equiv (this \neq null) \wedge (x = 0)$$

$$Z_1 \equiv (this \neq null) \wedge (-1 \leq x \leq 0)$$

$$Z_2 \equiv (this \neq null) \wedge (-2 \leq x \leq 0)$$

$$Z_3 \equiv (this \neq null) \wedge (-3 \leq x \leq 0)$$

$$Z_4 \equiv (this \neq null) \wedge (-4 \leq x \leq 0)$$

$$Z_5 \equiv (this \neq null) \wedge (-5 \leq x \leq 0)$$

$$Z_6 \equiv (this \neq null) \wedge (-6 \leq x \leq 0)$$

$$Z_7 \equiv (this \neq null) \wedge (-6 \leq x \leq 0)$$

Since $Z_6 \equiv Z_7$ we conclude that the predicate Z_6 is a fixpoint. Since $O \vee Z_6 \Rightarrow IS_K$ does not hold, we infer that the inheritance is not correct in this example.

4.2 Example 2

Now, let the class T be a class derived from the same class K in the following way:

```
public class T extends K {
   public void m() {
      if(x>=5) throw RuntimeException();
      x++;
   }
}
```

The superclass constructor leaves the object in the initial state that satisfies

$$IS_T^0 \equiv (this \neq null) \wedge (x = 0).$$

The recursive sequence of predicates for the method m is now:

$$Z_0 \equiv (this \neq null) \wedge (x = 0)$$

$$Z_1 \equiv (this \neq null) \wedge (0 \leq x \leq 1)$$

$$Z_2 \equiv (this \neq null) \wedge (0 \leq x \leq 2)$$

$$Z_3 \equiv (this \neq null) \wedge (0 \leq x \leq 3)$$

$$Z_4 \equiv (this \neq null) \wedge (0 \leq x \leq 4)$$

$$Z_5 \equiv (this \neq null) \wedge (0 \leq x \leq 5)$$

$$Z_6 \equiv (this \neq null) \wedge (0 \leq x \leq 5)$$

Since $Z_5 \equiv Z_6$, the predicate Z_5 is a fixpoint. Further, since $O \vee Z_5 \Rightarrow IS_K$ we conclude that the inheritance is correct in this example.

6. CONCLUSION

Since it is technically possible to derive any class from any other, we introduced the concept of correct inheritance and established the appropriate conditions that must be satisfied by the subclass. These conditions describe the global disposition of a subclass to its superclass. Based on them, we formulated and presented a recursive algorithm for automatic verification of inheritance. Our future work will be directed towards further research of interclass relations and their automatic verification.

ACKNOWLEDGMENTS

This work was partially supported by the Ministry of Science and Education of the Republic of Serbia within the projects: ON 174026 and III 044006.

7. REFERENCES

[1] Amadi, M. and Cardelli, L. (1996), *A Theory of Objects*, Springer-Verlag, New York, USA.

[2] Berg, H. K., Boebert, W. E., Franta, W. R., and Moher, T. G. (1982), *Formal Methods of Program Verification and Specification*, Prentice-Hall, Englewood Cliffs, New Jersey.

[3] Dijkstra, E. W. (1975), "*Guarded Commands, Nondeterminacy and Formal Derivation of Programs*", Comunications of the ACM 18, 8 (August), 453–457.

[4] Dijkstra, E. W. (1976), *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, USA.

[5] Eckel, B. (2000), *Thinking in C++*, (*2nd Edition*), Prentice-Hall, Englewood Cliffs, NJ, USA.

[6] Eckel, B. (2006), *Thinking in Java*, (*4th Edition*), Prentice-Hall, Englewood Cliffs, NJ, USA.

[7] Gries, D. (1987), *The Science of Programming*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[8] Gordon, M. J. C. (1988), *Programming Language Theory and its Implementation*, Prentice-Hall International (UK) Ltd., Hertfordshire, UK.

[9] Hoare, C. A. R. (1969), *An Axiomatic Basis for Computer Programming*, Comunications of the ACM, 12, 10 (October), 576–585.

[10] Hoare, C. A. R. and Jifeng, H. (1998), *Unifying Theories of Programming*, Prentice-Hall, London.

[11] Huizing, K. and Kuiper, R. (2000), "*Verification of Object Oriented Programs Using Class Invariants*", In Proceedings of the Third International Conference on Fundamental Approaches to Software Engineering: Held as Part of the European Joint Conferences on the Theory and Practice of Software (ETAPS 2000). Lecture Notes in Computer Science, vol. 1783, Springer-Verlag, March 25-April 02, 208–221.

[12] Kupusinac, A. (2010), "*Analysis of Characteristics of Dynamic Postconditions in Hoare Triples*", PhD thesis, (*in Serbian*), Faculty of Technical Sciences, Department of Computing and Control, Novi Sad, Serbia; Mentor: Prof. dr Dušan Malbaški.

[13] Liskov, B. and Wing, J. (1994), "*A behavioral notion of subtyping*", ACM Transactions on Programming Languages and Systems (TOPLAS), 16, num. 6 (November), 1811–1841.

[14] Meyer, B. (1997), *Object–Oriented Software Construction*, (*2nd Edition*), Prentice-Hall.

[15] Sun Microsystem, Inc. (2002), "*javadoc Tool Home Page*", available at: http://java.sun.com/j2se/javadoc/ (accessed: 15 July 2008).

[16] Tarski, A. (1955), "*A lattice-theoretical fixpoint theorem and its applications*", Pacific Journal of Mathematics, 5, 285–309.

Automatska verifikacija nasleđivanja

Aleksandar Kupusinac, Dušan Malbaški

Primljen (27.05.2011.); Recenziran (17.09.2011.); Prihvaćen (22.10.2011.)

Rezime

Realizacija nasleđivanja u programskim jezicima ne postavlja nikakva ograničenja u pogledu odnosa potklase i natklase, odnosno u praksi je moguće iz jedne klase izvesti bilo koju drugu klasu. Međutim, sa aspekta teorije, nasleđivanje modeluje vezu specijalizacije, a to znači da u filozofskom smislu potklasa treba da je specijalna vrsta natklase. Ako se pak to previdi i/ili prekrši, tada u praksi nastaje ozbiljan problem prilikom polimorfne zamene pretka potomkom. Zbog toga je neophodno uvesti pojam korektnog nasleđivanja i postaviti uslove koje potklasa mora da zadovolji. U ovom radu razmatramo formalne uslove i na bazi njih formulišemo rekurzivni algoritam koji omogućava automatsku verifikaciju nasleđivanja na nivou metoda potklase. Naše rešenje se bazira isključivo na predikatskoj logici prvog reda.

Ključne reči: *formalne metode, nasleđivanje, objektno-orjentisana programska verifikacija, programska korektnost*