VisualAI:
A Framework for Interactive Abstract Interpretation

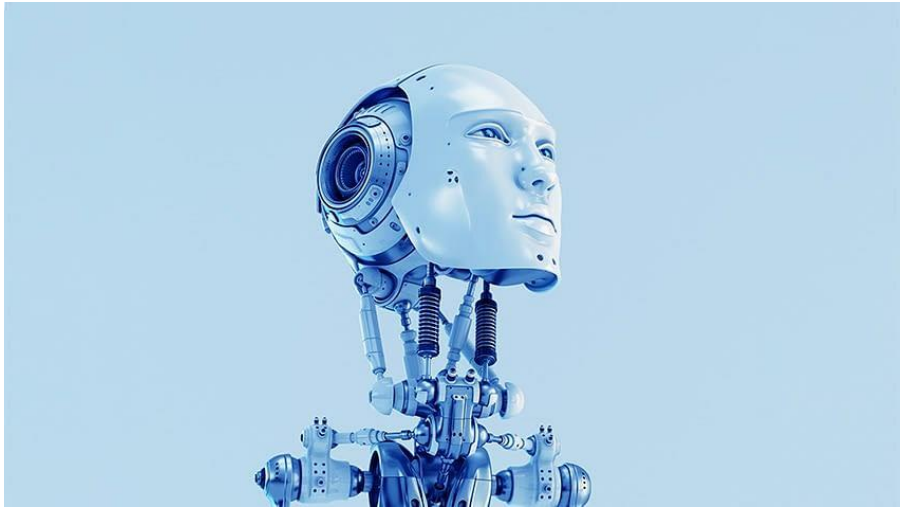
Jung Hyun Kim

SoftSec Lab., KAIST

IS661 Spring, 2024

AI?

AI?



Artificial Intelligence
(AI)

AI?

rtificial Intelligenc
(AI)

AI (Abstract Interpretation)

AI (Abstract Interpretation)

- Abstract Interpretation[1]: A formal method to evaluate a program.

[1] Cousot, Patrick, and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints." Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. 1977.

AI (Abstract Interpretation)

- Abstract Interpretation[1]: A formal method to evaluate a program.
- Allows us to reason about all possible executions of a program *even before running the program.*

[1] Cousot, Patrick, and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints." Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. 1977.

AI (Abstract Interpretation)

- Abstract Interpretation[1]: A formal method to evaluate a program.
- Allows us to reason about all possible executions of a program *even before running the program.*
- Widely-used for static program analysis: *type analysis, taint analysis, compiler optimization*, etc.

[1] Cousot, Patrick, and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints." Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. 1977.

AI (Abstract Interpretation)

- Abstract Interpretation[1]: A formal method to evaluate a program.
- Allows us to reason about all possible executions of a program *even before running the program.*
- Widely-used for static program analysis: *type analysis, taint analysis,*

AI is important in program analysis.

[1] Cousot, Patrick, and Radhia Cousot. "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints." Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. 1977.

Difficulties In Developing AI

Difficulties In Developing AI

1. It is hard to inspect *the inside of an AI process*.

Difficulties In Developing AI

1. It is hard to inspect *the inside of an AI process*.

Difficulties In Developing AI

1. It is hard to inspect *the inside of an AI process*.

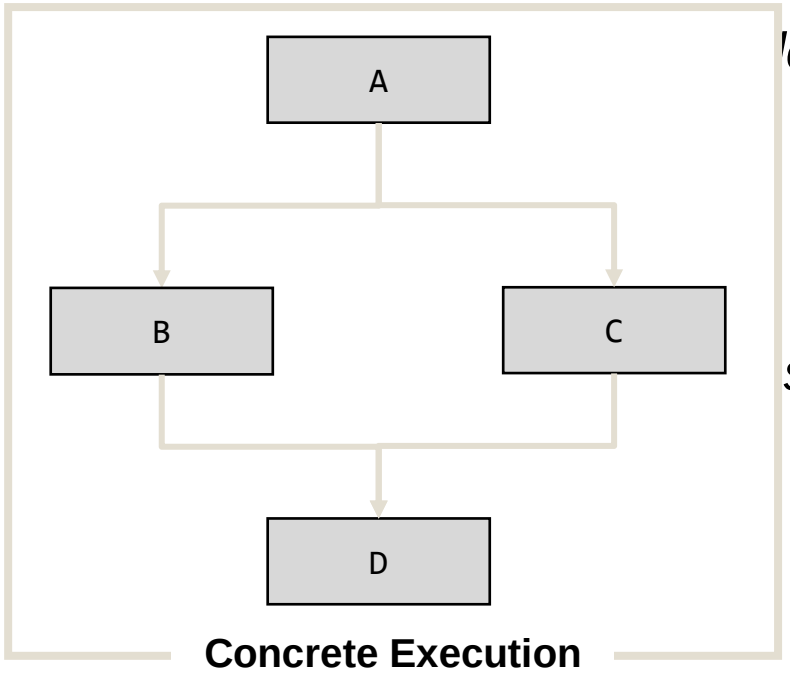
2. It is hard to follow *how states are propagated*.

Difficulties In Developing AI

1. It is hard to inspect *the inside of an AI process*.

2. It is hard to follow *how states are propagated*.

Difficulties In Visualizing AI



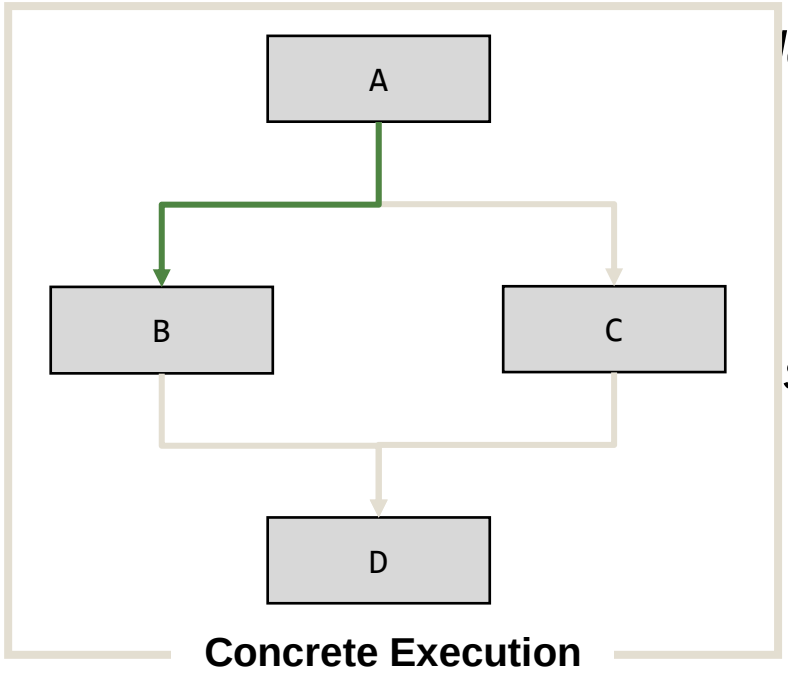
le of an AI process.

Lack of *Visualization*

s are propagated.

Lack of *Interaction*

Difficulties In Visualizing AI



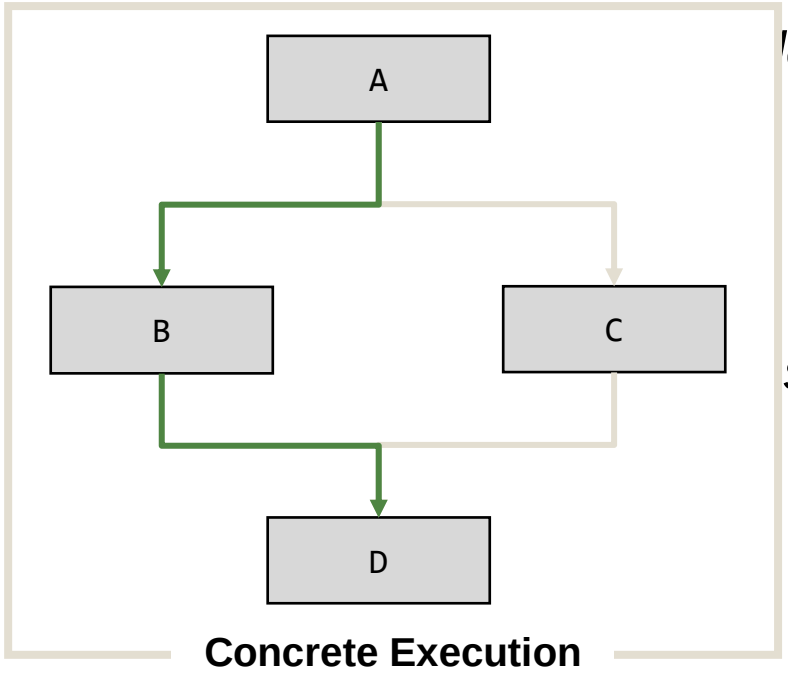
le of an AI process.

Lack of *Visualization*

s are propagated.

Lack of *Interaction*

Difficulties In Visualizing AI



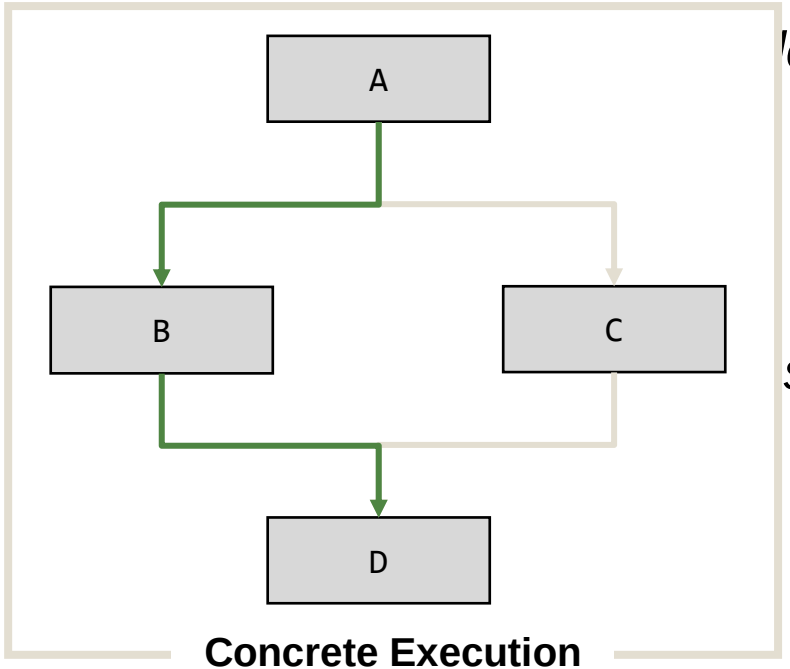
le of an AI process.

Lack of *Visualization*

s are propagated.

Lack of *Interaction*

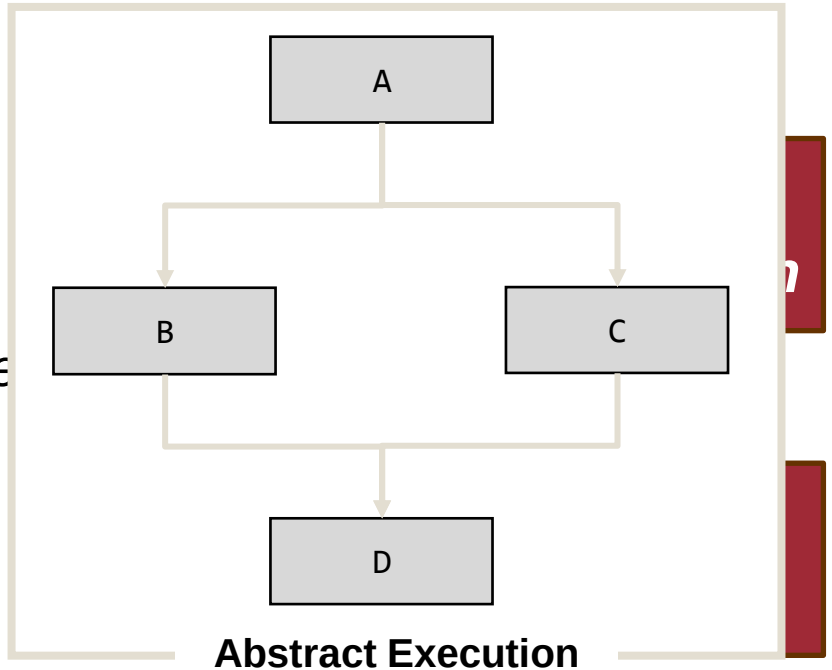
Difficulties In Visualizing AI



Concrete Execution

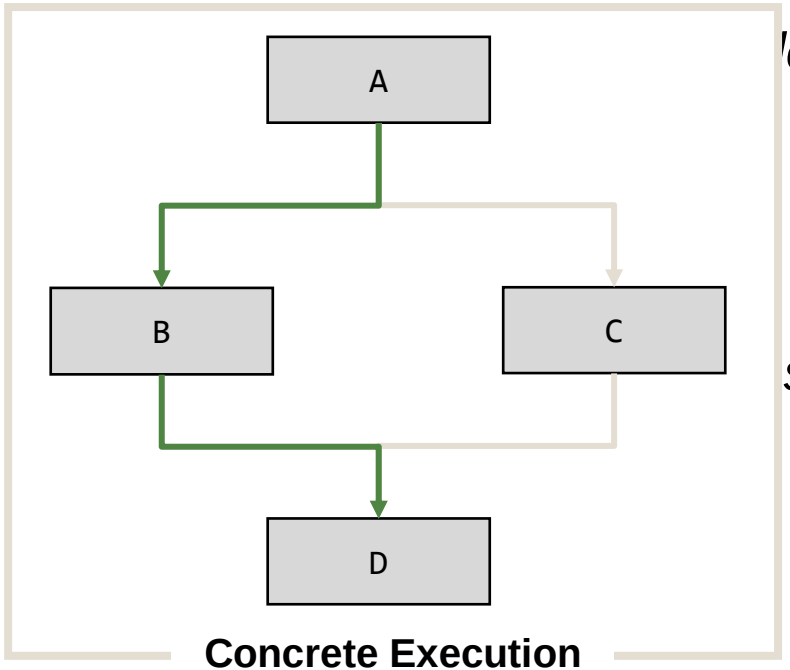
le of

s are

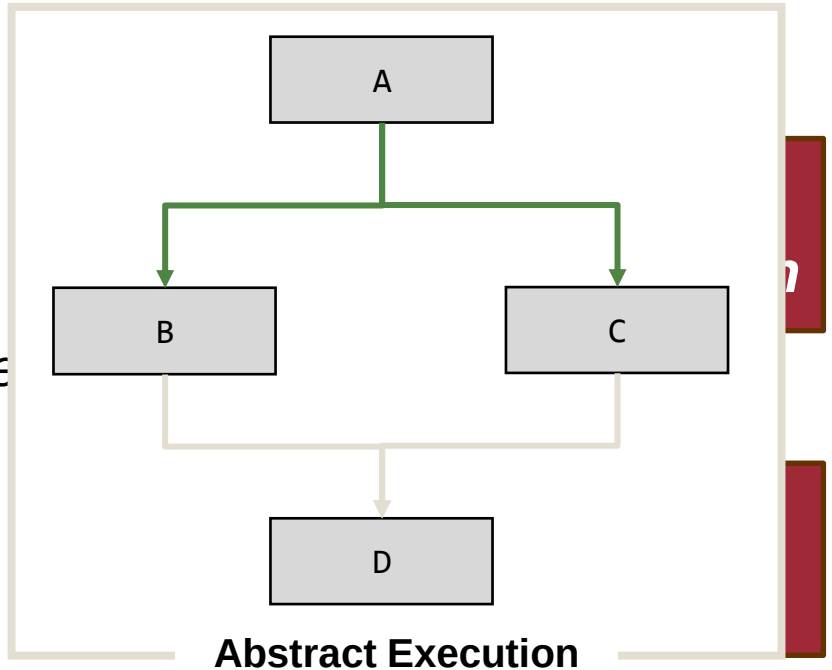


Abstract Execution

Difficulties In Visualizing AI

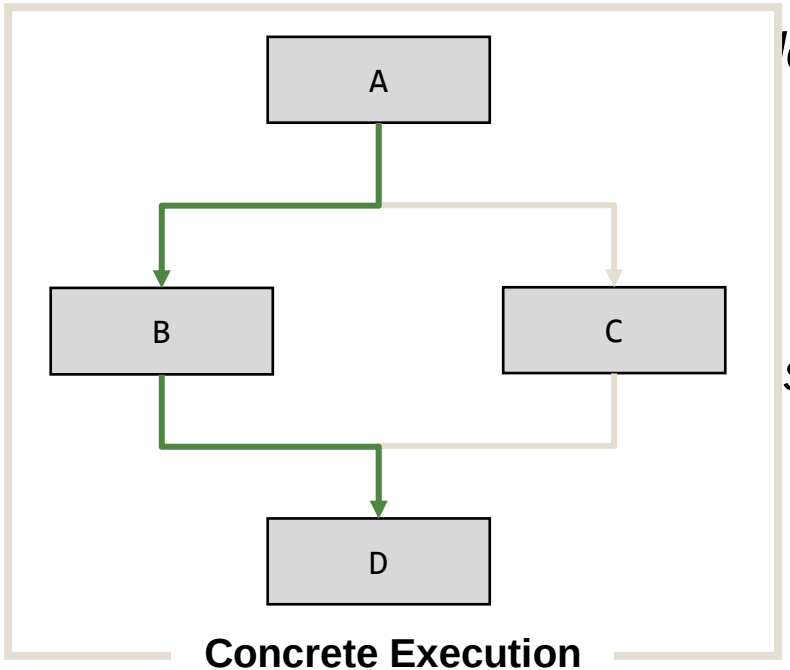


Concrete Execution

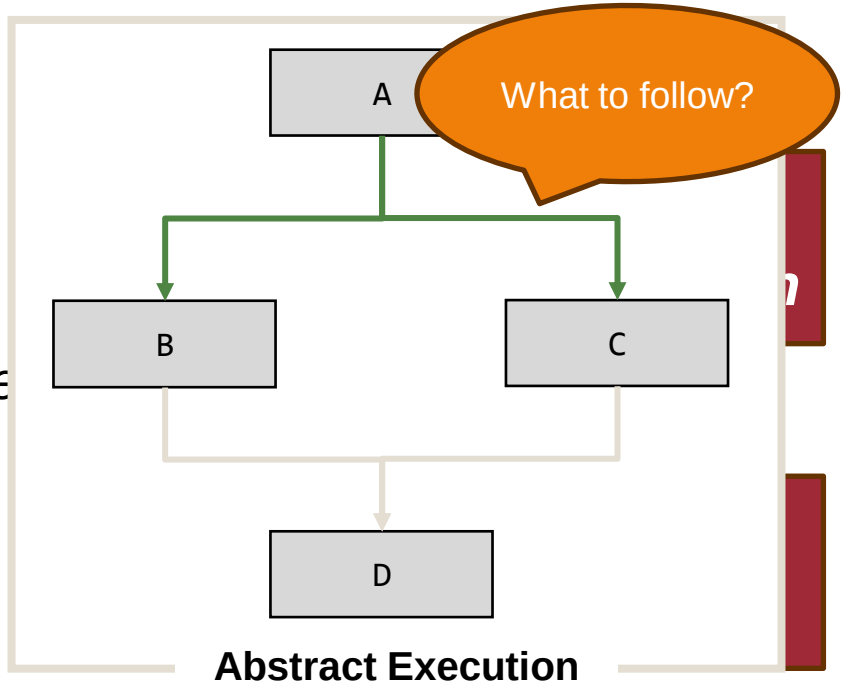


Abstract Execution

Difficulties In Visualizing AI

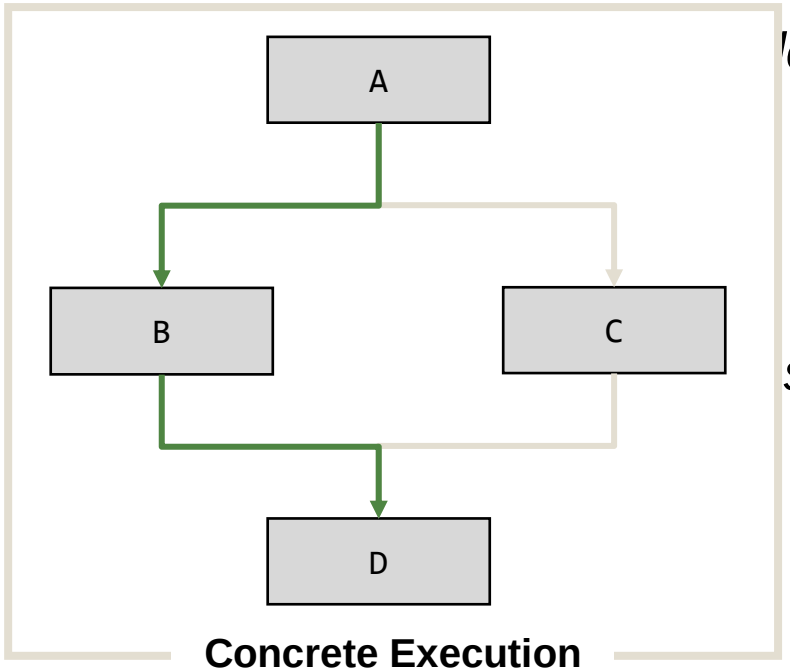


Concrete Execution

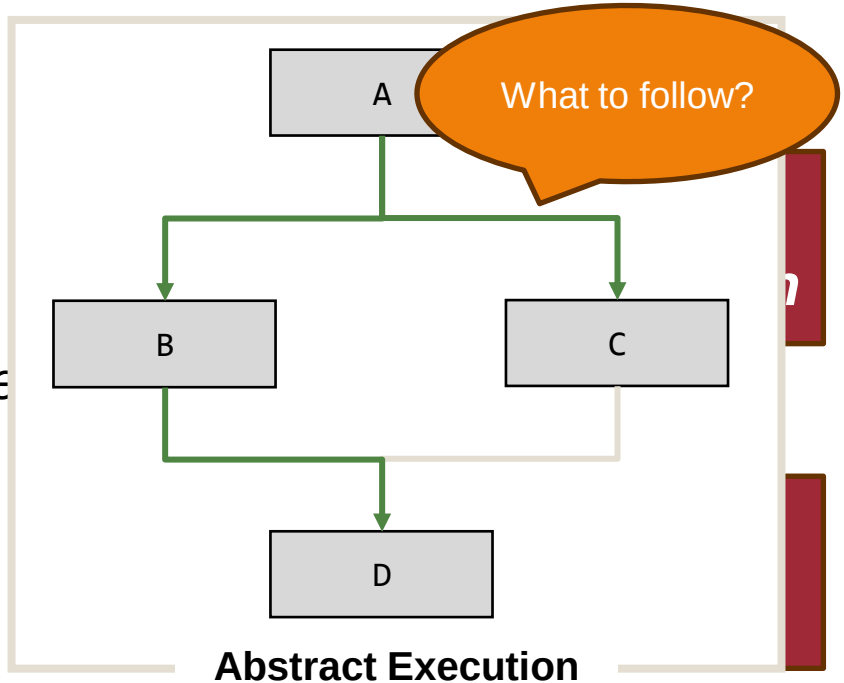


Abstract Execution

Difficulties In Visualizing AI

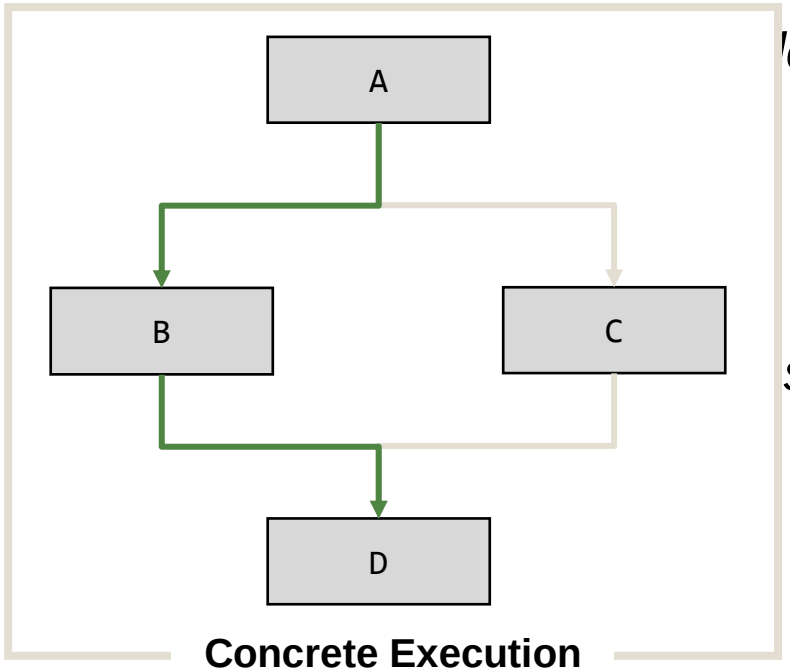


Concrete Execution

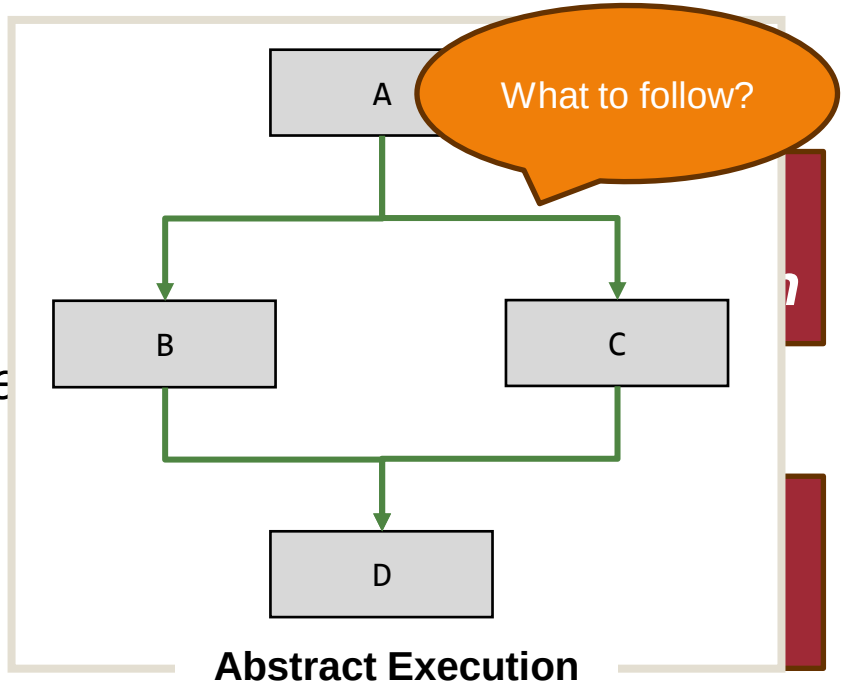


Abstract Execution

Difficulties In Visualizing AI

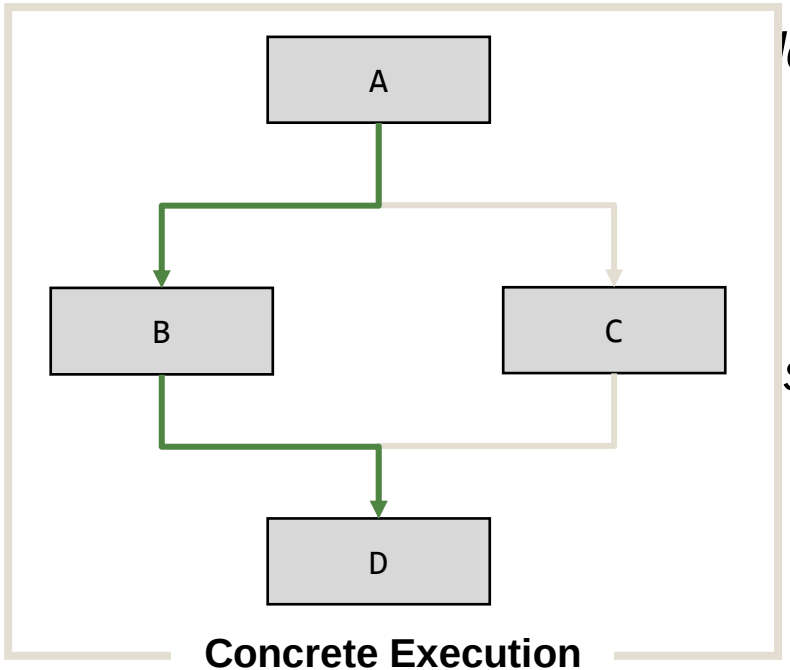


Concrete Execution

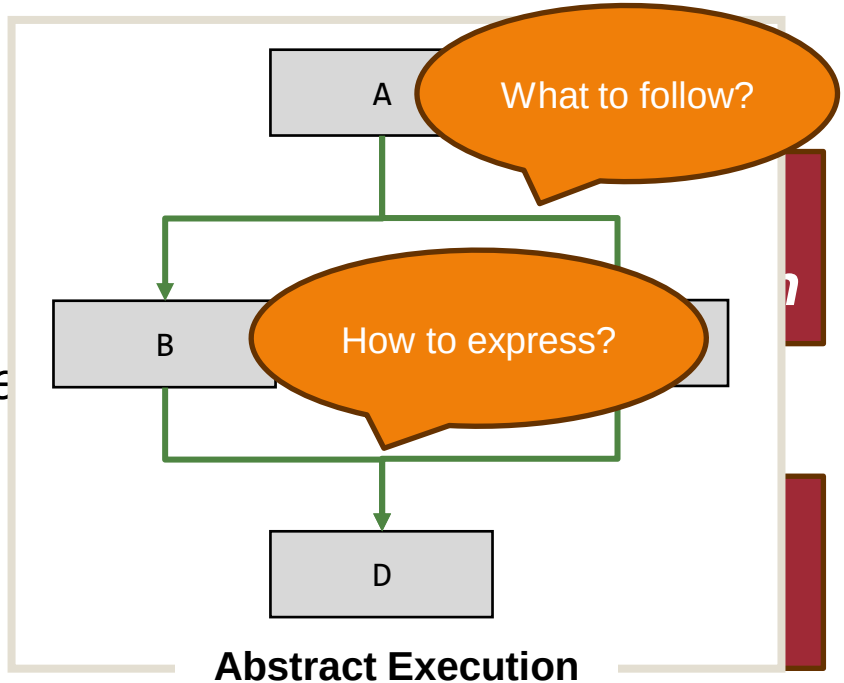


Abstract Execution

Difficulties In Visualizing AI

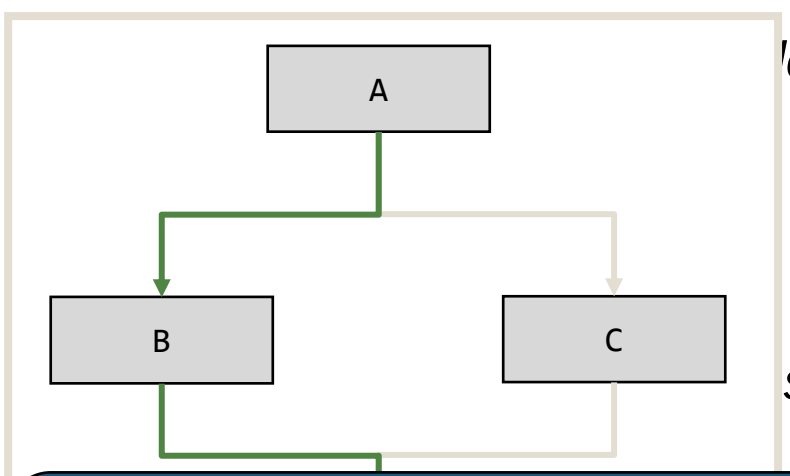


Concrete Execution



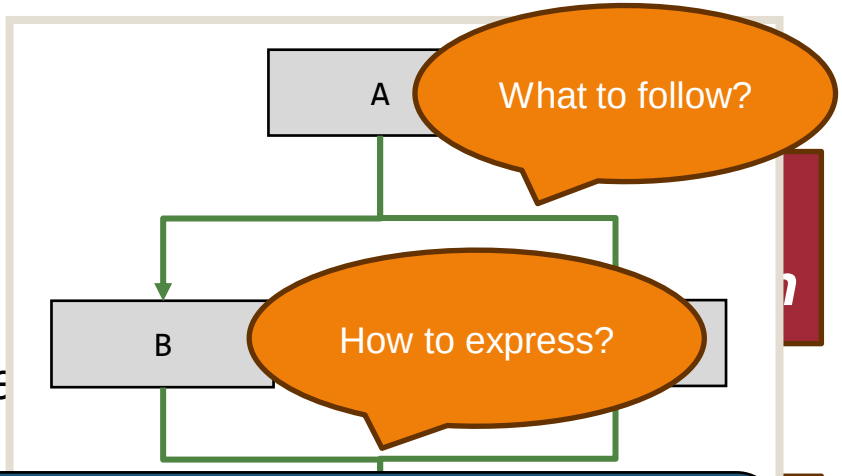
Abstract Execution

Difficulties In Visualizing AI



le of

s are



How could we *visualize* AI in an *interactive* manner?

VisualAI: An Interactive Visualizer for AI

VisualAI: An Interactive Visualizer for AI

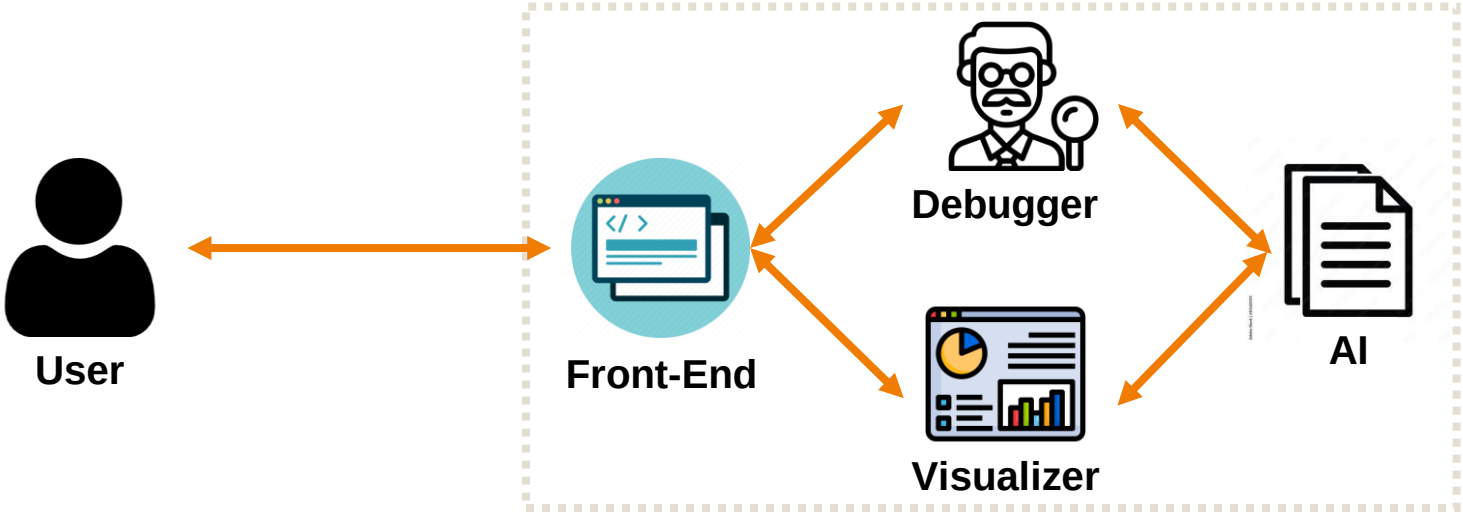
- Provides a *visualization* of an AI process.

VisualAI: An Interactive Visualizer for AI

- Provides a *visualization* of an AI process.
- Enables *interactive debugging* with the visualization.

VisualAI: An Interactive Visualizer for AI

- Provides a *visualization* of an AI process.
- Enables *interactive debugging* with the visualization.



VisualAI: An Interactive Visualizer for AI

- Provides a *visualization* of an AI process.
- Enables *interactive debugging* with the visualization.

Aims helping both students and researchers to understand AI better.

Expected Impact

Expected Impact

- *VisualAI* be a popular platform of AI development.

Expected Impact

- *VisualAI* be a popular platform of AI development.
- *VisualAI* be the first commercial model for AI development.

Expected Impact

- *VisualAI* be a popular platform of AI development.
- *VisualAI* be the first commercial model for AI development.
- Not only for research but also for study could it be useful.

Visualizer

Visualizer

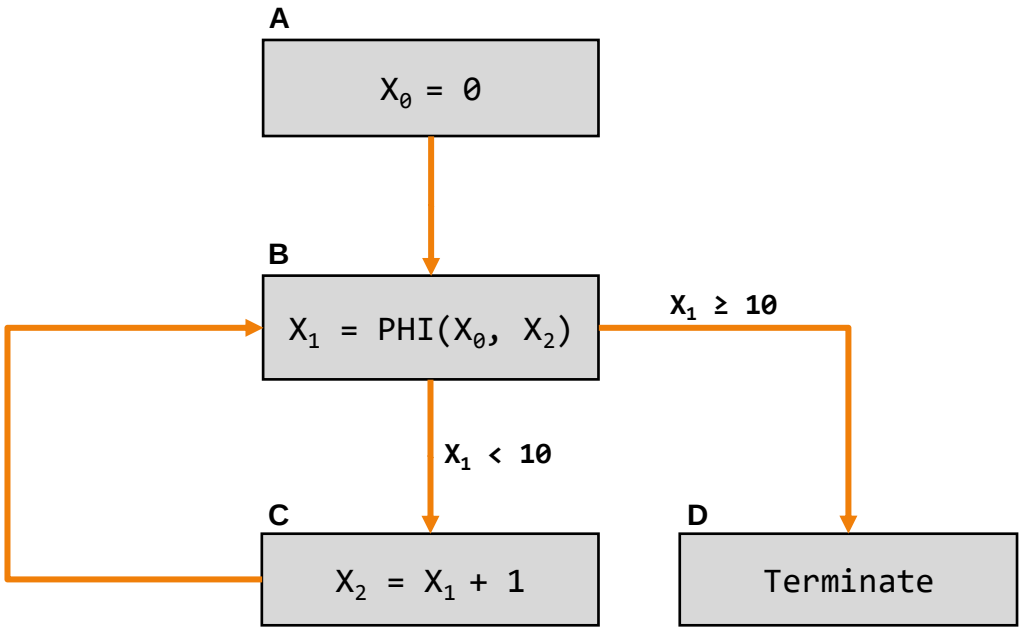
- Visualize each state that an AI process can have by using SSA Graph and State Dependence Graph (SDG).

Visualizer

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code

- V ng
  S



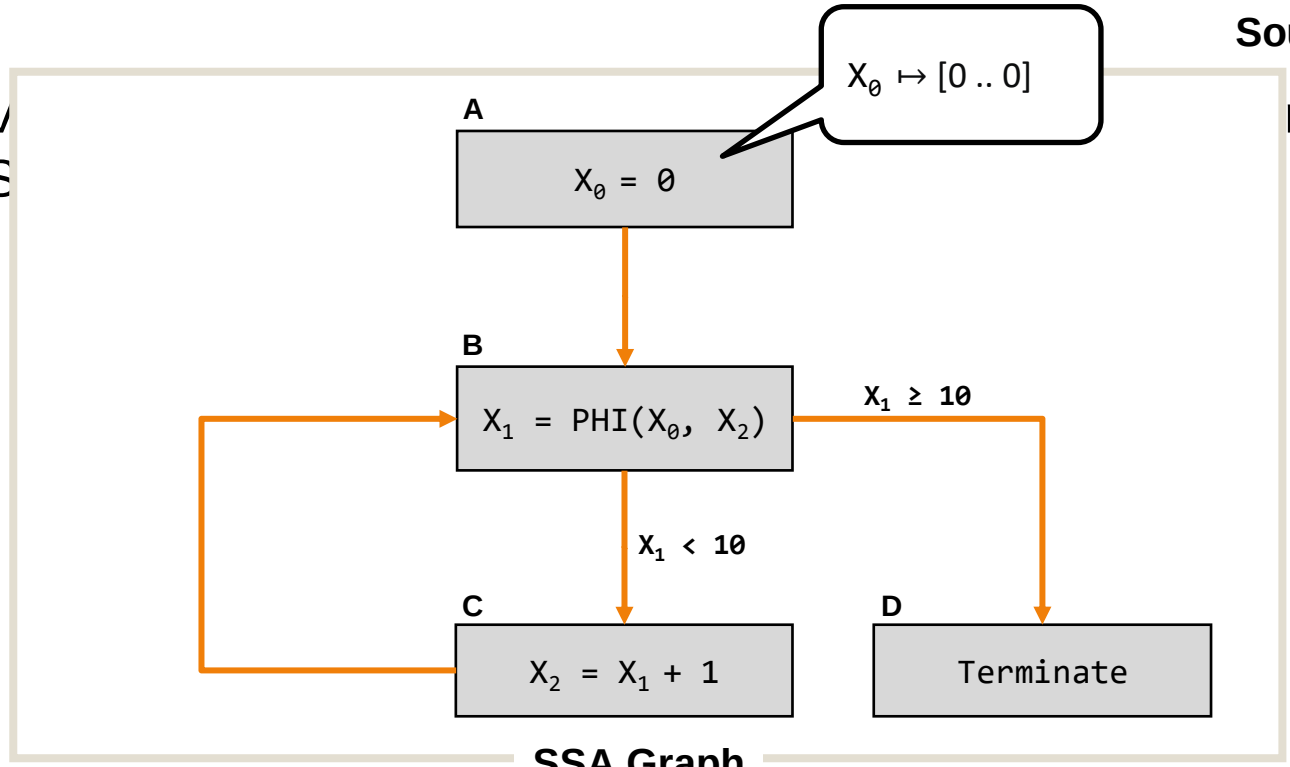
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



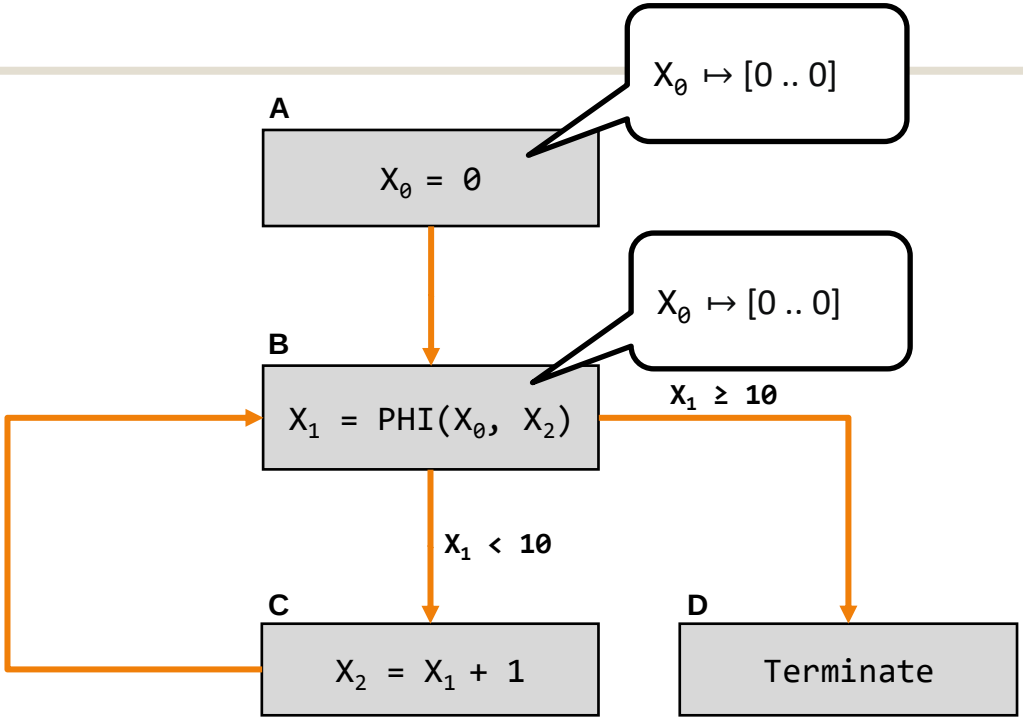
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



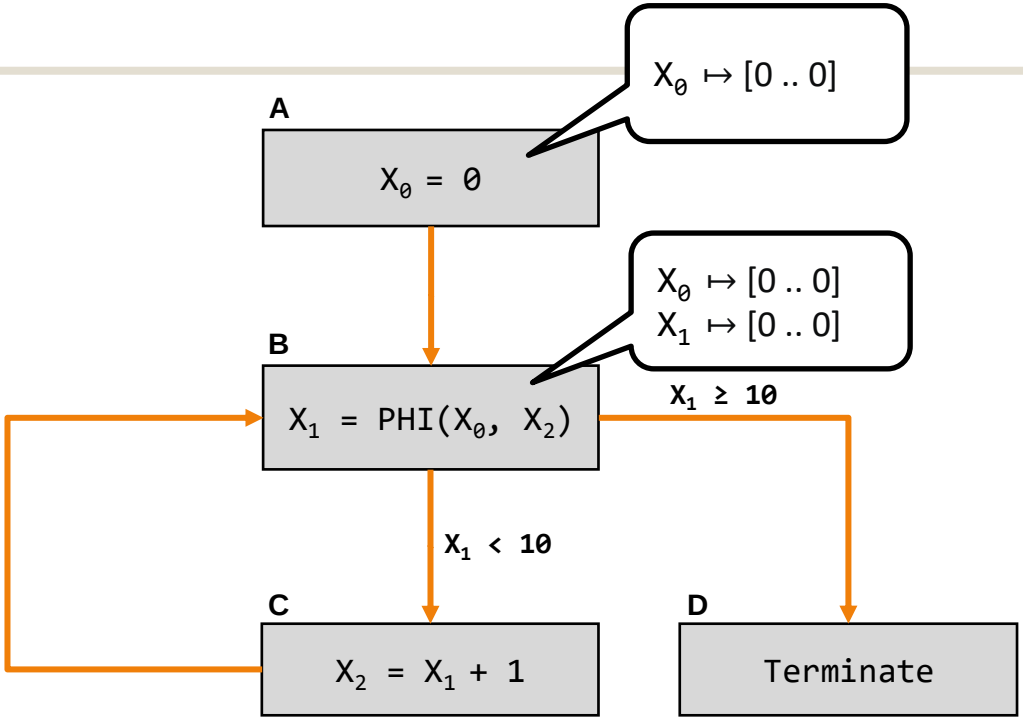
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



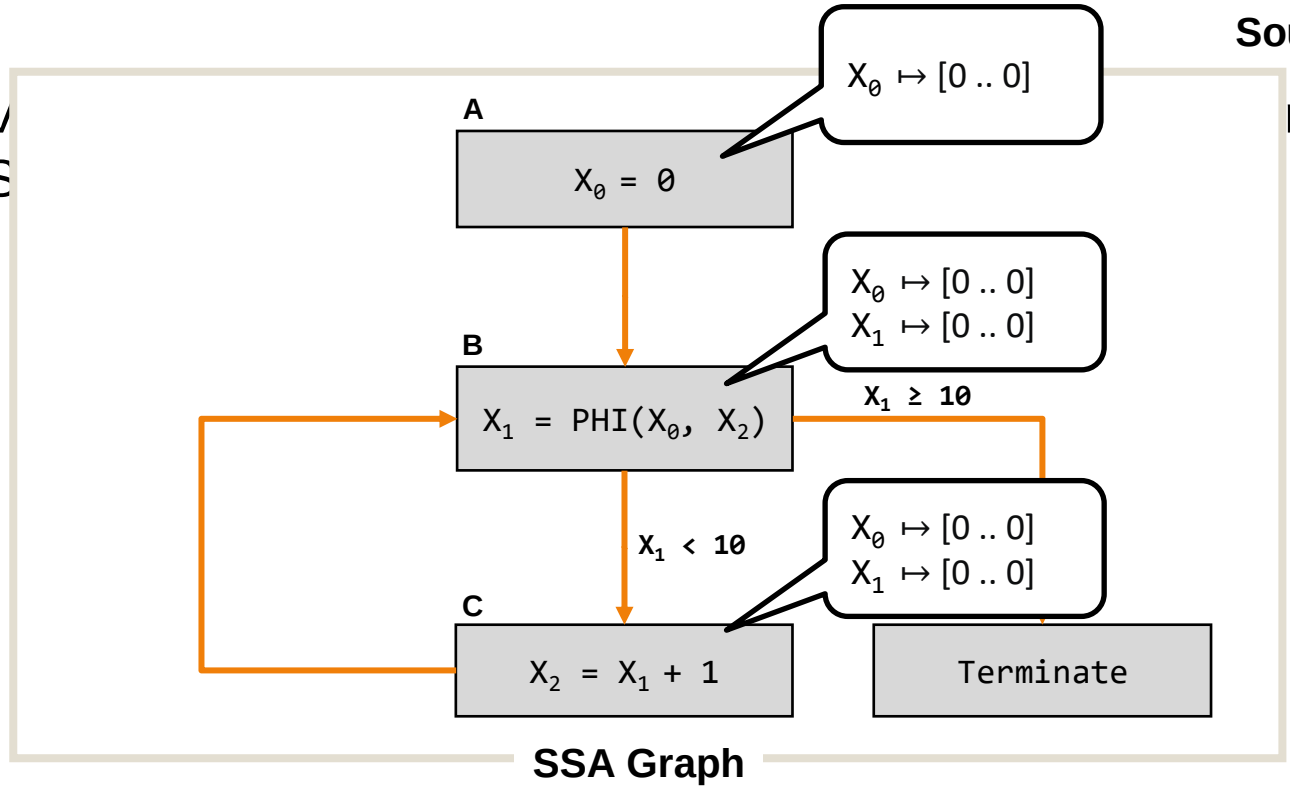
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



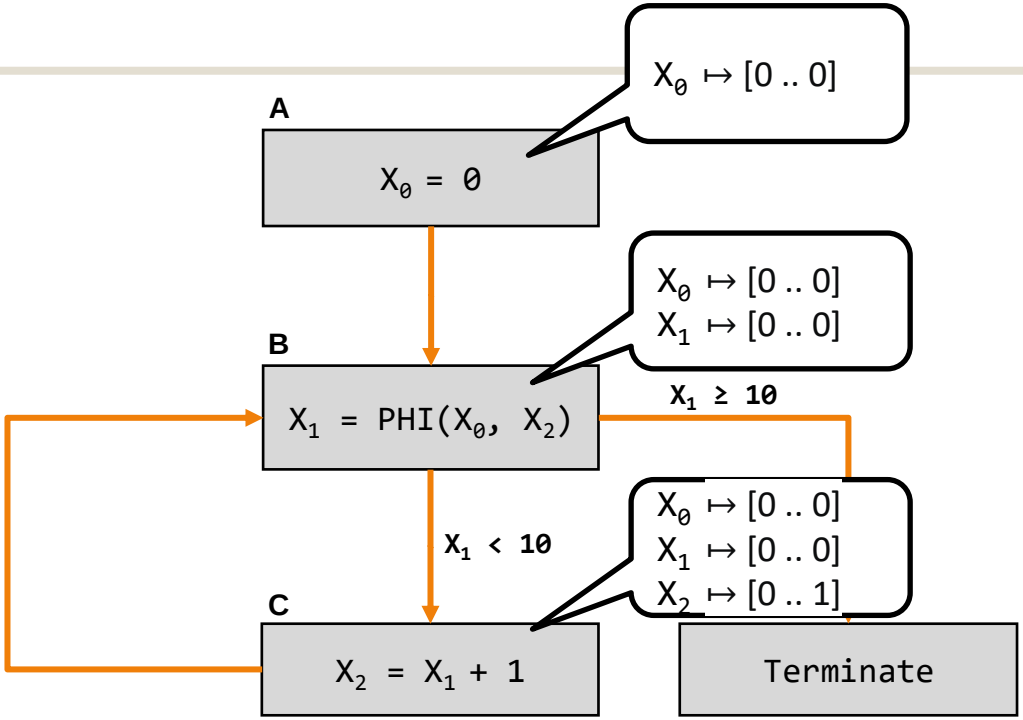
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



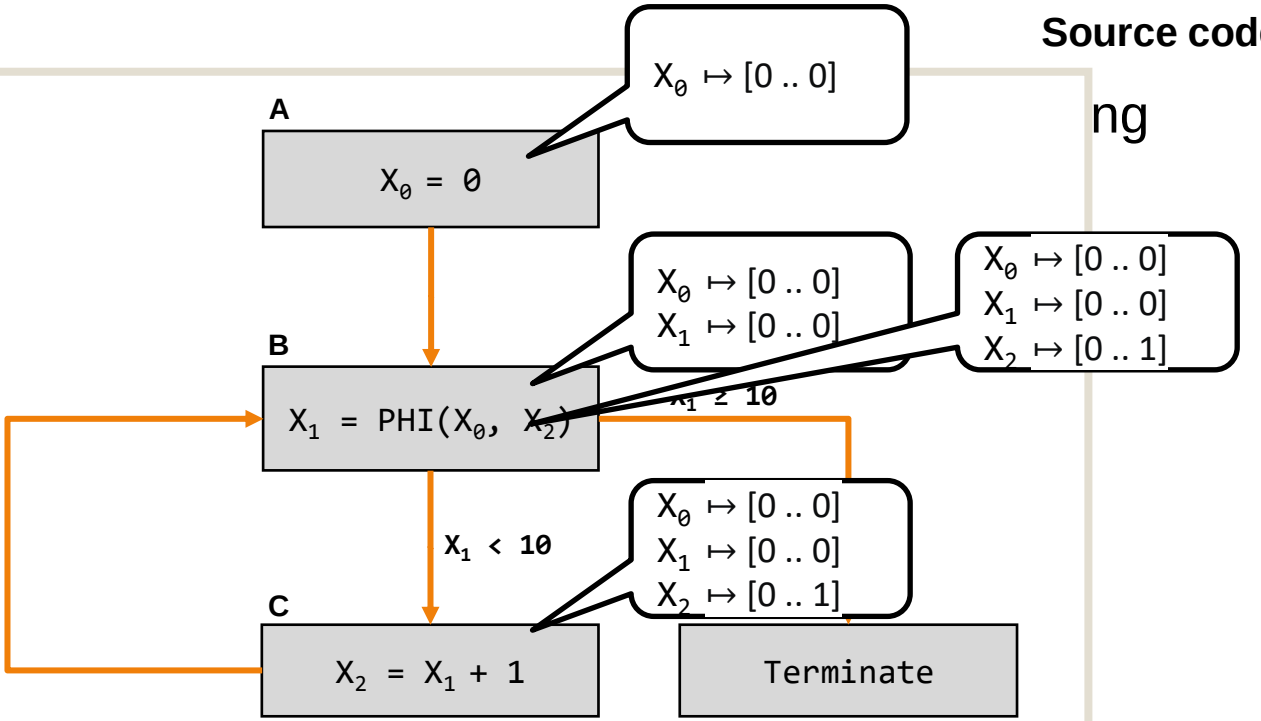
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



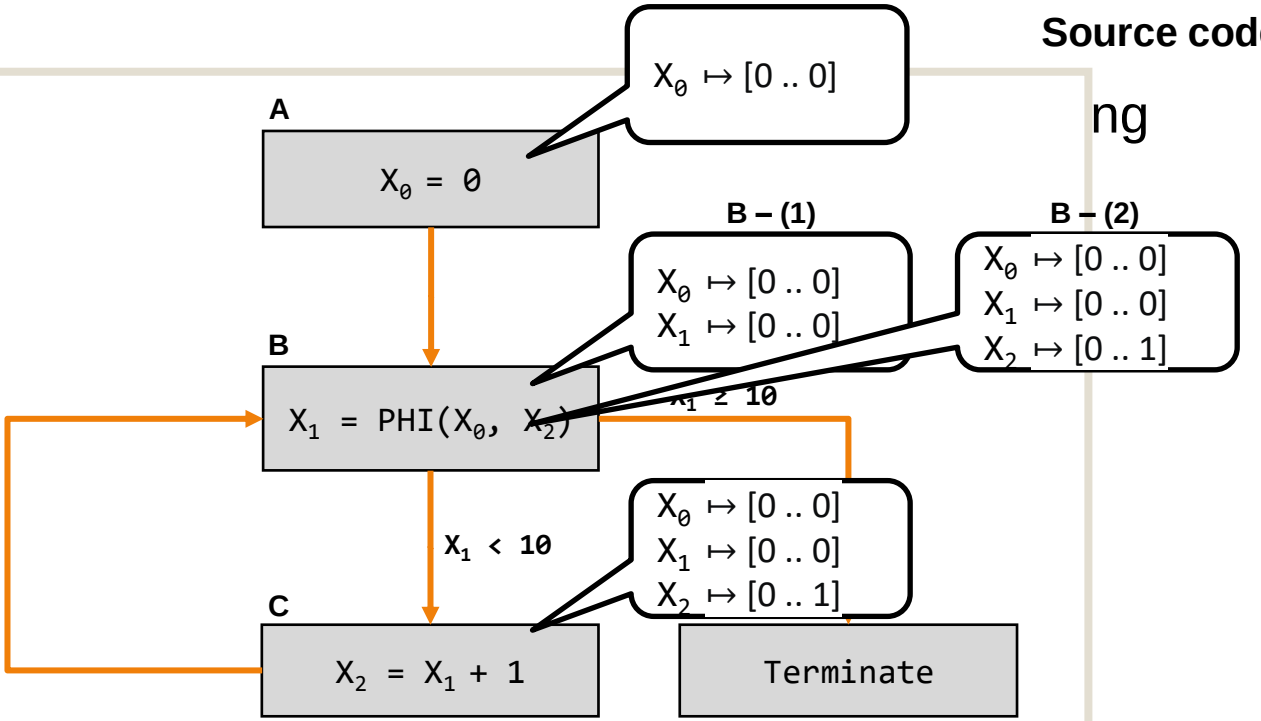
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



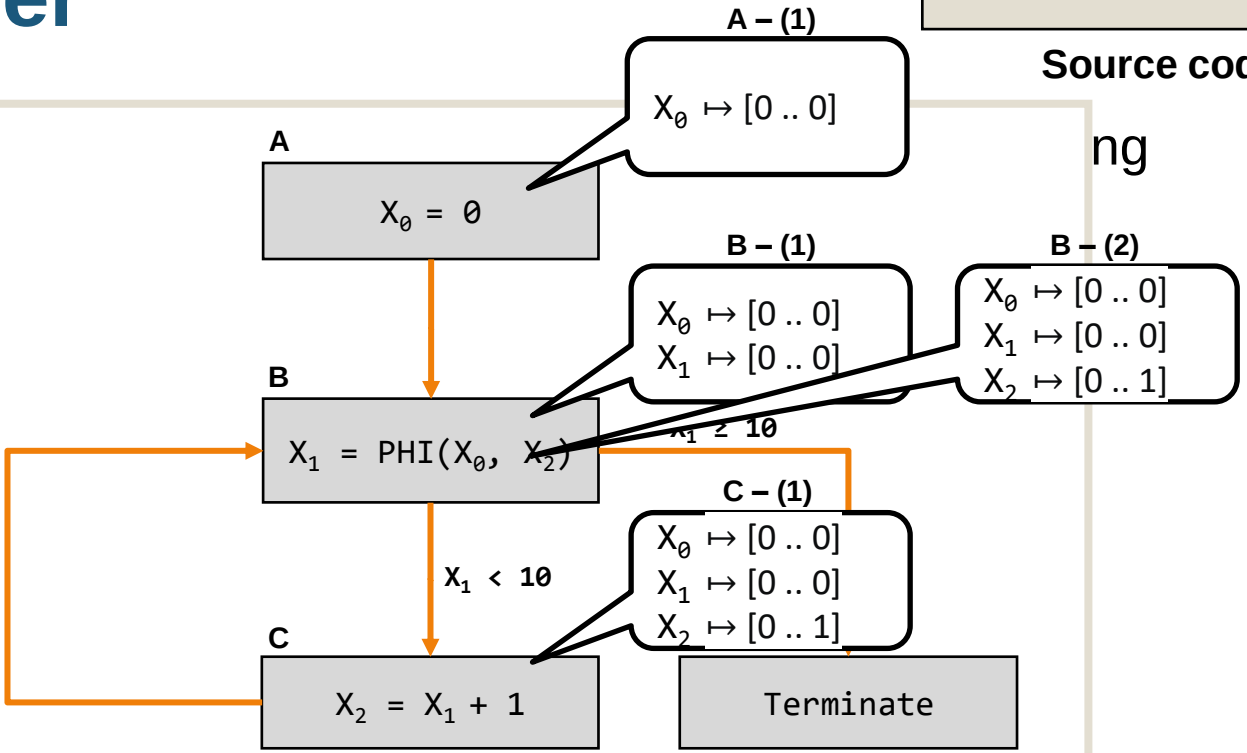
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



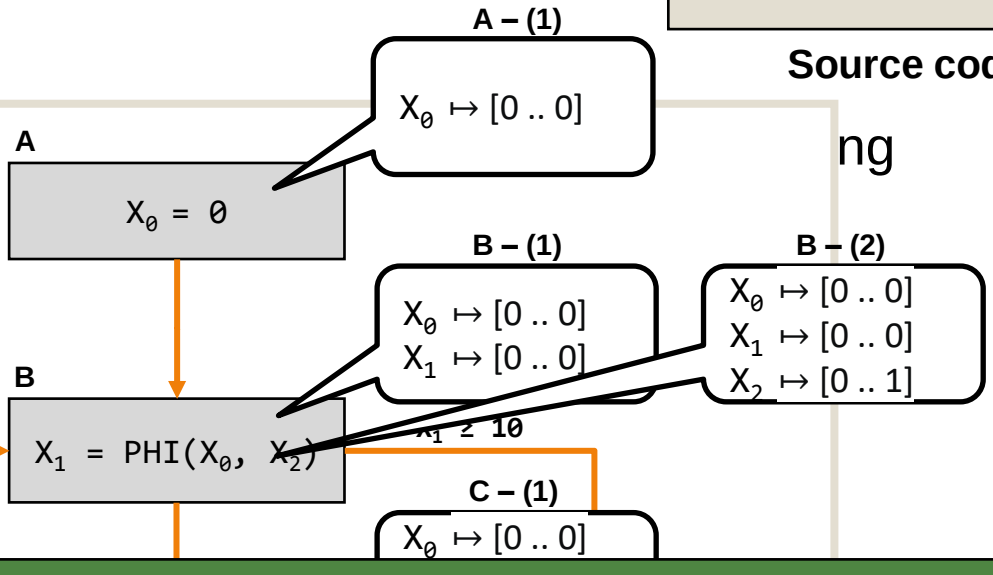
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



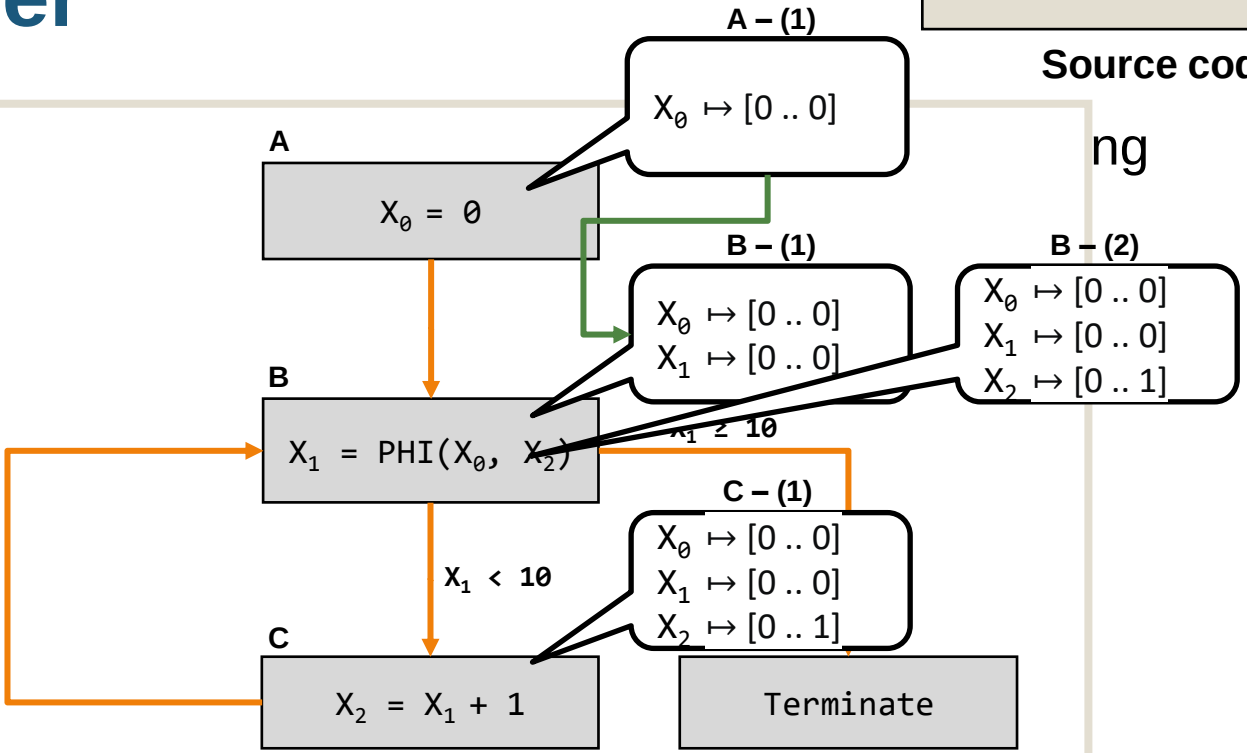
Each state is distinguished via unique identifier.

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



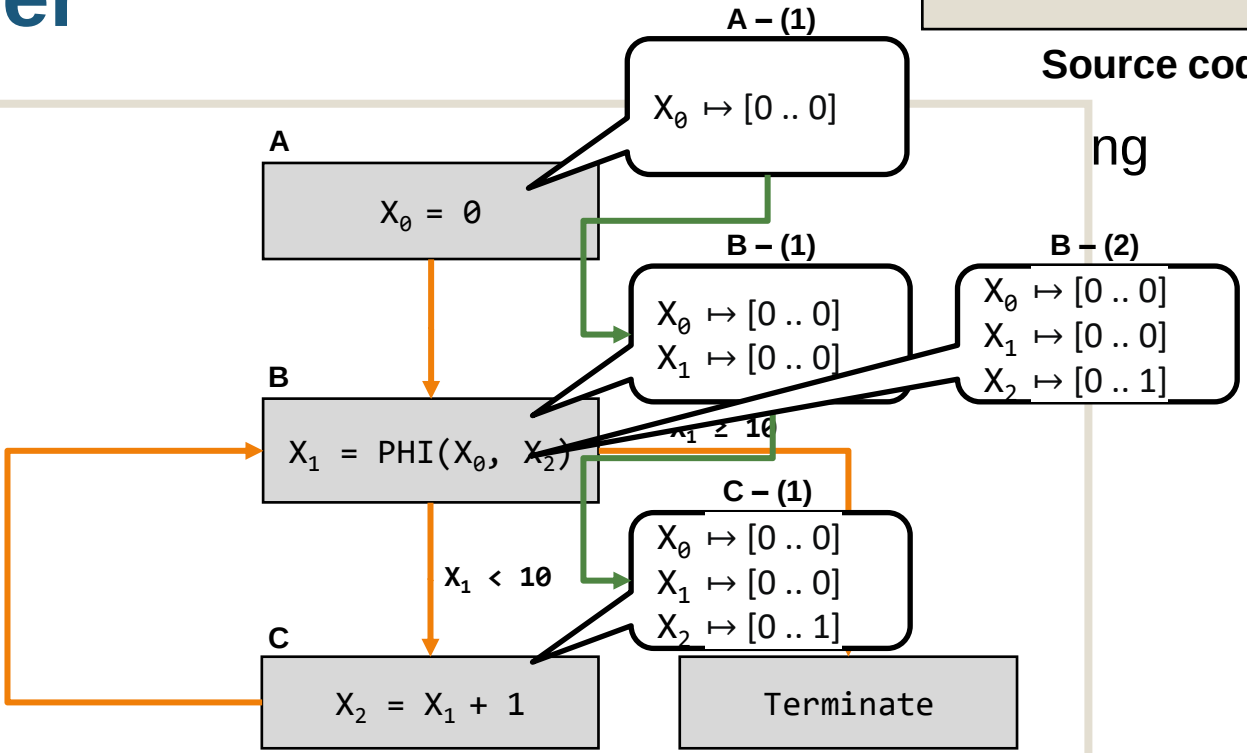
SSA Graph

Visualizer

- V ng
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



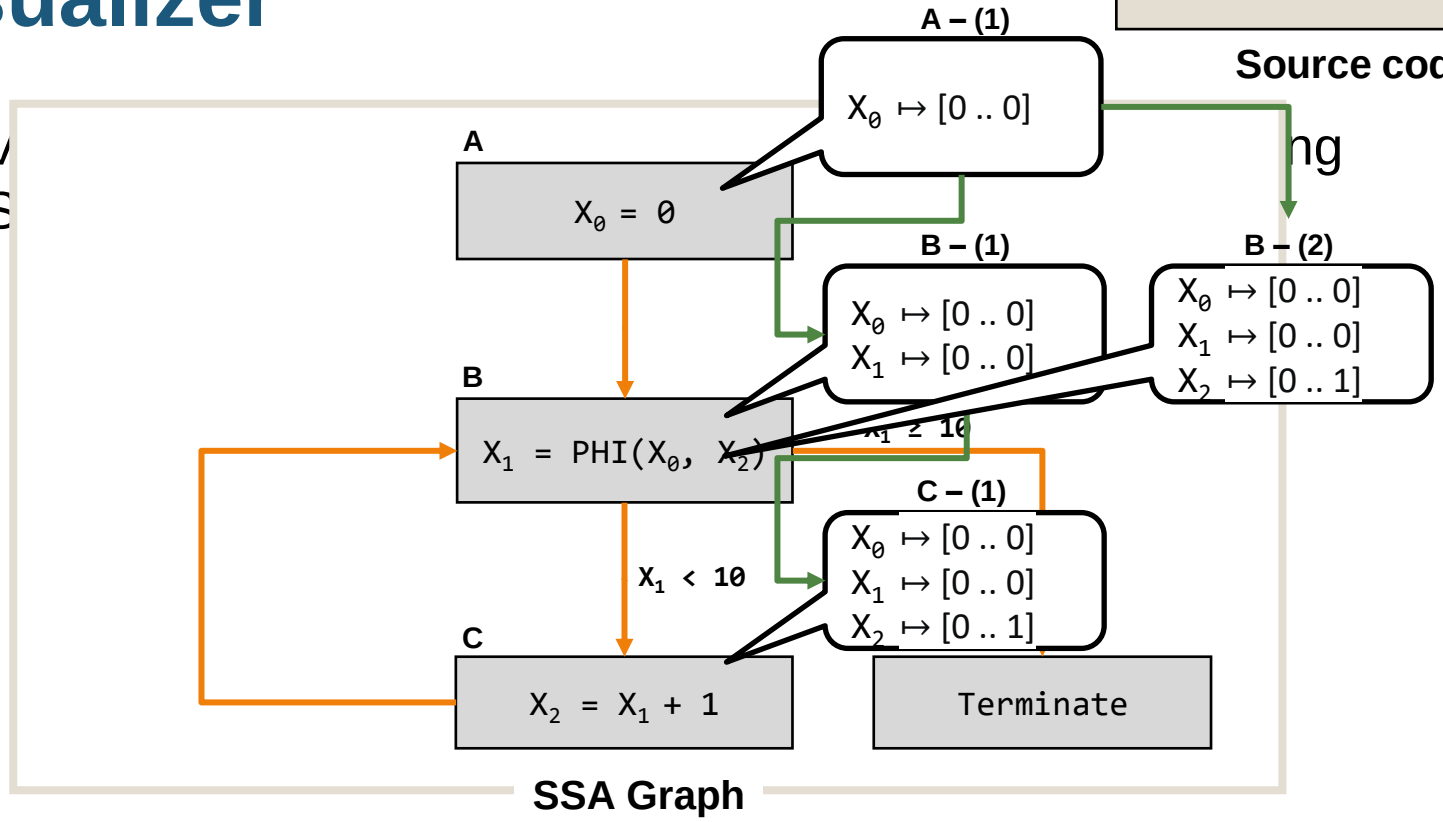
SSA Graph

Visualizer

- V
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



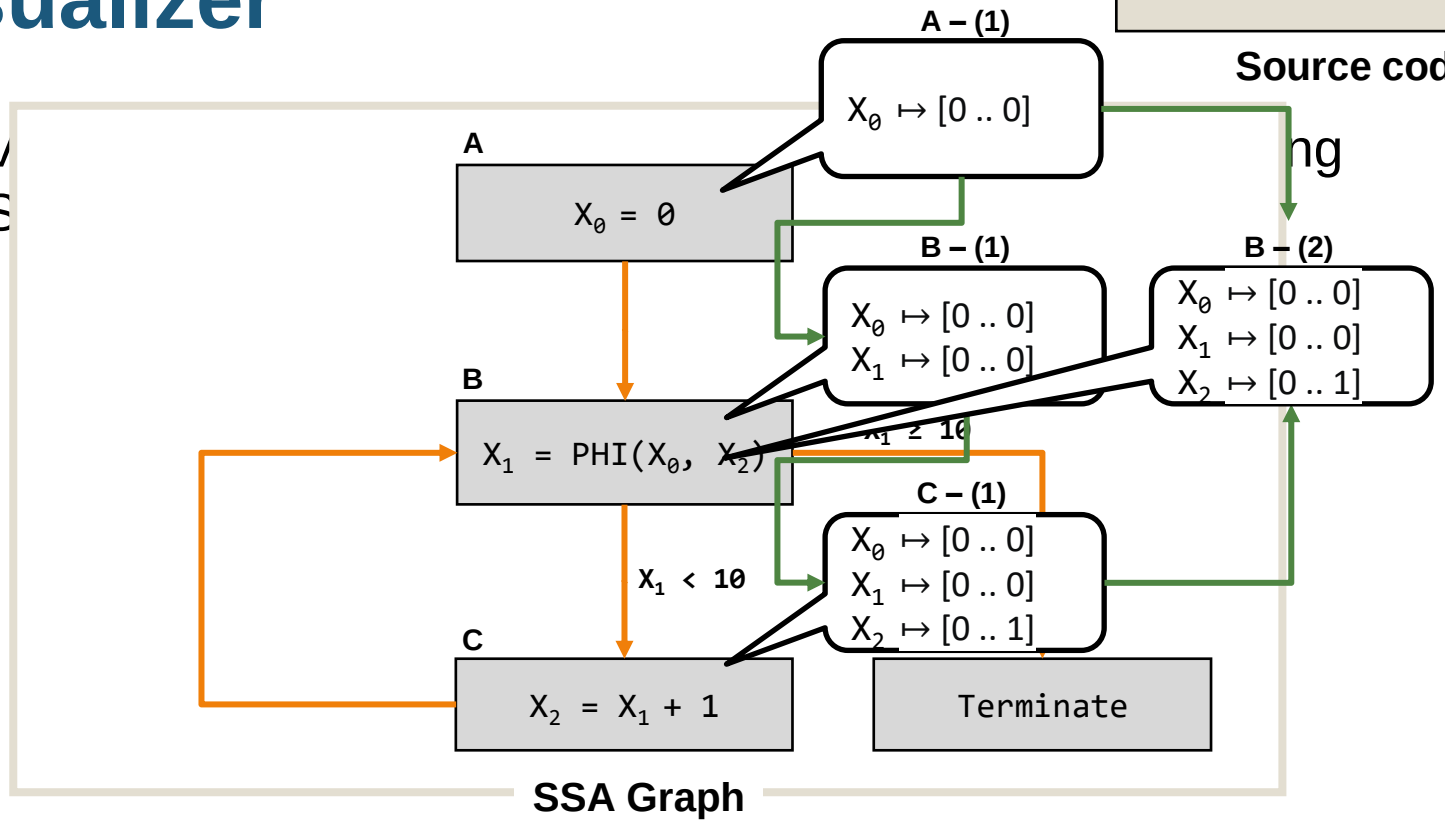
SSA Graph

Visualizer

- V
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



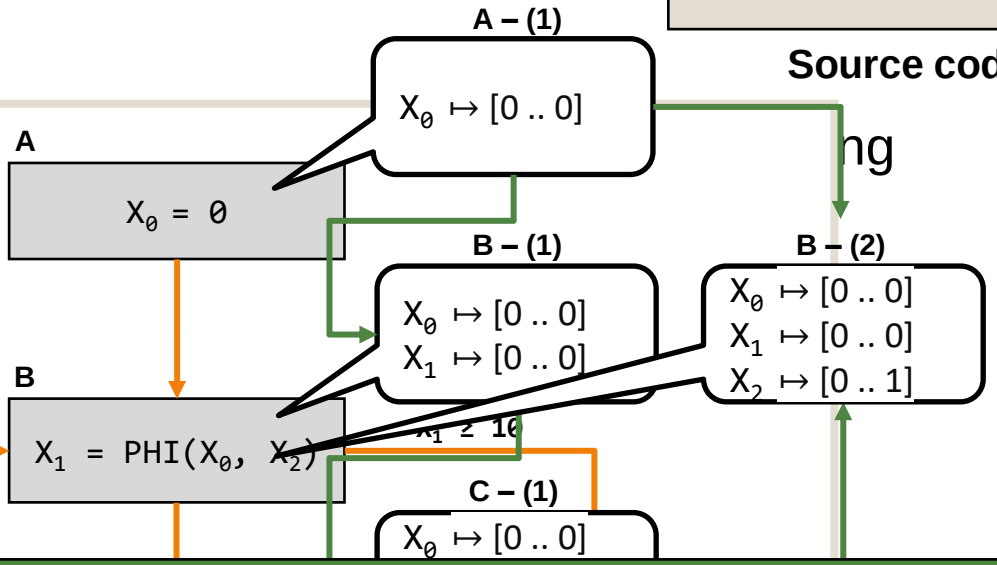
SSA Graph

Visualizer

- V
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

Source code



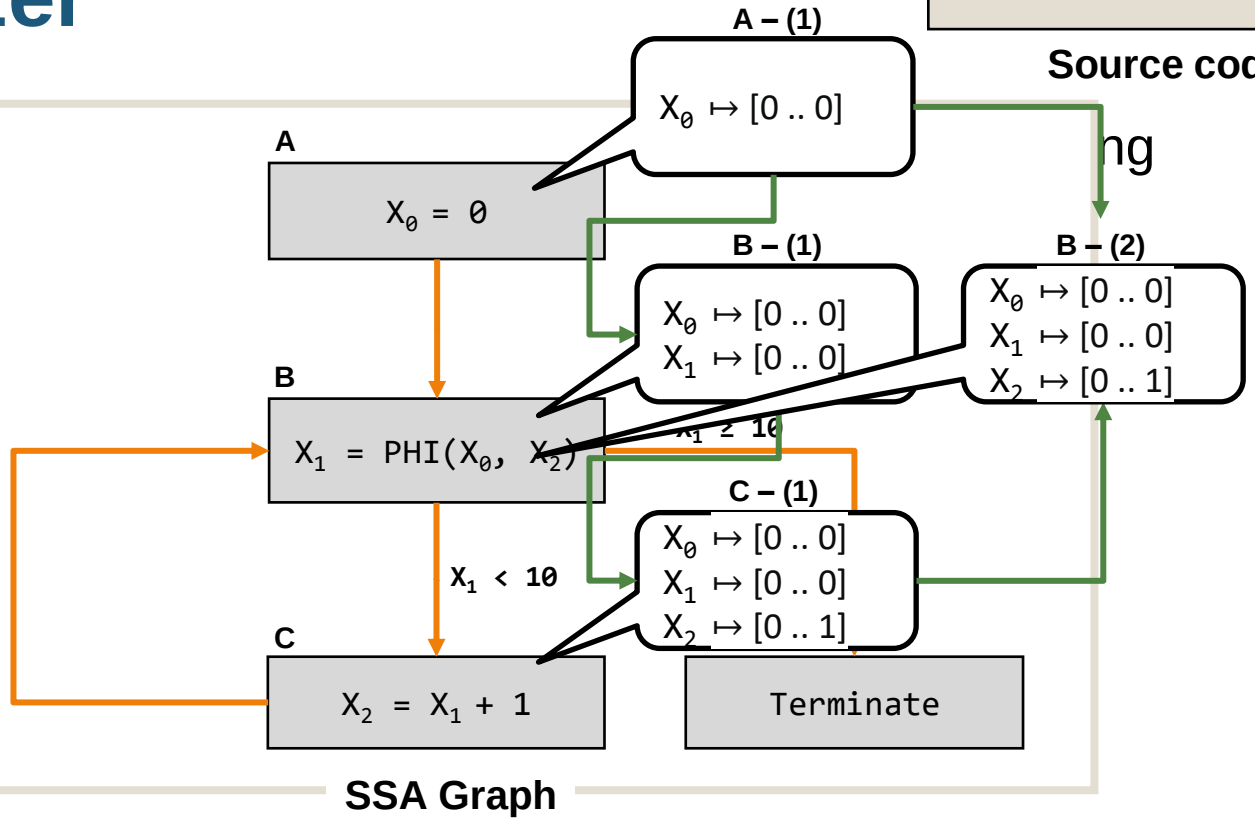
Unique identifiers help clarify the state transitions.

Visualizer

- V
  S

```
void main() {
  for(int i = 0; i < 10; i++);
}
```

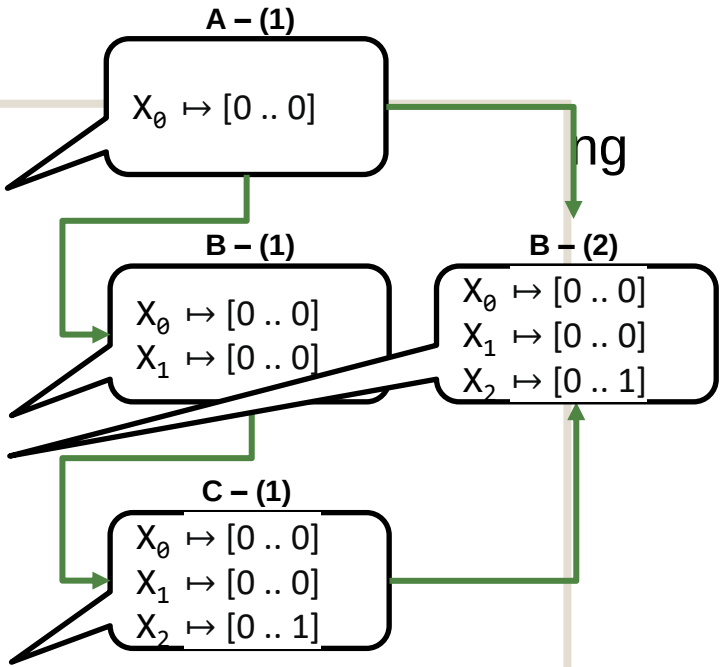
Source code



SSA Graph

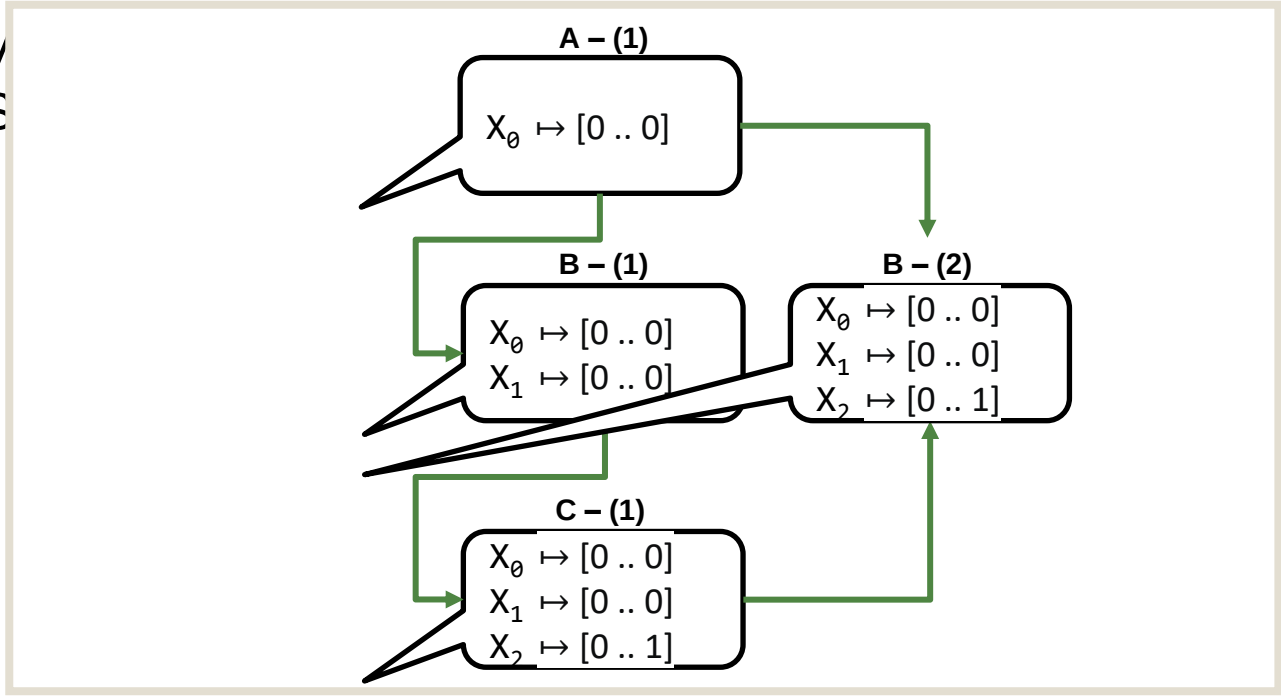
Visualizer

- V
  S



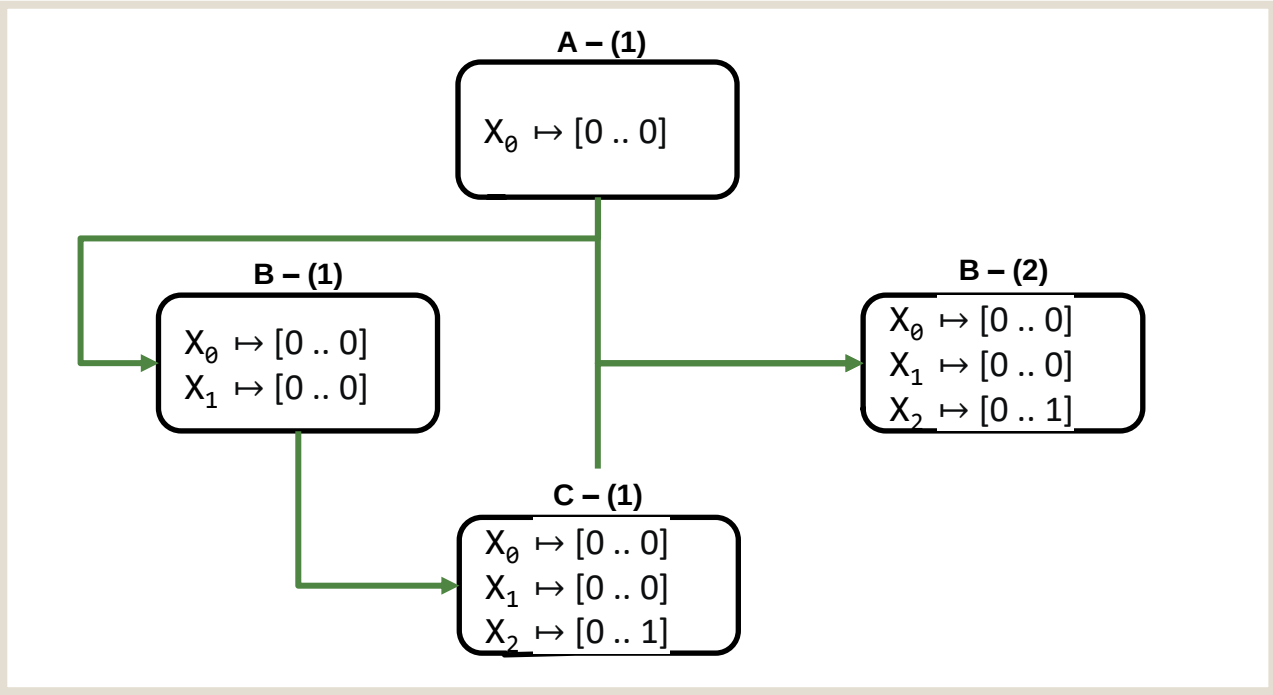
Visualizer

- V ng
  S



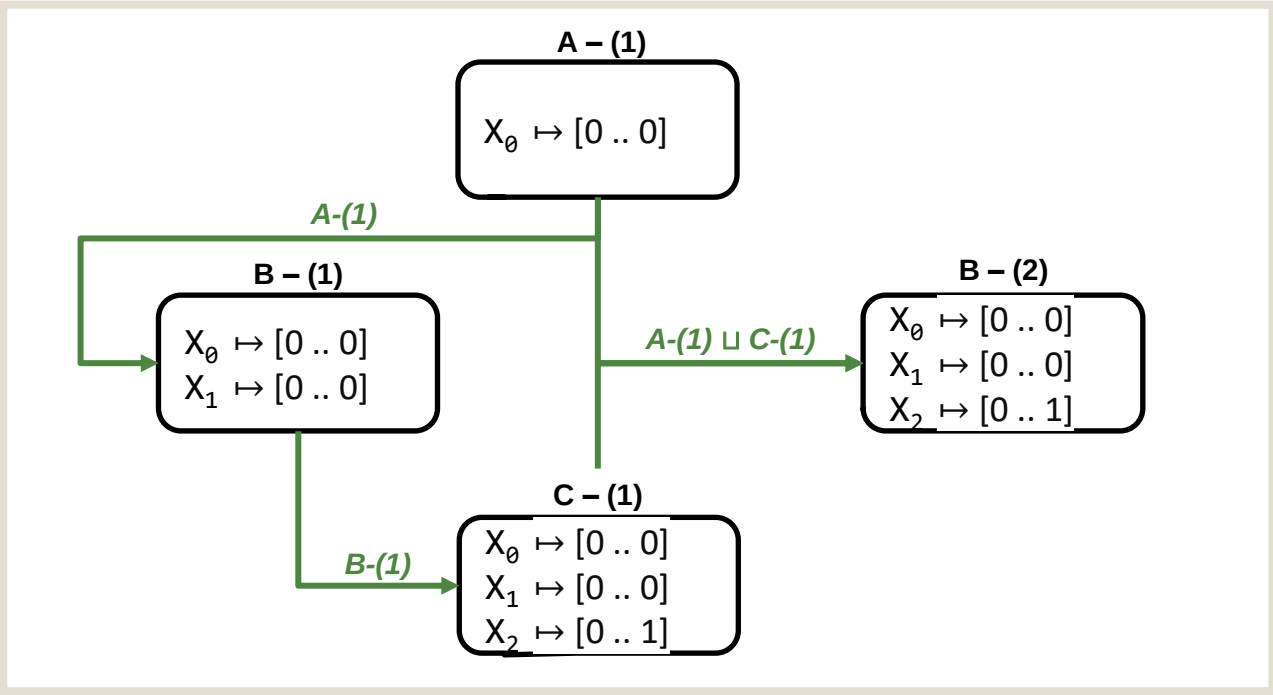
Visualizer

- V ng
  S



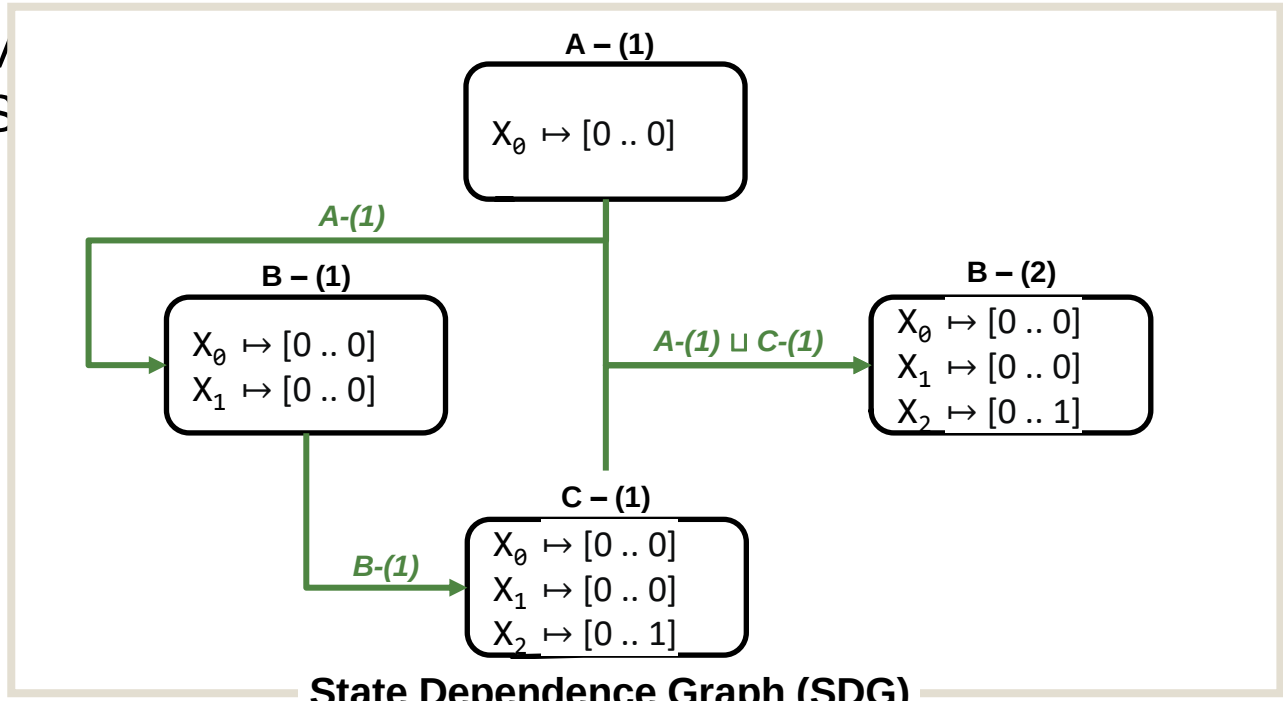
Visualizer

- V ng
  S



Visualizer

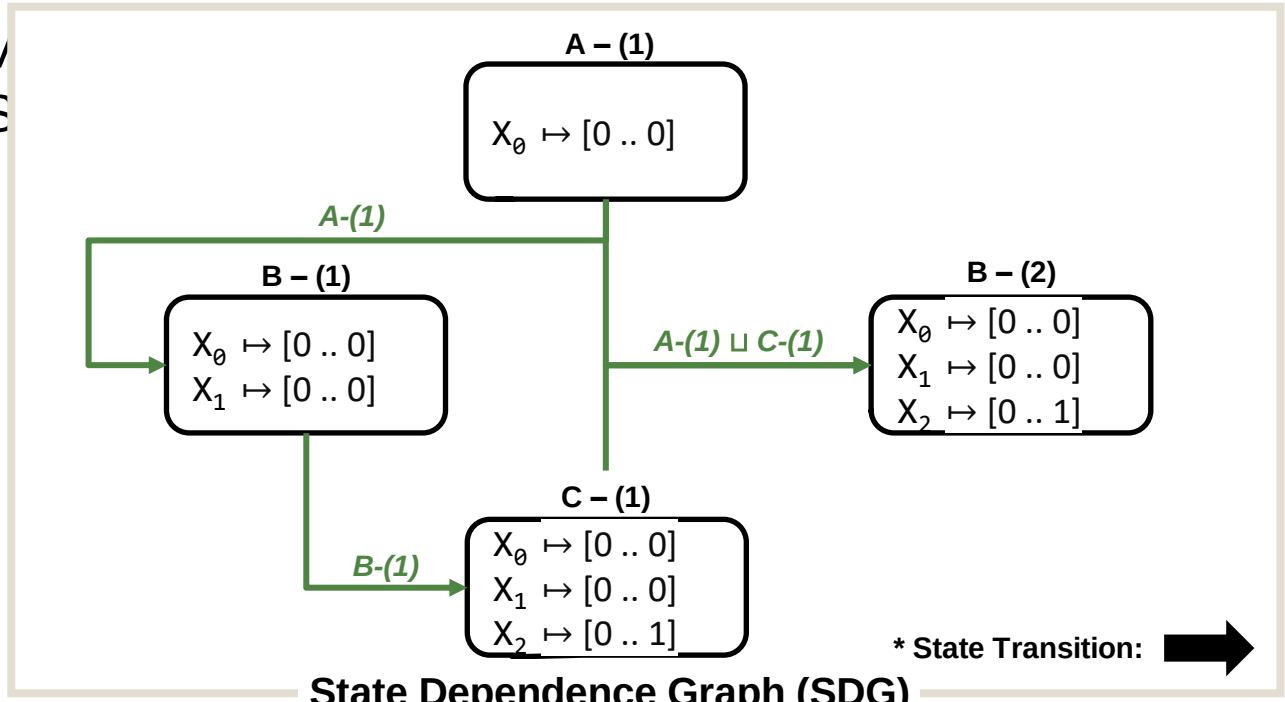
- V ng
  S



State Dependence Graph (SDG)

Visualizer

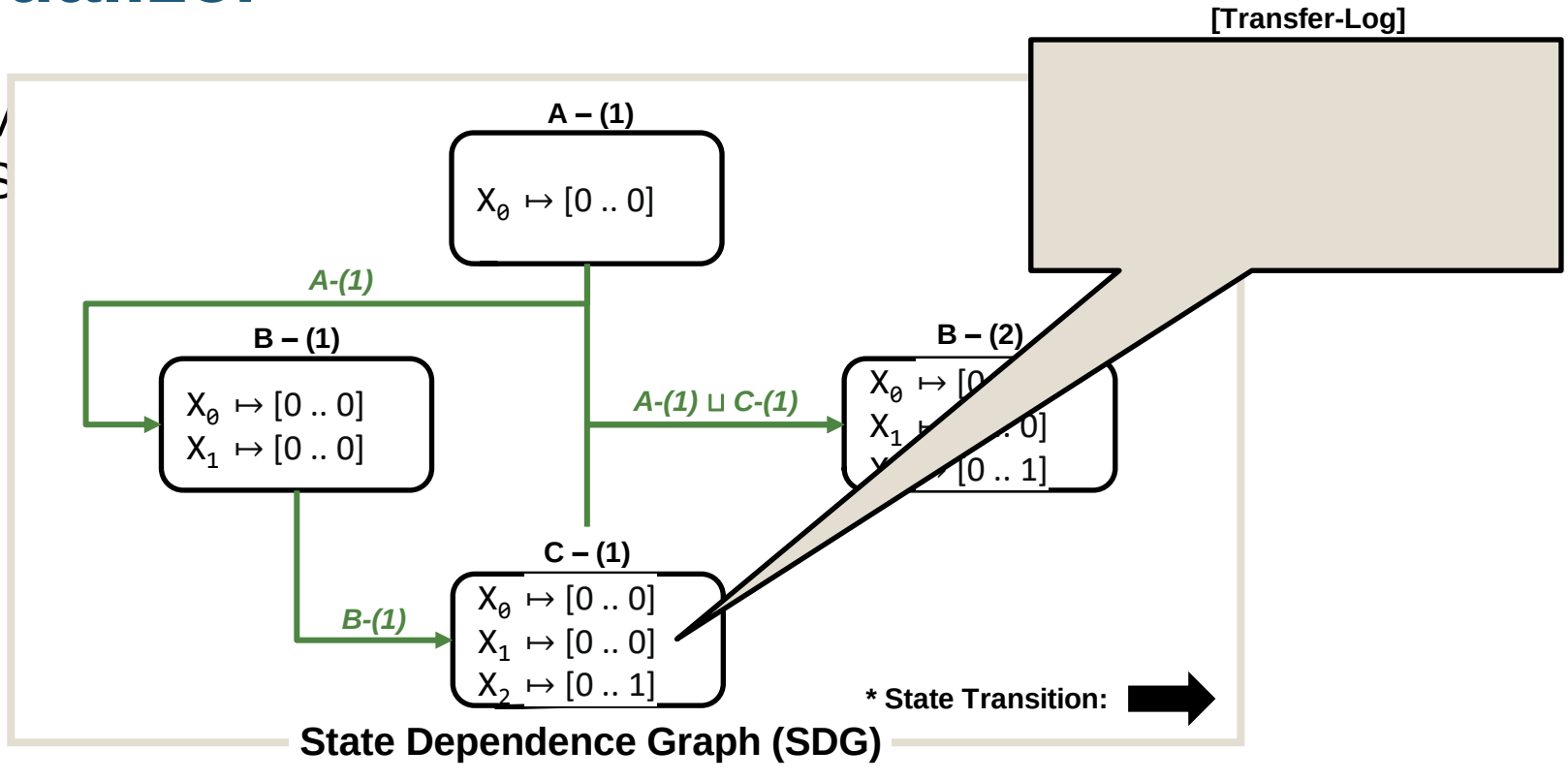
- V ng
  S



State Dependence Graph (SDG)

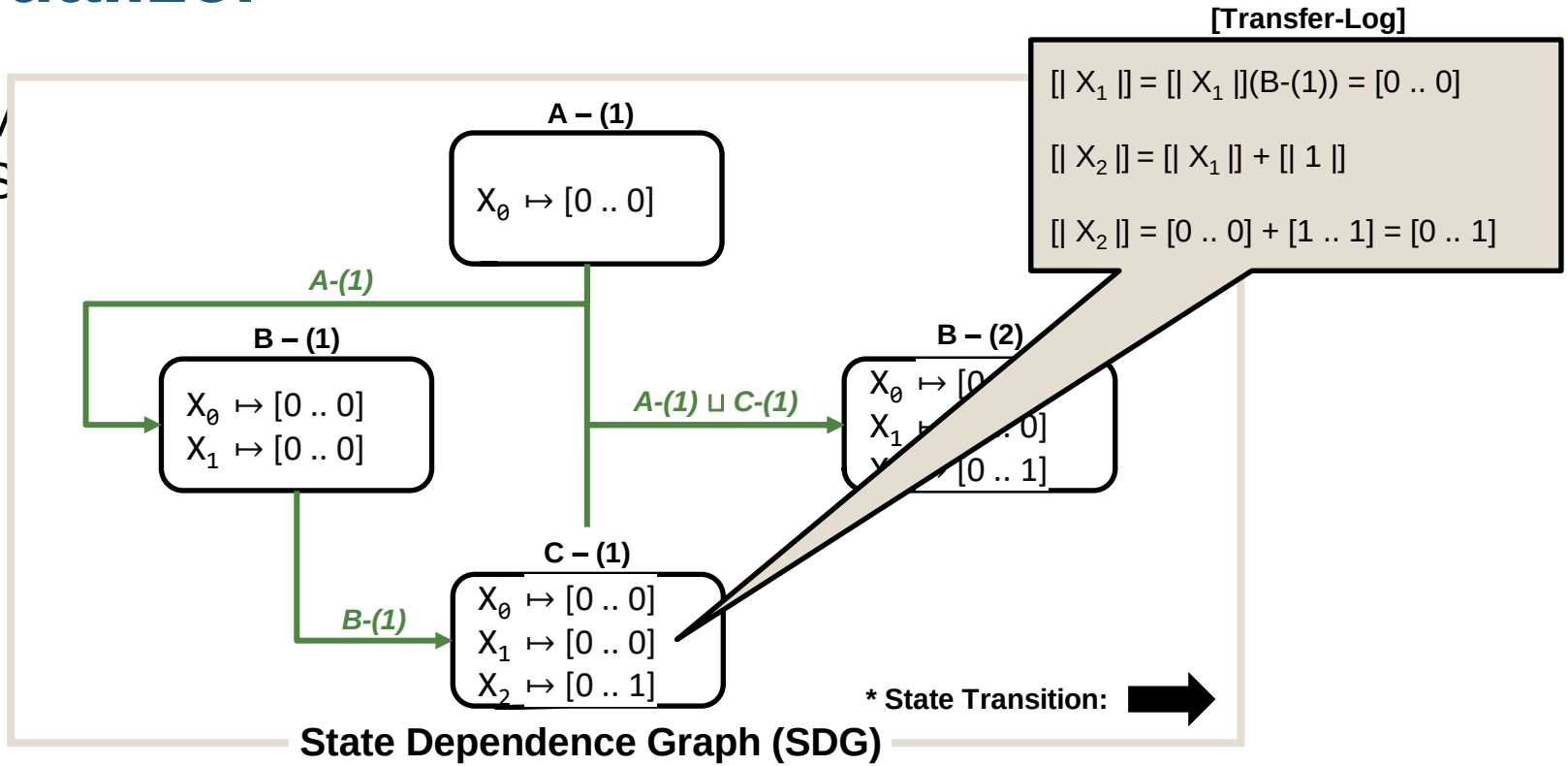
Visualizer

- V
  S



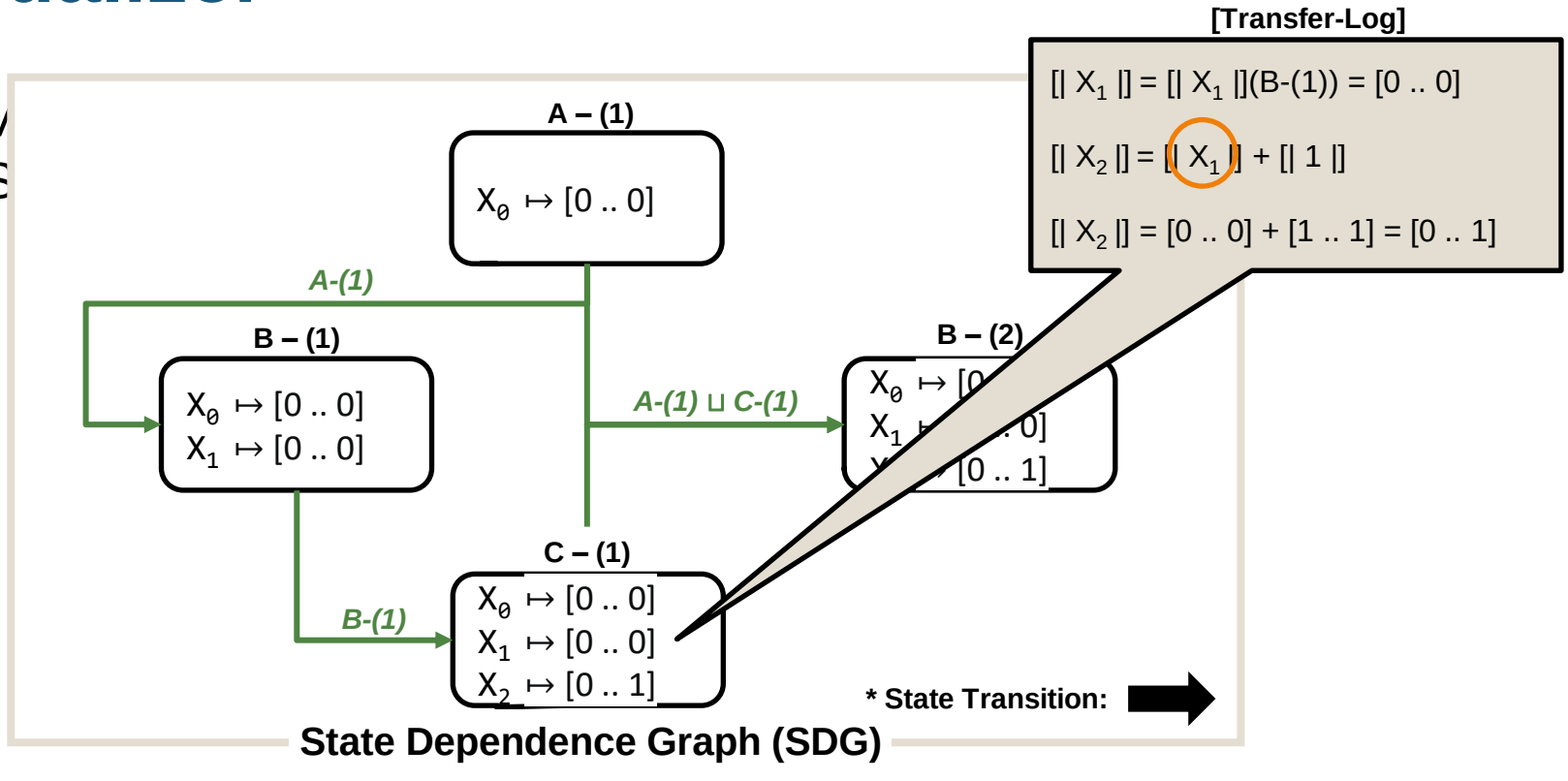
Visualizer

- V
  S



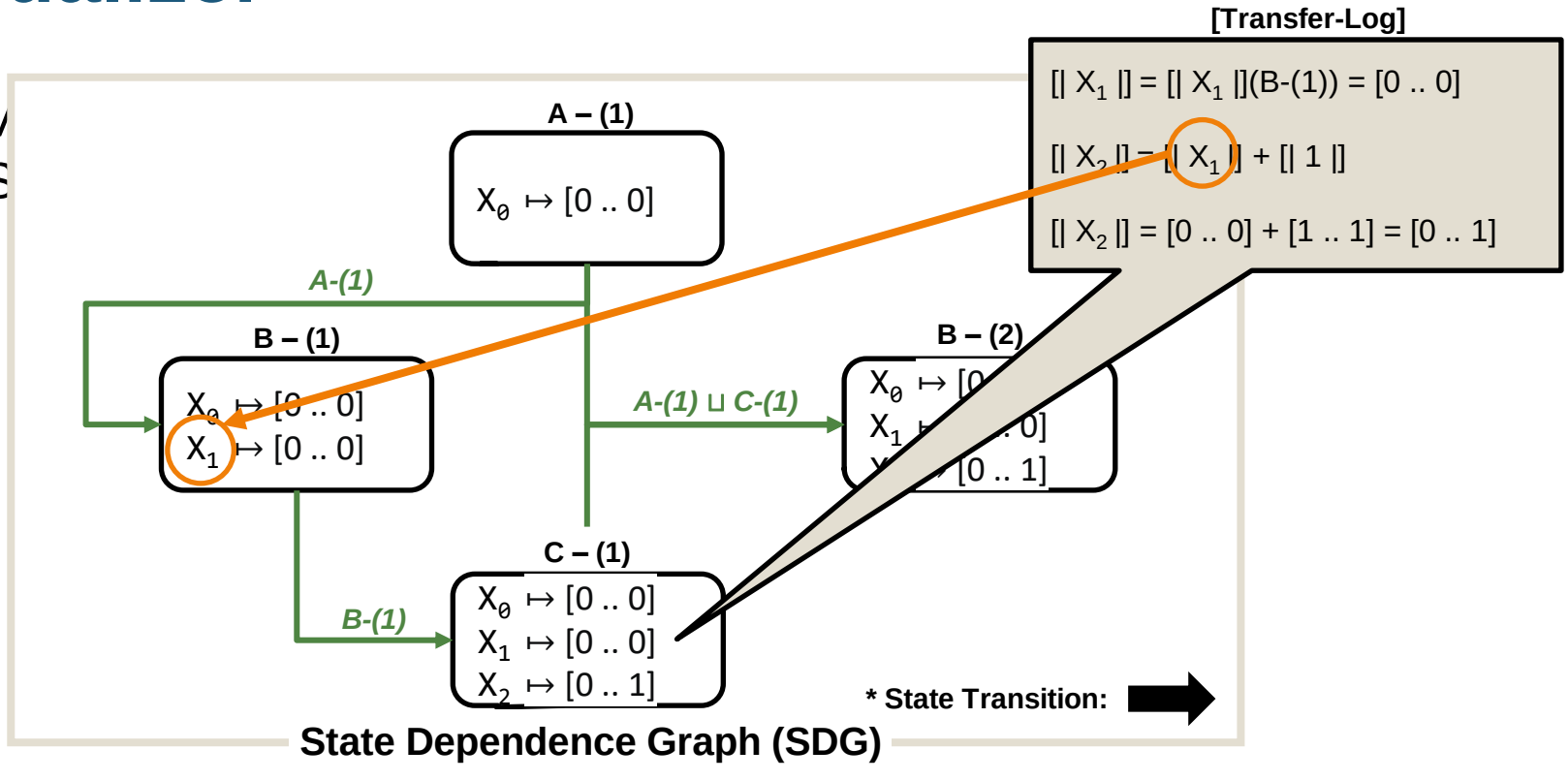
Visualizer

- V
  S

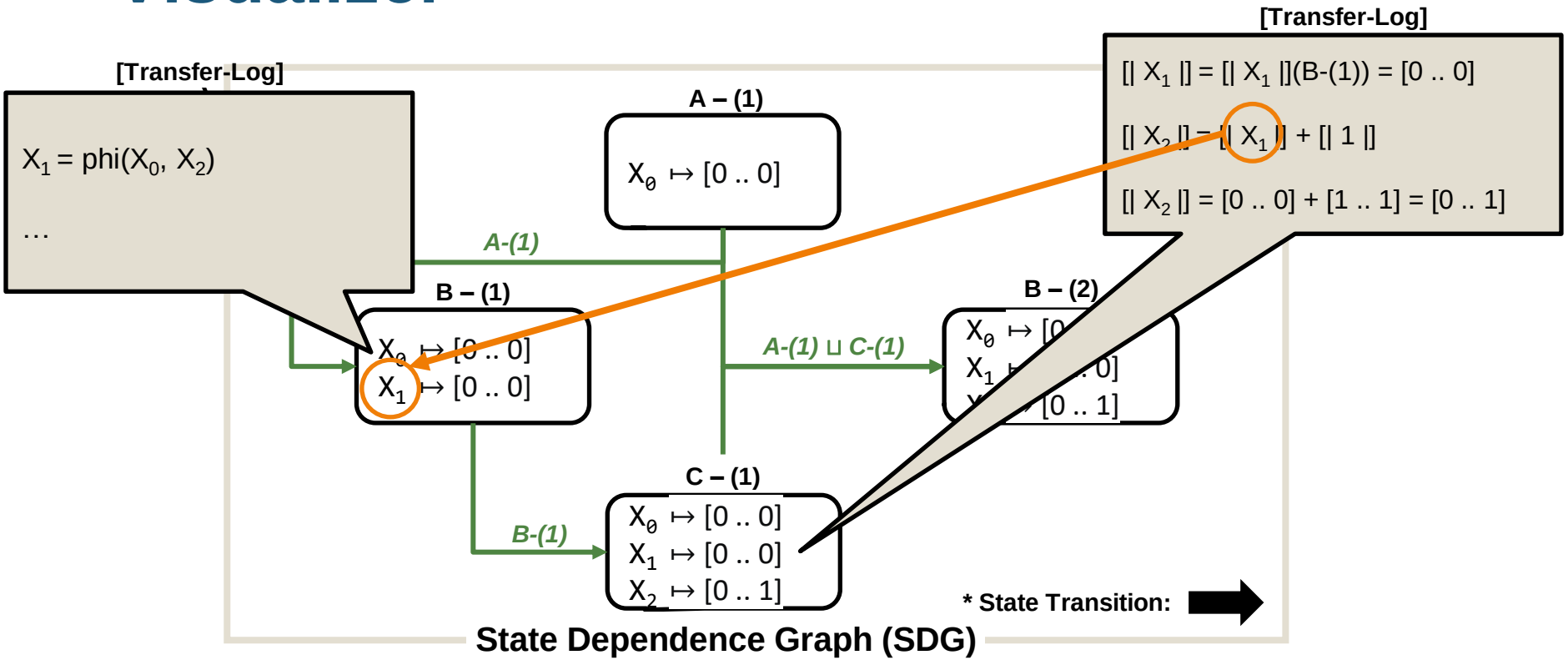


Visualizer

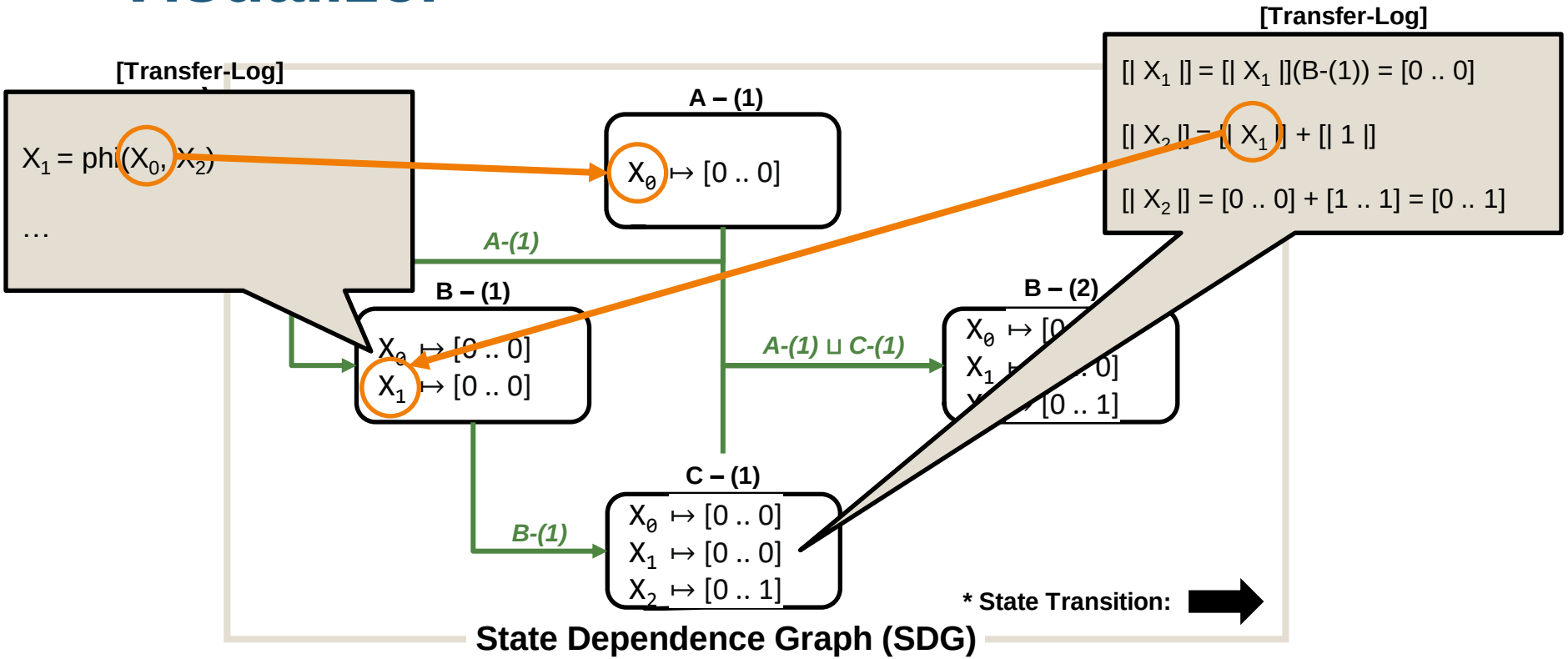
- V
  S



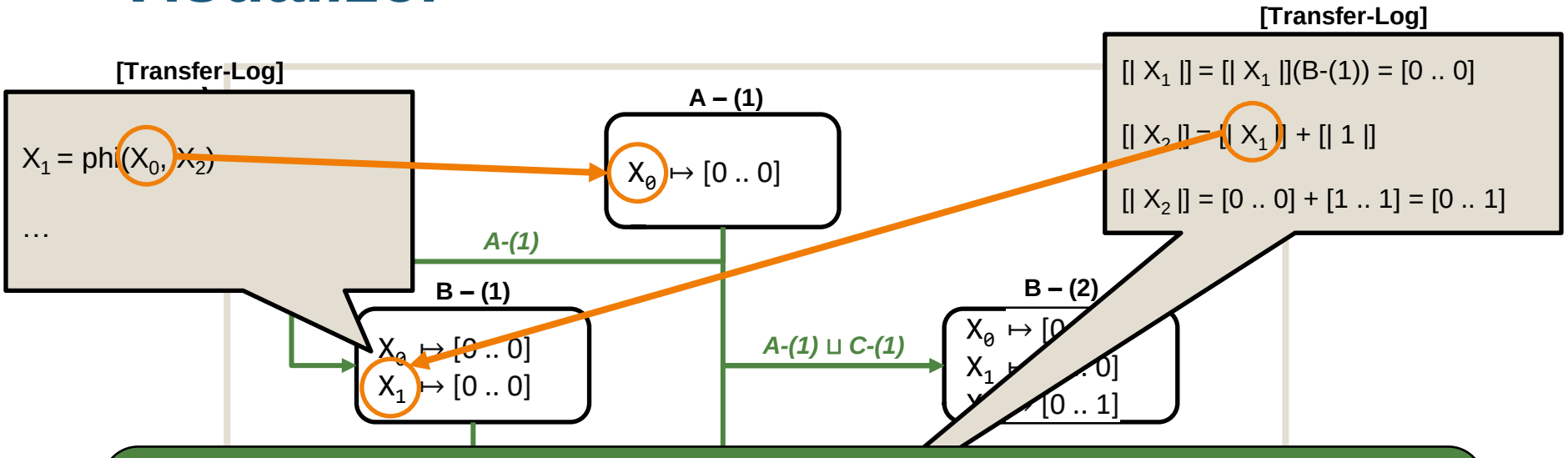
Visualizer


[Transfer-Log]
X1 = phi(X0, X2)
…
A – (1)
X0 ↦ [0 .. 0]
A-(1)
B – (1)
X0 ↦ [0 .. 0]
X1 ↦ [0 .. 0]
A-(1) ⊔ C-(1)
B – (2)
X0 ↦ [0
X1 ↦ [0 .. 0]
[0 .. 1]
B-(1)
C – (1)
X0 ↦ [0 .. 0]
X1 ↦ [0 .. 0]
X2 ↦ [0 .. 1]
[Transfer-Log]
[| X1 |] = [| X1 |](B-(1)) = [0 .. 0]
[| X2 |] = [| X1 |] + [| 1 |]
[| X2 |] = [0 .. 0] + [1 .. 1] = [0 .. 1]
* State Transition:
State Dependence Graph (SDG)

Visualizer


[Transfer-Log]
X1 = phi(X0, X2)
…
A – (1)
X0 ↦ [0 .. 0]
A-(1)
B – (1)
X0 ↦ [0 .. 0]
X1 ↦ [0 .. 0]
A-(1) ⊔ C-(1)
B – (2)
X0 ↦ [0
X1 ↦ [0 .. 0]
[0 .. 1]
B-(1)
C – (1)
X0 ↦ [0 .. 0]
X1 ↦ [0 .. 0]
X2 ↦ [0 .. 1]
[Transfer-Log]
[| X1 |] = [| X1 |](B-(1)) = [0 .. 0]
[| X2 |] = [| X1 |] + [| 1 |]
[| X2 |] = [0 .. 0] + [1 .. 1] = [0 .. 1]
* State Transition:
State Dependence Graph (SDG)

Visualizer



SDG helps to understand the flow of state transitions.

Visualizer

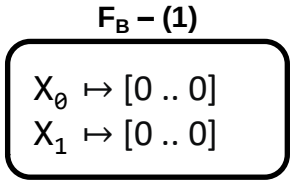
- Visualize each state that an AI process can have by using SSA Graph and State Dependence Graph (SDG).

- Sensitivities (e.g. context-sensitivity) are denoted using a prefix in a state identifier.

Visualizer

- Visualize each state that an AI process can have by using SSA Graph and State Dependence Graph (SDG).

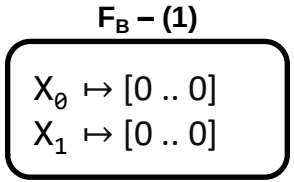
- Sensitivities (e.g. context-sensitivity) are denoted using a prefix in a state identifier.



No sensitivity

Visualizer

- Visualize each state that an AI process can have by using SSA Graph and State Dependence Graph (SDG).
- Sensitivities (e.g. context-sensitivity) are denoted using a prefix in a state identifier.

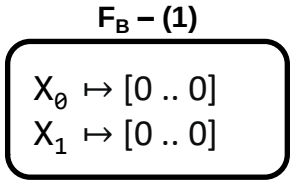


No sensitivity

Context-sensitivity w/ depth=1

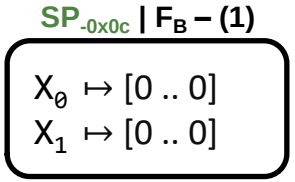
Visualizer

- Visualize each state that an AI process can have by using SSA Graph and State Dependence Graph (SDG).
- Sensitivities (e.g. context-sensitivity) are denoted using a prefix in a state identifier.



No sensitivity

Context-sensitivity w/ depth=1



Stack-pointer-sensitivity

Debugger

Debugger

- Move forward/backward between different states.

Debugger

- Move forward/backward between different states.
- Set a *breakpoint* with a specific condition he wants.

Debugger

- Move forward/backward between different states.
- Set a *breakpoint* with a specific condition he wants.
- Fetch the history of state transitions with *lazy evaluation*.

AI (User-Defined)

AI (User-Defined)

- AI consists of three parts: *Lattice*, *Transfer Function*, and *Sensitivity*.

AI (User-Defined)

- AI consists of three parts: *Lattice*, *Transfer Function*, and *Sensitivity*.

- We may introduce a new DSL (Domain Specific Language) for *Lattice* and *Transfer Function*.

AI (User-Defined)

- AI consists of three parts: *Lattice*, *Transfer Function*, and *Sensitivity*.

- We may introduce a new DSL (Domain Specific Language) for *Lattice* and *Transfer Function*.

  → But for the prototype, we use *F#.*

Implementation

- Uses *F#* as a main programming language.
- Uses *D3.js*[2] for an interactive visualization of AI processes.
- Uses *LLVMSharp*[3] to parse/handle LLVM IR codes.

[2] Zhu, Nick Qi. Data visualization with D3. js cookbook. Packt Publishing Ltd, 2013.
[3] LLVMSharp, https://github.com/dotnet/LLVMSharp, 2024.

Objectives

- To help students learn AI.
- To help researchers analyze their AI.
- To be a popular framework that is widely-used by people.
- To be integrated into frameworks that internally use AI.

Request For Funding

- Salary for one PhD student & one Master student:
  $3,000 × 2 per month
- Three workstations for development and evaluation of our work:
  $5,000 × 3

Our Plan

- We will finish our prototype until this June.
- We will integrate our prototype into *B2R2*[4] until this August.
- We will finish and commercialize our project until this December.

[4] B2R2, https://github.com/B2R2-org/B2R2, 2024.

Question?