Translating a switch statement to if-else statements

Suppose we have a classical and a very simple case of a switch statement as following:

```
x = 1;

switch(x) {
    case 1:
        console.log("case 1");
        break;
    case 2:
        console.log("case 2");
        break;
    default:
        console.log("default");
}

// case 1
```

At the very first glance, this piece of code can be rewritten as follows:

```
x = 1;

if (x === 1) {
    console.log("case 1");
}
else if (x === 2) {
    console.log("case 2");
}
else {
    console.log("default");
}

// case 1
```

I have implemented this kind of translation successfully. However, JavaScript is a flexible programming language. For instance, there can be no break statement in a switch case as it is shown below:

```
x = 1;

switch(x) {
    case 1:
        console.log("case 1");
    case 2:
        console.log("case 2");
        break;
    default:
        console.log("default");
}

// case 1
// case 2
```

The first idea was to translate this kind of code using labeled break statements as follows:

```
x = 1;
label: {
    if (x === 1) {
        console.log("case 1");
    }

    if (x === 2 || x === 1) {
        console.log("case 2");
        break label;
    }

    console.log("default");
}
```

```
14
15 // case 1
16 // case 2
```

But replacing break statements with labeled break statements cannot be easily done, as inside a switch case the break statement can be, for example, inside one or even several block statements or inside an if-else statement and etc. Also, there can be loops inside the switch case and the break statements relating to the loop inside the switch case rather than to the switch case itself. This makes any editing of the break statements quite problematic overall, as esprima library does not provide any information on relation between a break statement and an statement to which it relates. The examples of the switches are below:

```
1  x = 1;
2
3  switch(x) {
4      case 1:
5          console.log("case 1");
6          { { break; } }
7      case 2:
8          console.log("case 2");
9  }
10
11 // case 1
12
13 switch(x) {
14     case 1:
15         console.log("case 1");
16         while (true) {
17             break;
18         }
19         break;
20     case 2:
21         console.log("case 2");
22 }
23
24 // case 1
```

The solution to that problem cab be wrapping if-else statements in a loop rather than using labels:

```
1  x = 1;
2
3  for (_xyz123uvw in [1]) {
4      if (x === 1) {
5          console.log("case 1");
6          { { break; } }
7      }
8
9      if (x === 2 || x === 1) {
10         console.log("case 2");
11     }
12 }
13
14 // case 1
15
16 for (_xyz123uvw in [1]) {
17     if (x === 1) {
18         console.log("case 1");
19         while (true) {
20             break;
21         }
22         break;
23     }
24
25     if (x === 2 || x === 1) {
26         console.log("case 2");
27     }
28 }
29
30 // case 1
```

But this cannot be the final solution. The first problem is that the test expression can be checked against quite complex expressions rather than just literals or numbers. For example, it can be checked against function calls:

```
function case1() {
    console.log("function case1 called");
    return 1;
}


function case2() {
    console.log("function case2 called");
    return 2;
}

x = 1;

switch (x) {
    case case1():
        console.log("case 1");
    case case2():
        console.log("case 2");
        break;
}

// function case1 called
// case 1
// case 2
```

If we rewrite the code we will see that *case*1() function is called twice and this is bad, as the function can have side effects:

```
function case1() {
    console.log("function case1 called");
    return 1;
}


function case2() {
    console.log("function case2 called");
    return 2;
}

x = 1;

for (_xyz123uvw in [1]) {
    if (x === case1()) {
        console.log("case 1");
    }

    if (x === case1() || x === case2()) {
        console.log("case 2");
        break;
    }
}

// function case1 called
// case 1
// function case1 called
// case 2
```

We could solve the problem by checking the switch test expression against function calls and saving the results into variables and then reusing the variables in if-else test expressions:

```
function case1() {
    console.log("function case1 called");
    return 1;
}


```

```
function case2() {
    console.log("function case2 called");
    return 2;
}

x = 1;

for (_xyz123uvw in [1]) {
    _xyz124uvw = case1();
    _xyz125uvw = case2();

    if (x === _xyz124uvw) {
        console.log("case 1");
    }

    if (x === _xyz124uvw || x === _xyz125uvw) {
        console.log("case 2");
        break;
    }
}

// function case1 called
// function case2 called
// case 1
// case 2
```

But as one can see this is not a good solution, as we have to evaluate all the function calls preliminarily – both *case*1() and *case*2() – while the normal switch statement evaluates only *case*1(), that is it stops when it finds a suitable case branch. This is, in general, the first problem with rewriting switch statements.

The second general problem is a default branch of a switch statement. At the first glance, the solution with wrapping if-else statement in a loop works. Suppose we have a following switch statement:

```
x = 3;

switch(x) {
    case 1:
        console.log("case 1");
    case 2:
        console.log("case 2");
        break;
    default:
        console.log("default");
}

//default
```

Now we convert to series of if-else statements wrapped in a loop as follows:

```
x = 3;

for (_xyz123uvw in [1]) {
    if (x === 1) {
        console.log("case 1");
    }

    if (x === 2 || x === 1) {
        console.log("case 2");
        break;
    }

    console.log("default");
}

// default
```

However, unfortunately, the JavaScript syntax does not make one put a default case to the end of the switch statement. For example, we can legally have it between other two cases:

```
function case1() {
    console.log("function case1 called");
    return 1;
}


function case2() {
    console.log("function case2 called");
    return 2;
}

x = 3;

switch(x) {
  case case1():
    console.log("case 1");
  default:
    console.log("default");
  case case2():
    console.log("case 2");
    break;
}

// function case1 called
// function case2 called
// default
// case 2
```

This example clearly shows that, when translating the switch statement, the code of the default branch should be placed between two if-statements rather than after, as opposed to the previous example. At the same time the default code should be executed only after x is compared both to $case1()$ and to $case2()$ and this is again critical, as the functions can have side effects. All these subtleties cannot be satisfied with such an approach.

Also, one should remember that there can be nested switch statements inside a switch statement.

UPDATE

Finally, I found the correct approach solving all the problems stated above and all other problems further discovered. The approach is distinct from the above-discussed variants of solution but uses ideas learned from the research presented above. The implementation of the final solution is present in the repository containing this documentation file together with the thorough suite of unit-tests checking a *really* vast number of testing cases. When I have time I will append the description/explanation of the final solution to this documentation file.