数理解析研究所講究録
第1691巻 2010年 181-187

文脈を考慮したJavaプログラムの参照先解析に関する一考察

三重大学 大学院 工学研究科 田中 友幸 (Tomoyuki Tanaka), 大山口 通夫 (Michio Oyamaguchi)
Graduate School of Engineering, Mie University

1 概要

現在ソフトウェアはあらゆる業界で利用され，その利用域は今も拡大し続けている．そのなかでソフトウェアの実行効率や信頼性の向上は常に求められ，それらを実現するプログラム解析に関する研究が行われている．**参照先解析**はその根幹を担い，得られる情報は幅広く利用され，最終的にプログラムの最適化や信頼性向上につながる．そのため，より低コストでより正確な解析手法を求められ，これまで様々な手法が提案され盛んに研究されている．

参照先解析はプログラム中で使用される参照型の変数の取り得る値を求める静的解析ことである．本研究では2007年 Milanova によって提案された，Java プログラムに適した解析手法 [Mil07] を拡張し，計算量のオーダーを増加させずに，不正確な値を減少させることに成功した．

本稿において，変数とは主に参照型のフィールドやローカル変数のこと言う．フィールドとはクラス定義内で定義された変数のことを指し，スタティックフィールドとインスタンスフィールドを含む．またローカル変数はメソッド内で宣言された変数で，仮引数を含めた表現とする．

2 参照先解析 (Points-to Analysis)

参照先解析とはコンピュータ上で実行可能なプログラムを受け取り，使用される参照型の変数の取り得る値 (オブジェクト) を計算する．その結果は，変数とオブジェクトを表現する頂点と，それらの参照関係を表現する辺によって構成される参照先グラフで表現する．解析で扱うオブジェクトは生成文 (new 文) ごとに名前をつけて表現[1]する．ローカル変数は，変数名とそれが属するメソッド名，クラス名，パッケージ名と組み合わせ，フィールドはフィールド名とそれが属するオブジェクト名の組で表現する．

参照先解析は，数万から数十万行を越えるような大規模なプログラムを対象とする場合が多く，その場合でも現実的時間で実行可能な近似的手法が，様々研究されてきた．その近似では参照する可能性があるものは計算結果に含めるため，実際には参照しない誤った参照先が解析結果に現れる．その数が少ないほど正確だと言え，正確さと計算量はトレードオフの関係になるのが一般的である．従来手法は**フロー依存性**と**文脈依存性**により大きく4つに分類できる．

フロー依存性 (Flow-sensitivity) フロー (処理の流れ) を考慮するかどうかが問われる．流れを考慮するフロー依存解析はプログラムの文 (処理) ごとに変数の参照先を計算する[2]．一方，フロー非依存解析はすべての文から得られる結果をまとめて計算する．そうすることで，正確さは犠牲になるが，計算量やメモリ使用量を抑えることができることから，従来手法の多くがフロー非依存解析に基づいている．

文脈依存性 (Context-sensitivity) 手続きの，呼び出しの文脈に起因する，振る舞いの違いを考慮するかどうかが問われる．文脈非依存解析では，すべての呼び出しをまとめて計算するため，不正確な情報が生まれる．一方，文脈依存解析では手続きを文脈ごとに区別して考えられるため，引数やローカル変数の参照先がより正確になる．例えば $m(a)$; と $m(b)$; という呼び出しがある場合，文脈非依存では，(m の) 仮引数

[1] 文がループ内に存在している場合は1つの参照先と考える．
[2] コンパイラなどで利用されるデータフロー解析に類似する．

の値は a または b だが，文脈依存では，実引数が a である手続きと，b である手続きを別のものと考え，仮引数を区別できる．

文脈は「ある文脈 s1 で呼び出される手続き m1 の中の文脈 s2 で呼び出される手続き m2」というように入れ子構造になり，m2 の文脈は (s1,s2) と表現できる．この文脈の長さは再帰呼び出しなどで無限になるため近似が必要になる．文脈依存解析では，文脈を何で表現するか，文脈の長さをどう近似するかで正確さや計算コストが変わってくる．

3 従来手法

従来手法として，フロー非依存解析の中でより正確な Andersen-style の解析，文脈にオブジェクトを利用した文脈依存 (**オブジェクト依存**) 解析，さらにオブジェクトのアクセス関係を利用することで，計算量を抑えながら文脈依存を実現した Light Context-Sensitive 解析を紹介する．

3.1 Andersen-style の解析

Andersen は博士論文 [And94] の 4 章 "Pointer Analysis"(pp.111-152) の中で，C プログラムのポインタ解析について述べている．今までこの解析について，様々な研究が行われており [LH03, SF09]，現在でも注目を集める手法である．本稿では Andersen の解析に基づく，主に以下の 3 つの特徴をもつ解析のことを Andersen-style の解析と呼ぶ．

- フロー非依存
- 部分集合に基づく (subset-based)

  プログラム上の代入文において，代入する側が参照するオブジェクト集合は代入される側が参照する集合の部分集合であると考える．つまり代入文「$X = Y$」は「$X \supseteq Y$」と解釈する．

  対照的な手法としては等価性に基づいた (equality-based) 手法 [Ste96] があり，代入の両辺は同じものを参照するものとして扱う．そうすることで正確さは犠牲になるが，ほぼ線形時間で解が得られる．
- 文脈非依存

  Andersen は論文の中で文脈非依存解析にあたる手続き内 (intra procedural) 解析について述べている．さらにその拡張として手続き間 (inter procedural) 解析，ここで言う文脈依存解析にも言及しているが，今回は前者の解析に限定して述べる．

3.1.1 Java プログラムに対する解析

Java プログラムに対する Andersen-style の解析は，次の 4 種類の文[3]ごとに，参照先グラフに辺を追加する処理と考えられる．

文	追加する辺
(1) $s_i : l = newC$	$l \rightarrow O_i$
(2) $l = r$	$l \rightarrow O_i \in Pt(r)$
(3) $l.f = r$	$< O_i \in Pt(l), f > \rightarrow O_i \in Pt(r)$
(4) $l = r.f$	$l \rightarrow Pt(< O_i \in Pt(r), f >)$

(1) オブジェクト生成文において，s_iの i は生成文に対する通し番号で，生成されるオブジェクトは O_i と名付けられる．この文では変数 l から O_i への辺を追加する．(2) 代入文については，変数 l から，変数 r が参照するオブジェクト集合 $Pt(r)$ の要素すべてに辺を追加する．(3) フィールドへの書き込みに対しては，$< l$ が参照するオブジェクトと f との組 $>$ で表現されたフィールドから，r が参照するオブジェクトへ辺を追加する．(4) フィールドから読み出しでは，l から，$< r$ が参照するオブジェクトと f との組 $>$ で表現されたフィールドが参照するオブジェクトへの辺を追加する．

このように各文を処理するが，処理ごとに参照先が変化するため，各文を 1 回ずつ処理するだけでは不十分である．これを効率良く解く方法アルゴリズムについては [LH03, SF09] を参照されたい．

[3] メソッドの呼び出し文を考える場合には，対応する実引数から仮引数への代入文，返り値からメソッドが代入される変数への代入文を追加することで実現できる．

3.1.2 Andersen-style の解析の計算量

Andersen は自身の博士論文 [And94] の中で，解析の計算時間はプログラムサイズの多項式時間と述べるに留まっている．ここで述べるのは，より計算量が小さいと考えられている手法である．

Andersen-style の解析は，変数と参照先を頂点，代入文を有向辺と考えることでグラフの推移閉包を求める問題と考えられる．ただし，ポインタの Dereference により，推移閉包から求められない新たな辺が加わることがある．その場合，その辺に対する推移閉包を再び求める必要がある．このような枠組の問題は動的推移閉包 (Dynamic Transitive Closure) 問題と呼ばれている [SF09]．この問題は，辺の追加のみ，削除のみ，またはその両方を考慮する問題に分類される．この解析は辺の追加のみを考慮すれば良く，計算量は n を頂点数として $O(n^3)$ であることが証明されている [IK83]．

以上より Andersen-style の解析は $O(n^3)$ で計算できることがわかる．n はグラフの推移閉包を考える場合の頂点数だが，Andersen-style の解析に置き換えれば変数と参照先の数の和であり，プログラムサイズに比例する．

3.2 オブジェクト依存解析

Milanova らは 2002 年にメソッドが属するオブジェクト[4]を文脈として扱う文脈依存解析をオブジェクト依存 (Object Sensitive) 解析として提案している [MRR02, MRR05](以下，この手法を標準的なオブジェクト依存と言う意味で SObjectSens と呼ぶ)．メソッドを属するオブジェクトごとに区別して考えることで，文脈を考慮しない場合に生じる，オブジェクト指向型プログラム特有のカプセル化や継承による不正確さを改善することができる．

またこの手法はパラメータを与える事で，正確さや計算コストを調節できるように設計されている．パラメータとしてオブジェクトの抽象化の度合い (オブジェクトを生成文で表現するか，生成文と文脈の組で表現するか) や文脈で区別する変数を選ぶことができる．

論文 [LH06] によれば，他の文脈依存解析に比べ，オブジェクト依存解析は，無駄になる文脈が少なく，仮想メソッド呼出しの解決，キャスト安全性の解析などで効果的であるとされている．

3.2.1 オブジェクト依存解析の計算量

SObjectSens では解析にパラメータを与える事によって，解析の正確さと計算量を調節できる枠組みになっている．そのため計算量の考察は容易ではないが，ここでは最も単純な場合について考察する．

文脈依存解析はメソッドを文脈で区別するため，その文脈の数だけメソッドを複製した後，文脈非依存解析すると考える事ができる．SObjectSens の文脈はオブジェクトであり，最も単純な場合，オブジェクトは生成文で抽象化され，その総数は $O(n)$ である[5]．最悪の場合はプログラム中の全メソッドが，全オブジェクトに属している場合で，プログラムサイズは元のサイズの $O(n)$ 倍の $O(n^2)$ になる．

SObjectSens は文脈を考慮する以外は Andersen-style の手法を踏襲しているので，サイズ $O(n^2)$ のプログラムに対して 3 乗の解析を行うことになり，結果的に $O(n^6)$ の計算量が必要となる．

3.3 Light Context-Sensitive 解析

SObjectSens に比べ正確さは劣るが，計算コスト抑えた手法が Milanova によって 2007 年に提案されている [Mil07] (以下 LightSens と呼ぶ)．この手法はまず Andersen-style の解析を行い，その結果となる参照先グラフを作成すると共に，オブジェクト間のアクセス関係を近似した**オブジェクトグラフ**を作成し，その 2 つのグラフの**インターセクション (共通部分を取り出すこと)** により正確さを高める．

この手法で得られる情報の正確さは SObjectSens に劣るが，文脈を考慮しない Andersen-style の解析の正確さを改善することができ，時間計算量については文脈依存解析のなかで最も効率の良い部類に属する．

---

[4]呼び出しに利用される変数の参照先として計算される．

[5]オブジェクトをもっと詳細化するならば，ループの展開や文脈依存によって区別された文によって抽象化することができる．そうすることでより実行時に近いオブジェクトの抽象化が可能になり，解析の正確さは向上するが，オブジェクト総数は増加し，計算量にも影響を与える．

3.3.1 オブジェクトグラフ

オブジェクトグラフの頂点はオブジェクトと root である．root とはプログラムの開始地点となる main メソッドのことを表す．そして以下で述べる頂点間のアクセス関係を辺として表現する．

あるオブジェクト O1 が別のオブジェクト O2 にアクセス可能とは，O1 に属するフィールドや (メソッド内の) ローカル変数が O2 を参照している場合．それに加えて，O1 に属するメソッドから 1 回以上スタティックに呼び出されるメソッドのローカル変数が O2 を参照している場合である．

また，main メソッド，または main メソッドから 1 回以上スタティックに呼び出されるメソッドのローカル変数が O1 を参照している場合，root が O1 に対しアクセス可能と考える．

3.3.2 インターセクション

この手法のインターセクションはプログラム開始地点から到達可能なメソッド m 内のローカル変数 l に関する各種代入文によって以下の 2 つに場合に分かれる．

- $l = r, l = this.f, l = this.n(\cdots)$ の場合，$NewPt(l) = Pt(l) \cap Ag(m)$
- $l = r.f, l = r.n(\cdots)$ の場合，$NewPt(l) = Pt(l) \cap Ag(m) \cap Ag(r)$

ここで $Pt(l)$ は Andersen-style の解析から得られる l の参照可能なオブジェクト集合であり，$NewPt(l)$ は LightSens で得られる l の参照先集合である．$Ag(m)$ はメソッド m が属するオブジェクト集合 RC_m がアクセス可能なオブジェクト集合で $Ag(m) = \bigcup_{o \in RC_m} Ag(o)$ と計算される．$Ag(o)$ はオブジェクト o がアクセス可能なオブジェクト集合 (オブジェクトグラフから得られる)．$Ag(r)$ はローカル変数 r が参照するオブジェクト集合が，アクセス可能なオブジェクトの集合で $Ag(r) = \bigcup_{o \in Pt(r)} Ag(o)$ と計算される．

以上により，オブジェクトのアクセス関係により参照するオブジェクトを限定でき，正確さが向上する．

3.3.3 LightSens の例

LightSens が Andersen-style の解析の正確さを改善する例を挙げる．

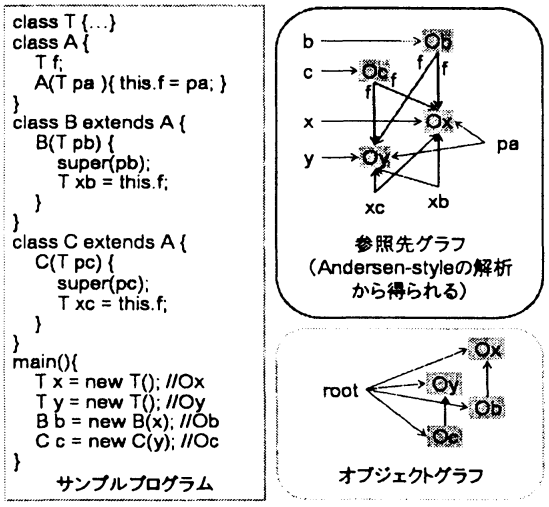

図 1: 参照グラフとオブジェクトグラフの例

図 1 はサンプルプログラムと，そのプログラムを入力として与えた場合の Andersen-style の解析から得られる参照先グラフ，LightSens で生成されるオブジェクトグラフを示している．

プログラムの main メソッドの中では 4 つのオブジェクト Ox，Oy，Ob，Oc が順に生成され，それぞれが変数 x，y，b，c に代入される．Ob，Oc の生成時には，x，y がコンストラクタに渡され，両方ともスー

パークラスAのコンストラクタAに渡る．このとき文脈を考慮しないAndersen-styleの解析では，Aの仮引数paがOxとOyの両方を参照することになり，ObとOcのフィールドfも，OxとOyの両方を参照することになる．そして，そのfが代入されるため，ローカル変数xbとxcはOxとOyの両方を参照することになる．これらの情報が図1の参照先グラフに反映されている．しかし，実際にはxbはOx，xcはOyしか参照しない．

図1のオブジェクトグラフではOx，Oy，Ob，Ocがmainメソッドで生成されるために，rootから辺が作られる．ObからOx，OcからOyへの辺は，Ob，Ocが生成されるとき，Ox，Oyがそれぞれ実引数として渡されることから作られる．

この例で改善されるのはxbとxcの参照先である．xbはOb，xcはOcのみに属していて，Obがアクセス可能なのはOx，OcがOy，であることがオブジェクトグラフから得られるため，インターセクションによって改善される．式で表現すると次のようになる．$NewPt(xb) = Pt(xb) \cap Ag(Ob) = \{Ox, Oy\} \cap \{Ox\} = \{Ox\}$，$NewPl(xc) = Pl(xc) \cap Ag(Oc) = \{Ox, Oy\} \cap \{Oy\} = \{Oy\}$．

3.3.4 LightSensの計算量

プログラムサイズをnとする．するとオブジェクトの数は生成文の数[6]となるため$O(n)$である．

LightSensはAndersen-styleの解析，オブジェクトグラフの生成，インターセクションを順次行うので計算量はそれらのうち最大のものに依存する．

Andersen-styleの解析は前述の通り$O(n^3)$である．

オブジェクトグラフは，最悪の場合，代入文の両辺(2つの変数)が参照し得るオブジェクト($O(n)$)の直積となる辺($O(n^2)$)を追加する場合がある．これをすべての文($O(n)$)について行うために$O(n^3)$となる．

インターセクションも各文について処理を行う．その処理のなかで変数の参照するオブジェクト集合がアクセス可能なオブジェクト集合$Ag(r)$を求める必要がある．この計算は$O(n)$のオブジェクト集合に対して$O(n)$の演算するため$O(n^2)$．これを$O(n)$の文について繰り返すために$O(n^3)$となる．

以上によりAndersen-styleの解析，オブジェクトグラフの生成，インターセクション全てが高々$O(n^3)$で計算可能なために，LightSens全体としても$O(n^3)$の計算量が必要になる．

4 LightSensの拡張

本稿ではLightSensを拡張した手法を提案する．この拡張では計算量のオーダーを増加させず，解析で得られる情報の正確さを向上させることが可能である．本手法ではLightSensで生成される参照先グラフとオブジェクトグラフを利用するため，LightSensを行った後に処理を付け加える．付け加える処理はローカル変数の複製とインターセクションである．実際には複製しつつインターセクションを計算する．

4.1 ローカル変数の複製について

LightSensではローカル変数はプログラム中に書かれた名前で区別される．しかし，それでは複数のメソッド呼び出しによって異なる値が代入される場合，解析結果として「複数の参照先を持つ可能性がある」としか言えない．これが参照先解析における不正確さとなる．そこで文脈依存解析では，さらにローカル変数を文脈によって細分化する事で正確さを向上させる．特にSObjectSensではローカル変数をその属する(メソッドが属する)オブジェクトで区別する．その考えを本拡張にも取り入れる．SObjectSensのような従来の文脈依存解析ではローカル変数を文脈に区別したのち代入される参照値の伝播を計算していたが，本拡張ではローカル変数を区別したのちにインターセクションを行う．その違いが計算量に影響を与える．

4.2 インターセクションについて

LightSensでもインターセクションを行うが，内容は少し異なる．LightSensでは代入文の種類によって処理を分けていたが，今回は文を参照する必要はなく，変数とその変数が属するオブジェクト，そのオブジェクトがアクセス可能なオブジェクト集合が計算できれば良い．

[6]文がループ内に存在している場合でも生成される参照先は1つと考える．

4.3 拡張処理アルゴリズム

図 2 は LightSens に付け加える処理のアルゴリズムである．1 行目の Reachable Methods とはプログラム開始地点から到達可能なメソッドの集合[7]である．2 行目では各メソッドの属するオブジェクト集合 RC_m の数が 1 より大きいかどうかを調べている．1 以下であればローカル変数を複製しても参照先を限定できないので，それ以降の処理は不要になる．

$Pt(v)$ は既に行っている LightSens から得られているローカル変数 v の参照可能なオブジェクト集合であり，$NewPt(v^o)$ はこの拡張手法で新しく得られるオブジェクト集合で，オブジェクト o に属するローカル変数 v の参照先である．

```
for (each Method m ∈ Reachable Methods)
    if (|RC_m| > 1)
        for (each Local Variable v in m)
            for (each o ∈ RC_m)
                NewPt(v^o) = Pt(v) ∩ Ag(O);
```

図 2: 拡張処理アルゴリズム

以上では複数のオブジェクトに属している全ローカル変数について，複製とインターセクションを行っているが，インターセクションにより結果が向上されないものについては参照先を複製する必要はなく，「向上しない」つまり「もとの参照先 (Pt) と同じ」ということを記録しておけばメモリの消費量は抑えられる．

またもとの参照先 (Pt) の数や属するオブジェクト (RC_m) の数が多い場合は，複製とインターセクションにより有用な結果が得られる可能性は低いと考えられる．そのため複製とインターセクションを行う条件に「$Pt(v)$，RC_m がそれぞれ一定数以下」を加えれば，計算量やメモリ消費量を節約できる．

4.4 拡張手法の例

拡張手法が LightSens の正確さを改善する例を挙げる．

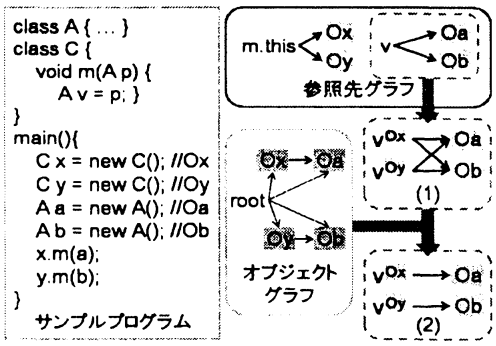

図 3: 拡張手法の改善例

図 3 はサンプルプログラムと，そのプログラムを入力として与えた場合 LightSens から得られる参照先グラフ，オブジェクトグラフ，さらに参照先グラフの一部を複製したグラフ (1)，(1) に対してインターセクションを行い誤った参照先が取り除かれたグラフ (2) を示している．

LightSens から得られる参照先グラフとオブジェクトグラフは 3.3.3 節の例と同様に生成される．参照先グラフ中の $m.this$ はメソッド m の暗黙のパラメータでメソッド m が属するオブジェクトを参照する．

参照先グラフの点線で囲まれた部分は「変数 v が Oa, Ob の両方を参照する可能性がある」ことを示しているが，LightSens ではこれ以上正確な情報を得られない．しかし，この変数 v を Ox に属する v^{Ox} と，Oy に属する v^{Oy} に複製する (図 3 の (1)) ことで，「Ox が Oa のみ，Oy が Ob のみアクセス可能」という情報を持つオブジェクトグラフとインターセクションをとることでより正確な参照先が得られる (図 3 の (2))．

[7] Andersen-style の解析で得られる．

4.5 拡張手法の計算量

今回提案した手法は先述のようにLightSensを行った後にローカル変数を属するオブジェクトごと複製してインターセクションを行う．ここでプログラムサイズを n とする．そうすると解析で扱うオブジェクトの数は生成文の数[8]となるため $O(n)$ である．

インターセクションは $O(n)$ の要素を持つオブジェクト集合に対する集合演算なので1回につき $O(n)$ の計算量が必要．本手法ではインターセクションはローカル変数の数だけ行う．ここでローカル変数の数はもともと $O(n)$ だが，複製することよって $O(n^2)$ に増加する．よって結果的に $O(n) \times O(n^2) = O(n^3)$ の計算量が必要になる．

LightSensは前述の通り $O(n^3)$，追加する処理も $O(n^3)$ なので拡張しても同じ $O(n^3)$ に抑えられる．

5 まとめ

本稿ではJavaプログラム[9]を対象とする参照先解析の新しい手法を提案し，その有用性を明らかにした．新しい手法では，LightSens[Mil07]を拡張し，計算量のオーダーを増加させることなく，正確さを向上させることに成功した．この計算量のオーダーは文脈非依存であるAndersen-styleの解析とも同じなので，計算量のオーダーを増加させずに文脈を考慮した結果が得られる手法と言える．

参考文献

[And94] Lars Ole Andersen. *Program Analysis and Specialization for the C Programming Language.* Phd thesis, DIKU, University of Copenhagen, May 1994.

[IK83] T. Ibaraki and N. Katoh. On-line computation of transitive closure for graphs. *Information Processing Letters*, Vol. 16, pp. 95–97, 1983.

[LH03] O. Lhotak and L. Hendren. Scaling Java points-to analysis using Spark. *Lecture Notes in Computer Science*, pp. 153–169, 2003.

[LH06] O. Lhotak and L. Hendren. Context-sensitive points-to analysis: Is it worth it? *Lecture Notes in Computer Science*, Vol. 3923, p. 47, 2006.

[Mil07] Ana Milanova. Light Context-Sensitive Points-to Analysis for Java. In *the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, June 2007.

[MRR02] Ana Milanova, Atanas Rountev, and Barbara G. Ryder. Parameterized Object Sensitivity for Points-to and Side-E ect Analyses for Java. In *the ACM SIGSOFT International Symposium on Software Testing and Analysis*, July 2002.

[MRR05] Ana Milanova, Atanas Rountev, and Barbara G. Ryder. Parameterized Object Sensitivity for Points-to Analysis for Java. *ACM Transactions on Software Engineering and Methodology*, Vol. 14, No. 1, pp. 1–41, 2005.

[SF09] Manu Sridharan and Stephen J. Fink. The Complexity of Andersen's Analysis in Practice. *Lecture Notes In Computer Science*, Vol. 5673, pp. 205–221, 2009.

[Ste96] B. Steensgaard. Points-to analysis in almost linear time. In *ACM Symposium on Principles of Programming Languages*, pp. 32–41, 1996.

[8] 文がループ内に存在している場合でも生成される参照先は1つと考える．
[9] オブジェクト指向型のプログラムならば，同様に解析可能である．