（情報処理学会誌 道しるべ　第６回，９月号原稿）

コンパイラ・インフラストラクチャ COINS を用いた SSA 最適化 (その 2) [1]

佐々政孝 sassa@is.titech.ac.jp

1 はじめに

静的単一代入形式（Static Single Assignment Form，以下**SSA形式**と略す）は，すべての変数の使用に対して，その値を定義（代入）している場所が 1 箇所しかないように変数の名前替えをした中間表現の形式である．通常の中間表現では変数の使用に対してその値を定義している場所が複数個あり得るのだが，SSA形式では，各変数の定義がプログラム上で１箇所しかなく，変数の使用と定義の関係が明確になる．

この性質を利用することにより，コンパイラにおけるデータフロー解析や最適化を見通しよく，容易に行うことができる．また最適化の実行効率もほとんどの場合に向上する．このためいくつかの最適化コンパイラ[6-ibm:RVM, 5-gcc]がその一部のパスでSSA形式を採用するようになってきている．

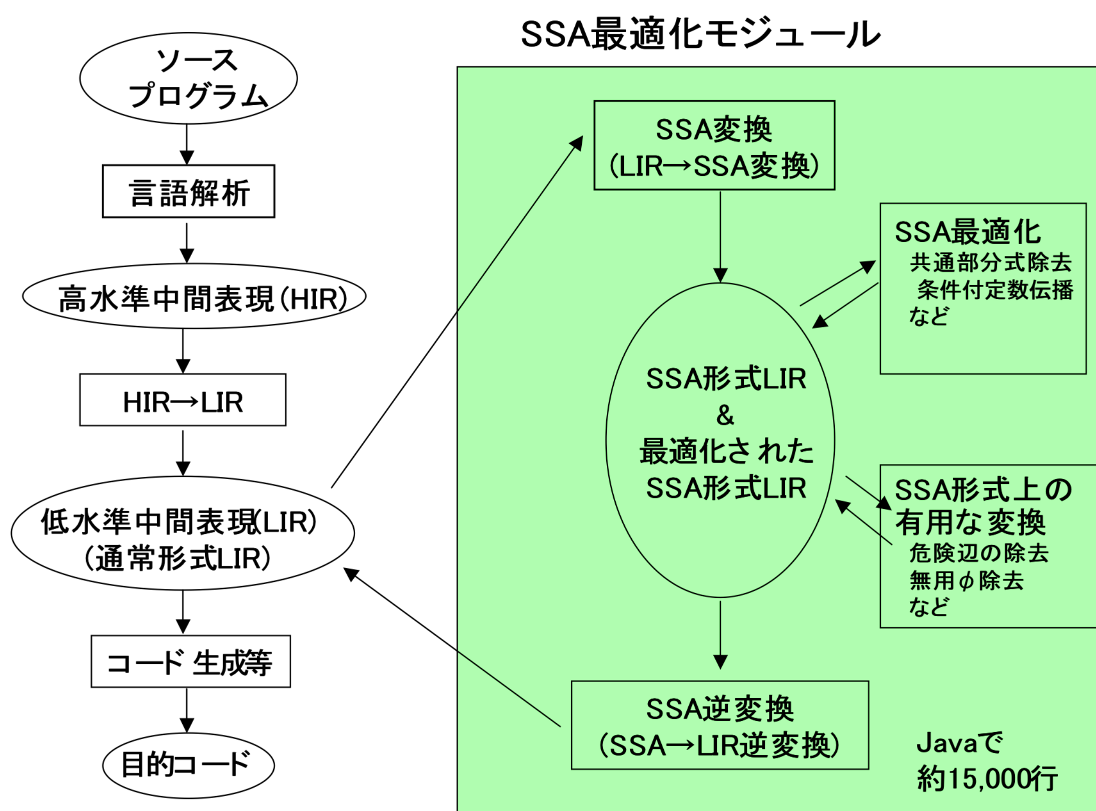


図１　COINS の SSA 最適化モジュール

[1] Copyright © 2006 Masataka Sassa　本稿は草稿です．［校正注．脚注は取ル］

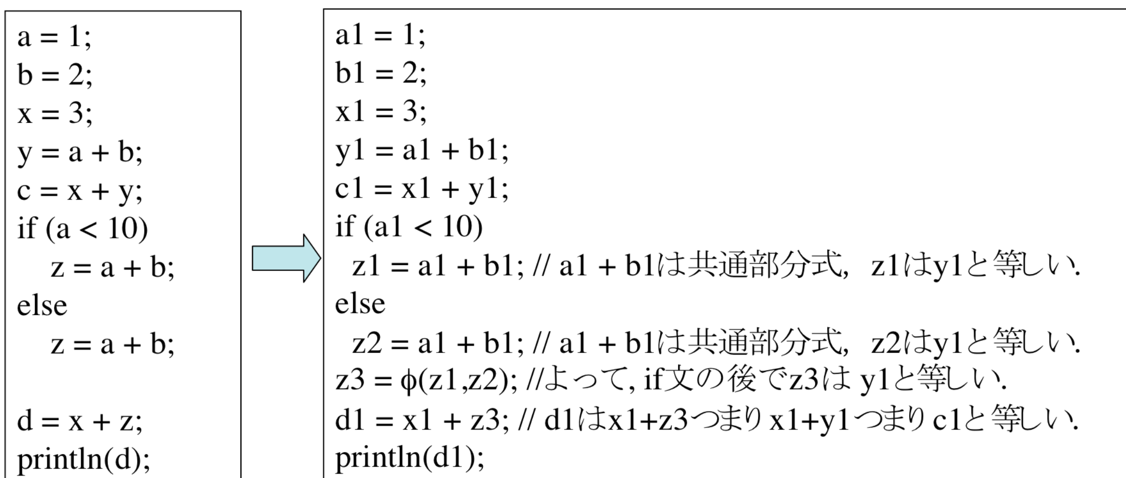
8 月号では，SSA 形式の概要，SSA 形式への変換，SSA 形式から通常形式への逆変換について述べた．今月号では，SSA 形式最適化のいくつかの例を述べる（SSA 形式を用いた最適化を SSA 形式最適化あるいは SSA 最適化と呼ぶ）．COINS での SSA 形式最適化モジュールについては 8 月号の図 8 に示したが，読者の便宜のため，それを図 1 に再掲する［2-coins，3-coinsssa］．今月号で扱うのは，この図 1 の右にある「SSA 最適化モジュール」という箱の中身である．

2 SSA 形式最適化

この節ではSSA 形式最適化の例を，COINS での最適化の実行例をまじえつついくつか挙げる．

2.1 共通部分式除去

2.1.1 共通部分式除去の仕組み



(a) ソースプログラム　　　　(b) SSA形式での解析結果

図 2　共通部分式除去（その 1）

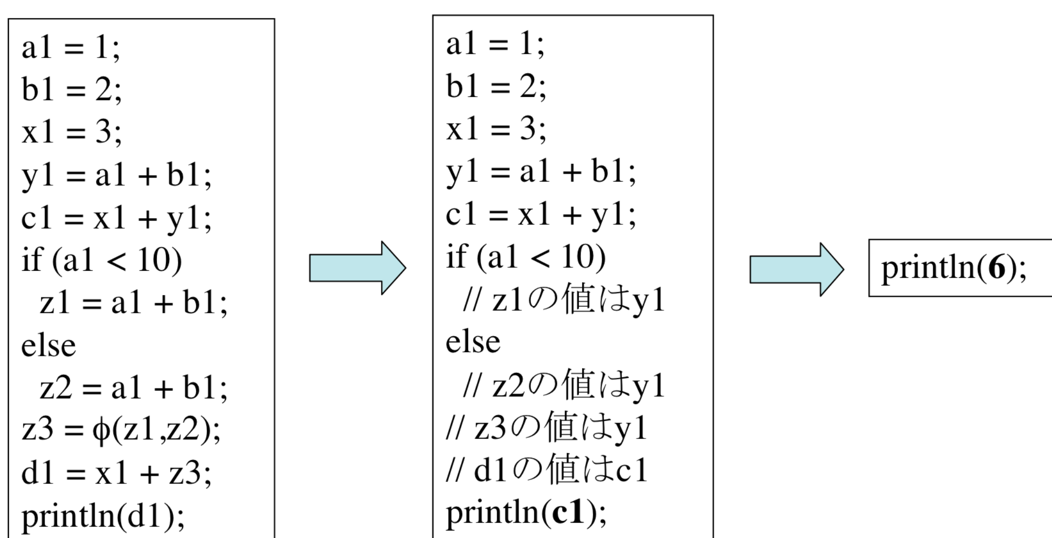
図 2(a)のソースプログラムがあったとする．これを SSA 形式中間表現に変換すると，図 2(b)となる．以下，バージョン名は下添字でなく a1，b1 等で示す．また本来は SSA 形式は中間表現であるが，読みやすさのためソースプログラムの形式で示す．

図 2(b)の上で共通部分式の解析を行うと，同図の「//」以降に書いた解析結果が得られる．たとえば，if 文中の「z1 = a1 + b1」では，右辺の「a1 + b1」が上で計算した「y1 = a1 + b1」の右辺と「字面」が同じであるので，共通部分式であることがわかり，z1 は y1 と等しいことが分かる．従来の通常形式による方法では，字面が等しいだけでは共通部分式を認識できず，その 2 つの式の間でそれらのオペランドの値

が変わらないことなどを解析する必要があるが，SSA 形式は単一代入なので，字面が等しければ値が等しいことがすぐに分かるのが特長である．

同様にして，z2 も y1 と等しいことが分かる．すると，「z3 = φ(z1,z2)」において，z1 も z2 もともに y1 と等しかったので，z3 が y1 と等しいことが分かる．

その次の行の「d1 = x1 + z3」では， z3 が y1 と等しかったので，d1 は「x1 + y1」と等しいことがわかり，これは上で計算した「c1 = x1 + y1」の右辺なので，結局 d1 は c1 と等しいことが分かる．



(c) SSA形式
(図2(b)と同じ)

(d) 共通部分式除去後

(e) 全最適化後

図 3　共通部分式除去（その 2）

これらを用いて共通部分式除去を行うと，図 3(d)が得られる．SSA 形式共通部分式除去のアルゴリズム[3-coinsssa, 8-中田 99]では，代入文の右辺が共通部分式であることが分かると，まず「その左辺の変数の値がその共通部分式を定義した変数にすでに格納されていることを表に登録して」，この文を消去する．たとえば図 3(d)の例では，「z1 = a1 + b1」を見ると，z1 の値は「a1 + b1」を計算した y1 と同じなので，「z1 の値が y1 に格納されていることを表に登録して」，その文「z1 = a1 + b1」を消去する．同様にして，「z2 の値が y1 にある」ことを表に登録して「z2 = a1 + b1」も消去する．以下同様に，「z3 の値が y1 にある」ことを表に登録して「z3 = φ(z1,z2)」も消去する．さらに，「d1 の値が c1 にある」ことを表に登録して「d1 = x1 + z3」も消去する．最終行の d1 は，表を見ると「d1 の値が c1 にある」と書いてあるので，c1 に置き換える．

図 3(d)にさらに定数伝播やコピー伝播，無用命令除去など SSA 形式最適化で行って

いる全最適化（後述の 2.2 節「条件分岐を考慮した定数伝播」や 3 節「COINS における SSA 形式最適化モジュール」参照）をかけると，図 3(e)が得られ，これはただの 1 行となる．

2.1.2　COINS を用いた共通部分式除去の例

2.1.1 節の例を COINS で実行してみよう．ソースプログラムは次である．なお，以下の例は C 言語を用いているが，連載の第 1 回ー第 2 回で扱った C0 言語を用いる場合でも同様のことが行える．詳しくは［11-デモの仕方］を参照されたい．
［例 1］共通部分式除去の例のソースプログラム（ファイルは csetest.c）．これは図 2(a)に対応する．
［校正注．タイプフェースで写植のこと．スペースも含め等幅．以下同じ．］

```
void println(int v);

int main ()
{
  int a;
  int b;
  int c;
  int d;
  int x;
  int y;
  int z;

  a = 1;
  b = 2;
  x = 3;
  y = a + b;
  c = x + y;
  if (a < 10)
    z = a + b;
  else
    z = a + b;
  d = x + z;
  println(d);
}
```

これを，次のコマンドでコンパイルする．8 月号でも述べたとおり，以下はシェルとして tcsh を仮定している．他のシェルを利用する場合は対応する同等のコマンドを用いてほしい．

```
% source aliases　（言語 C0 を用いる場合は aliasesc0 とする）
% cccse csetest.c
```

このコマンドを実行すると，最適化の途中の SSA 形式中間表現を C 言語風に表現したファイルがいくつか生成される．

［例 2］共通部分式除去の例 1 を SSA 形式中間表現にし，それを C 言語風に出力したもの．紙幅の関係で一部折り返してある．~~（？校正刷りを見て考えます）~~（ファイルは csetest-main-2.lir2c）．これは図 3(c)に対応する．

```
void main(void) {

（宣言部分省略，以下同じ）

_L1:
a_1__1 = ((int)( 1));
b_2__1 = ((int)( 2));
x_5__1 = ((int)( 3));
y_6__1 = ((int)(((int)(a_1__1 + b_2__1))));
c_3__1 = ((int)(((int)(x_5__1 + y_6__1))));
a_1__0 = ((int)( 0));
b_2__0 = ((int)( 0));
x_5__0 = ((int)( 0));
y_6__0 = ((int)( 0));
c_3__0 = ((int)( 0));
z_7__0 = ((int)( 0));
d_4__0 = ((int)( 0));
if ((a_1__1 <  10)) { goto _L3;}
else { goto _L4;}

_L3:
z_7__2 = ((int)(((int)(a_1__1 + b_2__1))));
goto _L5;

_L4:
z_7__1 = ((int)(((int)(a_1__1 + b_2__1))));
goto _L5;

_L5:
z_7__3 = phi(z_7__2:_L3, z_7__1:_L4);
d_4__1 = ((int)(((int)(x_5__1 + z_7__3))));
println(d_4__1);
goto _L6;

_L6:
```

```
return;
}
```

ここで，「a_1__0 = ((int)(0));」のように後で使用されない変数に 0（⊥(BOTTOM)の代用）が代入されているのは，8 月号で述べたのと同じ理由による．

また，SSA 形式の変数のバージョン名（添字）が図 3 と異なっているが，これは，実際の処理では制御フローグラフをある順番で辿りつつバージョン名をつけているためなので，気にしなくてよい．

中間表現は制御フローグラフなので，8 月号と同様にラベルや goto 文が出てくる．

上の例 2 に，共通部分式除去の最適化を施すと，その結果の SSA 形式中間表現は次のようになる．

［例 3］例 2 に共通部分式除去の最適化を施した結果の SSA 形式中間表現を，C 言語風に表現したもの．これは，図 3(d)に対応する．（ファイル csetest-main-3.lir2c）

```
void main(void) {

（宣言部分省略）

_L1:
a_1__1 = ((int)( 1));
b_2__1 = ((int)( 2));
x_5__1 = ((int)( 3));
y_6__1 = ((int)(((int)(a_1__1 + b_2__1))));
c_3__1 = ((int)(((int)(x_5__1 + y_6__1))));
a_1__0 = ((int)( 0));
b_2__0 = ((int)( 0));
x_5__0 = ((int)( 0));
y_6__0 = ((int)( 0));
c_3__0 = ((int)( 0));
z_7__0 = ((int)( 0));
d_4__0 = ((int)( 0));
if ((a_1__1 <  10)) { goto _L3;}
else { goto _L4;}

_L3:
goto _L5;

_L4:
goto _L5;
```

```
_L5:
println(c_3__1);
goto _L6;

_L6:
return;
}
```

SSA 形式で上記の共通部分式除去を行うとともに，SSA 形式上の標準的な最適化をすべて適用した結果は次のようになる．これはコマンド

```
% cccseallopt csetest.c
```

による．

［例 4］例 2 に，SSA 形式上の標準的な最適化をすべて適用した結果の SSA 形式中間表現を，C 言語風に表現したもの（ファイル csetest-main-4.lir2c）．これは図 3(e)に対応する．

```
void main(void) {

（宣言部分省略）

_L1:
println( 6);
goto _L6;

_L6:
return;
}
```

一見，無駄な飛び越し命令が残っているが，「goto _L6;」はコード生成の前に除去される．

2．2　条件分岐を考慮した定数伝播

2．2．１　条件分岐を考慮した定数伝播の仕組み

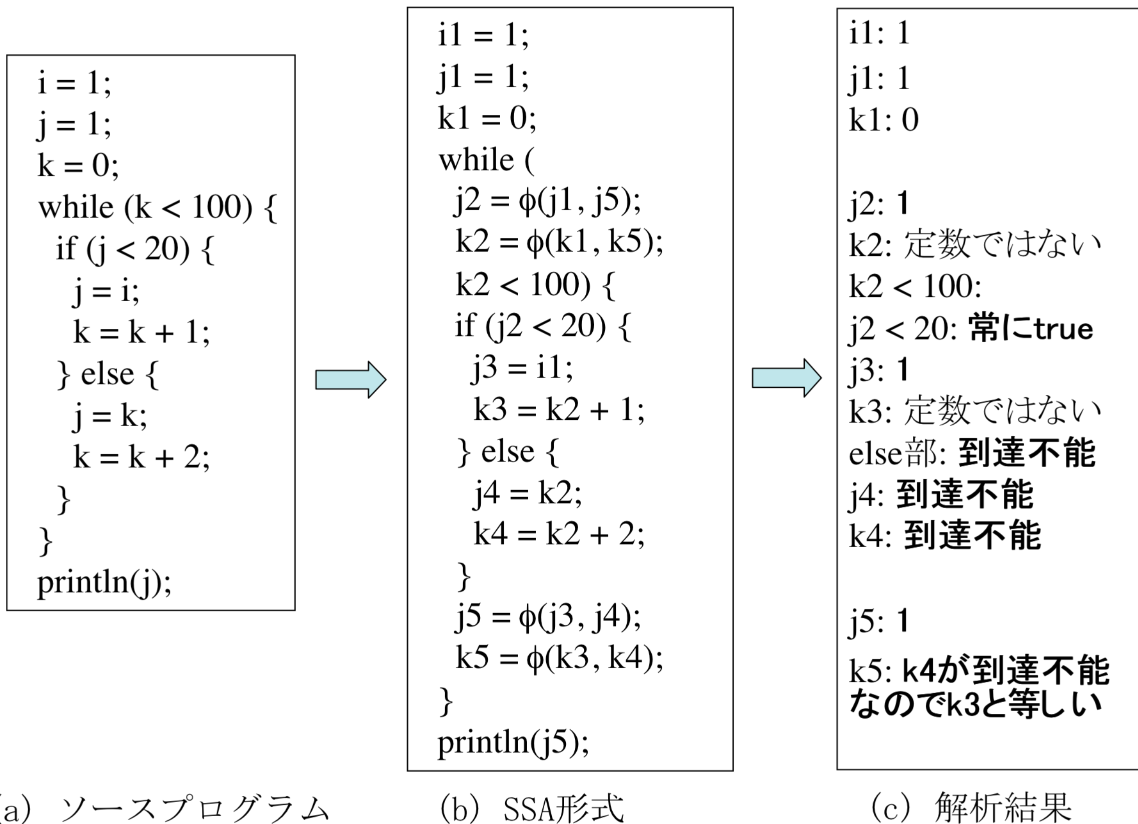


図 4　条件分岐を考慮した定数伝播（その 1）

条件分岐を考慮した定数伝播 [10-Wegman91, 1-Appel02, 8-中田 99, 3-coinsssa] の例として，図 4(a)のソースプログラムを挙げる．これは[1-Appel02]の図 19.4 にある例である．

これを SSA 形式に変換すると，図 4(b)が得られる．図 4(b)の SSA 形式プログラムに対して，条件分岐を考慮した定数伝播の解析を行うと，図 4(c)の情報が得られる．この解析の特徴は，定数である可能性があればさしあたり定数だと「楽観的」にみなしておき，その後，定数でないことがはっきりするまではそのまま定数だとみなすことである．このようなアルゴリズムは，否定されない限りは都合の良いように考えておくので，「**楽観的なアルゴリズム**」と言われる．

たとえば，図 4(b)の「j2=φ(j1, j5)」を初めて調べるときは，j1 の値が 1 なので，さしあたり j2 の値も 1 だとみなす．「j3=i1」を調べるときも，i1 の値が 1 なので，j3 の値もさしあたり 1 だとみなす．同様に，「j5=φ(j3, j4)」を調べるときも，j3 が 1 なので，j5 も 1 だとみなす．ループがあるので，その後ふたたび「j2=φ(j1, j5)」を調べることになるが，運良く j1 も j5 も 1 なので，最初の楽観的予想どおり，j2 は 1 とみなしておいて良かった，となる．j2 が 1 なので，「j2<20」はいつも true だと分かるので，else 部は到達不能だと分かる．このような解析を，みなした値が変化し

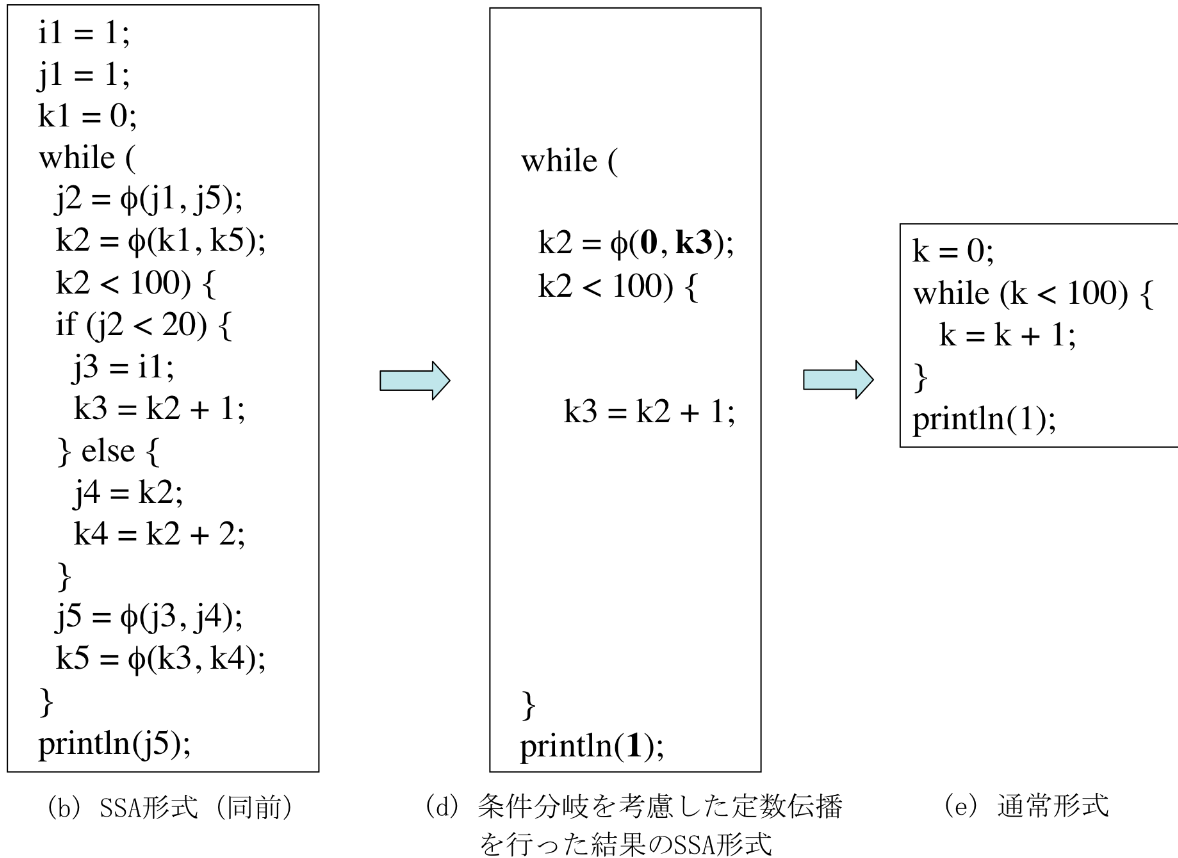
なくなるまで繰り返す．結果として，

・j2, j3, j5 は常に 1

・「j2<20」は常に true であるので，else 部には到達できない

・k4 への代入文が到達不能なので，k5 は k3 と等しい

であることが分かる．



(b) SSA形式（同前）　(d) 条件分岐を考慮した定数伝播を行った結果のSSA形式　(e) 通常形式

図 5　条件分岐を考慮した定数伝播（その 2）

このような解析も，SSA 形式で値が単一代入だからこそやりやすい．

図 4(c)の解析結果を使い，図 5(b)（図 4(b)と同じ）の SSA 形式を最適化すると図 5(d)が得られる．

COINS での条件分岐を考慮した定数伝播の最適化は他の最適化も同時に行っているので，（i）定数であることが分かった変数は，その値を記憶して，その変数への代入文を消去し，その変数の使用は記憶してあった定数で置き換える，（ii）ある変数 x の値が別の変数 y の値と等しいことが分かったら，x への代入文は消去し，以後の x の使用を y で置き換える，（iii）true あるいは false であることが分かった条件式の評価は省略する，（iv）到達不能なコードを削除する，ことを行う．

図 5(b)の例で説明すると，図 4(c)の解析の結果，i1，j1，k1，j2，j3，j5 は定数であることがわかったので，図 5(d)ではそれらの変数への代入文は消去する．また，k1 の使用は 0 で置き換え，j5 の使用は 1 で置き換える．k5 への代入文は消去し，k5

の使用は k3 で置き換える．if 文の条件式は常に true なので else 部は削除する．詳しいアルゴリズムは[3-coinsssa]を見られたい．

SSA 形式で行う最適化はここまでであるが，これを通常形式に戻すと，図 5(e)が得られる．

2. 2. 2 COINS を用いた条件分岐を考慮した定数伝播の例

2.2.1 節の例を COINS で実行してみよう．ソースプログラムは次のとおりである．

［例 5］ファイル appel.c

```
void println(int v);

int main ()
{
  int i;
  int j;
  int k;

  i = 1;
  j = 1;
  k = 0;

  while (k < 100) {
    if (j < 20) {
      j = i;
      k = k + 1;
    } else {
      j = k;
      k = k + 2;
    }
  }

  println(j);
}
```

以下，COINS における実行例は，［例 1］から［例 4］と同様であるので，抜粋して示す．興味のある読者は，［11-デモの仕方］を参照されたい．

［例 5］のソースプログラムを SSA 形式中間表現に変換し，それに条件分岐を考慮した定数伝播を施す．

```
% source aliases　（言語 C0 を用いる場合は aliasesc0 とする）
% cccstpnochangeloop appel.c
```

この結果，途中結果をC言語風に表現したファイルがいくつか生成される.

［例 6］例 5 を SSA 形式に変換した後，条件分岐を考慮した定数伝播の最適化を施す.以下はそれを C 言語風に表現したものである（ファイル appel-main-3.lir2c）.これは図5(d)に対応する.

```
void main(void) {

（宣言と初期化は略）

_L1:
goto _L3;

_L3:
k_3__2 = phi( 0:_L1, k_3__4:_L5);
if ((k_3__2 <  100)) { goto _L4;}
else { goto _L8;}

_L4:
goto _L5;

_L5:
k_3__4 = ((int)(((int)(k_3__2 +  1))));
goto _L3;

_L8:
println( 1);
goto _L9;

_L9:
return;
}
```

中間表現は制御フローグラフなので，前述のとおり while 文ではなく if 文と goto 文で表される．また，SSA 形式の変数のバージョン名が図 5(d)と異なるが，これも前述のとおり気にしなくてよい.

3 COINSにおけるSSA形式最適化モジュール

紙面の関係で省いたが，COINS の SSA 形式最適化部には，次のような SSA 形式最適化と変換を行うものが備えられている．これらは主要な SSA 形式最適化をほぼ網羅している．詳しくは[3-coinsssa]を参照されたい．後述のお勧めの最適化の説明のため

に，括弧内に最適化の略称を記した．

(SSA 形式最適化)

- 共通部分式除去(cse)：前述
- 条件分岐を考慮した定数伝播(cstp)：前述
- 質問伝播大域値番号付けと部分冗長性除去(preqp)：質問伝播という方法を用いてφ関数をまたいだ部分冗長性を判別し除去するとともに値番号付けを行う．これは貢献者の滝本宗宏氏が新たに開発したものである．
- 演算の強さの軽減と判定の置き換え(osr)：ループ内の帰納変数について，新たな帰納変数を導入し，乗算を加算で置き換え，新たな帰納変数を用いて終了判定を行う．
- コピー伝播(cpyp)：コピー文があったら，以後その左辺の変数の使用を右辺の変数で置き換える．
- ループ不変式移動(hli)：ループ内で値が変わらない式の計算をループの前に移動する．
- 無用命令除去(dce)：以後使われない変数の計算や実行されないコードを除去する．
- 大域的再結合(gra)：式の分配規則を用いて式を分け，ループ不変式を増やす．

(SSA 形式最適化に有用な変換や副アルゴリズム)

- 危険辺の除去(esplt)：危険辺（複数の後続ブロックを持つ基本ブロックから複数の先行ブロックを持つ基本ブロックへ向かう辺）に空の基本ブロックを挿入する．
- 無用φ除去(rpe)：無用なφ関数を除去する．
- 空ブロック除去(ebe)：中身が空の基本ブロックを除去する．
- 基本ブロックの連結(cbb)：必ず連続して実行される複数の基本ブロックを一つにまとめる．
- SSA グラフの作成(ssag)：SSA 形式をグラフ表現したものを作成する．
- 式の分割(divex)：右辺に複数の演算子を持つ式を右辺に 1 つの演算子だけを持つ式の列に分ける．

一般に，最適化の効果的な組合せおよび適用順序（以下たんに「組合せ」と呼ぶ）を一意的に決めることは難しい．COINS では，一般ユーザの便宜のためにお勧めの最適化の組合せを決めており，「-O3」などのコンパイル時オプションの指定により対応する最適化を適用できる．例えば「-O3」のオプションが指定されると，SSA 形式最適化では次の「/」で区切られた最適化がこの順に適用される．

divex/cse/cstp/hli/osr/hli/cstp/cpyp/preqp/cstp/rpe/dce

同じ最適化が 2 回以上適用されることにも注目されたい．

しかし，この最適化の組合せの効果は対象とするプログラムによって異なる．文献

[9-佐々06, 3-coinsssa]では, SSA 形式最適化の組合せを変えたときに, 種々のベンチマークでその効果がどのように変わるかについて述べている. 文献[9-佐々06]では, SPEC CPU2000 の C および F77 の 14 個のベンチマークについて, SSA 形式最適化の 4 つの組合せで実験を行った. その結果, 平均して良い結果を与えるような SSA 形式最適化の組合せを一つ決めたとしても, 4 個, 5 個あるいは 6 個のベンチマークについては他の SSA 形式最適化の組合せを採用した方が目的コードの実行時間が数パーセント程度短くなる, という逆転現象が起こることが示された. ただ, COINS の SSA 形式最適化はその後も改良が行われているので, 最新版ではこれほどの逆転は起こらないと予想される.

これらを考慮して, COINS では, ユーザがソースプログラムに応じて, オプションでの指定により, それぞれの最適化を任意の順序で任意の回数適用することができるようになっている. これにより, いろいろな最適化の適用順序を変えて, 効果を試してみることができる.

なお, 8 月号でも少し触れたように, コンパイラの最適化には様々なものがあり, SSA 形式最適化は万能ではない. たとえば, 配列の扱いやポインタ解析による別名の処理などは SSA 形式最適化の技法がまだ確立されておらず, 今後の研究が待たれる分野である. また, SSA 形式にするよりはソースプログラムに近い中間コード上で行ったほうが良いループ展開などの最適化や, 命令スケジューリングのようにコード生成部分で行う最適化もある.

4　おわりに

以上, 2 回にわたって SSA 形式の概要, SSA 形式への変換, 逆変換, SSA 形式での最適化の例を紹介した. もし興味を持っていただけたら幸いである.

次号では, HIR 中間表現での最適化、ループ並列化、SMP 並列化などについての解説が予定されている.

参考文献

［校正注. 文献番号は最終的には数にします］

[1-Appel02] Appel, A.: Modern Compiler Implementation in Java, second ed., Cambridge University Press, 2002.

[2-coins] 並列化コンパイラ向け共通インフラストラクチャ COINS : http://www.coins-project.org/

[3-coinsssa] 静的単一代入形式に基づく最適化に関する研究, 研究成果 (成果報告会) : http://www.is.titech.ac.jp/~sassa/coins-www-ssa/japanese/index.htmlより

[4-Cooper03] Cooper, K. and Torczon, L.: Engineering a Compiler, Morgan Kauffmann, 2003.

[5-gcc] GCC: http://gcc.gnu.org/
[6-ibm:RVM] IBM: Jikes Research Virtual Machine.
http://jikesrvm.sourceforge.net/
[7-伊藤05] 伊藤陽，小濱真樹，佐々政孝：静的単一代入形式からの逆変換アルゴリズムの比較と評価，情報処理学会論文誌：プログラミング，Vol. 46，No. SIG 14 (PRO 27)， pp. 30-42 (Oct. 2005).
[8-中田99] 中田育男：コンパイラの構成と最適化，朝倉書店，1999.
[9-佐々06] 佐々政孝，福岡岳穂，滝本宗宏：コンパイラ・インフラストラクチャにおける静的単一代入形式最適化部の実現，情報処理学会論文誌：プログラミング，Vol. 47，No. SIG 2 (PRO 28)，pp. 30-43 (Feb. 2006).
[10-Wegman 91] Wegman, M.N., and Zadeck, F.K.: Constant Propagation with Conditional Branches, ACM Trans. Prog. Lang. Syst., Vol. 13, No. 2, pp. 181-210, 1991.
[11-デモの仕方] http://www.is.titech.ac.jp/~sassa/coins-www-ssa/japanese/michishirube-ipsj-ssa/. 辿れない場合は，http://www.coins-project.org/ より「静的単一代入形式最適化部」を辿り，さらに，「21 世紀のコンパイラ道しるべ - COINS をベースとして　情報処理学会誌の連載 - SSA 最適化部」を辿る.