On the Complexity of Type Inference with Coercion

Mitchell Wand*

College of Computer Science
Northeastern University
360 Huntington Avenue, 161CN
Boston, MA 02115, USA

Patrick O'Keefe

ICAD, Inc.
1000 Massachusetts Avenue
Cambridge, MA 02139

Abstract

We consider the following problem: Given a partial order $(C, \leq)$ of base types and coercions between them, a set of constants with types generated from C, and a term M in the lambda calculus with these constants, does M have a typing with this set of types? This problem abstracts the problem of typability over a fixed set of base types and coercions (*e.g.* int $\leq$ real, or a fixed set of coercions between opaque data types). We show that in general, the problem of typability of lambda-terms over a given partially-ordered set of base types is NP-complete. However, if the partial order is known to be a tree, then the satisfiability problem is solvable in (low-order) polynomial time. The latter result is of practical importance, as trees correspond to the coercion structure of single-inheritance object systems.

1. Introduction

It is well known that the problem of type inference for the simply-typed lambda calculus reduces to the unification problem. The most general unifier gives a principal type: the typings of the terms are just the substitution instances of the principal type. If one adds coercions between base types, then John Mitchell has shown that type inference reduces to solving a set of inequalities between type variables and base type constants. The typings of the terms are those substitution instances of the principal type that satisfy the the inequalities.

If the set of coercions between base types is not specified in advance, then this is all that is needed to determine typability: one may always solve the inequalities by interpreting all the variables as a single base type. On the other hand, in real applications one typically deals with a fixed set of base types and coercions between them. For example, one might have the coercion int $\leq$ real, or some fixed set of coercions between opaque data types. In this case, it is necessary to determine whether the set of inequalities has a solution in the given base types.

In this paper we consider the complexity of typability over a given partial order of base types. We show the following:

- In general, the problem of typability of lambda-terms with constants over a given partially-ordered set of base types is NP-complete.
- However, if the partial order is known to be a tree, then the typability problem is solvable in (low-order) polynomial time.

The latter result is of practical importance, as trees correspond to the coercion structure of single-inheritance object systems.

In Section 2, we review the problem of type inference in the presence of coercions and show how it reduces to a problem of satisfiability over a partial order, which we call PO-SAT. In Section 3, we show that PO-SAT is NP-complete. In Section 4, we finish the reduction by showing that type inference with coercions is NP-complete as well. In Section 5, we show that an important special case, in which the partial order is known to be a tree, is solvable in low-order polynomial time. In Sections 6 and 7 we discuss related work and present conclusions.

2. Type Inference with Coercions

It is useful to think of the ordinary type inference rules (in the absence of let) as a set of constraints on the type expressions which appear in the derivation. In this view, we assign a type variable to every subterm and to every binding occurrence of a variable. The type inference rules may be stated as constraints on the types which can appear in the corresponding positions in the derivation. We write a constraint for each node in the parse tree (isomorphic, of course, to the derivation tree):

- for each applied occurrence of a variable x, generate the constraint $A(x) = t_x$, where t_x is the type variable corresponding to this applied occurrence of x and $A(x)$ is the type variable corresponding to the relevant binding occurrence of x.
- for each occurrence of a constant c, generate the constraint $t_c = t_M$, where t_M is the type variable corresponding to this occurrence of c and t_c is the type of the constant.

*Work supported by the National Science Foundation under grants numbered DCR-8605218 and CCR-8801591.

- for each occurrence of an application $(M\ N)$, generate the constraint $t_M = t_N \rightarrow t_{(MN)}$, where each type variable is the type variable corresponding to the occurrence of the indicated term.

- for each occurrence of an abstraction $\lambda x.M$, generate the constraint $t_{(\lambda x.M)} = t_x \rightarrow t_M$, where each type variable is the type variable corresponding to the indicated occurrence (a binding occurrence in the case of t_x).

It is easy to see that this formulation is equivalent to the usual inference rules, so that the solutions to the generated set of equations correspond to the possible type derivations. Thus the existence of most general unifiers implies the existence of principal types. This reduction is folkloric [e.g. Cardelli 85, Clément *et al.* 86], and is implicit in [Hindley 69, Milner 78].

To add coercion, one defines a coercion relation $\leq$ on types, with the intention that $\alpha \leq \beta$ means that values of type α are coercible to values of type β. One does this by defining the coercion relation on the base types and extending it to functional types by defining:

$$(\alpha \rightarrow \beta) \leq (\alpha' \rightarrow \beta') \iff (\alpha' \leq \alpha) \wedge (\beta \leq \beta')$$

One then adds the coercion rule

$$\frac{A \vdash M : \alpha \quad \alpha \leq \beta}{A \vdash M : \beta}$$

This violates the syntax-directed form of the usual typing rules, but it is easy to see that one may transform any type derivation using this rule into one in which the rule only occurs at the leaves. Therefore, the same typings are obtained by omitting the coercion rule and modifying the rules for identifiers and constants to read:

- for each applied occurrence of a variable x, generate the constraint $A(x) \leq t_x$, where t_x is the type variable corresponding to this applied occurrence of x and $A(x)$ is the type variable corresponding to the relevant binding occurrence of x.

- for each occurrence of a constant c, generate the constraint $t_c \leq t_M$, where t_M is the type variable corresponding to this occurrence of c and t_c is the type of the constant.

Mitchell [Mitchell 84] observed that the resulting inequations between terms could be reduced to solving inequations between variables. The basic observation was that because of the way $\leq$ was defined, any two comparable closed types had to be the same except perhaps for some inequations between base types that occurred at corresponding leaves of the trees. Therefore, Mitchell sketched an algorithm (later worked out in more detail in [Fuh & Mishra 88]) which did unification on trees "up to the leaves" and then generated the appropriate inequalities between leaf variables. The result was a principal type scheme and a set of inequalities between type variables. The types of the original program term were just those instances of the principal type scheme in which the values assigned to the type variables satisfied the inequalities.

If we say a term is typable iff there is some set of base types and coercions which gives the term a type, then this algorithm determines typability: just assign all the variables the same base type. However, if we are given a finite set of base types and coercibilities between them, then we need to know if there is an assignment of variables to types which satisfies the inequalities. There may not always be one: consider a set of two incomparable base types and a single variable which must be coercible to both. This leads us to the problem we call PO-SAT.

The problem PO-SAT may be stated as follows: Given a partial order $(C, \leq)$, and a set of inequations of the form $v \leq c$, $c \leq v$, and $v \leq v'$, where v, v', etc., are variables and c, etc, are constants drawn from C, is there an assignment from the variables to members of C that make all the inequations true?

This is not quite the same as the question asked for the type inference problem: typability asks for any assignment of types to the type variables, while PO-SAT permits only an assignment of base types. This gap is bridged by the following lemma:

Lemma. *Given a partial order $(C, \leq)$ of base types and a set of inequations of the form $v \leq c$, $c \leq v$, and $v \leq v'$, where v, v', etc., are variables and c, etc, are constants drawn from C, if there is any assignment of types to the variables that satisfies the equations, then there is an assignment in which every variable is assigned to a base type.*

Proof: Imagine there is an assignment of types to variables which satisfies the inequations. If some variable v is assigned a non-base type, then it must be incomparable to any base type, that is, no equation $v \leq c$ is deducible from the original equations. Assign all such variables the same base type, leaving all other assignments unchanged. By the preceding observation, this cannot violate any equation. *QED*

3. PO-SAT is NP-complete

We now proceed to consider the complexity of PO-SAT.

Theorem. *PO-SAT is NP-complete.*

Proof: PO-SAT is clearly in NP. To show that it is NP-hard, we give a direct reduction from 3-SAT. We first build a switch, which we call *one-of*$(v, c_1, \ldots c_n)$. This is a partial order and a set of inequations which are satisfied only by assigning v the value of one of $c_1, \ldots, c_n$.

To build *one-of*$(v, c_1, \ldots c_n)$, build a partial order consisting of $c_1, \ldots c_n$ and four other constants: a, a', b, b' under the ordering that a and a' are less than each of the c_j and each c_j is less than b and b'. Now add the following inequations:

$$a \leq v,\ a' \leq v,\ v \leq b,\ v \leq b'$$

It is clear that these equations are satisfied if and only if v is assigned one of the c_j. Furthermore, even if this partial order is embedded in a larger partial order, this property will be maintained so long as the embedding adds no new elements between the a, a' and b, b'. To ensure this, we will use fresh constants for a, a', b, b' whenever we build a new switch. We will sometimes call these *guard values*. A typical switch is shown in Figure 1.

Given a set of S of clauses over a set X of propositional variables, we construct an instance of PO-SAT as follows.

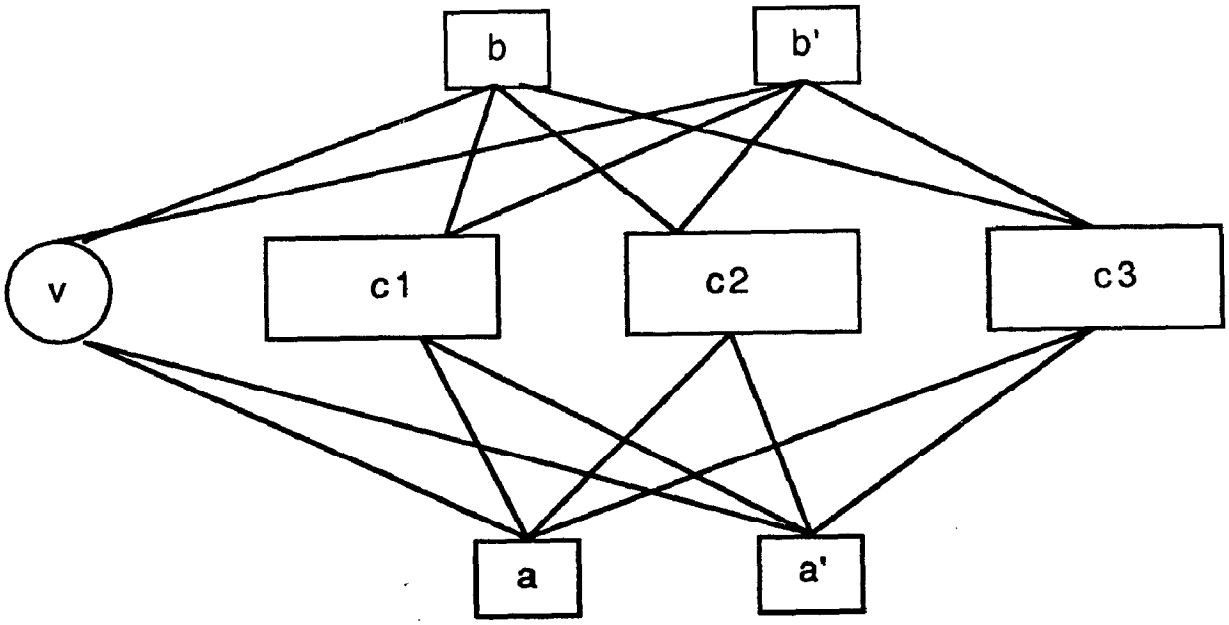

Figure 1: *one-of*$(v, c1, c2, c3)$

For each variable $x \in X$, we introduce two new constants labelled $(x = 0)$ and $(x = 1)$ and an ordinary variable (also called x), and build the switch *one-of*$(x, (x = 0), (x = 1))$. This is shown in Figure 2.

Next, given a clause, we will construct a partial order and a set of inequalities whose solutions correspond to the valuations that make the clause true. Each clause consists of three literals (variables or their negations). We introduce a variable g and a new constant for each of the seven valuations $I_1, \ldots I_7$ which make the clause true, and we construct the switch *one-of*$(g, I_1, \ldots, I_7)$. Thus a value for g is to correspond to a valuation making the clause true. For each variable in the clause, we construct a switch as above. We connect these switches as follows: for each propositional variable x, we make $(x = 1) \leq I_k$ whenever x is 1 in I_k (and similarly for $(x = 0)$). For each propositional variable x in the clause, we add the inequality $x \leq g$.

In this way, each x must be assigned a truth value, and g must be assigned a valuation, and $x \leq g$ under this assignment iff the truth values correspond to a valuation which makes the clause true. A picture of this network is shown in Figure 3.

Now, given a set S of clauses, we construct a set of switches for its variables and construct the network for each clause as above. Then the resulting system of inequations is solvable if and only if the original set of clauses is satisfiable. *QED*

4. Type Inference with Coercion is NP-Complete

The problem TIC (Type Inference with Coercion) may be stated as follows: Given a partial order $(C, \leq)$ of base types and coercions between them, a set of constants with types generated from C, and a term M in the lambda calculus with these constants, does M have a typing with this set of types?

So far we have shown that TIC reduces to PO-SAT and that 3-SAT reduces to PO-SAT. Now we complete the picture by showing that the reduction of 3-SAT to PO-SAT factors through TIC.

Theorem. *TIC is NP-complete.*

Proof: TIC is in NP, since it reduces to PO-SAT. Given a set S of clauses, we will construct an instance of TIC which is typable iff S is satisfiable. Given a set S of clauses, construct the partial order $(C, \leq)$ as before. Now we will construct a lambda-term M which yields the same set of inequalities as in the previous theorem.

We simplify our notation by letting the type variable associated with each binding occurrence of a lambda-variable v be just v. We will make sure that this does not lead to conflicts.

Let us introduce the following constants:

- a constant T of type $\alpha \to \alpha \to \alpha$ for all α.
- for each $c \in C$, a constant, also called c, of type c. (Note that this extends our convention on the types of bound variables to constants).
- for each $c \in C$, a constant $c^\natural$ of type $c \to c_0$ for some fixed c_0.

Observe that applying the algorithm in Section 2 to the term $((\lambda v.M)\,(T\,v_1\,v_2))$ yields the inequalities $v_1 \leq v$ and $v_2 \leq v$. Similarly, the term $\lambda v.(T\,(c^\natural\,v)(c'^\natural\,v))$ yields the inequalities $v \leq c$, $v \leq c'$. We introduce $(T\,M_1 \ldots M_n)$ as syntactic sugar for $(T\,M_1\,(T\,M_2 \ldots (T\,M_{n-1}\,M_n)))$ to extend T to an arbitrary number of arguments when needed. Last, let us introduce the notation

$$\text{let } x_1 = M_1 \ \ldots \ x_n = M_n \text{ in } N$$

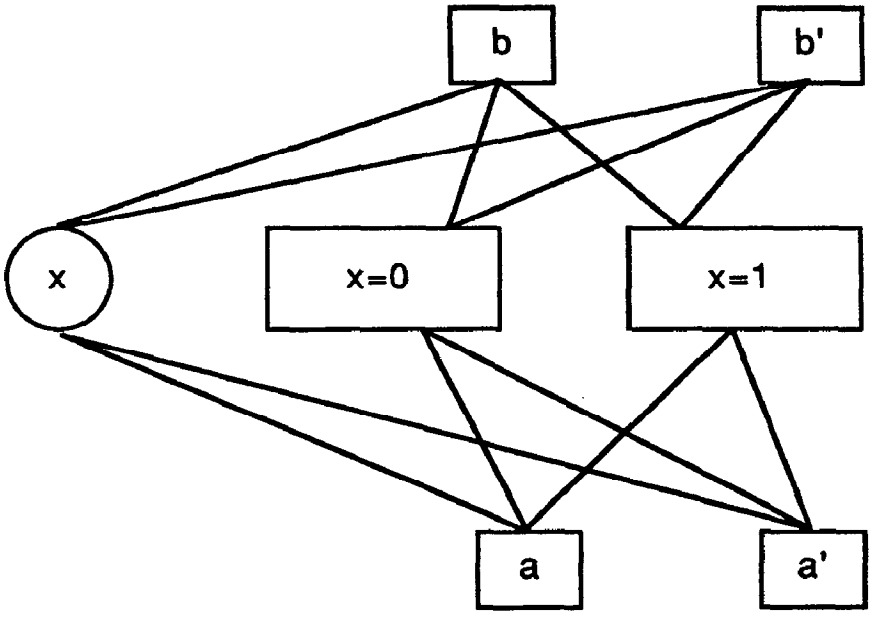

Figure 2: *one-of*$(x, (x = 0), (x = 1))$

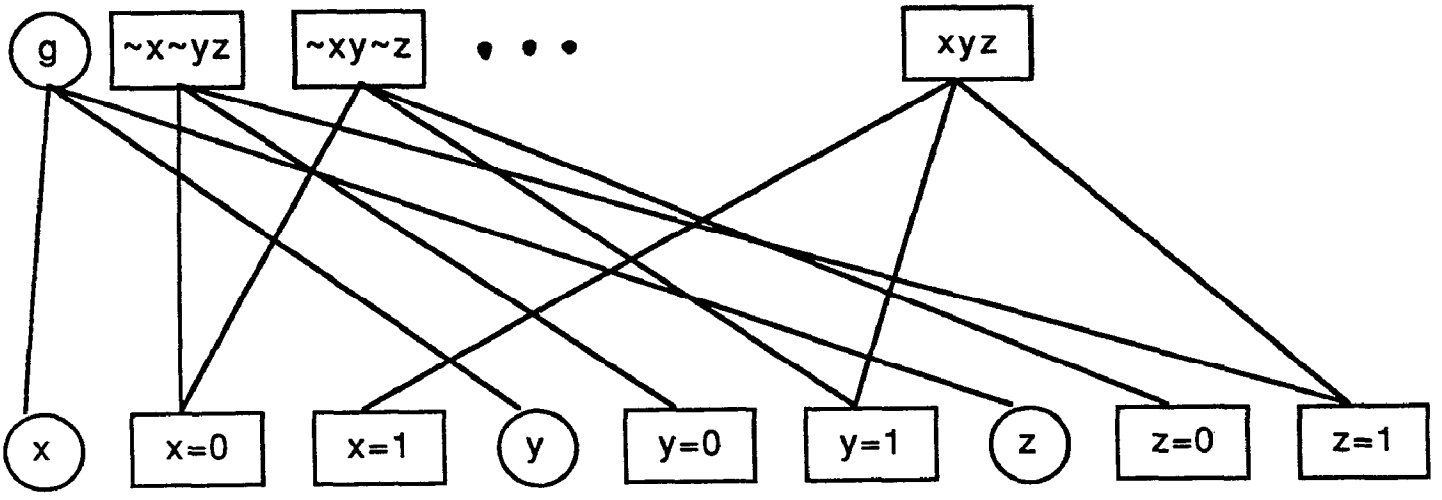

Figure 3: Partial order for the clause $x \vee y \vee z$

as syntactic sugar for

$$(\lambda x_1 \ldots \lambda x_n.N)\, M_1 \ \ldots \ M_n$$

Note that this is *not* the ML polymorphic let.

In order to produce the desired term, we need to have names for the elements in the various switches. We had a switch for each propositional variable x_i, with guard values $a_{x_i}, a'_{x_i}, b_{x_i}, b'_{x_i}$, and a switch for each clause g_j, with guard values $a_{g_j}, a'_{g_j}, b_{g_j}, b'_{g_j}$. The inequalities we want are:

$$a_{x_i} \leq x_i, a'_{x_i} \leq x_i, x_i \leq b_{x_i}, x_i \leq b'_{x_i}$$

for each propositional variable x_i, and

$$a_{g_j} \leq g_j, a'_{g_j} \leq g_j, g_j \leq b_{g_j}, g_j \leq b'_{g_j}$$

for each clause g_j. We can generate these inequalities by the following program:

let $x_i = (T\, a_{x_i}\, a'_{x_i})$ ...
in let $g_j = (T\, a_{g_j}\, a'_{g_j}\, x_{j_1}\, x_{j_2}\, x_{j_3})$ ...
in $(T\, (b_{x_i}{}^{\dagger}\, x_i)\, (b'_{x_i}{}^{\dagger}\, x_i) \ \ldots \ (b_{g_j}{}^{\dagger}\, g_j)\, (b'_{g_j}{}^{\dagger}\, g_j))$

Here $x_{j_1}, x_{j_2}, x_{j_3}$ are the variables in clause g_j.

This completes the reduction. *QED*

5. Solving Inequalities on Trees

In practical situations, however, we have some control over the structure of coercions in the set of base types C. For example, in a single-inheritance system, the set of coercions is generated by relations of the form $t_1 \leq t_0$ when t_0 is the parent of t_1. The coercion structure is that of a directed forest (a set of incomparable trees). An example is given in Figure 4. In this case, we can give a feasible algorithm for solving PO-SAT.

Theorem. *If the partial order $(C, \leq)$ is a tree, then PO-SAT is solvable in polynomial time.*

Proof: We first form the deductive closure of the given inequations and the inequations in the partial order. For each variable v, consider the set $U_v = \{c \mid v \leq c\}$ of upper bounds for v. Because C is a tree, either U_v has a greatest lower bound or no lower bound. v must be a lower bound for U_v, so if U_v has no lower bound, then there are no solutions. Otherwise, let σ_v denote the greatest lower bound of U_v.

Similarly consider the set $L_v = \{c \mid c \leq v\}$ of lower bounds of v. The value of v, if any, must be an upper bound of L_v. Therefore, if σ_v is not an upper bound of L_v, there are no solutions.

If each v passes this test, assign each v the value σ_v. We claim that this gives a solution. By construction, this assignment satisfies each equation of the form $v \leq c$ or $c \leq v$. Last, if one of the equations was of the form $v \leq w$, then $U_w \subseteq U_v$, so $\sigma_v \leq \sigma_w$, so this equation is satisfied also. *QED*

This algorithm applies to any partial order with the property that any subset with a lower bound has a greatest lower bound. This includes finite semilattices, directed forests, etc.

6. Related Work and Conclusions

This work was motivated by our desire to better understand [Mitchell 84]. [Jategaonkar & Mitchell 88] extend [Mitchell 84] to a system with records and pattern matching. [Fuh & Mishra 88] provide some of the details of the algorithms in [Mitchell 84] and raise the issue of deciding satisfiability in a fixed partial order. However, they leave the correctness of their algorithm as a conjecture (p. 112). Our examples illustrate the failure of their algorithm.

We have given an analysis of the complexity of type inference for lambda-terms with coercion. If there is no restriction on the partial order of coercion on the base types, then the problem of deciding whether a term is typable in a given partial order is NP-complete. If the partial order is restricted to a tree or similar structure, then the problem is decidable in low-order polynomial time.

These results hold only for lambda-calculus terms with constants. If ML polymorphic let is added, then the problem becomes PSPACE-complete [Kanellakis & Mitchell 89]. It remains open to what extent adding coercion to this language makes matters worse.

References

[Cardelli 85]
Cardelli, L. "Basic Polymorphic Typechecking," *Polymorphism Newsletter 2,1* (Jan, 1985). Also appeared as Computing Science Tech. Rep. 119, AT&T Bell Laboratories, Murray Hill, NJ.

[Clément *et al.* 86]
Clément, D., Despeyroux, J., Despeyroux, T., and Kahn, G. "A Simple Applicative Language: Mini-ML" *Proc. 1986 ACM Symp. on Lisp and Functional Programming*, 13–27.

[Fuh & Mishra 88]
Fuh, Y.-C., and Mishra, P. "Type Inference with Subtypes," *Proc. European Symposium on Programming* (1988), 94–114.

[Hindley 69]
Hindley, R. "The Principal Type-Scheme of an Object in Combinatory Logic," *Trans. Am. Math. Soc. 146* (1969) 29–60.

[Jategaonkar & Mitchell 88]
Jategaonkar, L.A., and Mitchell, J.C. "ML with Extended Pattern Matching and Subtypes," *Proc. 1988 ACM Conf. on Lisp and Functional Programming*, 198–211

[Kanellakis & Mitchell 89]
Kanellakis, P.C., and Mitchell, J.C. "Polymorphic Unification and ML Typing" *Conf. Rec. 16th Ann. ACM Symp. on Principles of Programming Languages* (1989), 105–115.

[Milner 78]
Milner, R. "A Theory of Type Polymorphism in Programming," *J. Comp. & Sys. Sci. 17* (1978), 348–375.

[Mitchell 84]
Mitchell, J.C. "Coercion and Type Inference (summary)," *Conf. Rec. 11th Ann. ACM Symp. on Principles of Programming Languages* (1984), 175–185.

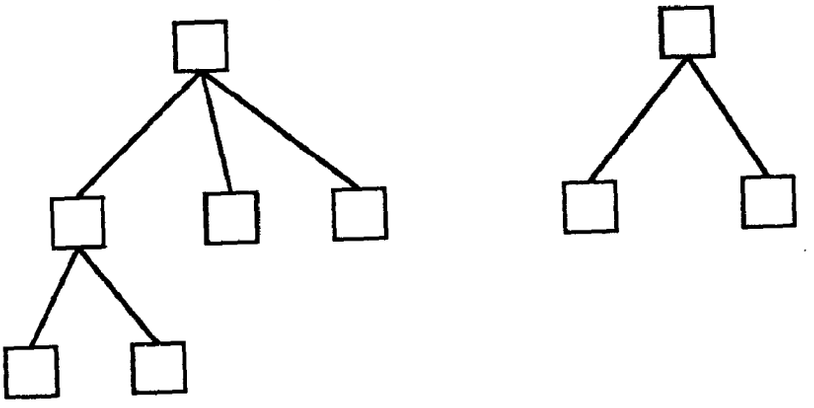

Figure 4: A single-inheritance hierarchy