Appendix C

Rules for the Basic Cases

In this appendix, we give some of the basic rules which were discussed in Section 5.3. For the sake of conciseness, we just explain one subcase (or two subcases if necessary). The case numbers refer to Table 5.1.

<u>**Case 2.5:**</u> The general format of this case is:

```
int *x[size1], *y[size2] ;
. . .
x[i] = y[j]
```

The rules applied in this case are as follows:

$$\begin{aligned}
&\text{kill} = \{\ (\text{x,x1,D}) \mid (\text{x,x1,D}) \in \text{input} \wedge \text{first_elem(x[i])} = \text{D}\ \} \\
&\text{gen} = \{\ (\text{x,y1,rel} \bowtie \text{first_elem([i])} \bowtie \text{first_elem([j]))} \mid (\text{y,y1,rel}) \in \text{input}\ \} \\
&\text{changed_input} = (\text{input - } \{\ (\text{x,x1,D}) \mid (\text{x,x1,D}) \in \text{input} \wedge \text{first_elem([i])= P}\}) \\
&\qquad\qquad\qquad\quad \cup \{\ (\text{x,x1,P}) \mid (\text{x,x1,P}) \in \text{input} \wedge \text{first_elem([i])= P}\} \\
&\textbf{return}(\ \text{gen} \cup (\text{changed_input - kill})\ )
\end{aligned}$$

Consider the following example:

```
f(int i) {
    int *x[70], *y[50] ;
    int a, b ;

    x[0] = &a ;        /* stmt1 */
    y[i] = &b ;        /* stmt2 */
    x[0] = y[0] ;      /* stmt3 */
}
```

Case 2.5 appears at statement 3. Using the given rules, the sets related to this statement are as follows:

input = {(x,a,D), (y,b,P)}

kill = {(x,a,D)}

gen = {(x,b,P)}

changed input = input

output = {(x,b,P), (y,b,P)}

Case 3.2: The general format of this case is:

```
struct{
    int *b ;
} *y ;
int *x ;
. . .
x = (*y).b ;
```

The rules applied in this case are as follows:

kill = { (x,x1,rel) | (x,x1,rel) $\in$ input }
gen = { (x,y2,rel1 $\bowtie$ rel2) | (y,y1,rel1), (y1.b,y2,rel2) $\in$ input};
return(gen $\cup$ (input - kill))

Consider the following example:

```
main()
{
    struct{
        int *f1 ;
    } *y, z ;
    int *x ;
    int a ;

    z.f1 = &a ;     /* stmt 1 */
    y = &z ;        /* stmt 2 */
    x = (*y).f1 ; /* stmt 3 */
}
```

Case 3.2 appears at statement 3. Using the given rules, the sets related to this statement are as follows:

input = {(z.f1,a,D), (y,z,D)}

kill = { }

gen = {(x,a,D)}

output = {(x,a,D), (z.f1,a,D), (y,z,D)}

Case 4.4: The general format of this case is:

```
struct{
    int *a ;
} *x ;
int y ;
. . .
(*x).a = &y ;
```

The rules applied in this case are as follows:

kill = { (x1.a,x2,rel) | (x,x1,D), (x1.a,x2,rel) ∈ input }
gen = { (x1.a,y,rel) | (x,x1,rel) ∈ input}
changed_input = (input - { (x1.a,x2,D) | (x,x1,P), (x1.a,x2,D) ∈ input})
∪ { (x1.a,x2,P) | (x,x1,P), (x1.a,x2,D) ∈ input}
return(gen ∪ (changed_input - kill))

Consider the following example:

```
main()
{
    struct{
        int *f1 ;
    } *x, y ;
    int a ;

    x = &y ;         /* stmt 1 */
    (*x).f1 = &a ; /* stmt 2 */
}
```

Case 4.4 appears at statement 2. Using the given rules, the sets related to this statement are as follows:

input = {(x,y,D)}

kill = { }

gen = {(y.f1,a,D)}

changed_input = input = {(x,y,D)}

output = {(y.f1,a,D), (x,y,D)}

Case 5.6: The general format of this case is:

```
struct{
    int *a ;
    int *b ;
} *x, y ;
. . .
(*x).a = y.b ;
```

The rules applied in this case are as follows:

kill = { (x1.a,x2,rel) | (x,x1,D), (x1.a,x2,rel) $\in$ input }
gen = { (x1.a,y1,rel1 $\bowtie$ rel2) | (x,x1,rel1), (y.b,y1,rel2) $\in$ input}
changed_input = (input - { (x1.a,x2,D) | (x,x1,P), (x1.a,x2,D) $\in$ input})
$\cup$ { (x1.a,x2,P) | (x,x1,P), (x1.a,x2,D) $\in$ input}
return(gen $\cup$ (changed_input - kill))

Consider the following example:

```
main()
{
    struct{
        int *f1 ;
        int *f2 ;
    } *x, y, z ;
    int a ;

    y.f2 = &a ;        /* stmt 1 */
    if (...)
        x = &z ;       /* stmt 2 */
    (*x).f1 = y.f2 ; /* stmt 3 */
}
```

Case 5.6 appears at statement 3. Using the given rules, the sets related to this statement are as follows:

input = {(x,z,P), (y.f2,a,D)}

kill = { }

gen = {(z.f1,a,P)}

changed_input = input = {(x,z,P), (y.f2,a,D)}

output = {(z.f1,a,P), (x,z,P), (y.f2,a,D)}

Case 5.7: The general format of this case is:

```
int **px , *y ;
. . .
(*px)[i] = y ;
```

The rules applied in this case are as follows:

kill = { (x1,x2,D) | (px,x1,D), (x1,x2,D) $\in$ input $\wedge$ first_elem([i])=D }
gen = {(x1,y1,rel1 $\bowtie$ rel2 $\bowtie$ first_elem([i])) | (px,x1,rel1), (y,y1,rel2) $\in$ input }
changed_input = (input - { (x1,x2,D) | [(px,x1,P), (x1,x2,D) $\in$ input]$\vee$
[(px,x1,D), (x1,x2,D) $\in$ input $\wedge$ first_elem([i])=P] })
$\cup$ { (x1,x2,P) | [(px,x1,P), (x1,x2,D) $\in$ input]$\vee$
[(px,x1,D), (x1,x2,D) $\in$ input $\wedge$ first_elem([i])=P] }
return(gen $\cup$ (changed_input - kill))

Consider the following example:

```
f(int i) {
    int **px, *y, z ;
    int *a[70], b ;

    a[i] = &b ;         /* stmt1 */
    px = &a[0] ;        /* stmt2 */
    y = &z ;            /* stmt3 */
    px[0] = y ;         /* stmt4 */
}
```

Case 5.7 appears at statement 4. Since `px` at statement 4 is a pointer to an array, we treat `px[0]` as `(*px)[0]`. Consequently, statement 4 is of the type 5.7. Using the given rules, the sets related to this statement are as follows:

input = {(a,b,P), (px,a,D), (y,z,D)}

kill = { }

gen = {(a,z,P)}

changed input = input

output = {(a,b,P), (a,z,P), (px,a,D), (y,z,D)}

<u>**Case 6.2:**</u> The general format of this case is:

```
struct{
    int *b ;
} *y ;
int **x ;
. . .
*x = (*y).b ;
```

The rules applied in this case are as follows:

kill = { (x1,x2,rel) | (x,x1,D), (x1,x2,rel) $\in$ input }
gen = { (x1,y2,rel1 $\bowtie$ rel2 $\bowtie$ rel3) | (x,x1,rel1), (y,y1,rel2), (y1.b,y2,rel3) $\in$ input }
changed_input = (input - { (x1,x2,D) | (x,x1,P), (x1,x2,D) $\in$ input})
$\cup$ { (x1,x2,P) | (x,x1,P), (x1,x2,D) $\in$ input}
return(gen $\cup$ (changed_input - kill))

Consider the following example:

```
main()
{
    struct{
        int *f1 ;
    } *y, z ;
    int **x ;
    int *a, b, c ;

    z.f1 = &c ;    /* stmt 1 */
    y = &z ;       /* stmt 2 */
    a = &b ;       /* stmt 3 */
    if (...)
        x = &a ;   /* stmt 4 */
    *x = (*y).f1 ; /* stmt 5 */
}
```

Case 6.2 appears at statement 5. Using the give rules, the sets related to this statement are as follows:

input = {(z.f1,c,D), (y,z,D), (a,b,D), (x,a,P)}

kill = { }

gen = {(a,z.f1,P)}

changed_input = {(z.f1,c,D), (y,z,D), (a,b,P), (x,a,P)}

output = {(a,z.f1,P), (z.f1,c,D), (y,z,D), (a,b,P), (x,a,P)}

Cases 7 and 8 did not appear in the previous chapter because they can not be broken down in to smaller cases (the reason was explained in Chapter 2).

Case 6.3: The general format of this case is:

```
int **x, **py ;
. . .
*x = (*py)[i];
```

The rules applied in this case are as follows:

kill = { (x1,x2,rel) | (x,x1,D), (x1,x2,rel) $\in$ input }
gen = { (x1,y2,rel1 $\bowtie$ rel2 $\bowtie$ rel3 $\bowtie$ first_elem([i])) |
[(x,x1,rel1), (py,y1,rel2), (y1,y2,rel3) $\in$ input] }

changed_input = (input - { (x1,x2,D) | (x,x1,P), (x1,x2,D) $\in$ input})
$\cup$ { (x1,x2,P) | (x,x1,P), (x1,x2,D) $\in$ input}
return(gen $\cup$ (changed_input - kill))

Consider the following example:

```
f(int i) {
    int **x, **py ;
    int *z, w ;
    int *a[70], b ;

    x = &z ;            /* stmt1 */
    z = &w ;            /* stmt2 */
    py = &a[0] ;        /* stmt3 */
    a[0] = &b ;         /* stmt4 */
    *x = py[i] ;        /* stmt5 */
}
```

Case 6.3 appears at statement 5 in the form of `py[i]`. Since `py` is a pointer to an array, we treat this statement as `*x=(*py)[i]`. Using the given rules, the sets related to this statement are as follows:

input = {(x,z,D), (z,w,D), (py,a,D), (a,b,D)}

kill = {(z,w,D)}

gen = {(z,b,P)}

changed input = input

output = {(x,z,D), (z,b,P), (py,a,D), (a,b,D)}

Case 7.1: The general format of this case is:

```
struct{
    int *b ;
} *y ;
int *x ;
. . .
x = & (*y).b ;
```

The rules applied in this case are as follows:

kill = { (x,x1,rel) | (x,x1,rel) $\in$ input }

gen = { (x,y1.b,rel) | (y,y1,rel) $\in$ input}
return(gen $\cup$ (input - kill))

Consider the following example:

```
main()
{
    struct{
        int f1 ;
    } *y, z ;
    int *x ;

    y = &z ;         /* stmt 1 */
    x = & (*y).f1 ; /* stmt 2 */
}
```

Case 7.1 appears at statement 2. Using the above rules, the sets related to this statement are as follows:

input = {(y,z,D)}

kill = { }

gen = {(x,z.f1,D)}

output = {(x,z.f1,D), (y,z,D)}

Case 8.1: The general format of this case is:

```
struct{
    int b ;
} *y ;
int **x ;
. . .
*x = & (*y).b ;
```

The rules applied in this case are as follows:

kill = { (x1,x2,rel) | (x,x1,D), (x1,x2,rel) $\in$ input }
gen = { (x1,y1.b,rel1 $\bowtie$ rel2) | (x,x1,rel1), (y,y1,rel2) $\in$ input}
changed_input = (input - { (x1,x2,D) | (x,x1,P), (x1,x2,D) $\in$ input})
$\cup$ { (x1,x2,P) | (x,x1,P), (x1,x2,D) $\in$ input}
return(gen $\cup$ (changed_input - kill))

Consider the following example:

```
main()
{
    struct{
        int *f1 ;
        int f2 ;
    } *x, y, *z, w ;
    int a, b ;

    x = &y ;                /* stmt 1 */
    y.f1 = &a ;             /* stmt 2 */
    if (...)
        z = &w ;            /* stmt 3 */
    w.f1 = &b ;             /* stmt 4 */
    (*x).f1 = & (*z).f2 ; /* stmt 5 */
}
```

Case 8.1 appears at statement 4. Using the above rules, the sets related to this statement are as follows:

input = {(x,y,D), (y.f1,a,D), (z,w,P), (w.f1,b,D)}

kill = {(y.f1,a,D)}

gen = {(y.f1,w.f2,P)}

changed_input = input

output = {(y.f1,w.f2,P), (x,y,D), (z,w,P), (w.f1,b,D)}