The model for a *current-controlled current source* is

```
branch (ps,ns) in, (p,n) out;
I(out) <+ A * I(in);
```

Unassigned Sources

If you do not assign a value to a branch, the branch flow, by default, is set to zero. In the following fragment, for example, when `closed` is true, `V(p,n)` is set to zero. When `closed` is false, the current `I(p,n)` is set to zero.

```
if (closed)
    V(p,n) <+ 0 ;
else
    I(p,n) <+ 0 ;
```

Alternatively, you could achieve the same result with

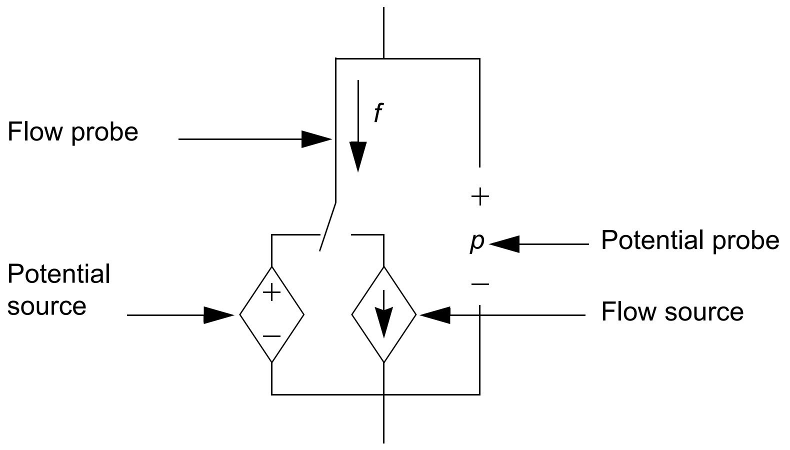
```
if (closed)
    V(p,n) <+ 0 ;
```

This code fragment also sets `V(p,n)` to zero when `closed` is true. When `closed` is false, the current is set to zero by default.

Switch Branches

Switch branches are branches that change from source potential branches into source flow branches, and vice versa. Switch branches are useful when you want to model ideal switches or mechanical stops.

To switch a branch to being a potential source, assign to its potential. To switch a branch to being a flow source, assign to its flow. The circuit model for a switch branch illustrates the effect, with the position of the switch dependent upon whether you assign to the potential or to the flow of the branch.



As an example of a switch branch, consider the module `idealRelay`.

```
module idealRelay (pout, nout, psense, nsense) ;
input psense, nsense ;
output pout, nout ;
electrical pout, nout, psense, nsense ;
parameter real thresh = 2.5 ;
analog begin
    if (V(psense, nsense) > thresh)
        V(pout, nout) <+ 0.0 ; // Becomes potential source
    else
        I(pout, nout) <+ 0.0 ; // Becomes flow source
    end
endmodule
```

The simulator assumes that a discontinuity of order zero occurs whenever the branch switches; so you do not have to use the discontinuity function with switch branches. For more information about the discontinuity function, see "Announcing Discontinuity" on page 135.

Contributing a flow to a branch that already has a value retained for the potential results in the potential being discarded and the branch being converted to a flow source. Conversely, contributing a potential to a branch that already has a value retained for the flow results in the flow being discarded and the branch being converted to a potential source. For example, in the following code, each of the contribution statements is discarded when the next is encountered.

```
analog begin
   V(out) <+ 1.0; // Discarded
   I(out) <+ 1.0; // Discarded
   V(out) <+ 1.0;
end
```

In the next example,

```
I(out) <+ 1.0;
V(out) <+ I(out);
```

the result of `V(out)` is not 1.0. Instead, these two statements are equivalent to

```
// I(out) <+ 1.0;
V(out) <+ I(out);
```

because the flow contribution is discarded. The simulator reminds you of this behavior by issuing a warning similar to the following,

```
The statement on line 12 contributes either a potential to a flow source or a flow
to a potential source. To match the requirements of value retention, the statement
is ignored.
```

Troubleshooting Loops of Rigid Branches

The following message might not actually indicate an error in your code.

```
Fatal error found by spectre during topology check.
The following branches form a loop of rigid branches (shorts)...:
```

Sometimes the simulator takes a too conservative approach to checking switch branches by assuming, when it is not actually the case, that all switch branches are in the voltage source mode at the same time. To disable this assumption, you can use the Cadence `no_rigid_switch_branch` attribute. To avoid convergence difficulties, however, do not use this attribute when you really do have multiple voltage sources in parallel or current sources in series.

To illustrate how the `no_rigid_switch_branch` can be used, assume that you have the following module.

```
// Verilog-A for sourceSwitch

`include "constants.vams"
`include "discipline.vams"

module sourceSwitch(vip1, vin1, vip2, vin2, vop1, von1);

    input vip1, vin1, vip2, vin2;
    output vop1, von1;
    electrical vip1, vin1, vip2, vin2, vop1, von1;
    parameter integer swState = 0;

//      (* no_rigid_switch_branch *) analog
    analog                  //this block causes a topology check error
    begin
        if ( swState == 0 )
        begin
            V(vop1, vip1) <+ 1.0;
            V(von1, vin1) <+ 1.0;
        end
        else if (swState == 1 )
        begin
            V(vop1, vip2) <+ 1.0;
            V(von1, vin2) <+ 1.0;
        end
```