「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1）」

佐々政孝（東京工業大学）
sassa@is.titech.ac.jp

はじめに

コンパイラでは，機械語の目的コードを生成するだけでなく，実行させたときにその目的コードが効率良く実行できるように，さまざまな変換を行う．これを「最適化」という．たとえば，連載第1回の「概要」のところでも紹介されているように，ループの中で実行しなくてもよい命令はループの外に出す，同じ計算は省略する，などの変換を行って目的コードの効率を向上させる手法がよく知られている．最適化は，現在のコンパイラで力が注がれている重要な技術の1つである．

最適化の方法としては，従来はデータの流れの解析と呼ばれる方法が使われていたが，最近は静的単一代入形式というものを用いた最適化の方法が注目を浴びている．

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箇所しかない．この性質を利用することにより，いろいろな最適化が見通しよく，容易にできるようになる．実際の例はSSA形式の章や9月号で述べる．SSA形式を利用した最適化を静的単一代入形式最適化（SSA最適化）という．SSA最適化では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),7)]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．

8月号と9月号では，SSA形式，SSA形式への変換と逆変換，SSA形式上での最適化，について，コンパイラ・インフラストラクチャCOINSでの例を挙げながら述べる．

用語の定義がされているところを太字で記す．

SSA形式

●通常形式とSSA形式

SSA形式とは，プログラム上のすべての変数の使用に対して，その使用に対する定義が1箇所しかないように表現した中間表現形式である．SSA形式では，変数の定義が字面上，つまりプログラムのテキスト上で唯一になる．この形式は静的に（つまり字面上で）単一代入なので，**静的単一代入形式**と呼ばれる．一方，SSA形式でないふつうの形式を**通常形式**と呼ぶことにする．

SSA形式のポイントは2点ある．1つ目は定義される変数に唯一の名前をつけること，2つ目はφ関数と呼ばれるものである．

1つ目のポイントは，すべての変数の使用に対して，その値を定義（代入や読み込みなど）している場所が1箇所しかないようにするために変数に唯一の名前をつけることである．たとえば，**図-1 (a)**のような通常形式のプログラムがあったとする．

図-1(a)に対するSSA形式は**図-1 (b)**のようになる（SSA形式は本来は中間表現形式であるが，分かりやすさのため，以後ソースプログラムの形式で表す）．代入があるごとに代入の左辺に現れる変数に新しく唯一な変数名を付け，a_1，a_2のように区別する．唯一な変数名をつける際は，慣例として，バージョン（version）と呼ばれる1，2などの添字をつけることが多いが，a_1，a_2とかa1，a2などとしてもよい．代入文の右辺で変数を使用する場合は，その変数がどの代入文で定義されたものである

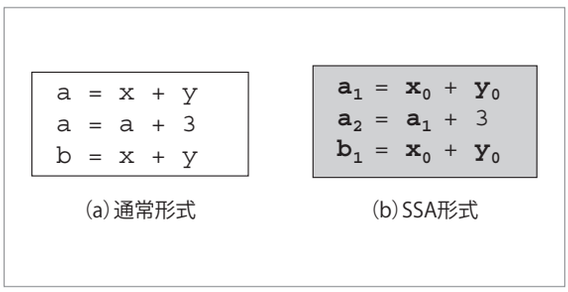

図 -1　SSA 形式での変数名の付け方

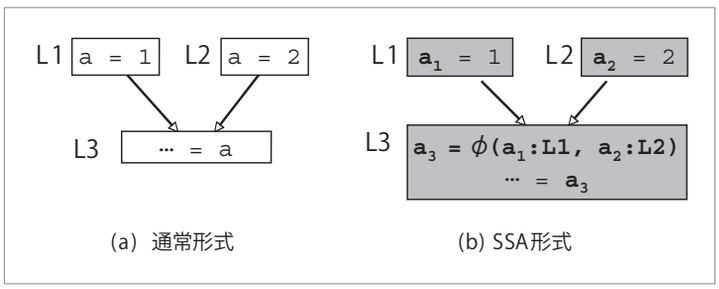

図 -2　SSA 形式での φ 関数

かを探して，対応する代入文の左辺の（唯一の）変数名を記す．たとえば，図 -1 (b) の 2 行目の右辺で使用している a は図 -1 (a) の 1 行目の代入文で定義した a なので，a_1 とする．

なお，SSA 形式にはいろいろな制限があるが，詳しくは文献 1）などを参照されたい．

SSA 形式での 2 つ目のポイントは，**φ 関数**（ファイかんすう，phi function) と呼ばれるものである．これは，制御の合流点に，通常形式で同じ変数であった変数の異なるバージョンが到達するときに用いられる．例を**図 -2** に示す．φ 関数を用いる理由は，図 -2 (a) のプログラムを SSA 形式に変換しようとすると，基本ブロック[☆1] L3 で使用している a に対応する定義を一意に決めることができないからである．そこで，SSA 形式では図 -2 (b) のような φ 関数を導入する．

「φ (a_1:L1, a_2:L2)」は，基本ブロック L1 から来たときは a_1 の値を返し，基本ブロック L2 から来たときは a_2 の値を返す（仮想的な）関数を表す．これにより，図 -2 (b) の L3 の最後の行の a_3 が，φ 関数で定義された唯一の a_3 を参照するようにできる．

なお，φ 関数の記述の煩雑さを避けるため，「a_3 = φ (a_1:L1, a_2:L2)」をたんに「a_3 = φ (a_1, a_2)」と略記することも多い．この記法は，φ の第 1 オペランドが図での左上から来ること，第 2 オペランドが図での右上から来ること，を仮定している．

SSA 形式について，1 つ注意を述べる．SSA 形式での各変数は，字面上あるいはプログラムのテキスト上で定義が唯一であるが，「値」が唯一であるわけではない．たとえば図 -1 がループの中にある場合を考えると，a_1, a_2, b_1 などの「値」はループを回るごとに変わり得る．前述のように，静的単一代入形式の「静的」とは，「字面上で」という意味で，動的（実行時）に値が唯一であるわけではない．図 -2 (b) の例でも，a_3 の「値」が実行時に唯一になるわけではないことは明らかであろう．

[☆1] 基本ブロックとは，途中に飛び越しがなくプログラムに書かれた順に連続して実行される文の列のこと．詳しくは教科書 [1), 5), 9)] を参照されたい．

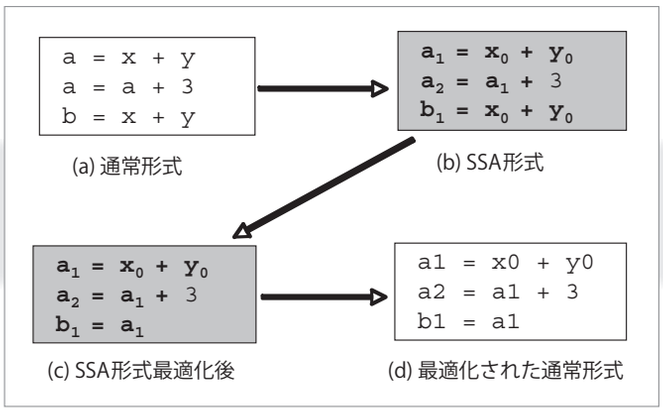

図 -3　SSA 形式での最適化（共通部分式除去）

● SSA 形式の利点

SSA 形式を用いると，プログラムの各変数の使用に対応する定義が 1 箇所だけになるので，変数の使用と定義の関係が明確になり，最適化の実現が容易となる．また最適化の実行効率が向上する．その例を**図 -3** に示す．

図 -3 (a) の通常形式から (b) の SSA 形式への変換は前述の通りである．図 -3 (b) を見ると，3 行目の「$x_0 + y_0$」が 1 行目の右辺と同じであることが分かる．つまり，この 2 つは共通部分式である．そこで，3 行目の「$x_0 + y_0$」を 1 行目の左辺の「a_1」で置き換えると，図 -3 (c) が得られ，共通部分式除去がなされる．これを後述の SSA 逆変換により通常形式に戻すと，図 -3 (d) が得られる．図 -3 (d) は，変数名は変わっているが図 -3 (a) を通常形式で最適化したものにほぼ相当する．

参考として，通常形式での共通部分式除去の最適化の例を**図 -4** に示す．SSA 形式を使わずに，図 -4 (a) の通常形式のままで共通部分式除去を行おうとすると，図 -4 (a) の 1 行目で代入された a の値が 2 行目で上書きされているので，簡単な処理では済まなくなる．ふつうは，図 -4 (a) の 1 ～ 3 行目の間で x，y の値が書き換えられないことを確認してから，一時変数 t を導入し，図 -4 (b) のような最適化がなされる．このように通常形式でも同様な最適化を行うことはできるが，SSA 形式で行った方が最適化の処理が容易となり，処理誤りも少なく，またその結果，最適化処理の効率も一般に向上する．

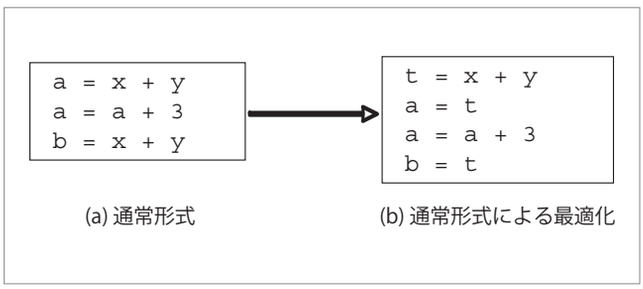

図 -4　通常形式による最適化（共通部分式除去）

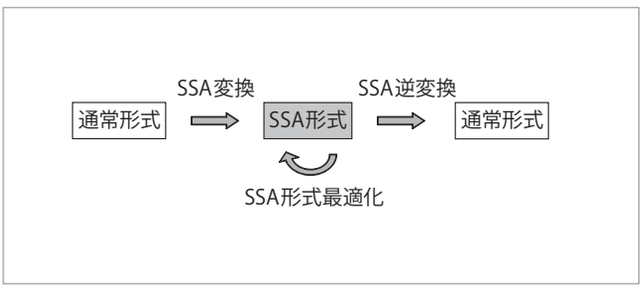

図 -5　SSA 形式を用いた最適化の流れ

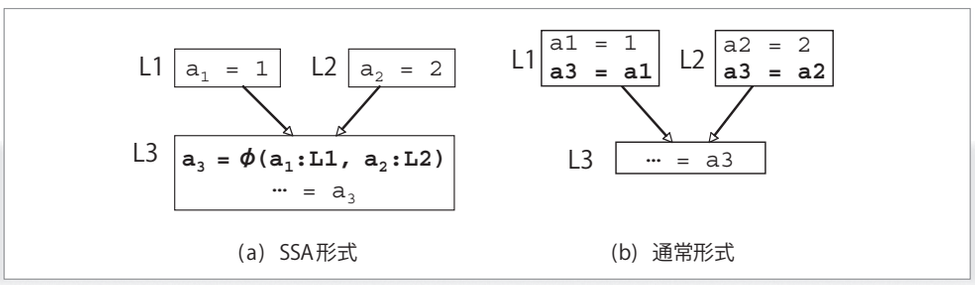

図 -6　Briggs 法による SSA 逆変換

ただし，SSA 形式最適化には不得手な分野もある．たとえば，配列の扱いやポインタによる別名の処理，など SSA 形式最適化の技法がまだ確立されていない分野もある．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よいループ展開などの最適化もある．

以上の例で示したように，SSA 形式はふつう**図 -5** のような流れで利用される．

SSA変換とSSA逆変換

●SSA変換

SSA 変換とは，通常形式から SSA 形式に変換することである．たとえば，図 -1 (a) を SSA 変換すると図 -1 (b) になり，図 -2 (a) を SSA 変換すると図 -2 (b) になる．

図 -1 や図 -2 は簡単な例であったが，プログラムの流れ☆2 が複雑になると，どこにφ関数を挿入すればよいか，とか変数にどのようにバージョン番号（添字）をつければよいか，などのアルゴリズムが必要となる．φ関数の挿入には，支配辺境（dominance frontier）というものを求める計算をすることになる．詳しくは，文献 1)，4)，5)，9）などを参照されたい．

●SSA逆変換

SSA 逆変換とは，SSA 形式から通常形式に戻す変換のことである．一般に，目的機械にはφ関数に相当する命令はないので，SSA 形式から直接コードを生成することはできない．そこで，SSA 形式で最適化を行ったあとには，SSA 逆変換が必要になる．

主な SSA 逆変換法には，Briggs らの方法と Sreedhar らの方法がある．

〈Briggsらによる SSA 逆変換の方法〉

Briggs らの方法（以下 Briggs 法）[2] による SSA 逆変換の例を**図 -6** に示す．

Briggs 法では，φ関数のあった基本ブロックの先行ブロックに，φ関数に対応するコピー文を挿入し，φ関数を消去する．この例では，L3 の

$a_3 = \phi\ (a_1{:}L1, a_2{:}L2)$

とは，制御が L1 から来たときは a_3 の値は a_1，L2 から来たときは a_3 の値は a_2，という意味だったので，それに相当する文を図 -6 (b) のそれぞれの先行ブロックに挿入している．

図 -6 (b) を見ると，コピー文が多くて無駄が多いように思えるが，Briggs らは，その後のレジスタ割当てフェーズで合併（coalescing）[1), 5), 9)] をすることで，これらのコピー文は合併され，図 -2 (a) と同じ結果になるので，かまわないと主張している．これは必ずしも正しくないのであるが，詳しい解析は文献 8）を見られたい．

また，「φ関数のあった基本ブロックの先行ブロックにコピー文を挿入する」という単純な方法では，SSA 形式に変換した後に最適化を施した結果の SSA 形式を正しく処理できない例がいくつもあることが知られている．Briggs 法はこれらの問題も解決したアルゴリズムである

☆2 プログラムの流れ，とは正確には，制御フローグラフのこと．制御フローグラフとは，プログラムを，基本ブロックを節，飛び越し文を有向辺，として表したグラフである．詳しくは本連載の 6 月号の図 -6 の前後の説明や教科書 [1), 5), 9)] を参照されたい．

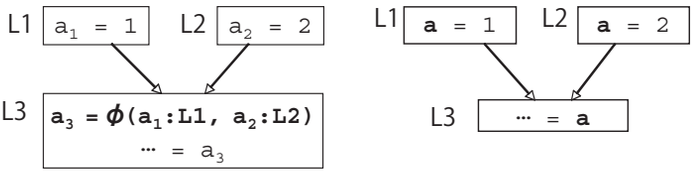

図 -7 Sreedhar 法による SSA 逆変換

が，詳しくは文献 2），4），8）などを参照されたい．

〈Sreedhar らによる SSA 逆変換の方法〉

一方，Sreedhar らの方法（以下 Sreedhar 法）[10] による SSA 逆変換の例を**図 -7** に挙げる．

基本的な手法は，φ関数とその左辺にある変数をすべて単一の変数に置き換え，φ関数を除去するものである．この例では，a_3 と a_1 と a_2 をすべて a に置き換え，φ関数を除去している．図 -7 (b) の結果は，図 -2 (a) と同じである．このように，SSA 変換直後の SSA 形式を Sreedhar 法により SSA 逆変換すると，もとの通常形式とまったく同じものが得られる．Sreedhar 法は，Briggs 法におけるような合併を行わずとも，無駄の少ない通常形式が得られることが利点である．

さて，図 -7 の例だけを見ると，Sreedhar 法は簡単なように見えるが，実は，φ関数とその左辺にある変数をすべて単一の変数に置き換えることができるためには，ある条件を満たしていないといけない．それは，「φ関数とその左辺に現れる変数の生存区間☆3 に重なりがない（干渉がない）」という条件である．一般に，SSA 形式に変換したあとに最適化を施すと，この条件が満たされなくなることが多い．そこで，この条件が満たされないときは，φ関数を書き換え，新たなコピー文を挿入する，という操作が必要になる．詳細は，文献 4），8），10）を見られたい．

COINSにおけるSSA最適化モジュール

連載第 1 回の再掲になるが，COINS コンパイラ・インフラストラクチャの全体構成は**図 -8** のようになっている．

このうち，COINS の SSA 最適化モジュール（以下 SSA 部とも呼ぶ）の部分を少し詳しく表したものを**図 -9** に示す．SSA 部は，低水準中間表現 LIR を受け取り，LIR レベルの SSA 形式に変換する．LIR レベルの SSA 形式に SSA 最適化を施した後，最適化された SSA 形式を SSA 逆変換して再び通常形式の LIR に戻す．その LIR からコード生成等のモジュールにより目的コードを生成する．

☆3 生存区間とは，変数が定義されてから最後に使用されるまでの区間のこと．詳しくは教科書 [1), 5), 9)] を参照されたい．

COINSを用いたSSA変換とSSA逆変換の実行例

SSA 最適化の例については 9 月号で述べることとし，この章では，COINS における SSA 変換と逆変換の実例を示す．

●COINSを用いたSSA変換の例

簡単な例を用いて SSA 変換の例を示す．ここで取り上げるプログラミング言語は C 言語である．COINS のサイトからダウンロードした C 言語のコンパイラ（以下 COINS の C コンパイラと呼ぶ）[3] を用いて実験できる．

なお，ここでの例は，連載の第 1 回～第 2 回で扱った C0 言語を利用する場合にも対応できるようにしてあるので，C0 言語であっても，同様な結果が得られる．詳しくは文献 11）を参照されたい．

［例 1］入力した数の絶対値を出力するプログラム
（ファイル ssatransback.c）

```
void println(int v);
int read();
int main ()
{
  int a, b;
  a = read();
  if (a >= 0) {
      b = a;
  } else {
      b = -a;
  }
  println(b);
}
```

これを COINS の C コンパイラを用いてコンパイルし，

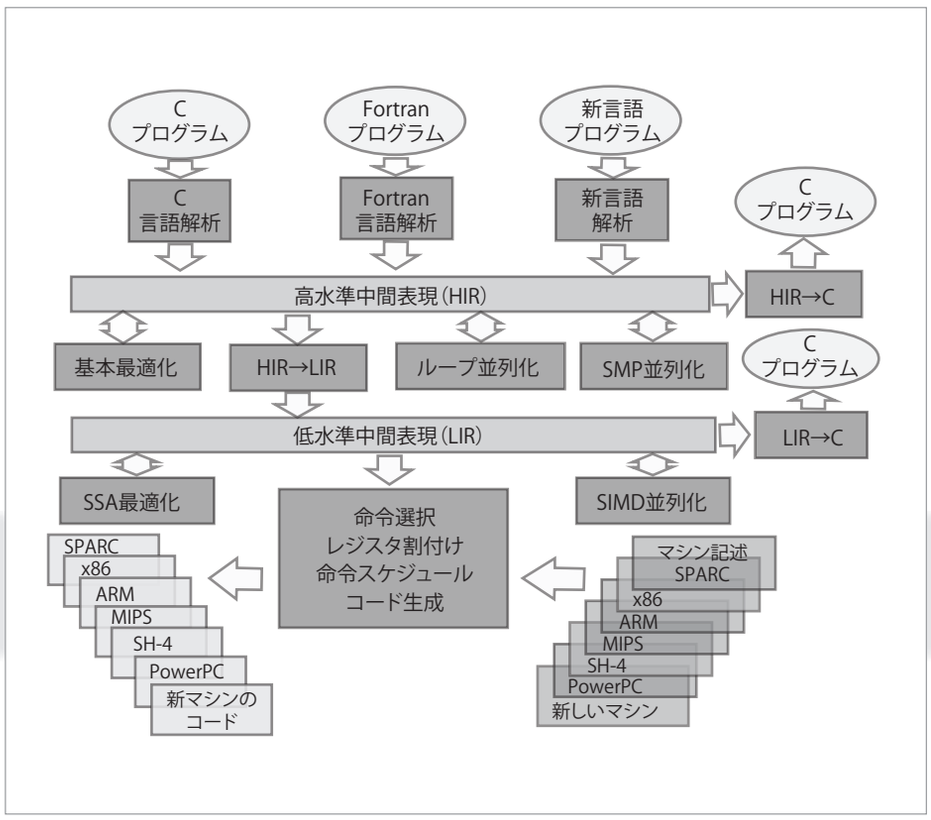

図 -8 COINS の全体構成

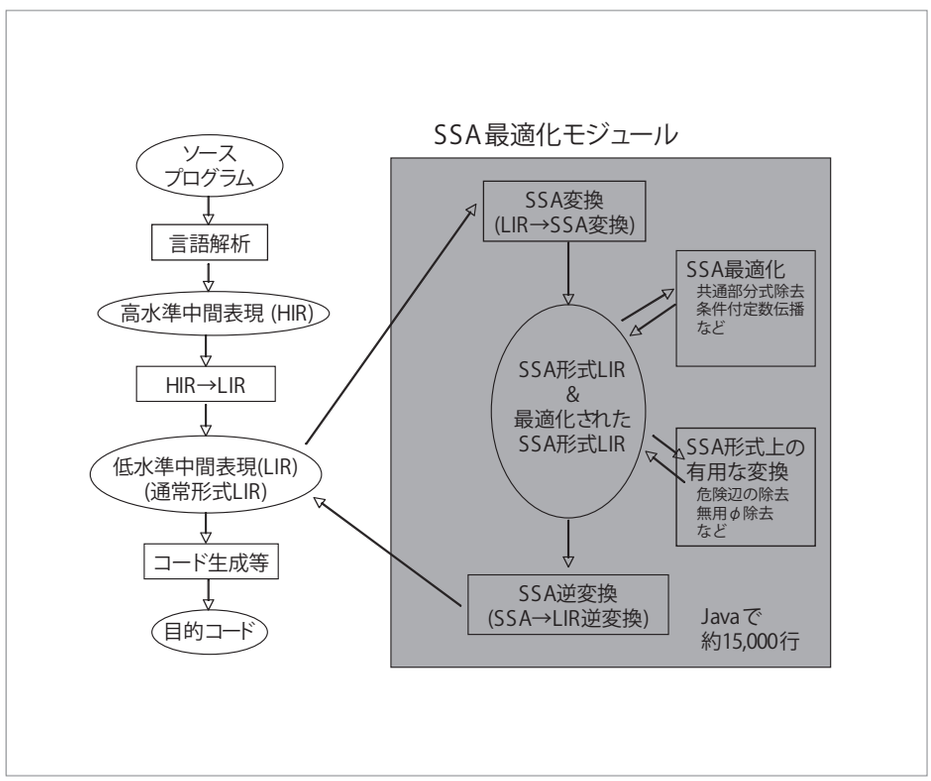

図 -9 COINS の SSA 最適化モジュール

SSA 部の処理の途中の過程をたどってみる．

COINS の SSA 変換処理を行う部分では，デフォルトではコピー畳み込みという最適化を同時に行っているので，それをはずすオプションをつけてコンパイルする [4]．

また，SSA 最適化部は，LIR という中間表現の上で動くので，変換前や変換後の形式は LIR であるが，読者の見やすさのために，その C 言語風表現を示す．この C 言語風表現は，COINS の C コンパイラで SSA オプシ

ョンに lir2c をつけることで出力される．

以下，コマンドのオプションをすべて記載すると長くなるので，コマンドの別名ファイルを読み込んで短いコマンドを使うことにしてある．詳しくは文献 11）を参照されたい．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．「%」は UNIX コマンドラインでのプロンプトを表す．

```
% source aliases
% ccprunsrd3nocf ssatransback.c
```

連載第 1 回～第 2 回で扱った言語 C0 を用いる場合は，「% source aliases」の代わりに「% source aliasesc0」を用いる，詳しくは文献 11）参照．

上のコマンドを実行すると，ssatransback.c に対する通常形式 LIR を作成し，それを SSA 変換し（SSA 最適化はしないで），SSA 逆変換を行う．また，途中結果をそれぞれ C 言語風に表示したファイルがいくつか作られる．

［例 2］次は，作成された SSA 変換前の通常形式 LIR を C 言語風に出力したものである（関連部分のみ示す，以下同じ）．

```
% cat ssatransback-main-1.lir2c

_L1:
a_1_ = read();                  // a_1_ = read()
if ((a_1_ >= 0)) { goto _L3;} // a_1_ >= 0 なら
                                // _L3 へ行く
else { goto _L4;}               // そうでなければ
                                // _L4 へ行く

_L3:
b_2_ = ((int)(a_1_));           // b_2_ = a_1_
goto _L5;

_L4:
b_2_ = ((int)((-(a_1_))));      // b_2_ = - a_1_
goto _L5;

_L5:                            // 合流
println(b_2_);
goto _L6;

_L6:
return;
```

ここで，「//」より後は説明のために加えたコメントである．全体としてやや見づらいかもしれないが，LIR よりは分かりやすいのでご了解願いたい．コンパイラ内では LIR が制御フローグラフの形になっているので，ラベルや goto が多数出てくる（ラベルで始まり goto で終わる部分が 1 つの基本ブロックになっている）．しかし本質的には例 1 と同じであることが見て取れよう．また，中間表現は型の情報を含むため，キャストが挿入されている．

［例 3］次は例 2 を SSA 変換したものである．

```
% cat ssatransback-main-2.lir2c

_L1:
a_1__1 = read();
a_1__0 = ((int)( 0));           // 不定値 0 で初期化
b_2__0 = ((int)( 0));           // 不定値 0 で初期化
if ((a_1__1 >= 0)) { goto _L3;}
else { goto _L4;}

_L3:
b_2__2 = ((int)(a_1__1));
goto _L5;

_L4:
b_2__1 = ((int)((-(a_1__1))));
goto _L5;

_L5:                            // 合流
b_2__3 = phi(b_2__2:_L3, b_2__1:_L4); // b_2__3 =
                                // φ (b_2__2, b_2__1)
println(b_2__3);
goto _L6;

_L6:
return;
```

変数名が唯一になり添字がついていること，φ関数（phi）が挿入されていることが見て取れる．また，「//不定値 0 で初期化」の文は，宣言された変数が未定義のまま使用されたときの処理のために SSA 変換が挿入するものである．詳しくは文献 4）を参照されたい．

●COINSを用いたSSA逆変換の例

COINS の SSA 部は，逆変換法として Briggs 法も Sreedhar 法も提供しているが，後者がお勧めとなっており，以下の例も Sreedhar 法を用いている．

［例 4］次は，Sreedhar 法による SSA 逆変換により，上の例 3 の SSA 形式を再び通常形式に戻したものである．

```
% cat ssatransback-main-3.lir2c

_L1:
a_1__1 = read();
a_1__0 = ((int)( 0));            // 不定値 0 で初期化
b_2__0 = ((int)( 0));            // 不定値 0 で初期化
if ((a_1__1 >= 0)) { goto _L3;}
else { goto _L4;}

_L3:
b_2__2 = ((int)(a_1__1));
goto _L5;

_L4:
b_2__2 = ((int)((-(a_1__1))));
goto _L5;

_L5:                             // 合流
println(b_2__2);
goto _L6;

_L6:
return;
```

この SSA 逆変換の結果は，添字や a_1__0 や b_2__0 の初期化を除いて，例 2 とほとんど同じ通常形式に戻っていることが見て取れる．a_1__0 や b_2__0 の初期化はこの後のフェーズで無用コードとして除去される．

おわりに

静的単一代入形式（SSA 形式）について，SSA 形式のあらまし，SSA 形式への変換，SSA 形式からの逆変換，について，コンパイラ・インフラストラクチャ COINS での例を挙げながら述べた．9 月号では，SSA 形式最適化の例について述べる予定である．

謝辞　コメントをいただいた，中田育男，滝本宗宏，中谷俊晴の各氏，COINS グループの各位と閲読者に感謝する．

参考文献

1）Appel, A.: Modern Compiler Implementation in Java, second ed., Cambridge University Press (2002).
2）Briggs, P., Cooper, K., Harvey, T. and Simpson, T. : Practical Improvements to the Construction and Destruction of Static Single Assignment Form, Softw. Pract. Exper., Vol.28, No.8, pp.859-881 (1998).
3）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/
4）静的単一代入形式に基づく最適化に関する研究：http://www.is.titech.ac.jp/~sassa/coins-www-ssa/japanese/index.html
5）Cooper, K. and Torczon, L. : Engineering a Compiler, Morgan Kauffmann (2003).
6）GCC : http://gcc.gnu.org/
7）IBM : Jikes Research Virtual Machine. http://jikesrvm.sourceforge.net/
8）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).
9）中田育男：コンパイラの構成と最適化，朝倉書店 (1999).
10）Sreedhar, V. C., Ju, R. D. -C., Gillies, D. M. and Santhanam, V. : Translating Out of Static Single Assignment Form, in Cortesi, A. and File, G. (Eds.) SAS'99, Lec. Notes in Comp. Sci., Vol.1694, pp.194-210 (1999).
11）http://www.is.titech.ac.jp/~sassa/coins-www-ssa/japanese/michishirube-ipsj-ssa/　たどれない場合は，http://www.coins-project.org/ より「静的単一代入形式最適化部」をたどり，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」をたどる．

（平成 18 年 6 月 20 日受付）