

# SSA形式（静的単一代入形式）とは

## 1. SSA形式

SSA形式 (static single assignment form, 静的単一代入形式) とは, 変数の定義がプログラムの字面上で唯一になるようにしたプログラムの表現形式である. 静的とは, プログラムの字面上で, という意味である. 変数の定義が唯一になるように変数の名前替えを行うが, これはふつう変数に添字をつけて表す. この結果, SSA形式では, プログラム (あるいは中間表現) の上で, 各変数の定義 (代入やread文など) が1箇所だけになる.

[例] SSA形式の簡単な例

$$a_1 = x_0 + y_0;$$

$$a_2 = a_1 + 3;$$

$$b_1 = x_0 + y_0;$$

## 2. SSA変換

通常形式からSSA形式へ変換することをSSA変換という.

[例] 簡単な例

次のような通常形式のプログラムがあったとする.

$$a = x + y;$$

$$a = a + 3;$$

$$b = x + y;$$

(図1 通常形式)

↓

これをSSA形式に変換すると次のようになる.

$$a_1 = x_0 + y_0;$$

$$a_2 = a_1 + 3;$$

$$b_1 = x_0 + y_0;$$

(図2 図1のSSA形式)

たとえばaについて見ると, 代入があるごとに変数の新しいversionを作り,  $a_1$ ,  $a_2$ のように添字をつけて区別する.

[例] 制御の合流がある例

次のような通常形式のプログラムがあったとする.

```
x = ...;
if (x > 0)
    a = x;
else
    a = - x;
print(a);
```

(図3 通常形式)

↓

これをSSA形式に変換すると次のようになる.

```
x1 = ...;
if (x1 > 0)
    a1 = x1;          (#1)
else
    a2 = - x1;        (#2)
a3 = φ(a1, a2);    (#3)
print(a3);
```

(図4 図3のSSA形式)

(#3)では、 $a_1$ と $a_2$ の値が合流するので、それ以降に $a$ が使われていると、 $a_1$ か $a_2$ かが決められなくなる。そこで、SSA形式では $\phi$ （あるいは $\Phi$ , phi, ファイ）関数と呼ばれる仮想的な関数を導入する。

$a_3 = \phi(a_1, a_2);$ により、これ以後使われる $a$ は $a_3$ であることを表す。また、この $\phi$ 関数 $\phi(a_1, a_2)$ は、(#1)から来たときは $a_1$ の値を返し、(#2)から来たときは $a_2$ の値を返す関数を表す。

### 3. SSA形式に基づく最適化

SSA形式では、変数の使用に対する定義が1つだけなので、SSA形式を用いると、コンパイラにおけるデータフロー解析や種々の最適化が見通しよく行なえる。

[例] 共通部分式除去

```
a = x + y;
```

$a = a + 3;$

$b = x + y;$

(図1 通常形式. 再掲)

↓

$a_1 = x_0 + y_0;$

$a_2 = a_1 + 3;$

$b_1 = x_0 + y_0;$

(図2 SSA形式. 再掲)

↓

$a_1 = x_0 + y_0;$

$a_2 = a_1 + 3;$

$b_1 = a_1;$

(図5 図2のSSA形式に共通部分式除去を適用した結果)

上記の図5の3行目の右辺では，共通部分式 $x_0 + y_0$ が $a_1$ に置き換えられている。

ちなみに，SSA形式でない通常形式（図1）では，3行目の右辺の $x + y$ を $a$ に置き換えようとしても， $a$ の値が2行目で変えられているので，図5のような最適化は簡単にはできない。

#### 4. SSA逆変換

SSA形式を通常形式に戻すことをSSA逆変換（translating out of SSA form）という。

$\phi$  関数がないSSA形式を通常形式に戻すのは簡単である。

[例] 簡単なSSA逆変換

上の図5に対するSSA逆変換は，以下のようになる。

$a_1 = x_0 + y_0;$

$a_2 = a_1 + 3;$

$b_1 = a_1;$

(図5 SSA形式. 再掲)

↓

$a1 = x0 + y0;$

$a2 = a1 + 3;$

$b1 = a1;$

(図6 図5のSSA逆変換の結果)

$\phi$  関数をそのまま実行できるアーキテクチャはないので、目的コードを生成する前に  $\phi$  関数の除去を行う必要がある。  $\phi$  関数がある場合のSSA逆変換は、あらかし次のように行う。

[例]  $\phi$  関数がある場合のSSA逆変換

前述の図4に対するSSA逆変換は、以下のようになる。

```
x1 = ...;
if (x1 > 0)
    a1 = x1;
else
    a2 = -x1;
a3 =  $\phi$ (a1, a2);    (#3)
print(a3);
```

(図4 SSA形式. 再掲)

↓

```
x1 = ...;
if (x1 > 0) {
    a1 = x1;
    a3 = a1;    (#4)
} else {
    a2 = - x1;
    a3 = a2;    (#5)
}
print(a3);
```

(図7 図4のSSA逆変換の結果)

前述したとおり、図4の  $a_3 = \phi(a_1, a_2); (#3)$  は、どちらの基本ブロックからこの合流点へ入って来たかによって  $a_3$  の値を決めるものであった。そこで、上の例では、通常形式でそれを復元するために、合流点に入ってくるそれぞれの基本ブロックのさいごに (#4) と (#5) のコピー一文を置き、 $\phi$  関数を消去する。

ただし、このままだと無駄なコピー文があるので、合併(coalescing)という手法を使って、それらを取り除くことが多い。図7で  $a_1$ ,  $a_2$ ,  $a_3$  を合併すると、元の通常形式である図3と同じものが得られる。

もう一つ大切なことがある。SSA逆変換では、上記の素朴な方法がうまくいかない場合が知られている。詳しくは [Briggs et al. 98] を参照されたい。

## 5. 参考文献

SSA形式全般については、[Appel 02] [中田99]がわかりやすい。

[Appel 02] Appel, A.: Modern Compiler Implementation in Java, second ed., Cambridge University Press, 2002.

[Briggs et al. 98] Briggs, P., Cooper, K.D., Harvey, T.J. and Simpson, L.T.: Practical improvements to the construction and destruction of static single assignment form, Software - Practice and Experience, Vol. 28 (8), pp. 859-881, 1998.

[Cooper et al. 03] Cooper, K. and Torczon, L.: Engineering a Compiler, Morgan Kaufmann, 2003.

[中田99] 中田育男: コンパイラの構成と最適化, 朝倉書店, 1999.