オブジェクト指向データベースにおける型検査問題の計算量

清水 將吾[†]，石原 靖哲[†]，関 浩之[†]，伊藤 実[†]

† 奈良先端科学技術大学院大学

630-01 奈良県生駒市高山町 8916-5

E-mail: shougo-s@is.aist-nara.ac.jp

内容梗概

オブジェクト指向プログラミング言語の本質的な特徴の一つとして，メソッド呼出し機構が挙げられる．この機構は，データのカプセル化やコードの再利用のために重要なものであるが，実行時に型エラーが起こる危険性がある．特に，オブジェクト指向データベース（OODB）の問合わせプログラムの場合は，実行時に型エラーが起こると，ロールバックを行う必要がある．したがって，与えられた OODB スキーマが型整合性を持つこと，すなわち，どんなデータベースインスタンスの下でも問合せプログラムの実行中に型エラーが起こらないことを保証できることが望ましい．本稿では，Hull らによって提案された更新スキーマを OODB スキーマのモデルとして採用し，(1) 停止性を満たすスキーマに対する型検査問題, および, (2) メソッド定義中に更新操作がないスキーマに対する型検査問題がいずれも決定不能であることを示す．

1. まえがき

オブジェクト指向プログラミング言語の本質的な特徴の一つとして，メソッド呼出し機構が挙げられる．この機構は，クラス階層に沿った継承によるメソッドの多重定義と遅延束縛に基づいている．多重定義により，あるメソッド名 m に対して，複数のクラスが m の異なった定義を持つことができる．オブジェクト o に対して m が呼び出されたとき，o が属するクラスに従って m の定義のうちの一つが選択され，実行時に m に束縛される（遅延束縛または動的束縛）．この機構は，データのカプセル化やコードの再利用のために重要なものであるが，実行時に型エラー（例えば，m に束縛すべき定義が存在しない場合など）が起こる危険性がある．特に，オブジェクト指向データベースの問合わせプログラムの場合は，実行時に型エラーが起こると，ロー

ルバックを行う必要がある．したがって，与えられたオブジェクト指向データベーススキーマ S が型整合性を持つこと，すなわち，どんなデータベースインスタンスの下でも問合わせプログラムの実行中に型エラーが起こらないことを，コンパイル時（実行前）に保証できることが望ましい．以下，S が型整合性を持つか否かを判定する問題を，S に対する型検査問題と呼ぶ．型検査問題を解くためには，通常，各メソッドの返り値やメソッド定義（プログラム）中の変数の値がどのクラスのオブジェクトになり得るかを解析する必要がある．

オブジェクト指向データベーススキーマのモデルとしては，Hull らにより更新スキーマと呼ばれるモデルが提案されている [1]，[9]．更新スキーマは，クラス階層，継承，複合オブジェクトといったオブジェクト指向データベースの基本的な特徴をすべて持つ．メソッドの定義は手続き型オブジェクト指向プログラミング言語に基づく．したがって，データベースインスタンスの更新はオブジェクトの属性へのオブジェクトの代入として容易にモデル化される．

更新スキーマに対する型検査問題は，一般には決定不能であることが知られている [1]，[9]．さらに，以下のような更新スキーマの部分クラスに対する型検査問題の計算量が知られている．

- クラス階層の高さが 0 のとき，多項式時間可解 [10]．
- クラス階層の高さが高々1 であっても，決定不能 [10]．
- 停止性を満たし，かつメソッド定義中に更新操作がないとき，多項式時間可解 [14]．
- メソッド定義中に再帰がないとき，co 非決定性指数時間完全 [10]．

本稿では，以下のような更新スキーマの部分クラスに対する型検査問題が決定不能であることを示す（図 1参照）．

1. 停止性を満たし，かつクラス階層の高さが高々1 であるスキーマ．
2. メソッド定義中に更新操作がなく，かつクラス階層の高さが高々1 であるスキーマ．

本稿では，更新スキーマに以下の 3 つの制約をおく．まず，多重継承は存在しないという制約をおく．しかし，適当な継承機構を仮定すれば，本稿の結果は多重継承が

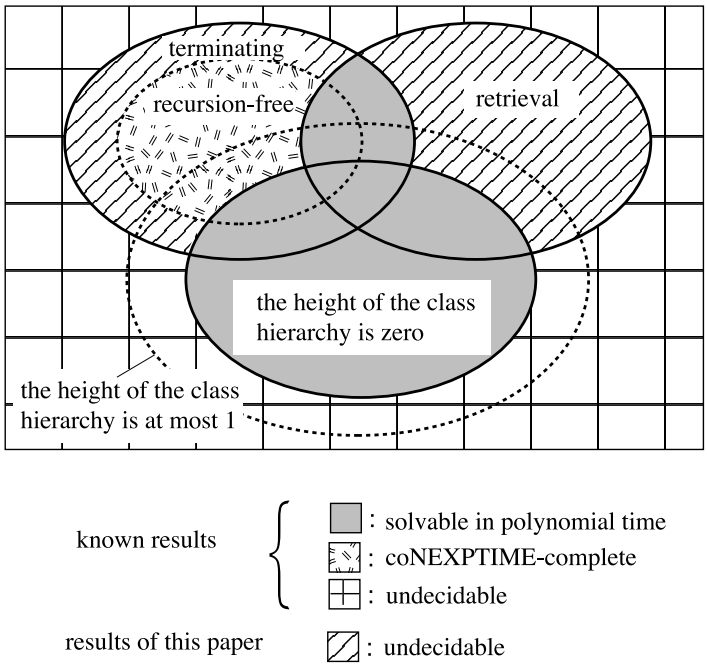

図 1　型検査問題の計算量

ある場合についても成り立つ．すなわち，この制約は議論の簡単のためにおいた制約である．次に，メソッド定義中には条件文が使えないという制約をおく．しかし，2.2 の例 2で示すように，更新操作を用いて if-then 文を模倣することができる．最後に，各メソッドの引数の個数は 1 であるという制約をおく．この制約をおかない場合，文献 [2] の結果により，停止性を満たし，かつメソッド定義中に更新操作がないスキーマに対する型検査問題が決定不能となることが容易に示せる．その一方で，クラス階層の高さが 0 のスキーマに対する型検査問題は多項式時間可解であることが示せる．また，再帰なしスキーマに対しても，決定可能であることが予想される．

関連研究としては，まず，Abiteboul らの文献 [2] が挙げられる．この文献では，メソッドスキーマと呼ばれる関数型オブジェクト指向プログラミング言語モデルが提案されており，そのいくつかの部分モデルに対する型検査問題の計算量が示されている．さらに，文献 [15] では，再帰なしメソッドスキーマの型検査を行う最適なインクリメンタルアルゴリズムが示されている．また，文献 [12] では，手続き型オブジェクト指向プログラミング言語に対する型推論アルゴリズムが提案されている．このアルゴリズムでは，プログラム中の各式 e に対して e の型を表す変数 $[\![e]\!]$ を導入し，こ

れらの変数間の関係を表す等式の最小解を求めることによってプログラムの型整合性を解析する（文献 [11]，[13] も参照されたい）．なお，文献 [4], [5], [14] などで，オブジェクト指向データベースにおけるメソッドベースのアクセス権モデルが提案されているが，これらのモデル上でのアクセス権の判定には型検査の技術が必要であることが知られている．

一方，文献 [3] や [6] では，各メソッドのシグネチャ（引数のオブジェクトのクラスと返り値のオブジェクトのクラスとの関係）を入力として，それらが contravariance などの条件を満たしているかどうかを検査する問題について考察している．しかし，この問題は，我々や Abiteboul らが扱っている問題と違い，プログラムの内容を解析せずに解くことができる．また，[7], [8] では，プログラム中の変数や実行文などの型を推論するための公理系を与えているが，型検査問題を解くための計算量についてはほとんど議論されていない．

2. メソッド実行モデル

2.1 構文

データベーススキーマは，次の 6 項組 $\mathbf{S} = (C, \leq, Attr, Ad, Meth, Impl)$ である．

1. Cはクラス名の有限集合．
2. $\leq$はクラス階層を表す C上の半順序．$c \leq c'$のとき，c は c'の下位クラス（または，c'は c の上位クラス）であると言う．議論の簡単のため，$\leq$は森である（すなわち，もし $c \leq c_1$かつ $c \leq c_2$ならば，$c_1 \leq c_2$または $c_2 \leq c_1$）と仮定する．$\leq$ の高さを，異なる $c_0, c_1, \ldots, c_n$に対して $c_0 \leq c_1 \leq \ldots \leq c_n$が成り立つ最大の n と定義する．
3. $Attr$は属性名の有限集合．
4. $Ad : C \times Attr \to C$は属性宣言を表す部分関数．$Ad(c, a) = c'$は，「クラス c の任意のオブジェクトの属性 a の値は，クラス c' またはその下位クラスのオブジェクトである」ことを表す（詳しくは 2.2のデータベースインスタンスの定義を参照）．
5. $Meth$はメソッド名の有限集合．

6. $Impl : C \times Meth \to WFS$はメソッドの実行プログラムを表す部分関数．ここで$WFS$は，以下で定義する，構文的に正しいプログラム全体の集合である．任意の$c \in C$と$m \in Meth$に対して，$Impl(c, m)$を，cで定義されたmのメソッド定義と呼ぶ．

各実行文は，次の6つの形式のいずれかである．

1. $y := y'$,
2. $y := \mathrm{self}$,
3. $y := \mathrm{self}.a$,
4. $y := m(y')$,
5. $\mathrm{self}.a := y'$,
6. $\mathrm{return}(y')$.

ここで，y, y'は変数，$a \in Attr$，$m \in Meth$，selfはメソッドが呼び出されたオブジェクト（実引数となるオブジェクト）を表す予約語である．このうち，2.2で述べるように，形式5の実行文だけがデータベースの更新を行うことができる．以下，形式5の実行文を更新操作と呼ぶ．

$s_1; s_2; \cdots; s_n\ (= \alpha)$を実行文の有限系列とする．もし，(1) どの実行文$s_i\ (1 \leq i \leq n)$も値が未定義の変数を参照せず，かつ，(2) 最後の実行文s_nだけが$\mathrm{return}(y)$の形式であるならば，αを構文的に正しいプログラムと言う．

一般性を失うことなく，一時変数を省略して書くことがある．例えば，"$y' := \mathrm{self}.a; y := m(y');$"を，一時変数$y$を略して"$y := m(\mathrm{self}.a);$"と書く．

2.2 意味

メソッド定義の継承を表す部分関数$Impl^* : C \times Meth \to WFS$を次のように定義する．$c \in C, m \in Meth$とする．$c'$を，$Impl(c', m)$が定義されている$c$の（$\leq$に関する）最小の上位クラスとする．すなわち，$c'$は，「もし$Impl(c'', m)$が定義されていてかつ$c \leq c''$ならば，$c' \leq c''$」を満たすようなクラスである．このとき，$Impl^*(c, m) = Impl(c', m)$と定義する．もしそのような$c'$が存在しなければ，$Impl^*(c, m)$は未定義とする．

同様に，属性宣言の継承を表す部分関数 $Ad^* : C \times Attr \to C$を次のように定義する．$c \in C, a \in Attr$とする．もし $Ad(c', a)$ が定義されている c の最小の上位クラス c'が存在するならば $Ad^*(c, a) = Ad(c', a)$ とし，さもなければ $Ad^*(c, a)$ は未定義とする．

$\mathbf{S}$ のデータベースインスタンスは，次の 2 項組 $\mathbf{I} = (\nu, \mu)$ である．

1. νは Cを定義域とする関数で，$\nu(c)$ $(c \in C)$ はオブジェクトの有限集合．ただし，各クラス間のオブジェクト集合は互いに素（すなわち，もし $c \neq c'$ならば $\nu(c) \cap \nu(c') = \phi$）とする．各 $o \in \nu(c)$ を，($\mathbf{I}$ における）クラス c のオブジェクトと呼ぶ．更に，$O_{\mathbf{S},\mathbf{I}} = \bigcup_{c \in C} \nu(c)$ とし，各 $o \in O_{\mathbf{S},\mathbf{I}}$を，単に（$\mathbf{I}$ における）オブジェクトと呼ぶ．$cl(o)$ により，$o \in O_{\mathbf{S},\mathbf{I}}$の属するクラス，すなわち，$o \in \nu(c)$ となる c を表す．

2. μは有向グラフ$\mu = (V, E)$ である．ここで，Vはオブジェクトを表す頂点の集合 $O_{\mathbf{S},\mathbf{I}}$であり，$E$は以下の条件を満たすラベル付き有向辺の集合である．

   - $o \in \nu(c)$ とする．$Ad^*(c, a) = c'$ならば，ちょうど一つの $o' \in \nu(c'')(c'' \leq c')$ に対して，$o \xrightarrow{a} o' \in E$．$Ad^*(c, a)$ が未定義ならば，o はラベル a をもつ出射辺をもたない．

   $o \xrightarrow{a} o' \in E$のとき，$o'$をオブジェクト o の属性 a の値と呼び，$o.a$ で表す．

以下では，$\mathbf{I}$ の下での $\mathbf{S}$ の実行を簡単に説明する（形式的な定義は [1]，[9] または [10] を参照）．オブジェクト o に対してメソッド m が呼び出されたとき，o を，その呼出しにおけるセルフオブジェクトと言い，self と略記する．6 種類の各実行文に対して，次のような意味を与える．

$y := y'$ は，y'の内容を yに代入する．

$y :=$ self は，self を変数 yに代入する．

$y :=$ self$.a$ は，self の属性 a の値を yに代入する．

$y := m(y')$ は，変数 y'に代入されているオブジェクトに対するメソッド m の呼出しを表し，次のように実行される．変数 y'に代入されている値を $o \in \nu(c)$ とする．もし $Impl^*(c, m)$ が未定義ならば型エラーで停止する．もし $Impl^*(c, m) = \alpha$ならば，α において o を self に束縛し，αを実行する．返り値は yに代入する．

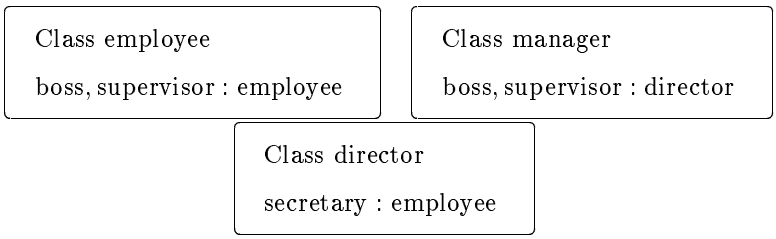

図 2　Ad_1の定義

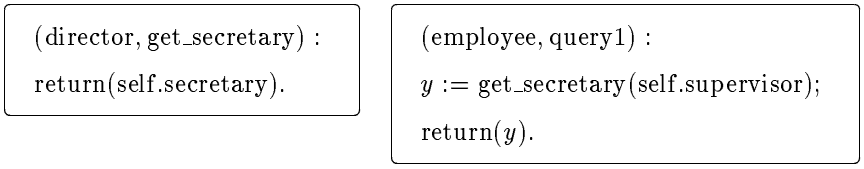

図 3　$Impl_1$の定義

実行文 self.$a := y'$**を考える．**self**の属するクラスを** c **とし，**y'**に代入されている値を** o **とする．**$Ad^*(c, a) = c'$**のとき，もし** o **が** $c'' \leq c'$**を満たすクラス** c''**のオブジェクトならば，**self **の属性** a **の値を** o **とする．さもなければ，型エラーで停止する．**

return(y) **は変数** y**の内容を返り値とすることを表す．**

S 中のどのメソッドをどんなデータベースインスタンスのもとで実行しても型エラーが起こらないとき，S は型整合性をもつと言う．但し，最初のメソッドの呼び出しとしては，$Impl^*(cl(o), m)$ **が定義されているような** o **に対する** m **の実行のみを考える．また，S 中のどのメソッドをどんなデータベースインスタンスのもとで実行しても必ず停止するとき，S は停止性を満たすと言う．**

例 1: データベーススキーマ $\mathbf{S}_1 = (C_1, \leq_1, Attr_1, Ad_1, Meth_1, Impl_1)$ **を考える．ここで，**

- $C_1 = \{\text{director}, \text{manager}, \text{employee}\}$ **であり，**director $\leq_1$ manager $\leq_1$ employee**．**
- $Attr_1 = \{\text{boss}, \text{supervisor}, \text{secretary}\}$ **であり，**Ad_1 **は図** 2**で定義される．**
- $Meth_1 = \{\text{get\_secretary}, \text{query1}\}$ **であり，**$Impl_1$**は図** 3**で定義される．**

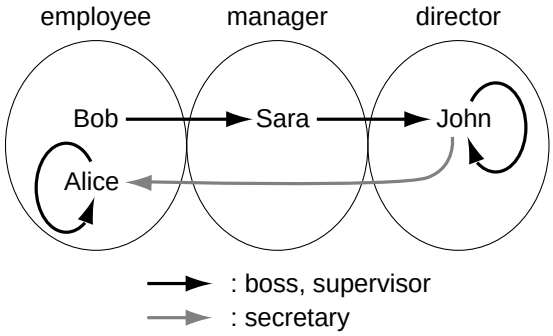

図 4　$\mathbf{I}_1$のデータベースインスタンス

(employee, calc_supervisor) :
$y :=$ calc_supervisor(self.boss);
return(y).

(director, calc_supervisor) :
return(self).

(director, get_secretary) :
return(self.secretary).

(employee, query2) :
self.supervisor := calc_supervisor(self);
$y :=$ get_secretary(self.supervisor);
return(y).

図 5　$Impl'_1$の定義

図 4は $\mathbf{S}_1$のデータベースインスタンス $\mathbf{I}_1 = (\nu_1, \mu_1)$ を表している．この図において，Bob**，**Sara**，... はオブジェクトであり，**Bob $\rightarrow$ Sara は $\mu_1(\text{Bob}, \text{boss}) = \mu_1(\text{Bob}, \text{supervisor}) = \text{Sara}$ であることを表している．**Bob に対する** query1 **の実行を考える．$\mu_1(\text{Bob}, \text{supervisor}) = \text{Sara} \in \nu_1(\text{manager})$ であり $Impl^*_1(\text{manager}, \text{get\_secretary})$ は未定義であるから，型エラーが起こる．したがって，$\mathbf{S}_1$は型整合性を持たない．また，$\mathbf{S}_1$が停止性を満たすことは容易に示せる．**

$\mathbf{S}'_1 = (C_1, \leq_1, Attr_1, Ad_1, Meth'_1, Impl'_1)$ **を考える．ここで，$Meth'_1 = \{\text{calc\_supervisor}, \text{get\_secretary}, \text{query2}\}$ であり，$Impl'_1$ は図 5で示される．このとき，$\mathbf{I}_1$は $\mathbf{S}'_1$のインスタンスでもある．**Bob **に対する** calc_supervisor **の実行は正常に終了し，**John **が返される．一方，**Alice **に対する** calc_supervisor **の実行は停止しない．** calc_supervisor **の実行が停止するとき，常にクラス** director **のオブジェクトが返され**

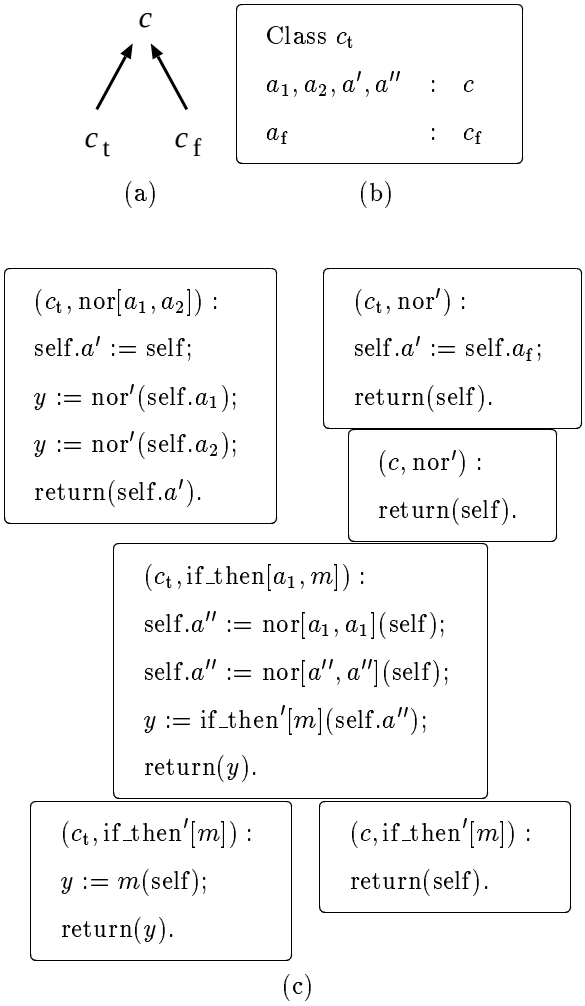

図 6　$\mathbf{S}_2$の定義

る．次に Bob **に対する** query2 **の実行を考える．図** 5**の** (employee, query2) **の第** 2 **文より，**Bob.supervisor **の値は** John $\in \nu_1$(director) **となり，**Bob **に対する** query2 **の実行は正常に終了する．**$\mathbf{S}_1'$**が型整合性を持つことは容易に示せる．** □

例 2: 以下のようなデータベーススキーマ $\mathbf{S}_2 = (C_2, \leq_2, Attr_2, Ad_2, Meth_2, Impl_2)$ **を考える．**

- $C_2 = \{c, c_t, c_f\}$ **であり，** $c_t \leq_2 c, c_f \leq_2 c$ **とする（図** 6（a)）**．**
- Ad_2**は図** 6（b）**で定義される．**

$\mathbf{S}_2$において論理値表現を次のように定める．o をクラス $c_{\rm t}$のオブジェクトとする．o の各属性 $a \in \{a_1, a_2, a', a'', a_{\rm f}\}$ について，$o.a = o$ のとき真，そうでないとき偽を表すものとする．$o.a_{\rm f}$は，$Ad_2(c_{\rm t}, a_{\rm f}) = c_{\rm f}$という宣言（図 6（b））により $o.a_{\rm f} = o$ とはなり得ないから，$o.a_{\rm f}$は常に偽を表す．

このとき，2 つのメソッド $\mathrm{nor}[a_1, a_2]$ と $\mathrm{if\_then}[a_1, m]$ を図 6（c）のように定義する．$\mathrm{nor}[a_1, a_2]$ は $o.a_1$ と $o.a_2$の NOR を計算し，結果が真のときは o を，偽のときは $o.a_{\rm f}$を返す．すべての論理演算子は NOR を用いて表すことができるので，$\mathrm{nor}[a_1, a_2]$ を用いれば任意の命題論理式を計算するメソッドを構成することができる．また，$\mathrm{if\_then}[a_1, m]$ は if-then 文の模倣を行い，$o.a_1 = o$ のときかつそのときに限り，o に対して m を呼び出す． □

3. 型検査問題の計算量

3.1 停止性を満たすスキーマ

定理 1: S を停止性を満たすデータベーススキーマとする．このとき S の型検査問題は，クラス階層の高さを高々1 に限定した場合でも決定不能となる． □

ここでは，証明の概略についてのみ述べる（証明の詳細は付録 A参照）．本定理の証明をチューリング機械の受理問題からの帰着により行う．具体的には，任意のチューリング機械 Mとそれに対する入力 x に対し，次の条件を満たすスキーマ $\mathbf{S}_{M,x}$を構成する．

- 停止性を満たす．
- クラス階層の高さが 1．
- Mが x を受理するとき，かつそのときのみ，$\mathbf{S}_{M,x}$が型整合性を持たない．

$\mathbf{S}_{M,x}$のクラス階層を図 7，属性宣言を図 8のように定義する．$\mathbf{S}_{M,x}$はクラス階層の高さが 1 である．メソッド TM を図 9のように定義する．メソッド test 以外のメソッドは，$\mathbf{S}_{M,x}$のすべてのクラスに対して定義しておく．test はクラス $c_{\rm f}$に対してのみ定義しておく．

以下，データベースインスタンス $\mathbf{I} = (\nu, \mu)$ の下で，$o \in O_{\mathbf{S}_{M,x},\mathbf{I}}$を引数としてメソッド m を実行することを $m(o)$ と書く．

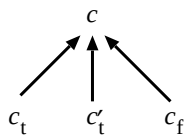

図 7　$\mathbf{S}_{M,x}$のクラス階層

Class c_{t}		
$a_{\Rightarrow}$	:	c
$a_1, \ldots, a_K$	:	c
$a'_1, \ldots, a'_K$	:	c
$a''_1, \ldots, a''_K$	:	c
$a'''_1, \ldots, a'''_K$	:	c
a_{cont}	:	c
$a_{\mathrm{yes}}, a'_{\mathrm{yes}}$	:	c
a_{f}	:	c_{f}
$a_{\mathrm{ws}}, a'_{\mathrm{ws}}$	:	c

Class c'_{t}		
a'_{yes}	:	c
a_{f}	:	c_{f}

Class c		
a'_{t}	:	c'_{t}

図 8　$\mathbf{S}_{M,x}$の属性宣言

オブジェクト $o \in \nu(c_{\mathrm{t}})$ の属性 a_{ws}は，メソッド m の定義中に再帰がある場合でも，m の実行が無限ループに陥らないことを保証するために用いられる．$\mathbf{S}_{M,x}$中の再帰を含む各メソッド m の動作を以下のように定義する．

- **$o \in \nu(c_{\mathrm{t}})$ に対して m が呼び出されたとき，$o.a_{\mathrm{ws}}$の値が真であれば，$o.a_{\mathrm{ws}}$の値を偽とした後，$m(o)$ の実行を続ける．そうでなければ，$m(o)$ の実行をこれ以上続けない．**

$O_{\mathbf{S}_{M,x},\mathbf{I}}$は有限集合であるので，これにより $\mathbf{S}_{M,x}$が停止性を満たすことが示される．また，$m(o)$ の実行が終了する直前に $o.a_{\mathrm{ws}}$の値を真に戻しておくことで，他のメソッドも属性 a_{ws}を利用できる．

ここで，定理 1の証明を行う前に，チューリング機械の定義をしておく．

定義 1: 決定性チューリング機械 Mは 3 字組 (Q, Σ, δ) で表される．ここで，

```
(c_t , TM) :
y := get_ws(self);
y := init_ws(self);
y := step(self);
y := accept(self);
y := test(y);
return(self).
```

図 9　メソッド TM の定義

- Q は状態の有限集合であり，初期状態 q_0，受理状態 q_{yes} という 2 つの特別な状態を含む．
- Σはテープ記号の有限集合であり，空白記号 B，先頭記号 $\triangleright$ という 2 つの特別な記号を含む．
- δは $(Q-\{q_{\mathrm{yes}}\})\times\Sigma$から $Q\times\Sigma\times\{\leftarrow,\rightarrow,-\}$ への関数である．$\delta(q,\ \triangleright)=(q',\sigma,d)$ ならば，$\sigma=\ \triangleright$かつ $d=\rightarrow$であるとする．つまり，テープヘッドがテープの左端から落ちることはないものとする．

□

定義 2: Mの時点表示 Iは有限列 $\langle q_1,\sigma_1\rangle,\ldots,\langle q_k,\sigma_k\rangle\ (k\geq 1)$ で表される．ここで，$q_i\in Q\cup\{\perp\},\sigma_i\in\Sigma\ (1\leq i\leq k)$，かつ，$\sigma_1=\ \triangleright$であり，ちょうど 1 つの q_iだけが Q の要素となっている（i はテープヘッドの位置を表す）．時点表示 Iの i 番目の要素対 $\langle q_i,\sigma_i\rangle$ を $I[i]$ と書く．時点表示間の遷移関係$\vdash_M$は通常のように定義する．また，M が入力 x に対し j回遷移した後の時点表示を I_jで表す．　□

以下，メソッド TM の各実行文の動作を説明する．

$o_0\in\nu(c_{\mathrm{t}})$ とする．$\mathrm{get\_ws}(o_0)$ の実行により，与えられたデータベースインスタンスから，Mの模倣を行うための次のような作業領域を得る．

- データベースインスタンスにより定まるある $k\ (\geq 1)$ に対して，$o_i.a_{\Rightarrow}=o_{i+1}\in\nu(c_{\mathrm{t}})\ (0\leq i\leq k-2), o_{k-1}.a_{\Rightarrow}=o_k\in\nu(c'_{\mathrm{t}})$．