For CC 2012

A New Method For Program Inversion

Cong Hou, George Vulov, Daniel Quinlan,
David Jefferson, Richard Fujimoto, Richard Vuduc

Georgia Tech College of Computing
Computational Science and Engineering

hpcgarage

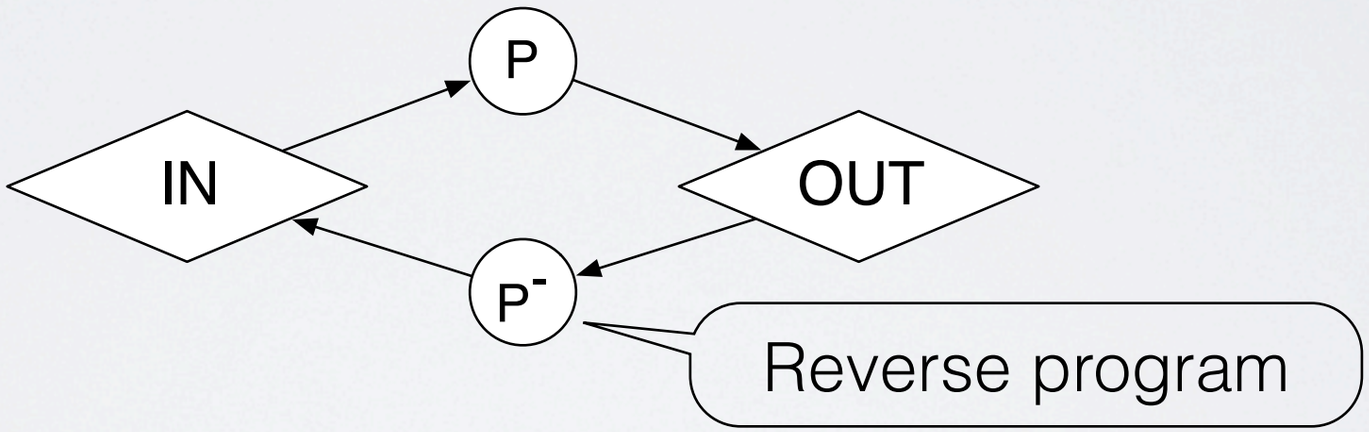
Lawrence Livermore National Laboratory

Program Inversion

- Given a program *P*, and its inverse P^-, then we have

$$P \; ; \; P^- = \text{no-op}$$

- Assume the input and output of *P* are *IN* and *OUT*:



- Program inversion is about retrieving values of *IN* from *OUT*.

Program Inversion

- What if *P* is not reversible?

```
IN: a

a = 0;

OUT: a
```

P

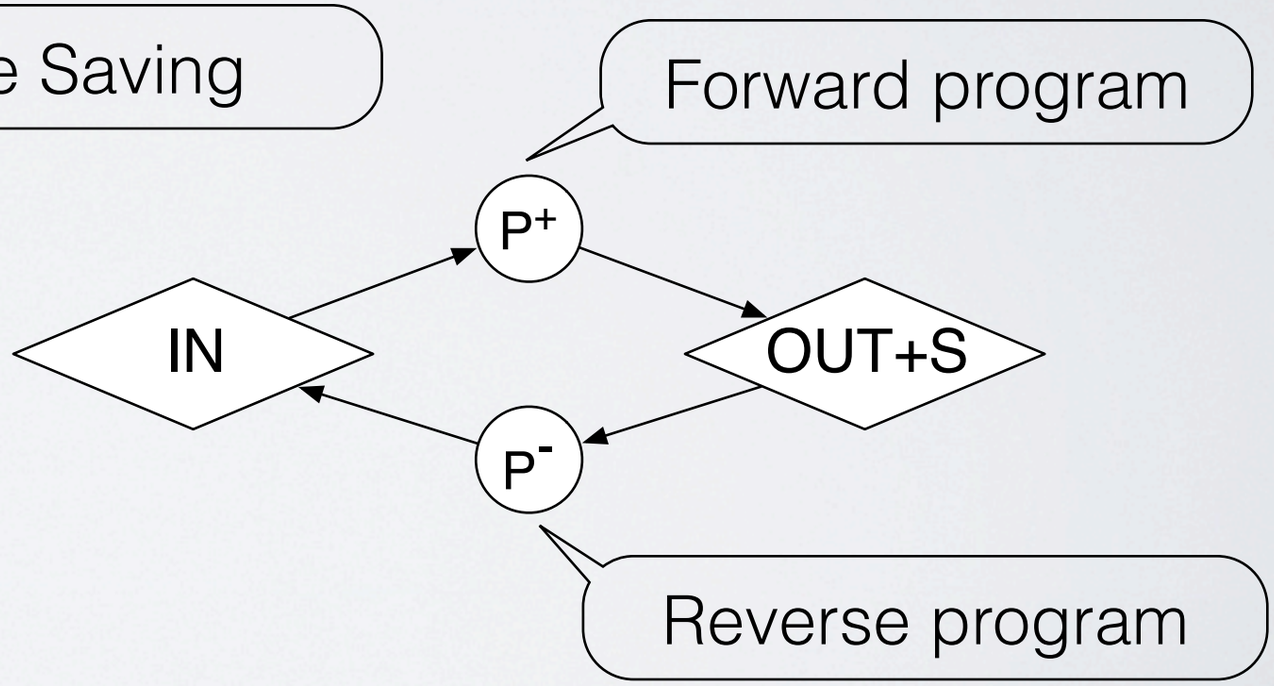
- Make it reversible!

```
IN: a

s = a;
a = 0;

OUT: a, s
```

P+

```
IN: a, s

a = s;

OUT: a
```

P-



Applications

- Our application: optimistic parallel discrete event simulation (OPDES).
  - Program inversion is used to realize the roll-back.
- Other applications:
  - Reverse debugging.
  - Program synthesis.
  - …

Our Goal

- Given a program, our C/C++ source-to-source compiler Backstroke can synthesize its forward and reverse programs automatically.

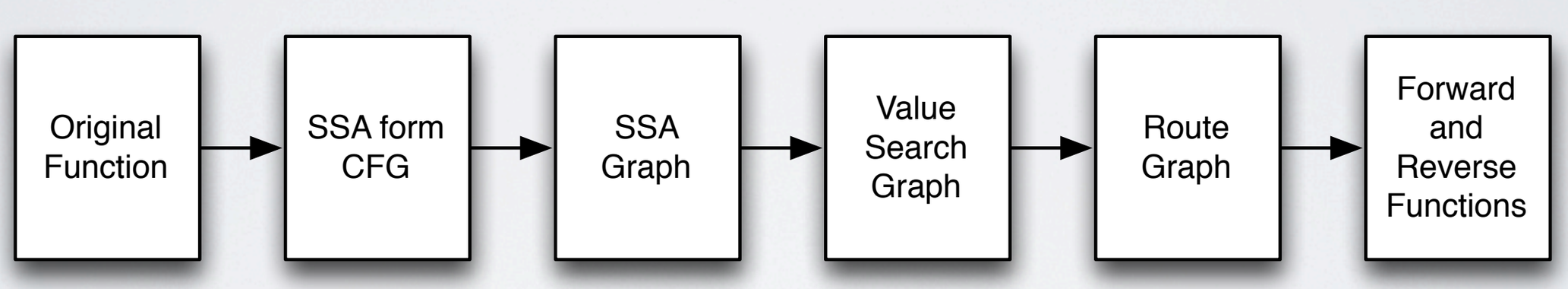
- The forward program contains as less state savings as possible.

Restrictions

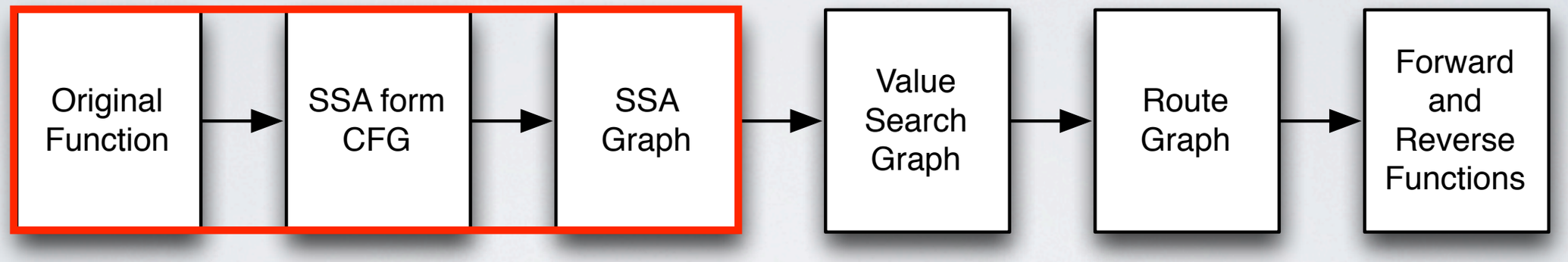
- The original program can only contain scalar data types. (No aliasing, arrays, and object accesses).
  - Current and future work: Arrays and object accesses.
- No function calls in the program.
- We cannot generate loops in the reverse program.
  - Under the review: Generating loops in the reverse program.

Overview

- Our approach:

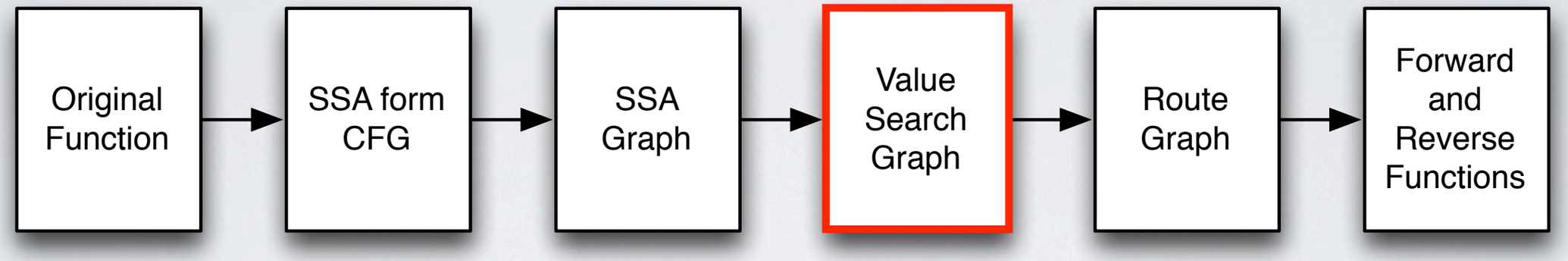


Overview



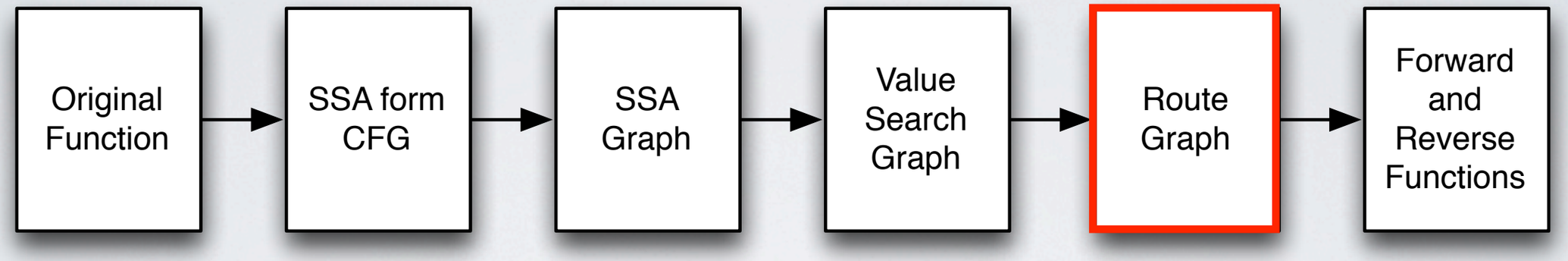
- We turn the program into the **SSA form**, so that each variable is defined only once and can represent a distinct value.

Overview



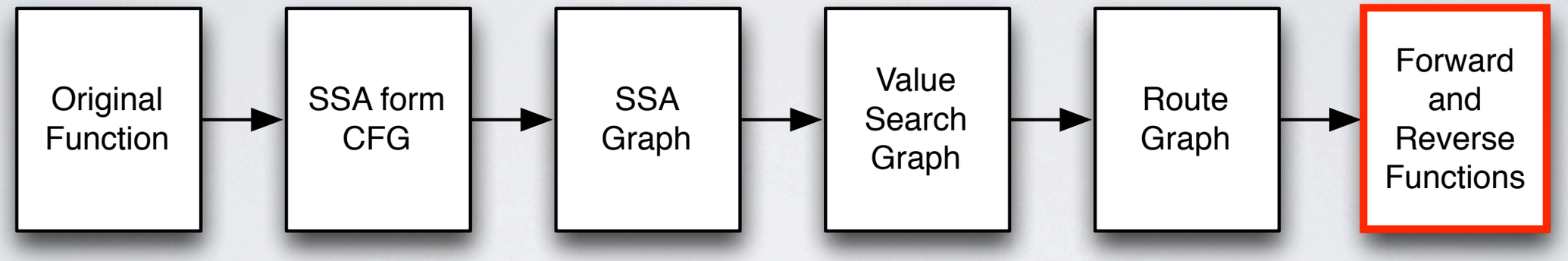
- We build a graph showing all **equality relations** between values in the program. Finding the inverse becomes a search problem in this graph. Each equality is constrained by a condition represented by a set of CFG paths.

Overview



- We then search for the desired values by **following the equalities** until available values are reached. The search should guarantee that each value is retrieved for all CFG paths.

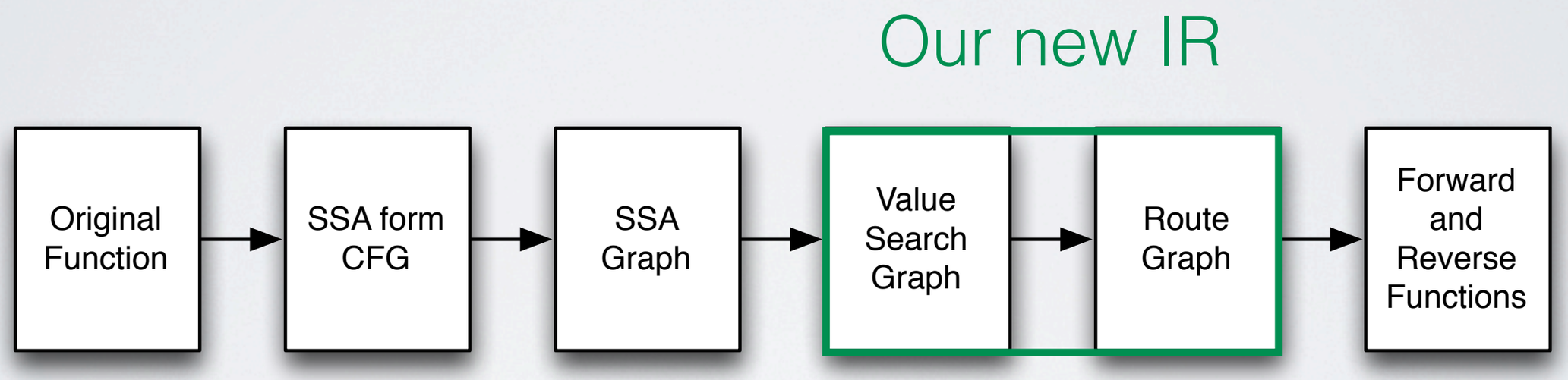
Overview



- The forward and reverse programs are built based on the search result.

Overview

- Our approach:

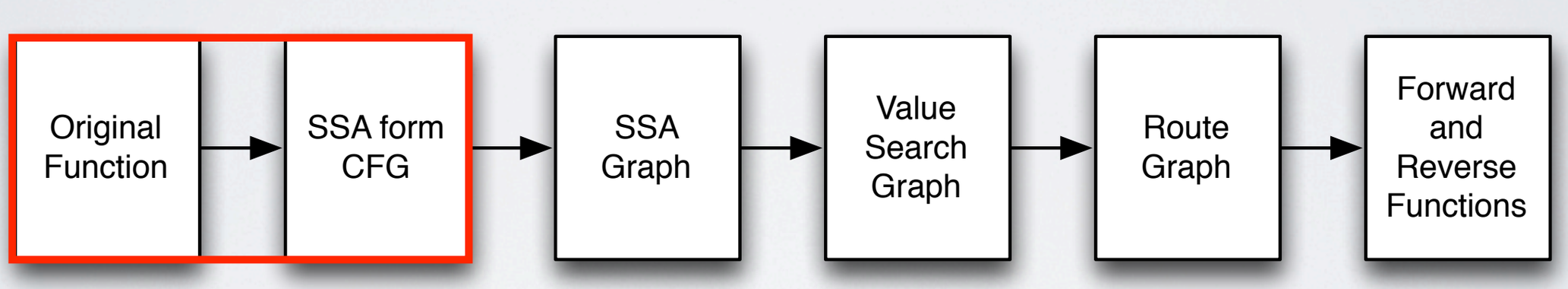


Running Example

```
IN: a, b

void foo()
{
  if (a == 0)
    a = 1;
  else
  {
    b = a + 10;
    a = 0;
  }
}

OUT: a, b
```

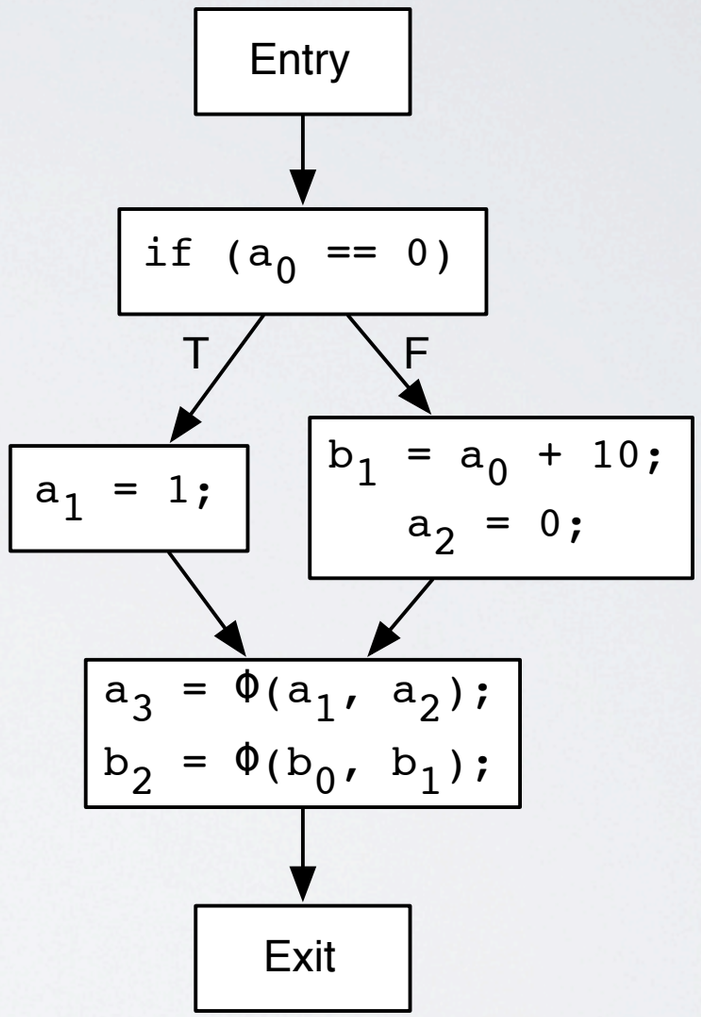
Overview

- Our approach:



SSA Form CFG

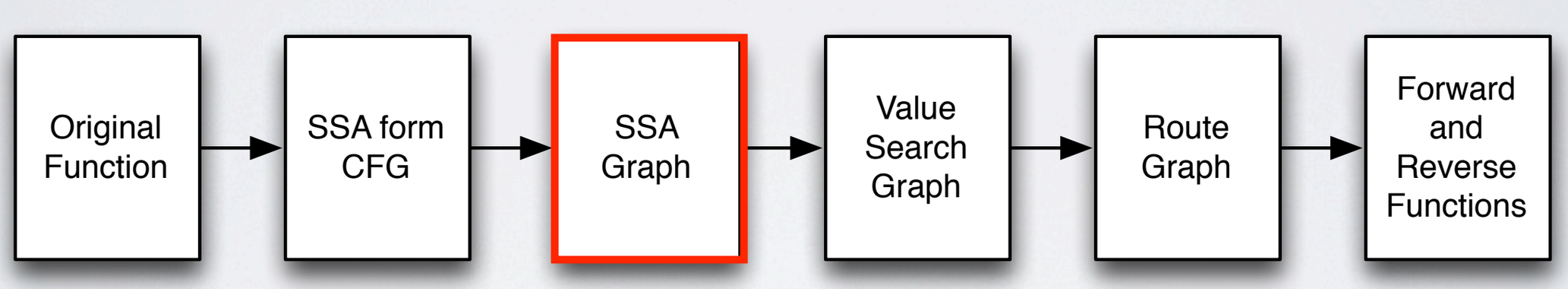
```
IN: a, b

void foo()
{
  if (a == 0)
    a = 1;
  else
  {
    b = a + 10;
    a = 0;
  }
}

OUT: a, b
```



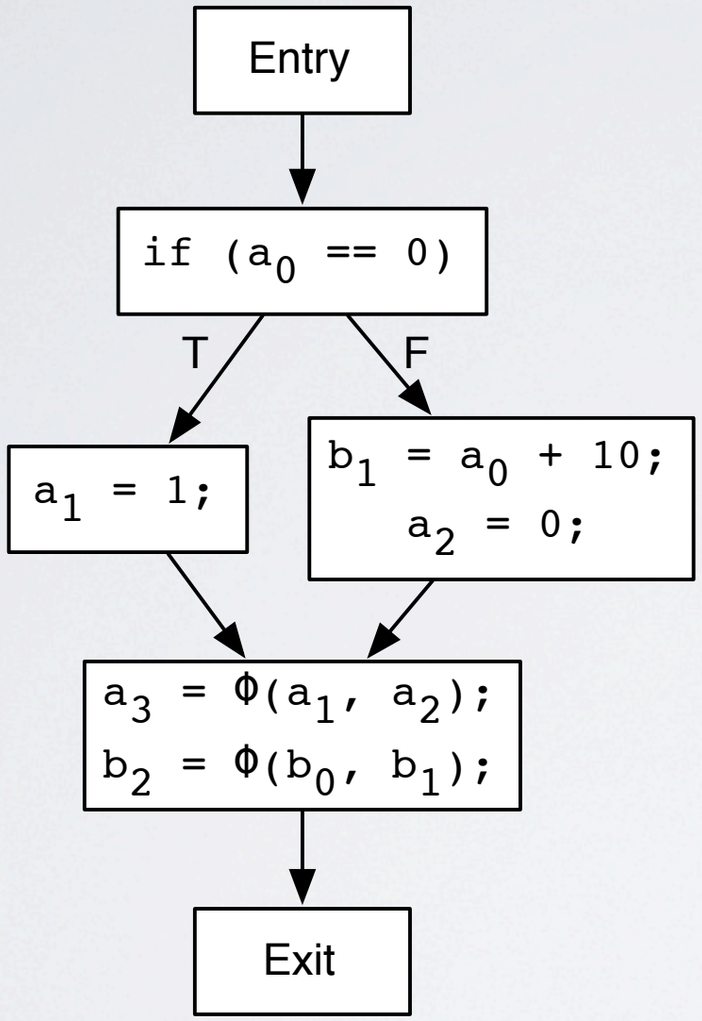
SSA form CFG

Overview

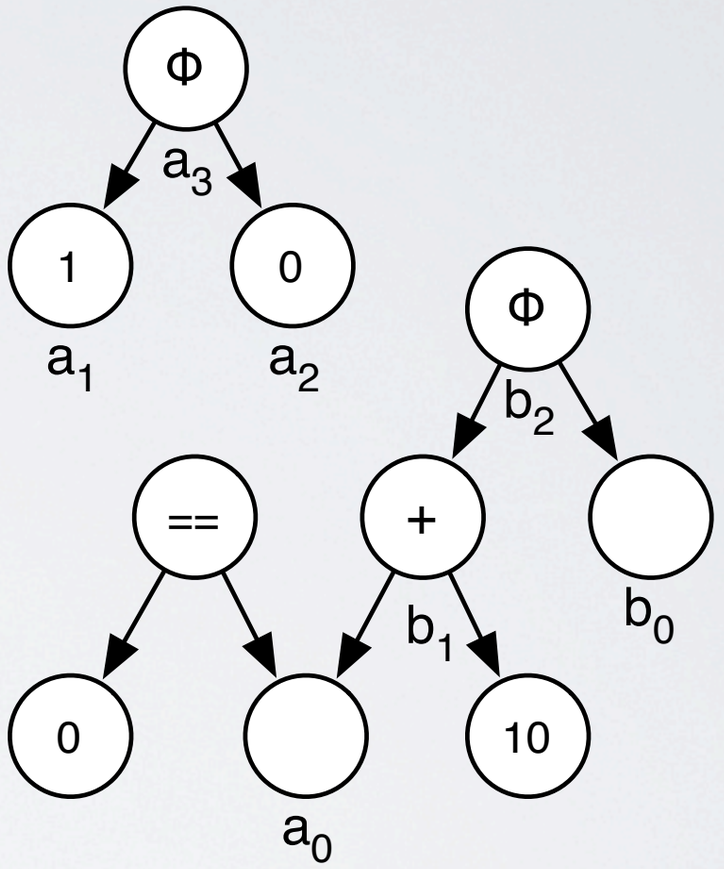
- Our approach:



SSA Graph



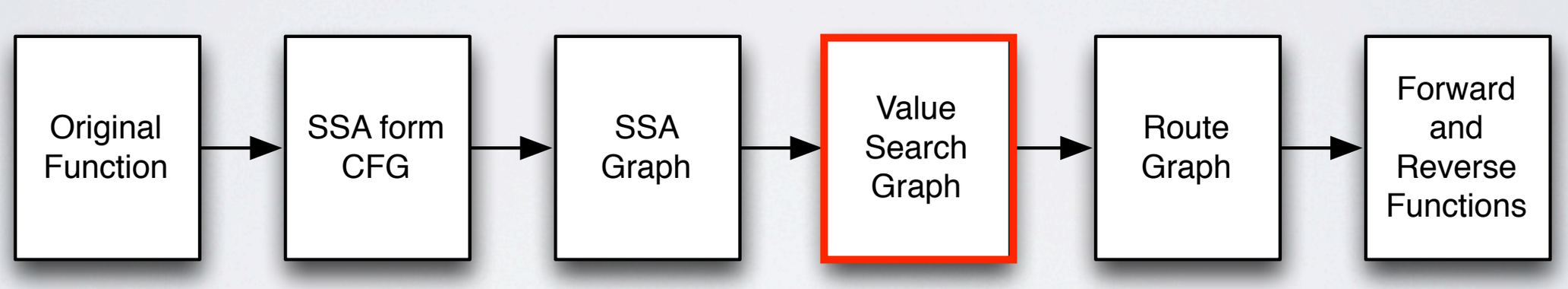
SSA form CFG



SSA Graph

Overview

- Our approach:



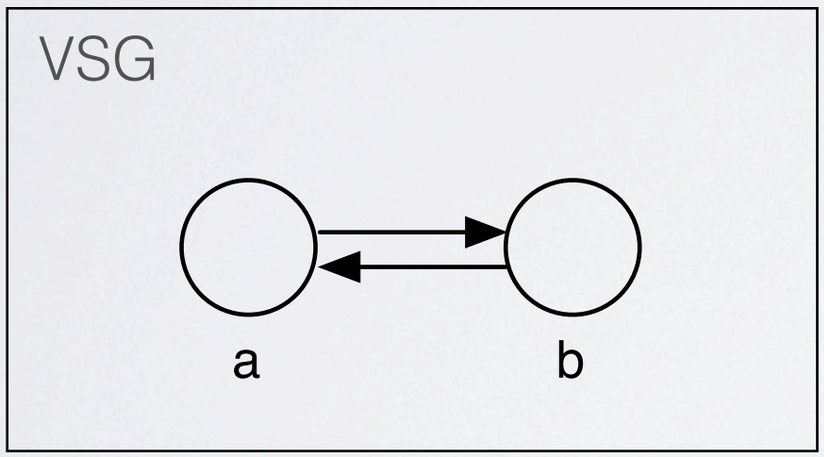
Value Search Graph

- A **value search graph** (**VSG**) is a directed graph similar to an SSA graph.
- The VSG explicitly represents equality relationships between values.
  - Two kinds of nodes:
    - Value nodes: represent values from variables or constants.
    - Operation nodes: represent operations.
  - Edges connect value nodes and operation nodes.
- Path information is attached to VSG edges indicating the condition of each equality relationship.
- Each edge is weighted to indicate the approximate cost of retrieving the value through it.

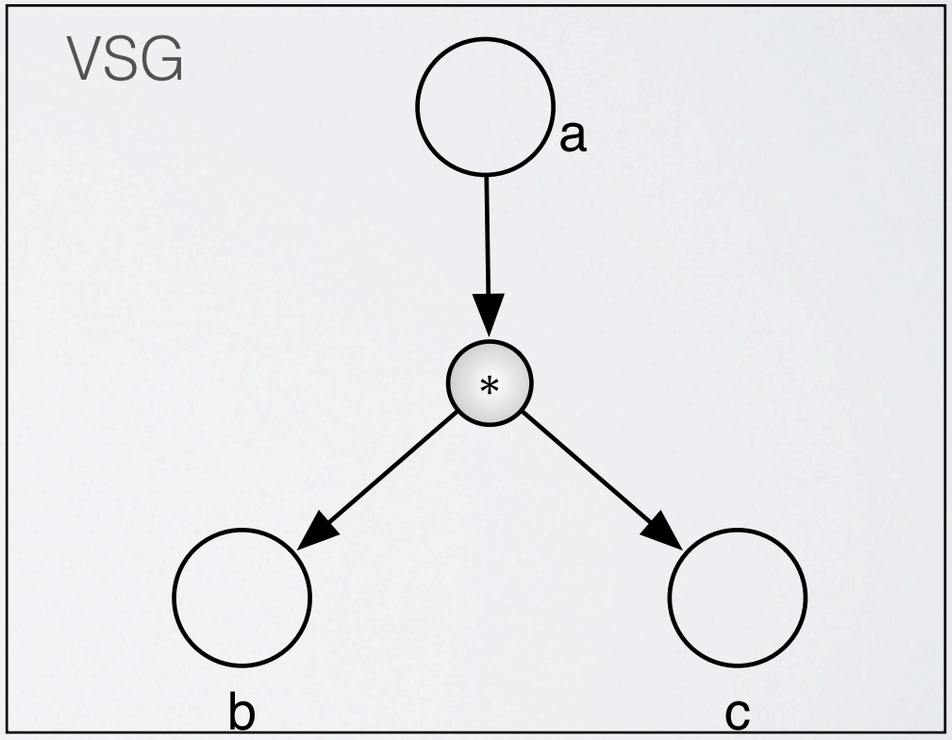
Equalities

- From where do equalities come?
  - Assignment.

```
a = b
```



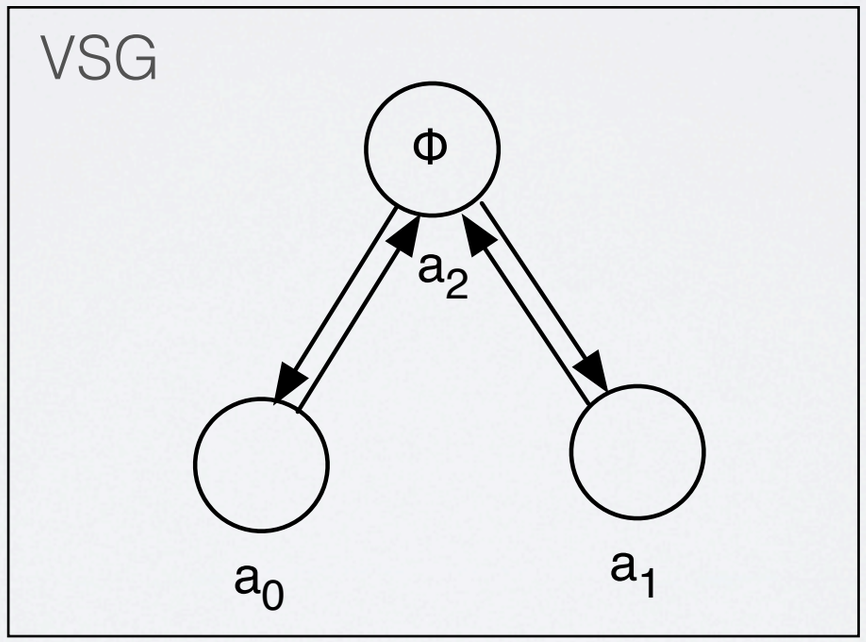

```
a = b * c
```



Equalities

- From where do equalities come?
  - Phi function.

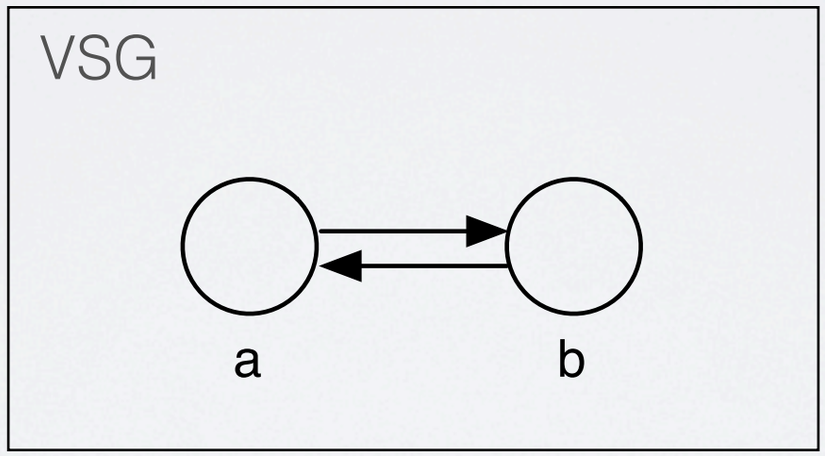
$$a_2 = \varphi(a_0, a_1)$$



Equalities

- From where do equalities come?
  - Comparison.

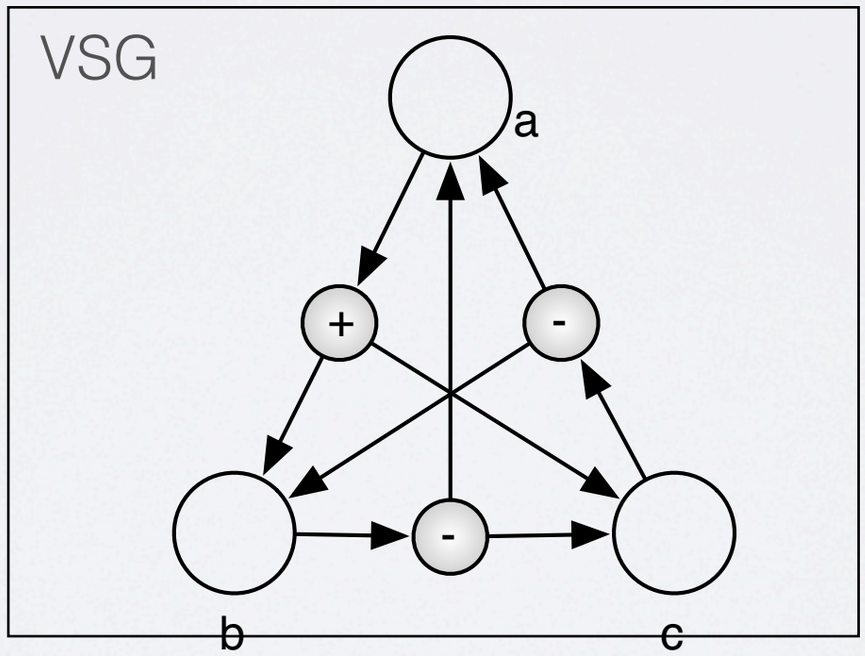
```
if (a == b) ...
if (a != b) ...
```



Equalities

- From where do equalities come?
  - Re-association.

```
a = b + c   →   b == a - c   &   c == a - b
```



Equalities

- All special operations.

Operation Name	Original Operation	New Added Operations
Unary Plus	`a = +b`	`b = +a`
Unary Minus	`a = -b`	`b = -a`
Unary Bit-Not	`a = ~b`	`b = ~a`
Unary Logical-Not	`a = !b`	`b = !a`
Unary Increase	`++a`	`--a`
Unary Decrease	`--a`	`++a`
Binary Add	`a = b + c`	`b = a - c` `c = a - b`
Binary Minus	`a = b - c`	`b = a + c` `c = b - a`
Binary Bit-Xor	`a = b ^ c`	`b = a ^ c` `c = a ^ b`

Equalities

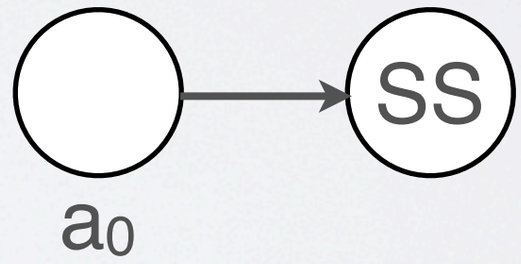
- Other equalities being explored in the future:
  - Global value numbering (GVN).
  - Solving equations.
  - Solving constraints.

Enable State Saving In VSG

- We introduce a special **state saving node** (**SS node**).
- Since we can store any value in the forward program, we connect each value node to the SS node.
- Each edge to the SS node (SS edge) has a cost proportional to the size of the memory needed to save the corresponding value.

initial value: a_0

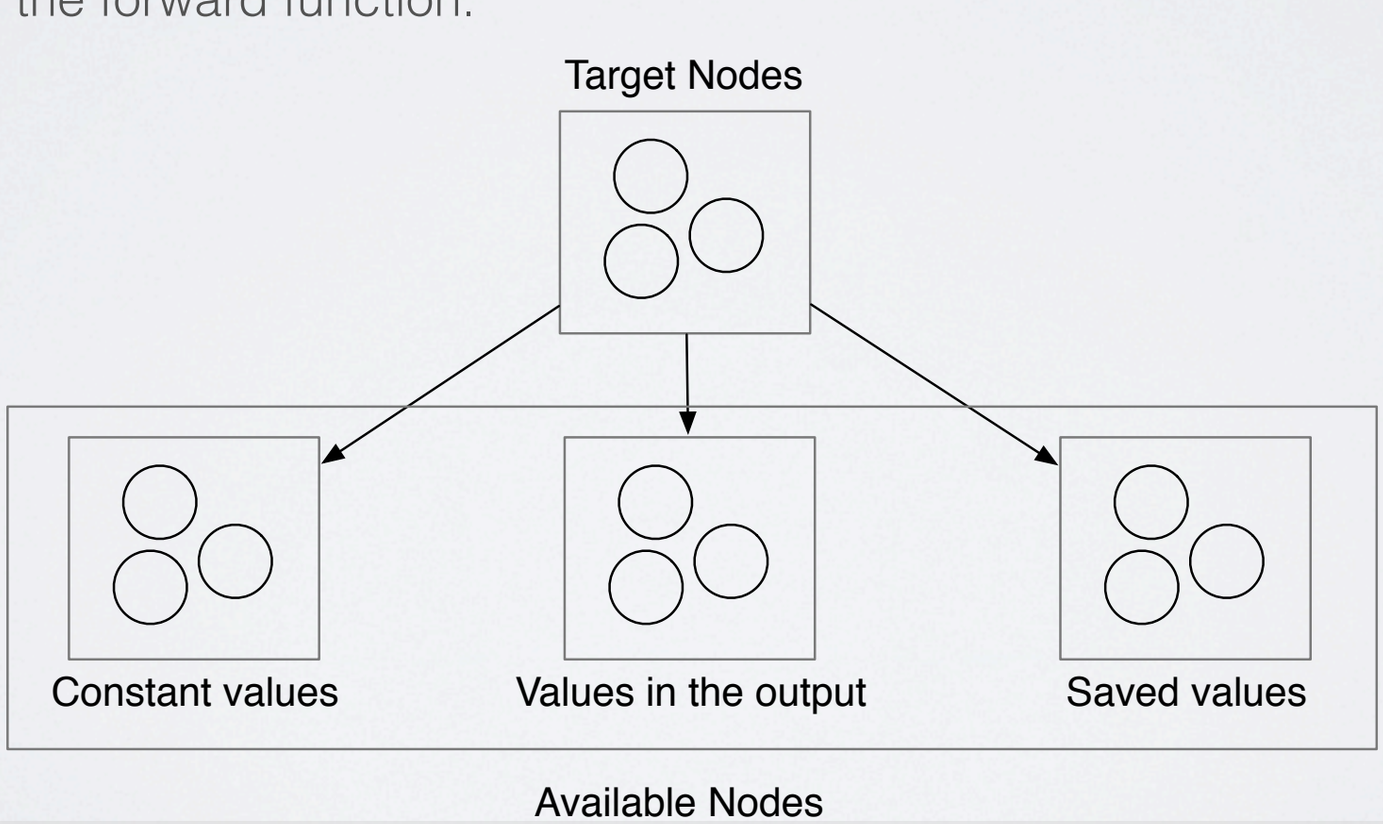
$a_1 = 0;$



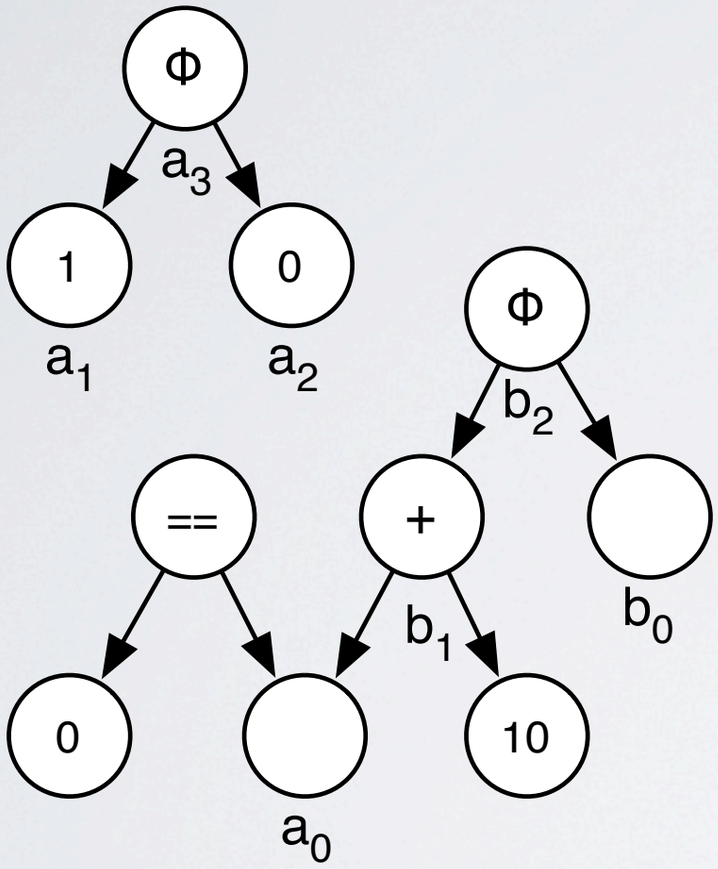
VSG

Target Nodes And Available Nodes

- The beginning and end of the search: target nodes and available nodes.
  - Target nodes: nodes containing the values we want to recover.
  - Available nodes: value nodes with available values and SS node.
    - Constants.
    - The output of the forward function.

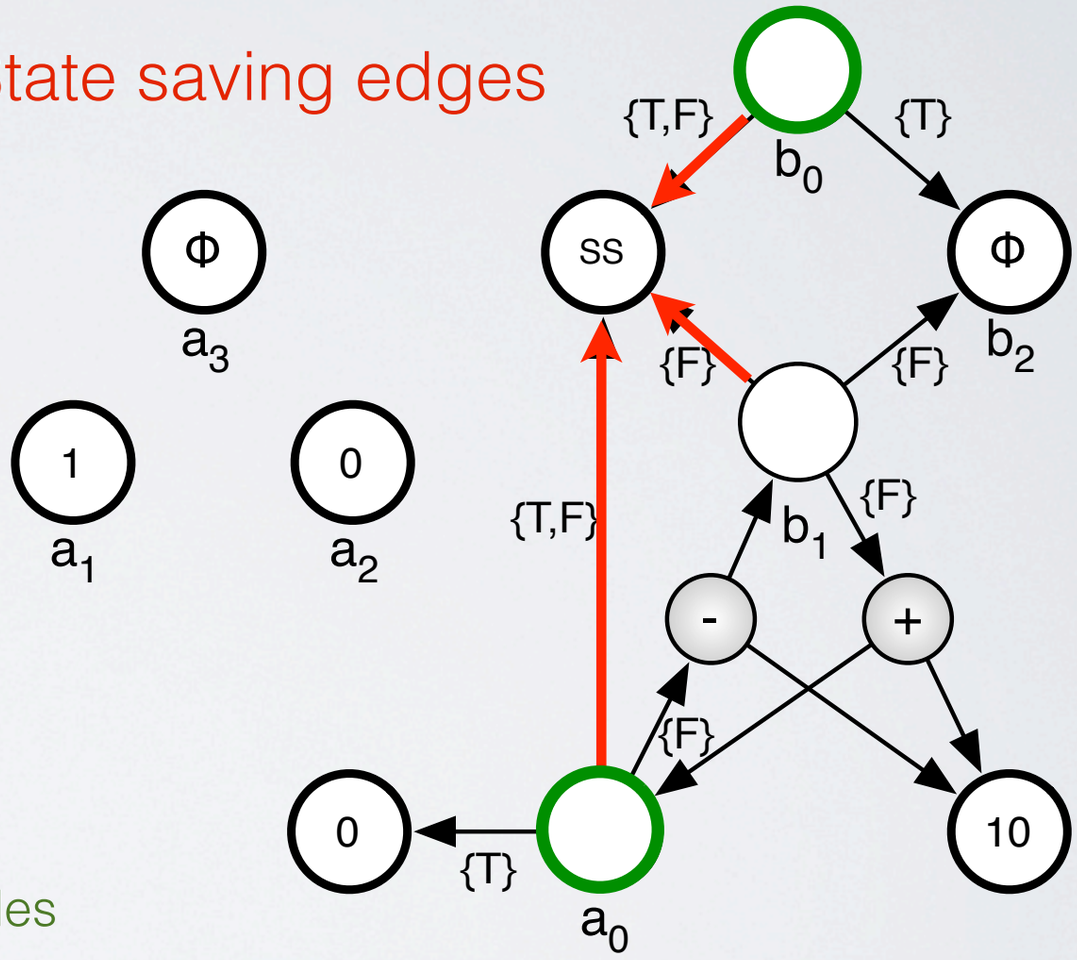


Building Value Search Graph



SSA Graph

State saving edges



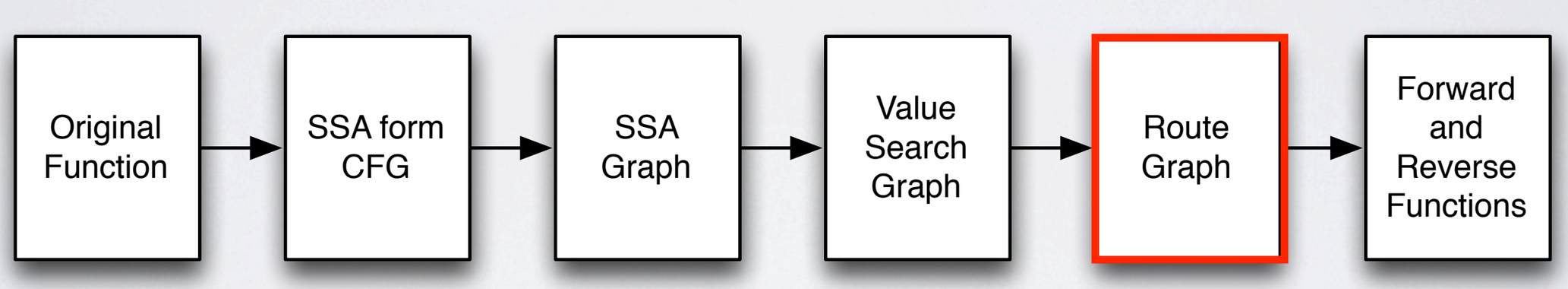
Value Search Graph

Target nodes

Available nodes

Overview

- Our approach:



Route Graph

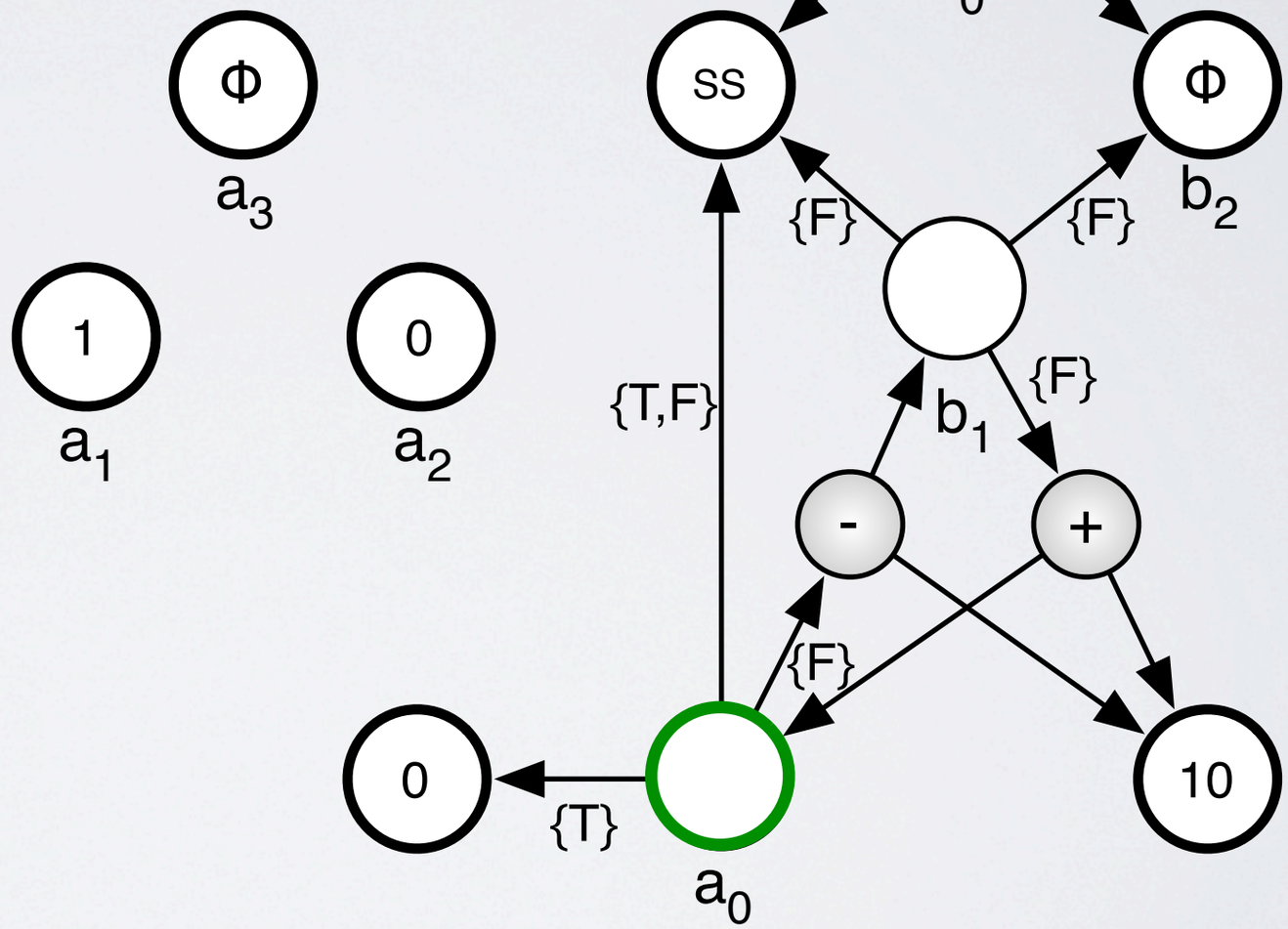
- A **route graph** (**RG**) is a subgraph of a VSG connecting all target nodes to available nodes.
- In the RG, each target value should be restored for all CFG paths. And for each CFG path, there should be only one way to get each target value through the RG.
- To guarantee the correct data dependences, the RG is acyclic for each control flow path.
- The RG is obtained by a search algorithm performed on the VSG.

Searching On VSG

- Starts from a target node and ends at available nodes.
- The search should make sure each target value can be retrieved on all control flow paths.
  - The state saving node makes sure each value can always be retrieved using state saving on all control flow paths on which this value exists.
- On each control flow path, the search guarantees each target value is retrieved only once and with the least cost (to avoid state saving).

Searching On VSG

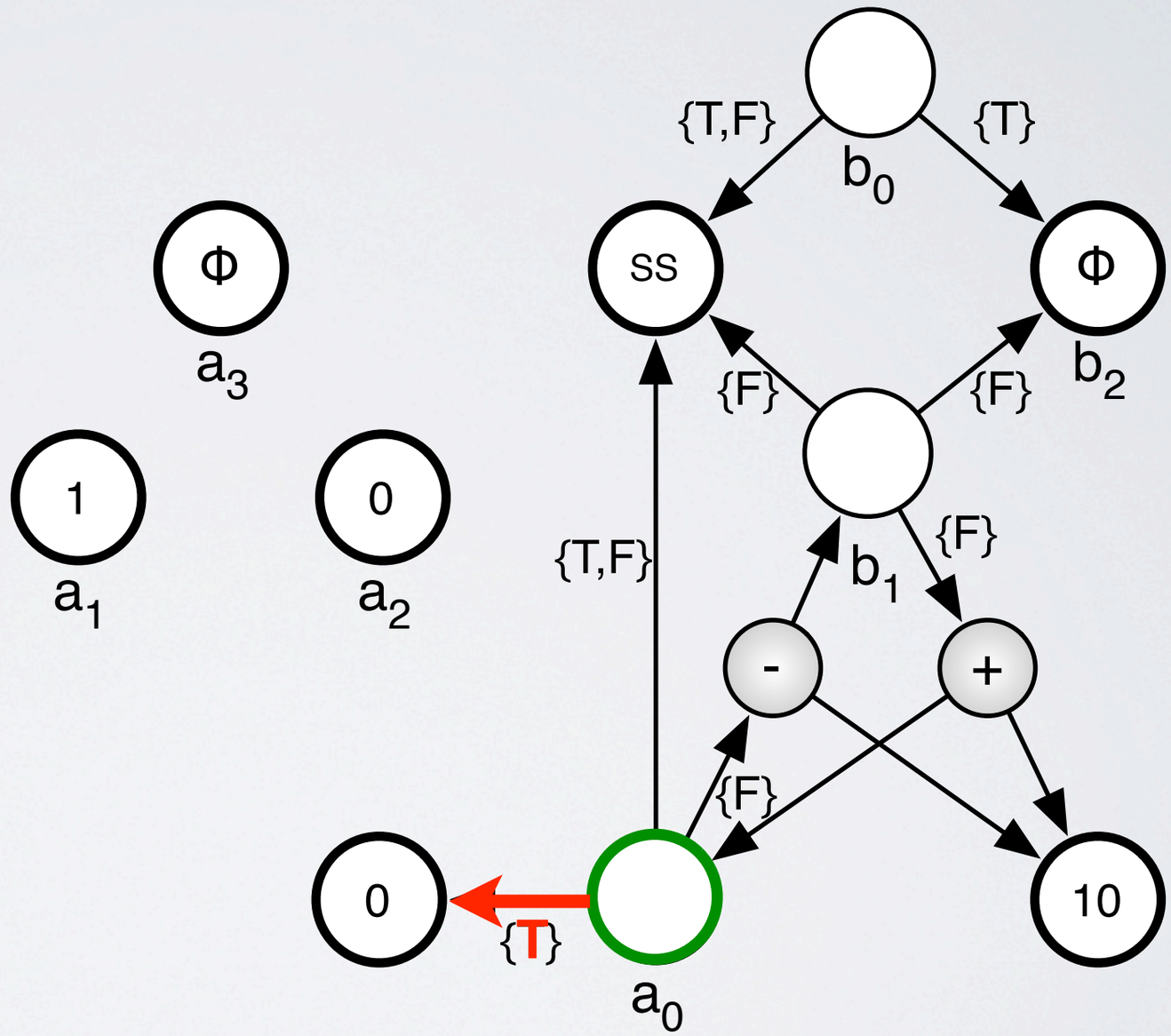
```
IN: a, b

void foo()
{
   if (a == 0)
      a = 1;
   else {
      b = a + 10;
      a = 0;
   }
}

OUT: a, b
```



Forward Function

Searching On VSG

```
IN: a, b

void foo()
{
   if (a == 0)
      a = 1;
   else {
      b = a + 10;
      a = 0;
   }
}

OUT: a, b
```

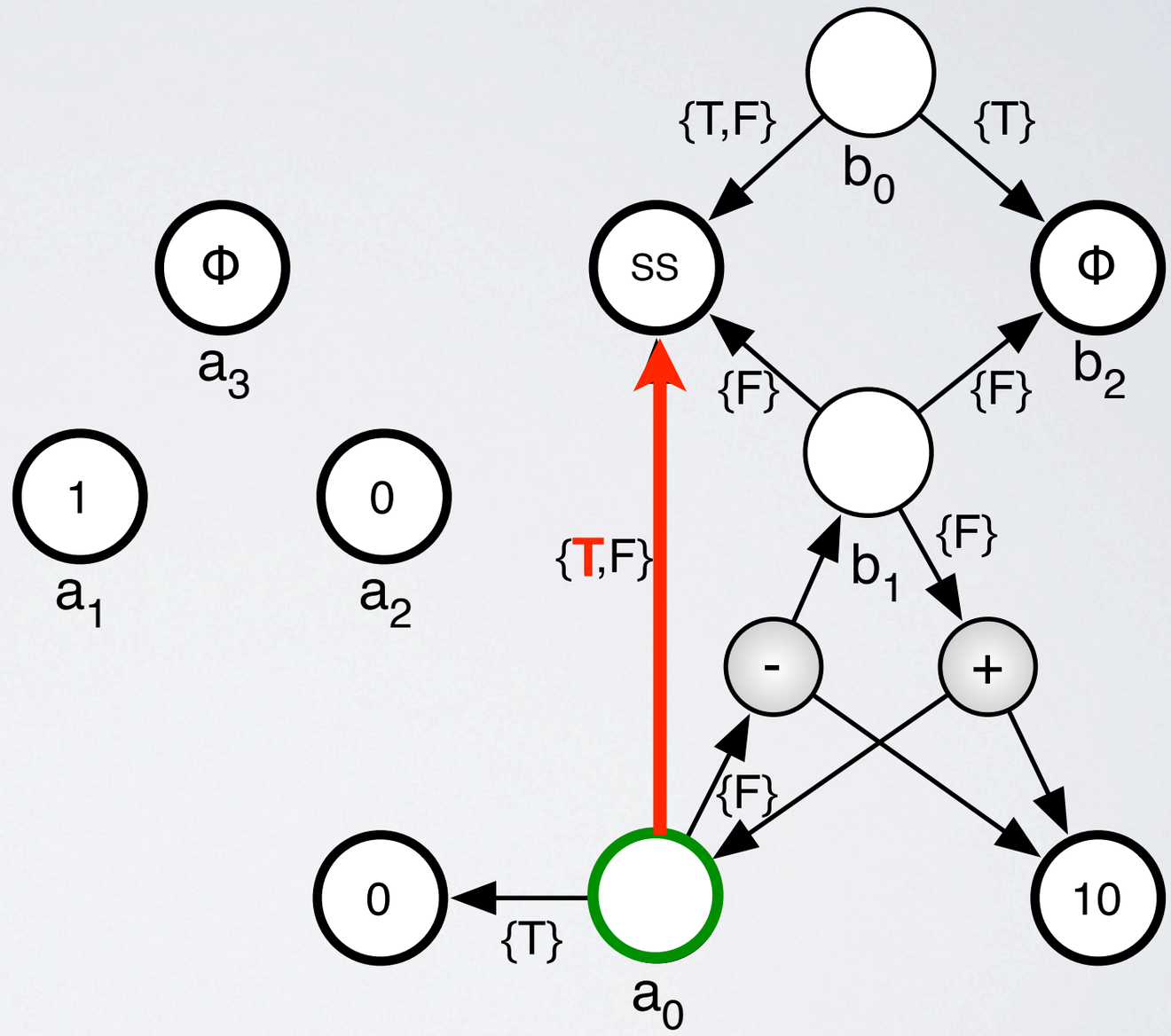


Forward Function

Searching On VSG

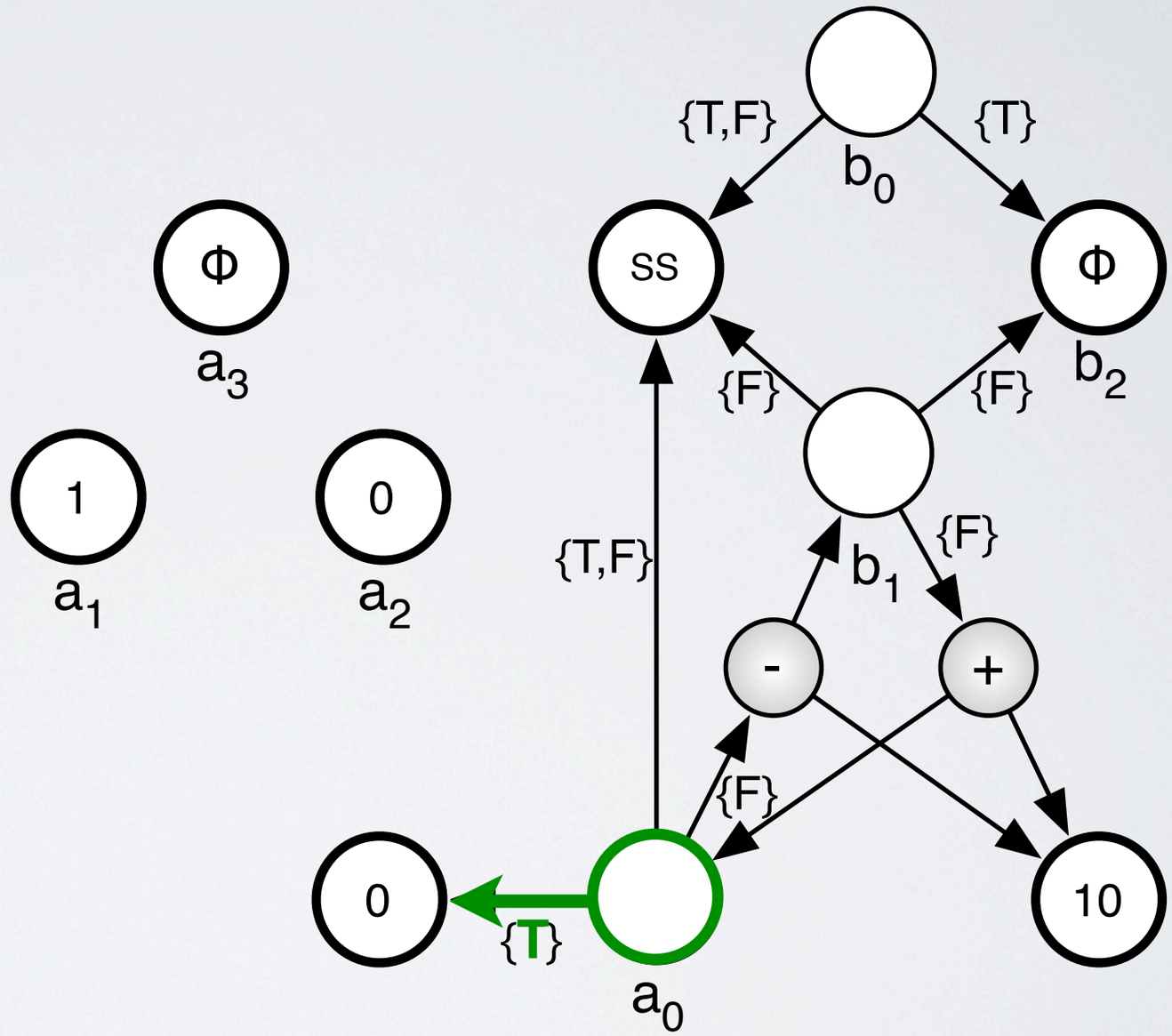
```
IN: a, b

void foo()
{
   if (a == 0) {
      store(a);
      a = 1;
   }
   else {
      b = a + 10;
      a = 0;
   }
}

OUT: a, b
```

Forward Function



Searching On VSG

```
IN: a, b

void foo()
{
   if (a == 0)
      a = 1;
   else {
      b = a + 10;
      a = 0;
   }
}

OUT: a, b
```

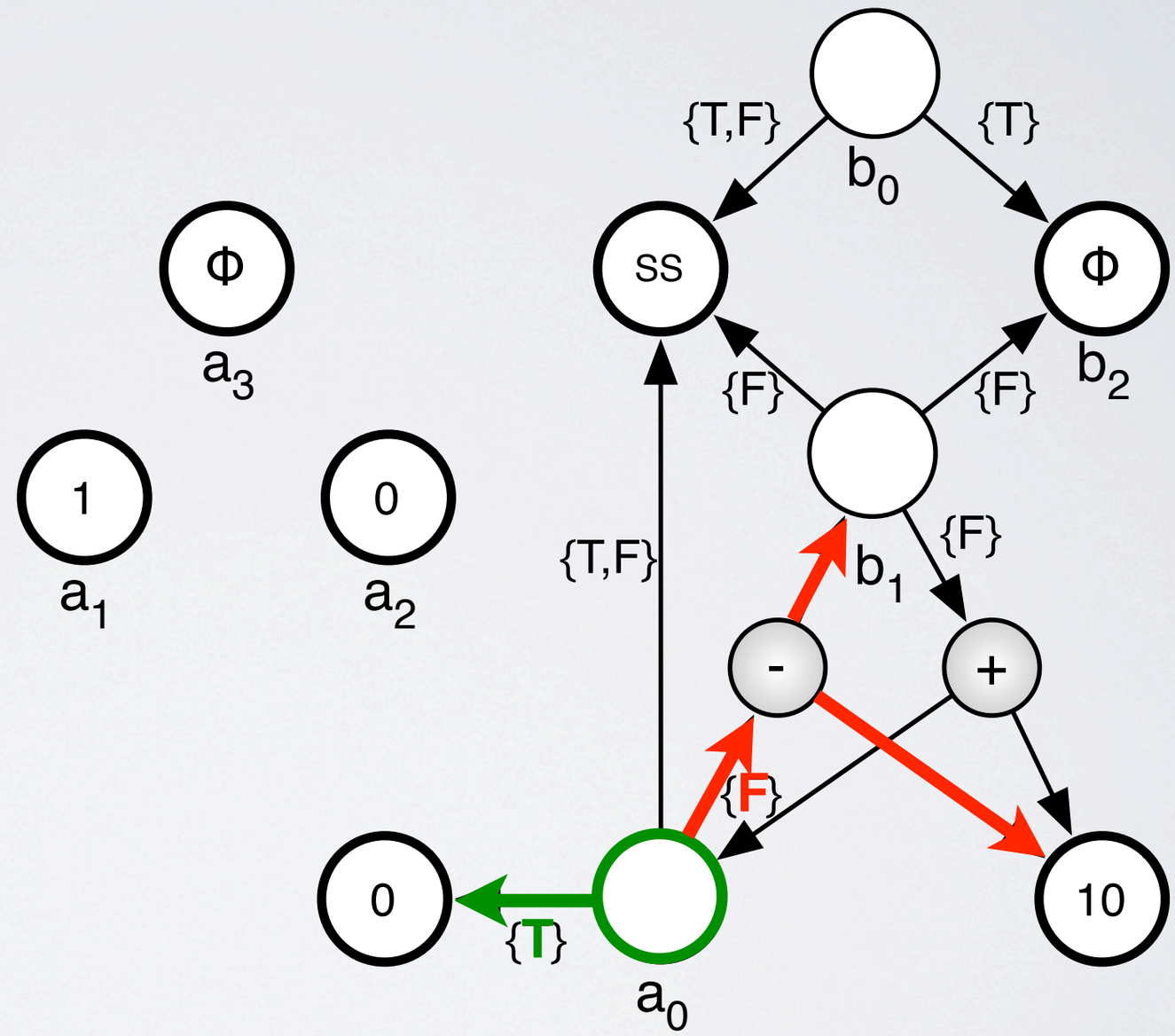


Forward Function

Searching On VSG

```
IN: a, b

void foo()
{
  if (a == 0)
    a = 1;
  else
  {
    b = a + 10;
    a = 0;
  }
}

OUT: a, b
```

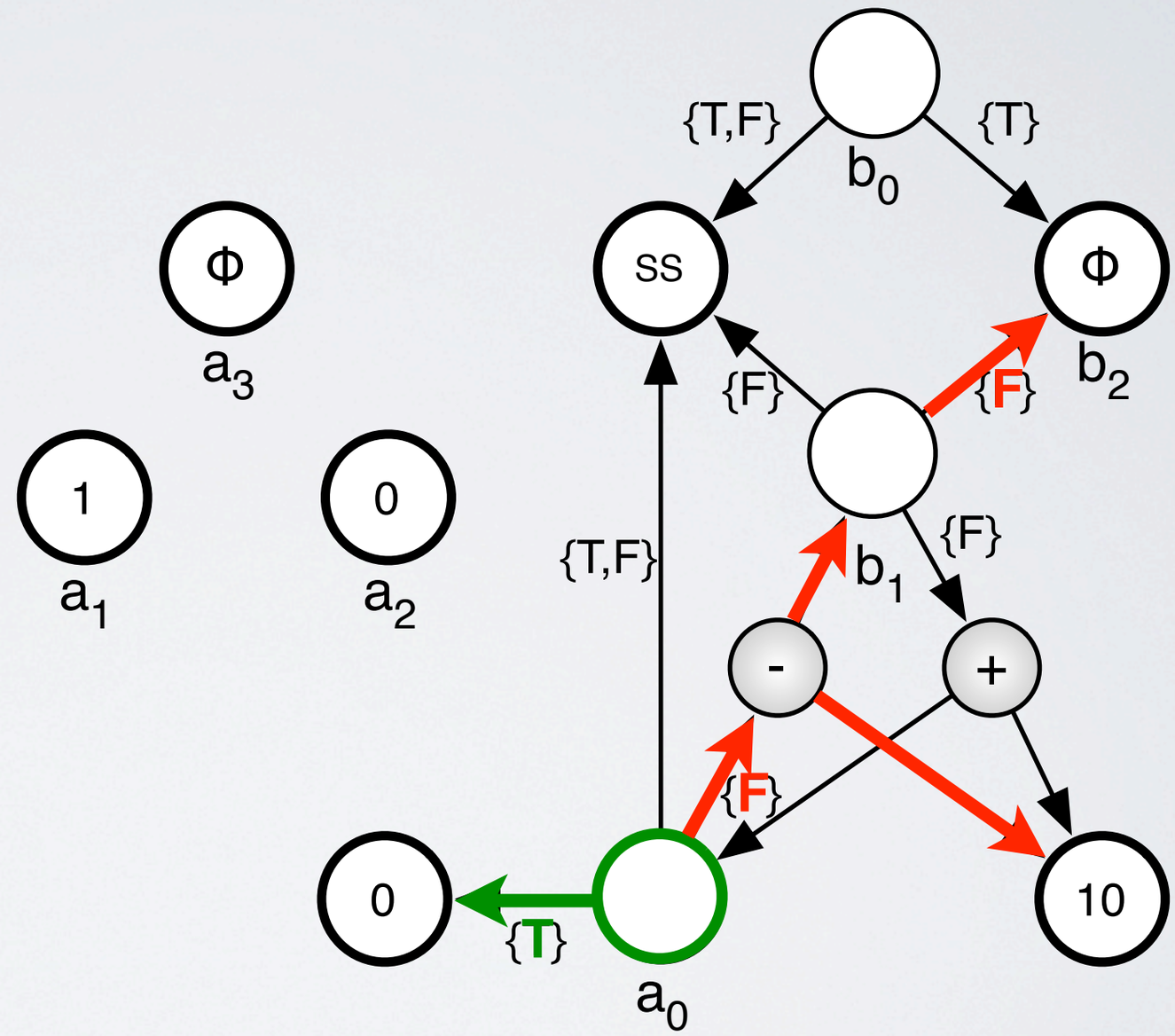
Forward Function



Searching On VSG

```
IN: a, b

void foo()
{
   if (a == 0)
      a = 1;
   else
   {
      b = a + 10;
      a = 0;
   }
}

OUT: a, b
```

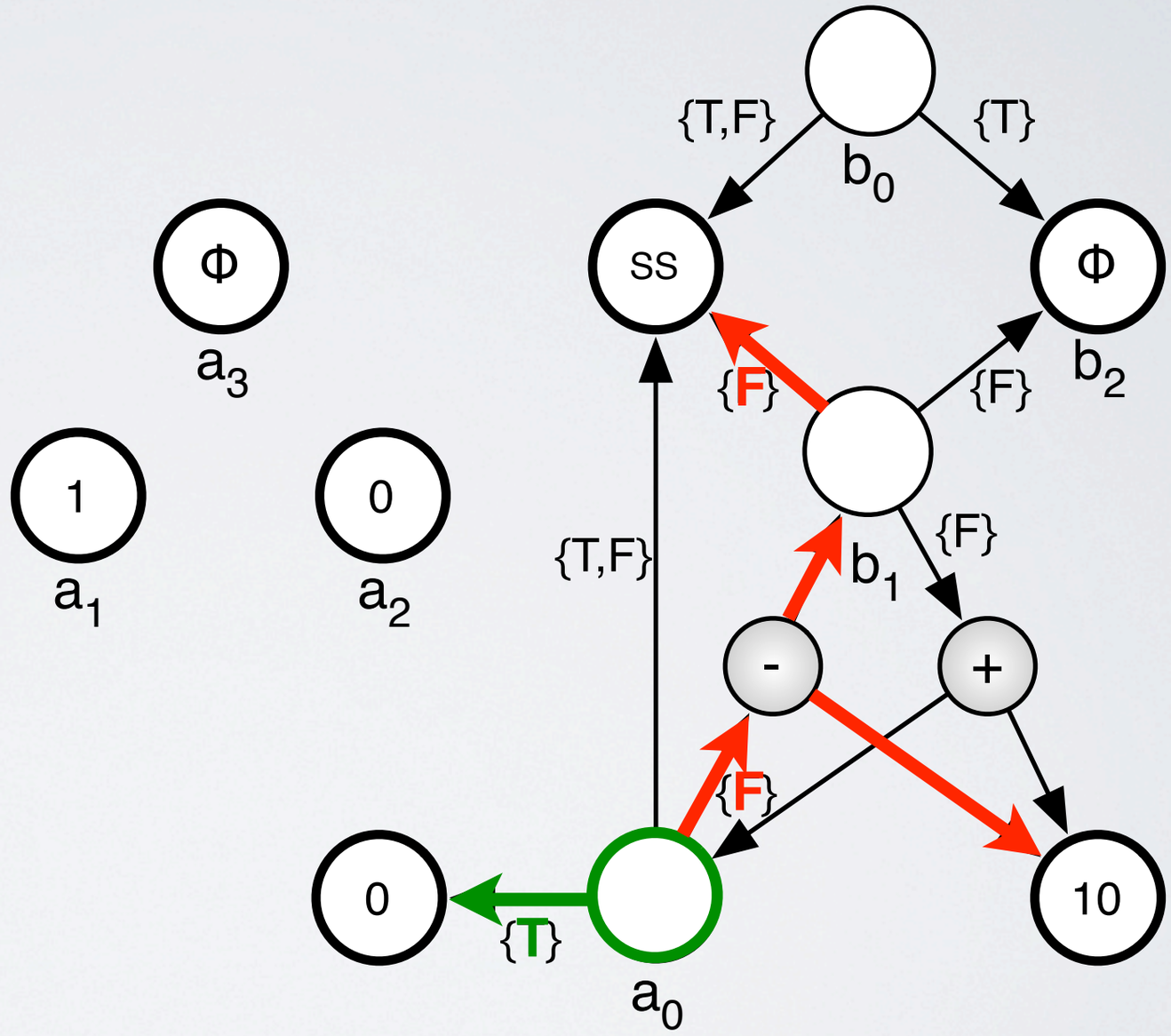
Forward Function



Searching On VSG

```
IN: a, b

void foo()
{
   if (a == 0)
      a = 1;
   else
   {
      b = a + 10;
      store(b);
      a = 0;
   }
}

OUT: a, b
```

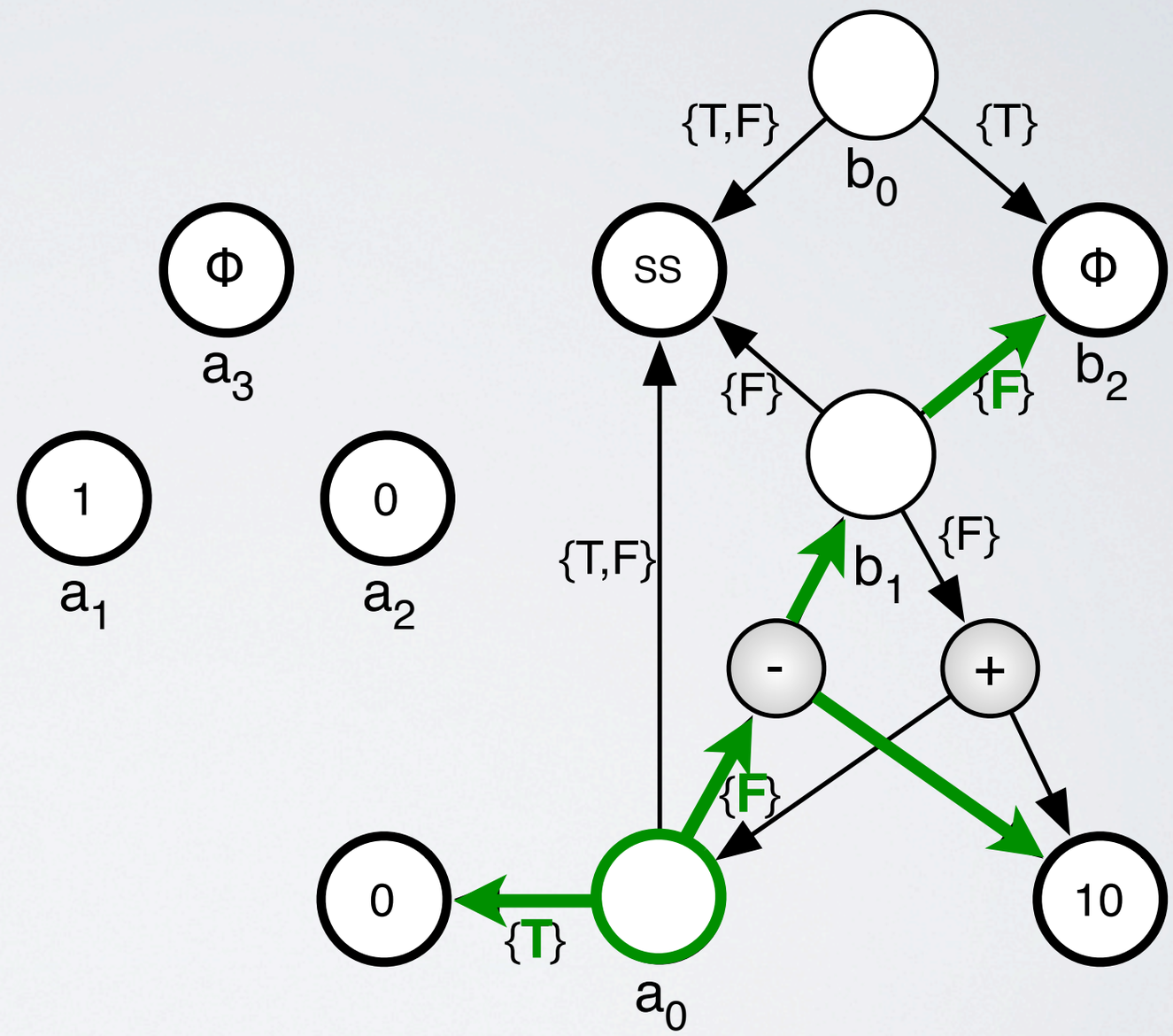
Forward Function



Searching On VSG

```
IN: a, b

void foo()
{
   if (a == 0)
      a = 1;
   else
   {
      b = a + 10;
      a = 0;
   }
}

OUT: a, b
```

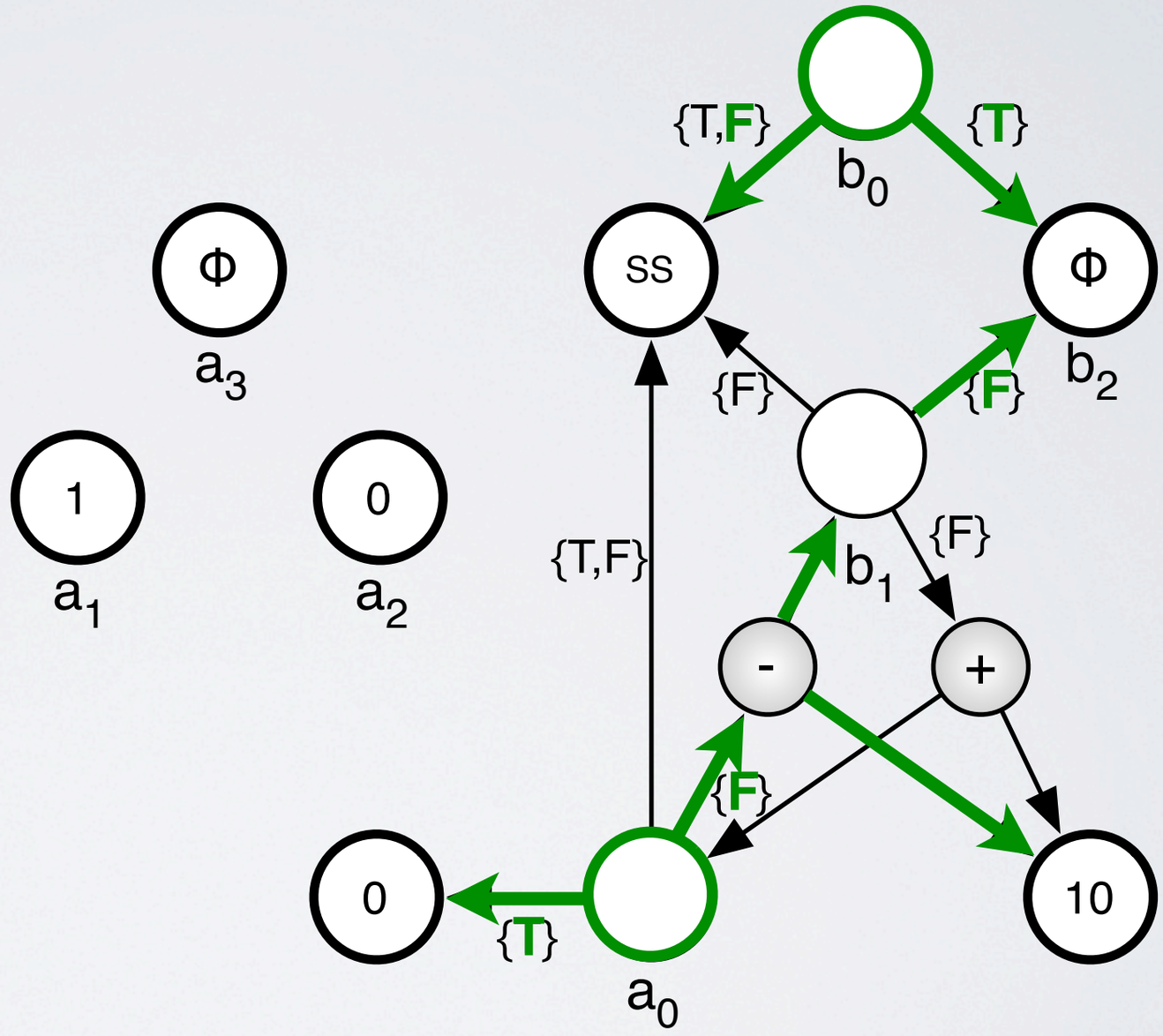
Forward Function



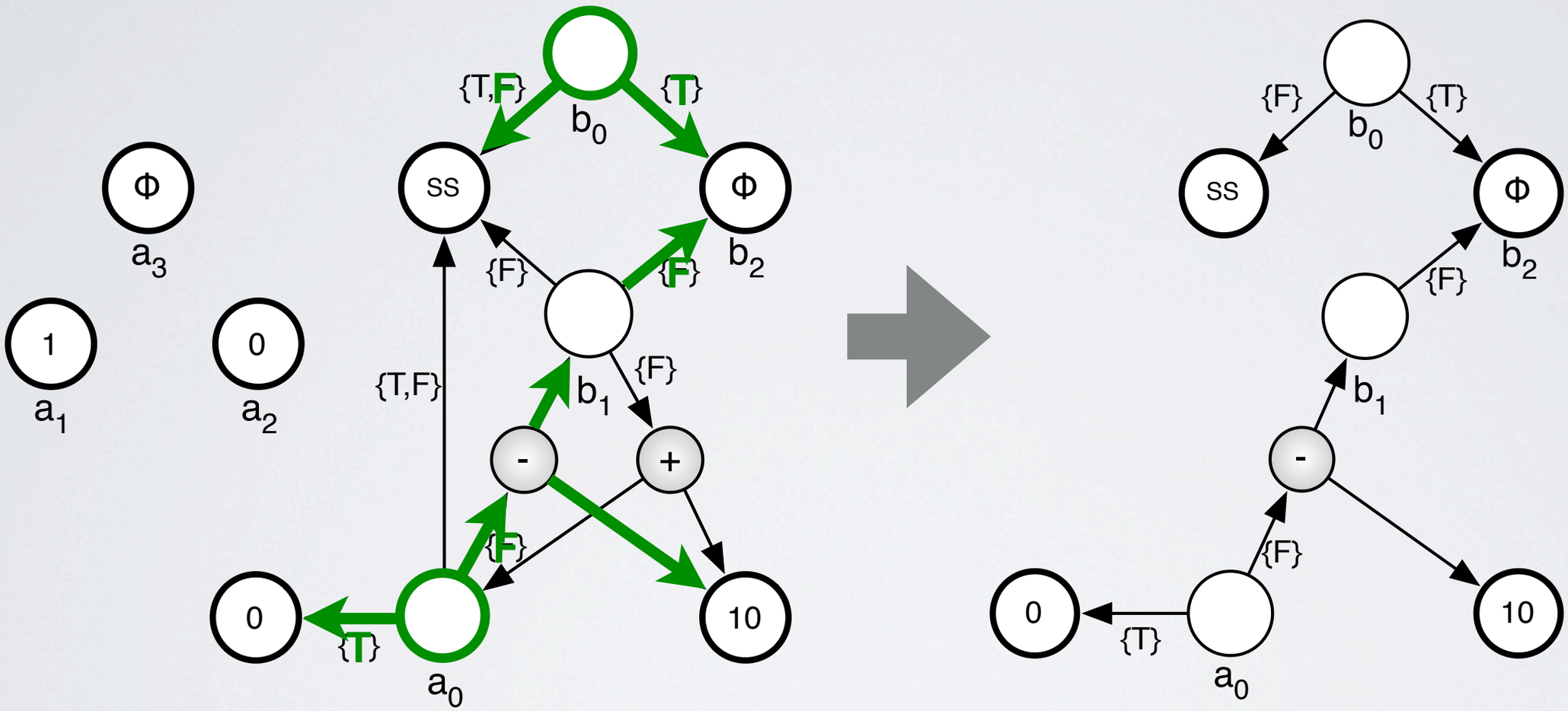
Searching On VSG

```
IN: a, b

void foo()
{
   if (a == 0)
      a = 1;
   else
   {
      store(b);
      b = a + 10;
      a = 0;
   }
}

OUT: a, b
```

Forward Function



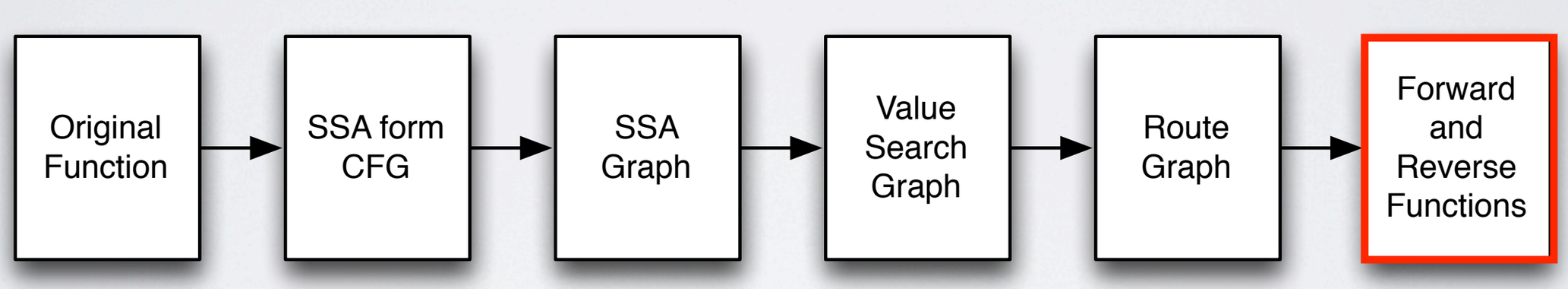
Searching Result



Route Graph

Overview

- Our approach:



Generating Forward And Reverse Programs

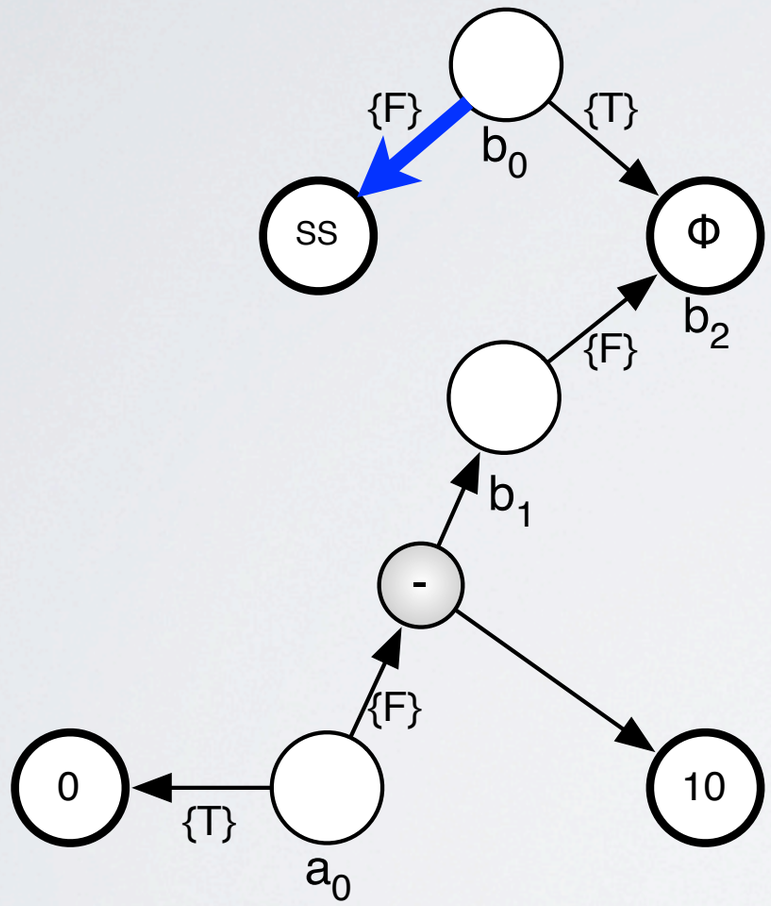
- Generating the forward program:
  - Recording path information.
  - Performing state saving.
- Generating the reverse program:
  - First generating a CFG of the reverse program from the RG.
  - Translating the CFG into source code.

Control Flow Path Recording

- A bit vector is used to record the control flow path in the forward program.
  - For each two way branch node, one bit is used to record whether the true or false body is selected at runtime.

- This bit vector will be stored at the end of the forward function, and then used to build the predicates in the reverse program.

Generate The Forward Program

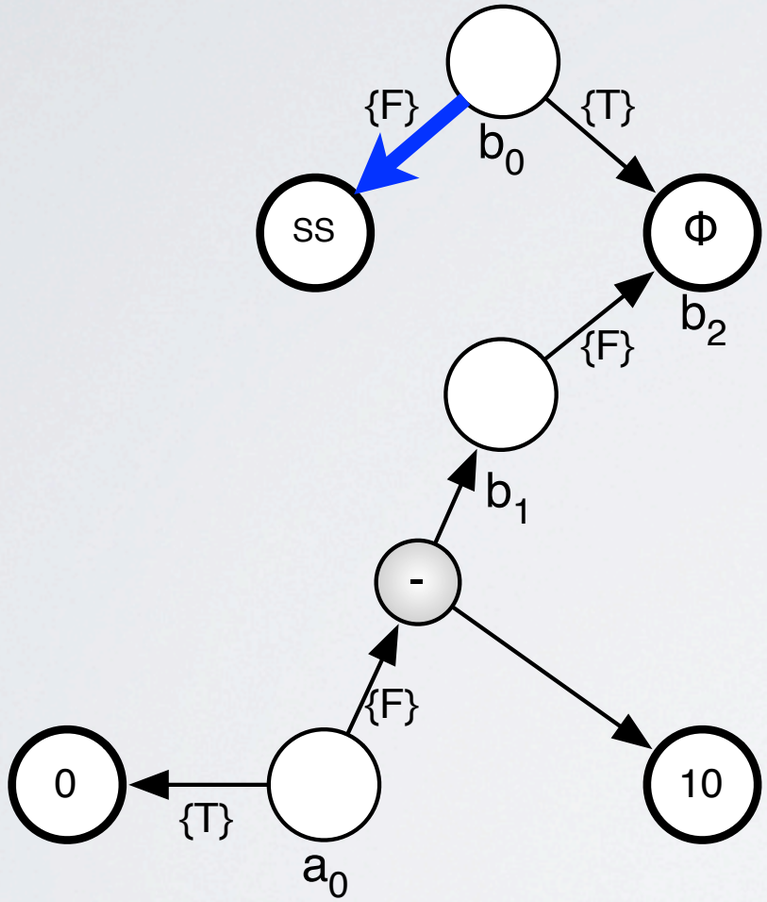


Route Graph

```
void foo_forward()
{
  int trace = 0;
  if (a == 0)
  {
    trace |= 1;
    a = 1;
  }
  else
  {
    store(b);
    b = a + 10;
    a = 0;
  }
  store(trace);
}
```

Red: Path recording. Blue: State saving.

Generate The Reverse Program



Route Graph

```
void foo_reverse()
{
   int trace;
   restore(trace);
   if ((trace & 1) == 1)
      a = 0;
   else
   {
      a = b - 10;
      restore(b);
   }
}
```

Generated Forward And Reverse Functions

```
void foo()
{
  if (a == 0)
    a = 1;
  else
  {
    b = a + 10;
    a = 0;
  }
}
```

```
void foo_forward()
{
  int trace = 0;
  if (a == 0)
  {
    trace |= 1;
    a = 1;
  }
  else
  {
    store(b);
    b = a + 10;
    a = 0;
  }
  store(trace);
}
```

```
void foo_reverse()
{
  int trace;
  restore(trace);
  if ((trace & 1) == 1)
    a = 0;
  else
  {
    a = b - 10;
    restore(b);
  }
}
```

Red: Path recording. Blue: State saving.

Dealing With Loops

- In the CFG, each loop region is reduced into a single node, making the program loop-free.
- The VSG doesn't contain any value defined in the loop, but contains the input and output values of each loop.

```
void foo()
{
   for (int i = 0; i < N; ++i)
       a += i;
}
```

Conclusion

- Pros:
  - Can deal with reversible and irreversible programs.
  - Generates efficient forward and reverse programs.
  - Can handle arbitrary control flows.
- Cons:
  - Forces instrumentations in the forward program: recording control flows.
  - Needs extensions to SSA form to deal with aliasing, arrays, etc..

Current And Future Work

- Loops (done).
- Arrays (currently working on).
- High level information assistance.

- Our compiler: Backstroke. (Based on ROSE compiler: www.rosecompiler.org)

Thank You!