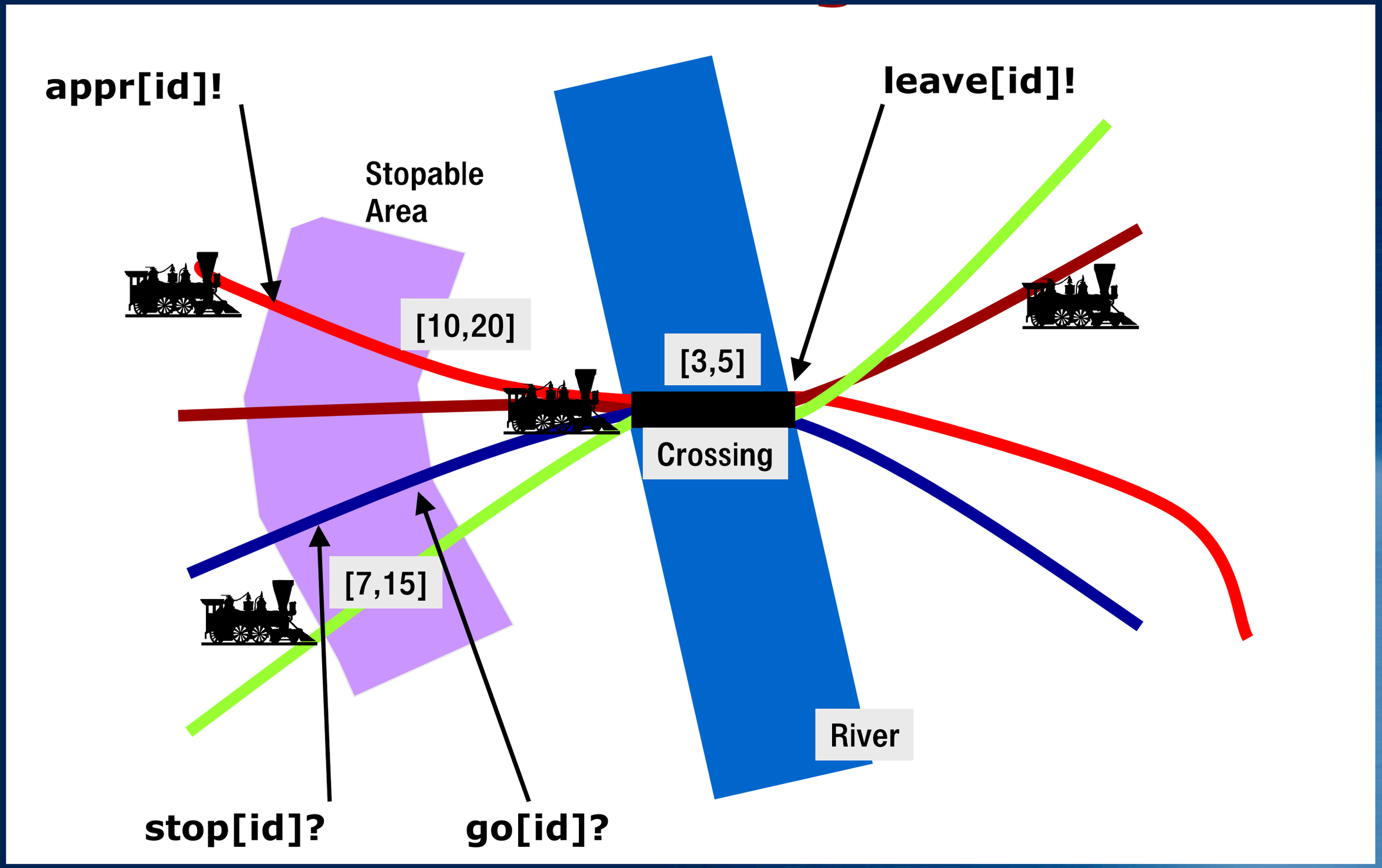
Slicing for UPPAAL

Claus Thrane, Uffe Sørensen and Kim G. Larsen

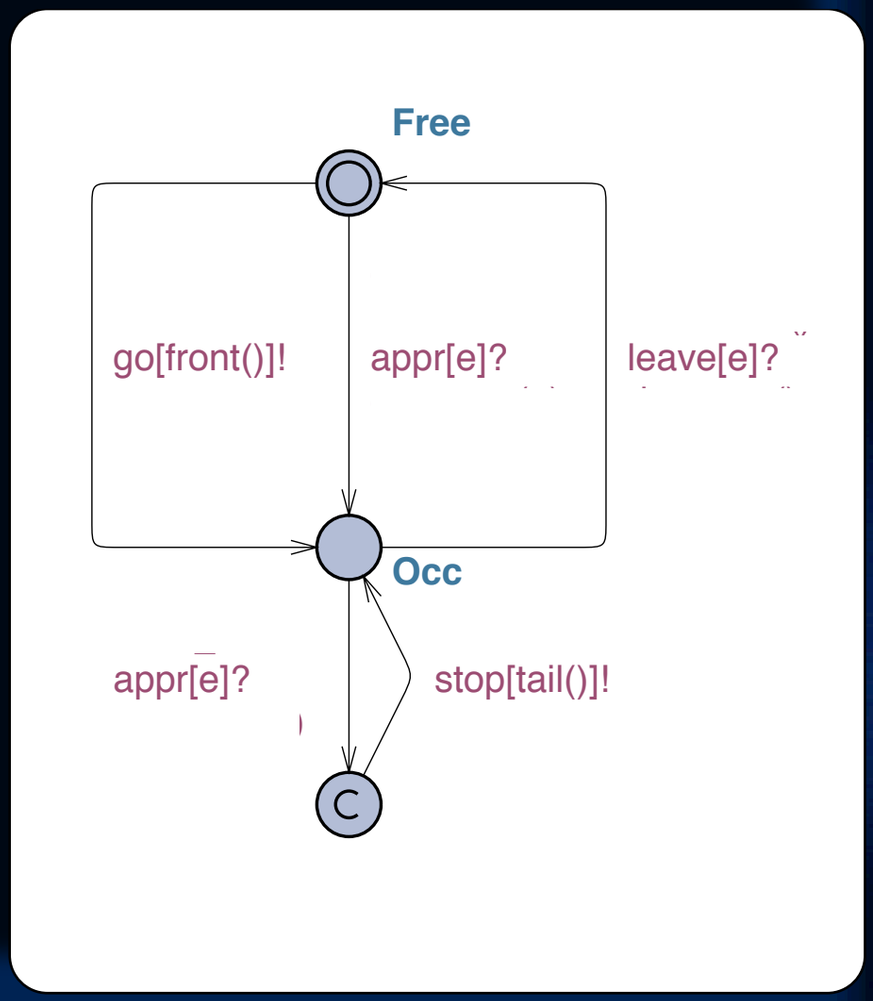
NWPT'08

Modeling Real-Time Systems

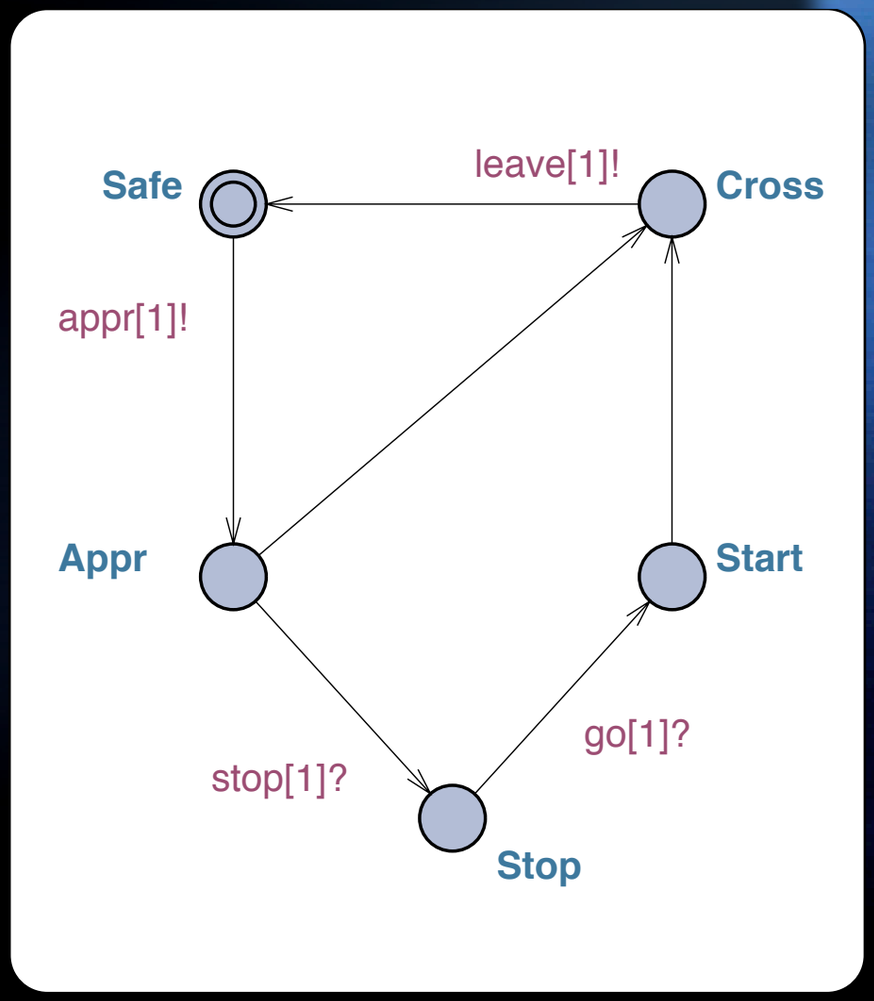


Communicating FSA

Gate

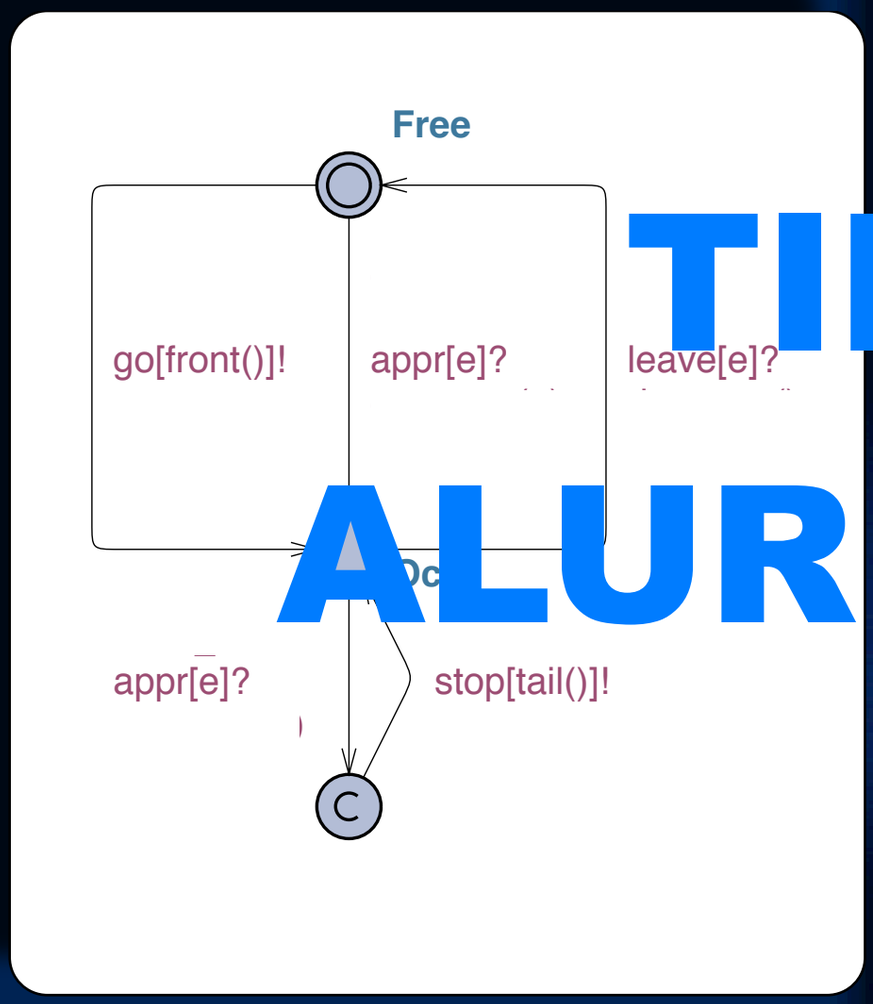


Train

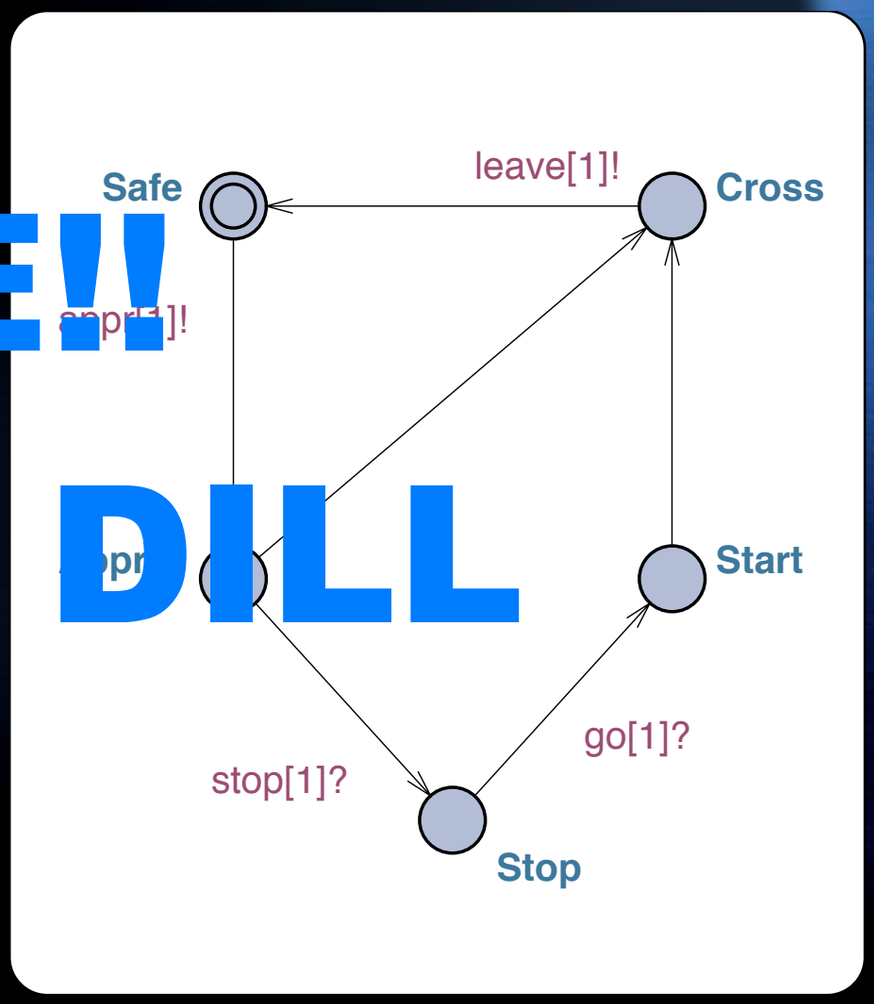


Communicating FSA

Gate



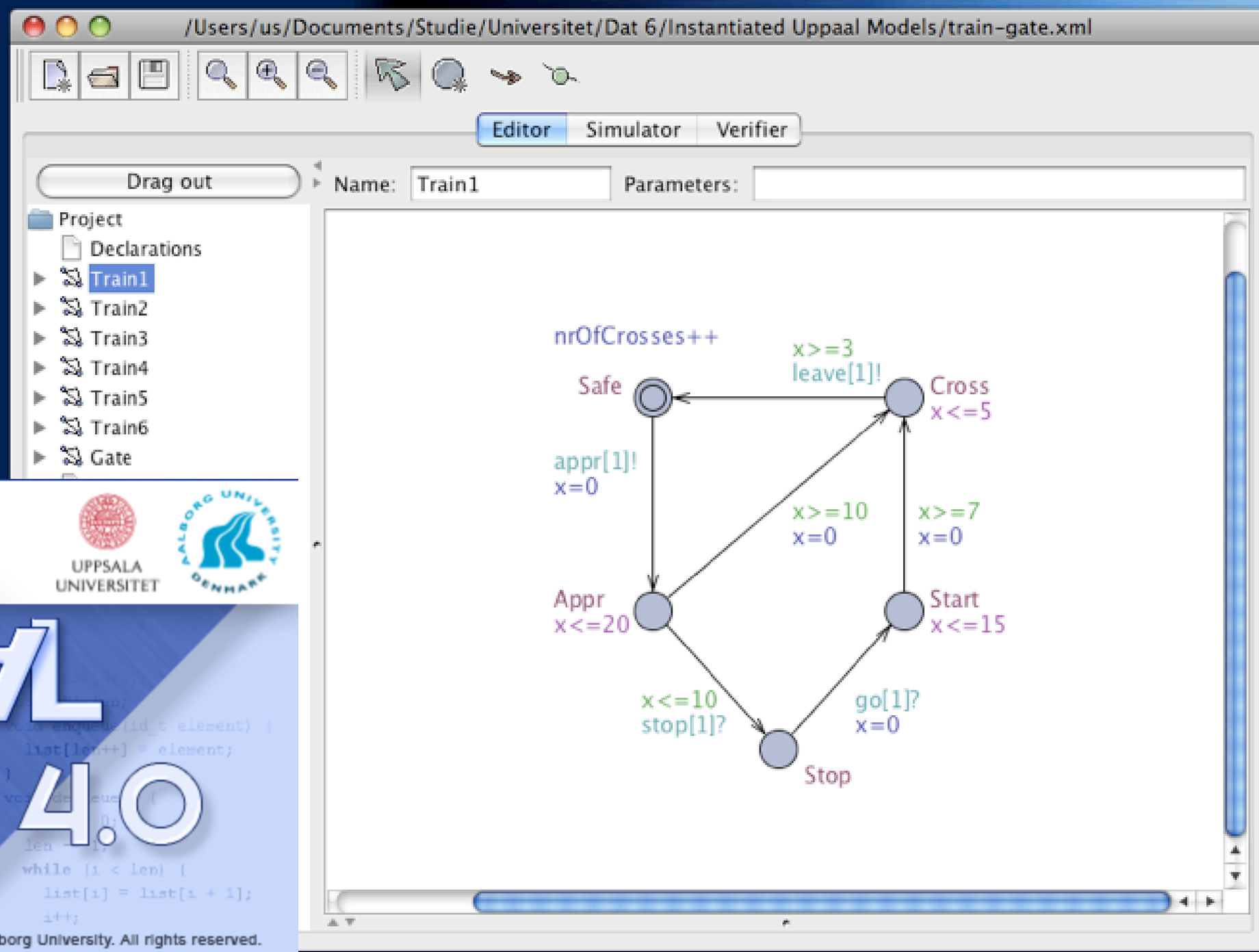
Train

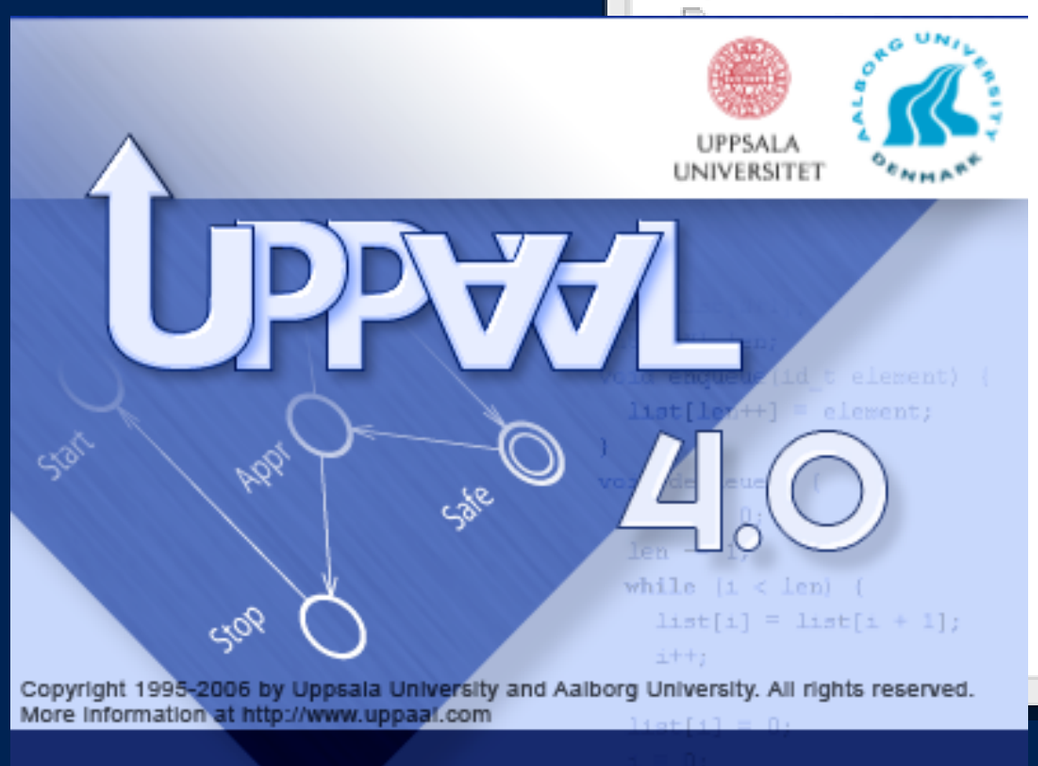


TIME!!

ALUR & DILL

UPPAAL


/Users/us/Documents/Studie/Universitet/Dat 6/Instantiated Uppaal Models/train-gate.xml
Editor
Simulator
Verifier
Drag out
Name: Train1
Parameters:
Project
Declarations
Train1
Train2
Train3
Train4
Train5
Train6
Gate
nrOfCrosses++
x>=3
leave[1]!
Safe
Cross
x<=5
appr[1]!
x=0
x>=10
x=0
x>=7
x=0
Appr
x<=20
Start
x<=15
x<=10
stop[1]?
go[1]?
x=0
Stop


UPPSALA UNIVERSITET
AALBORG UNIVERSITY DENMARK
UPPAAL
4.0
Start
Appr
Safe
Stop
Copyright 1995-2006 by Uppsala University and Aalborg University. All rights reserved.
More information at http://www.uppaal.com

Imperative code

```
clock x1, x2, .... ,xn;
```

```
int[0,5] i1, i2, .... ,in;
```

```
const int delay = 5;
```

```
int a[4] = {1,8,7,42};
```

```
bool b;
```

```
struct{int a; int b;}a = {9, 42};
```

```
typedef int[1,7] id_t;
```

```
if(...){...}
```

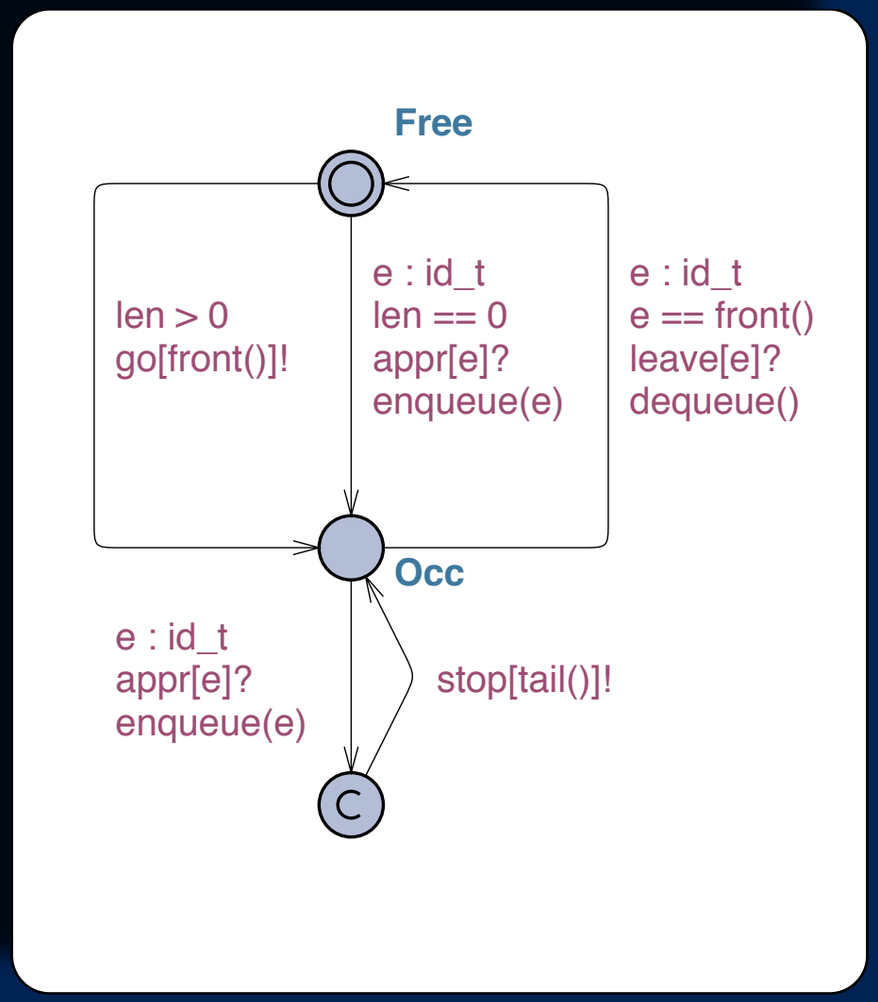
```
while(...){...}
```

```
for(.;.;.){...}
```

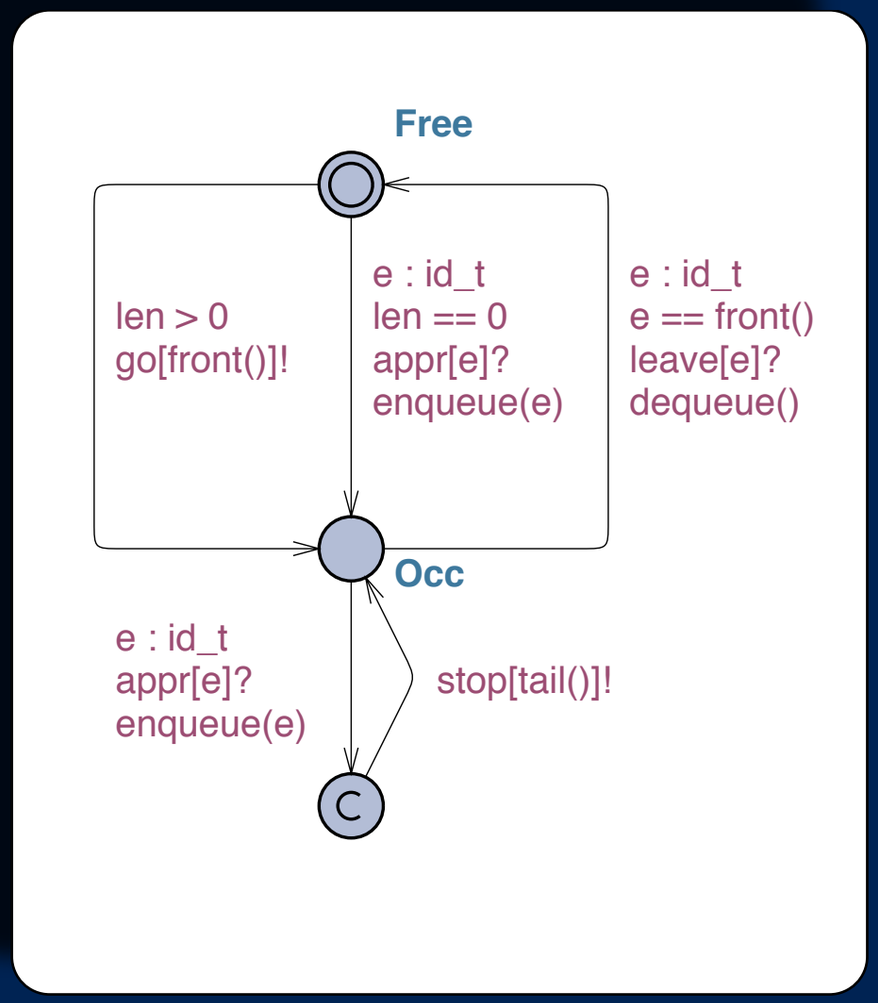
```
id_t foo(...)
{
 ...
 return ..
}
```

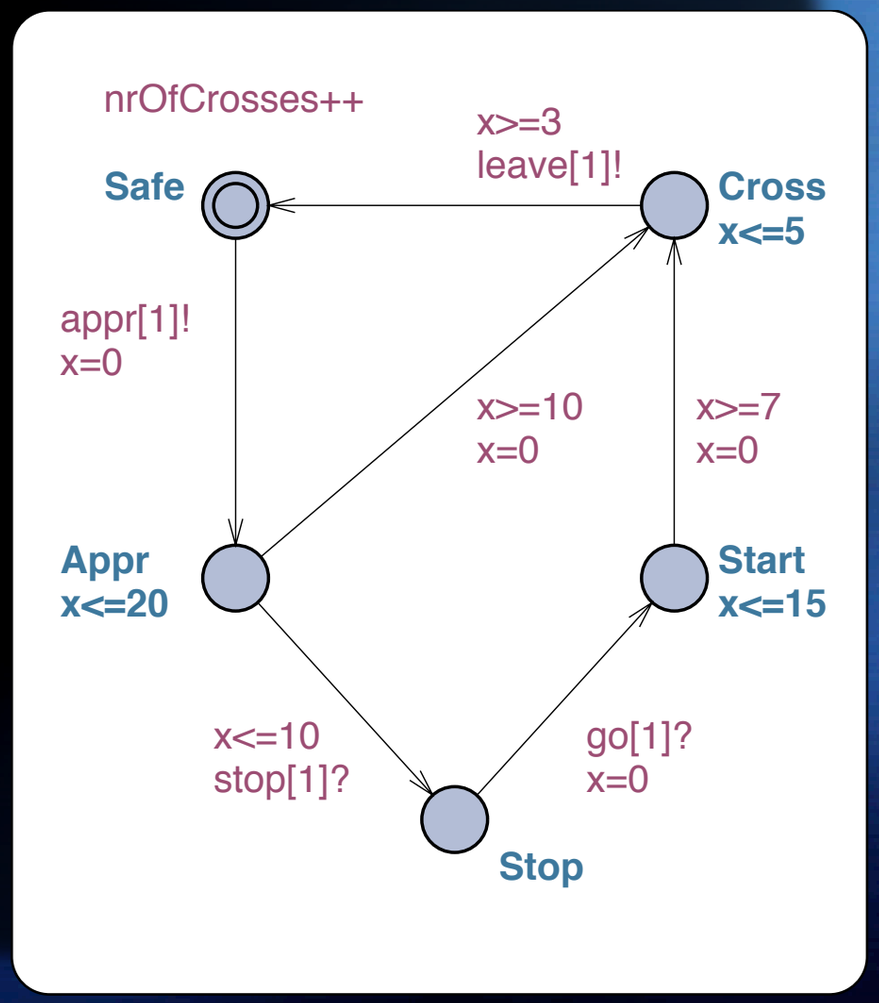
UPPAAL Model

UPPAAL Model

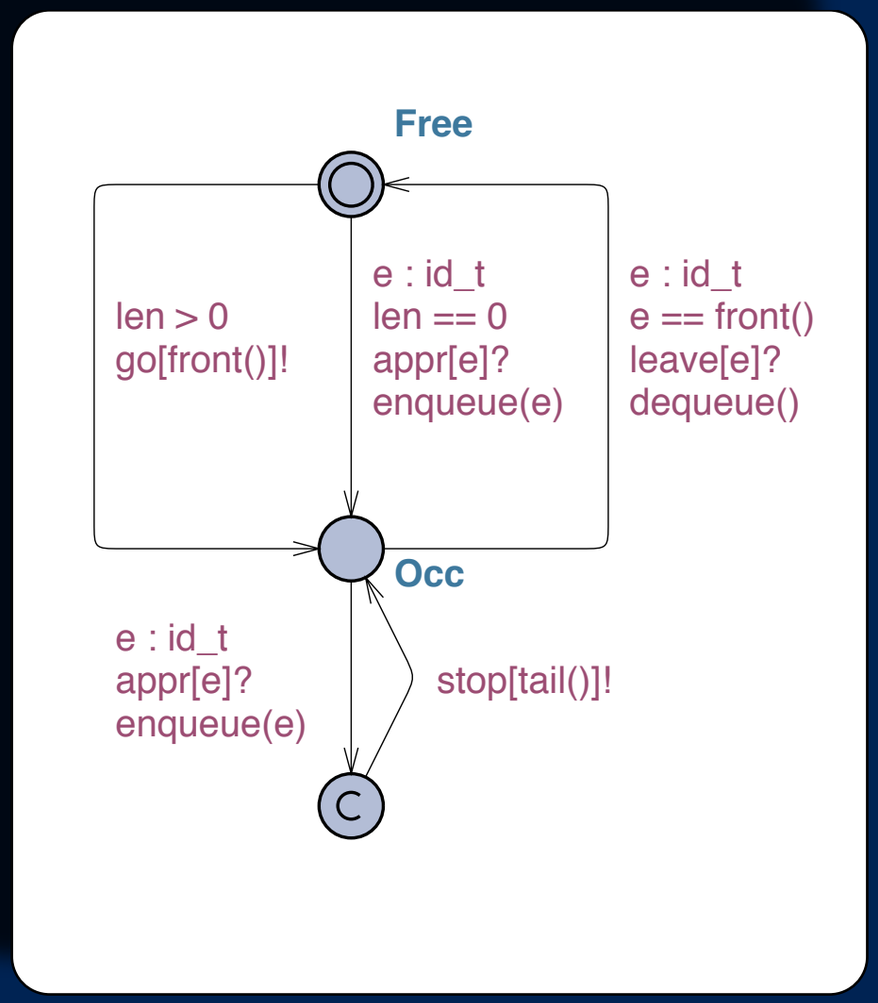


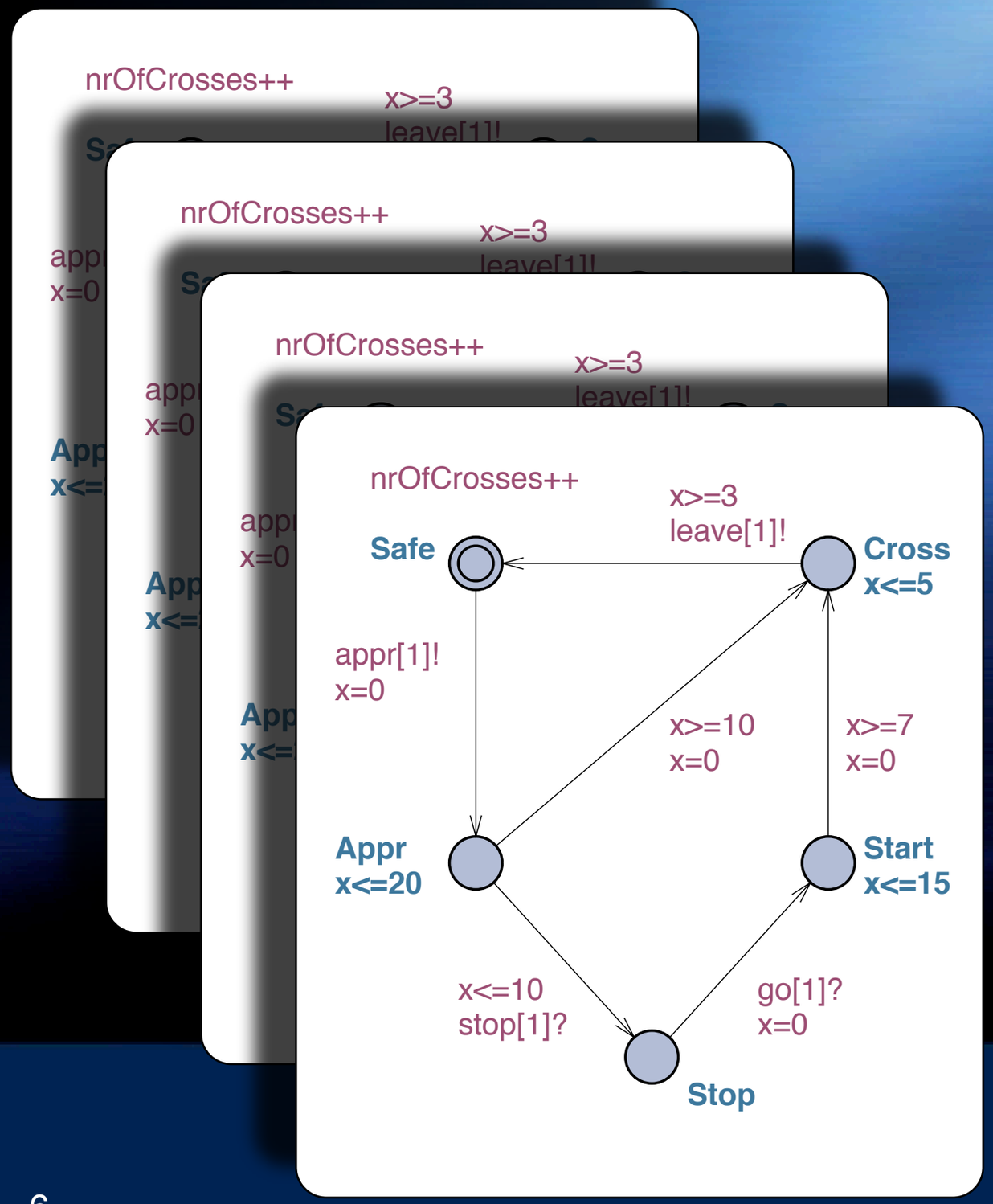
UPPAAL Model





UPPAAL Model

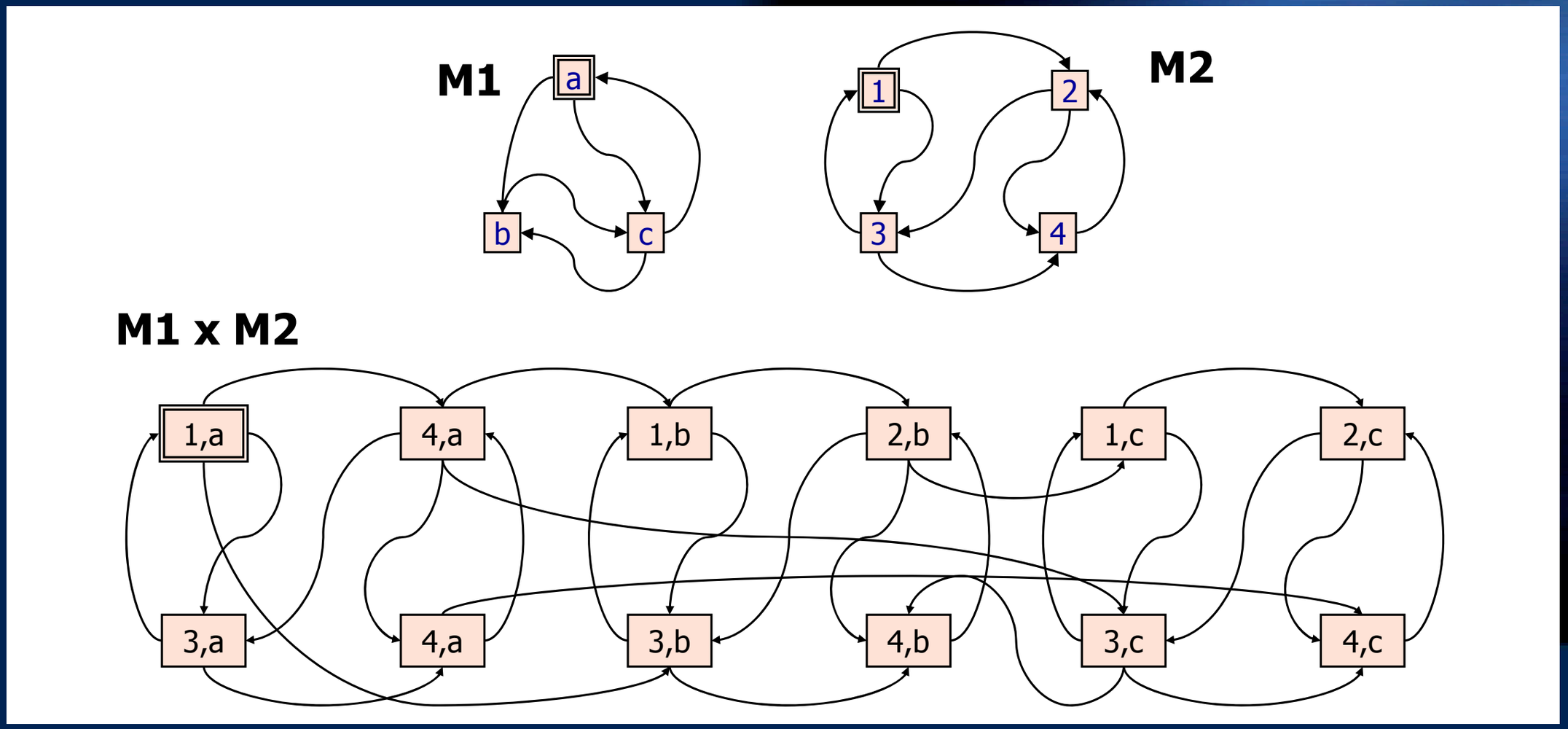




Motivation

Model-checking Timed Automata is PSPACE-Complete

State-space explosion problem



We propose

- Static analysis, specifically slicing can be used on the “hybrid” modeling language of UPPAAL - to improve the tool’s performance.

Slicing

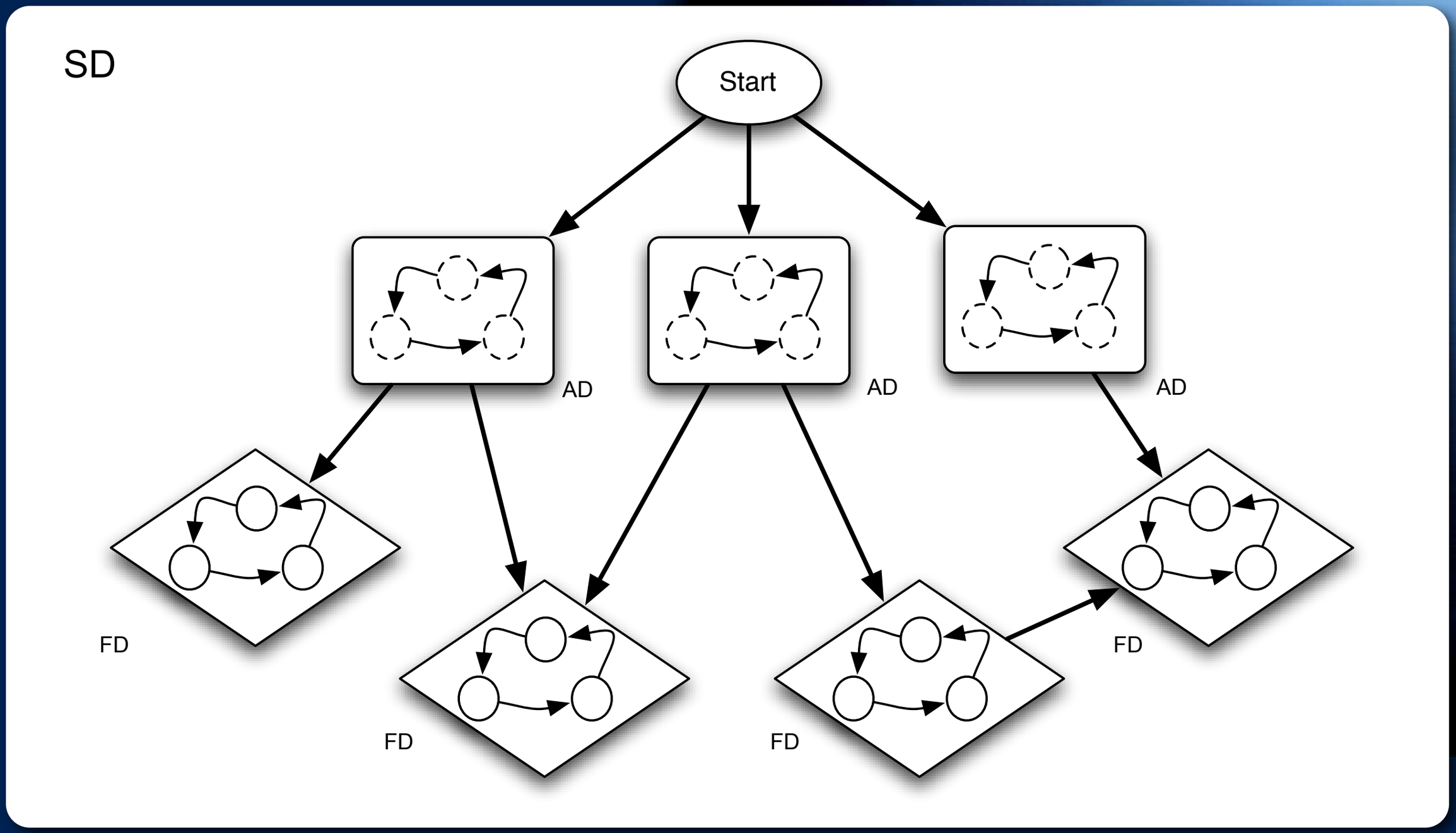
Weiser: "*a slice corresponds to the mental abstractions made by people while debugging programs*"

Traditionally used for removing irrelevant components during debugging and testing

- Slicing for TA w. simple data - Janowska and Janowski
- Used in SPIN and IF (KRONOS) to reduce the syntactic size of models prior to verification

System Dependency Graph

[S. Horwitz, T. Reps, and D. Binkley]



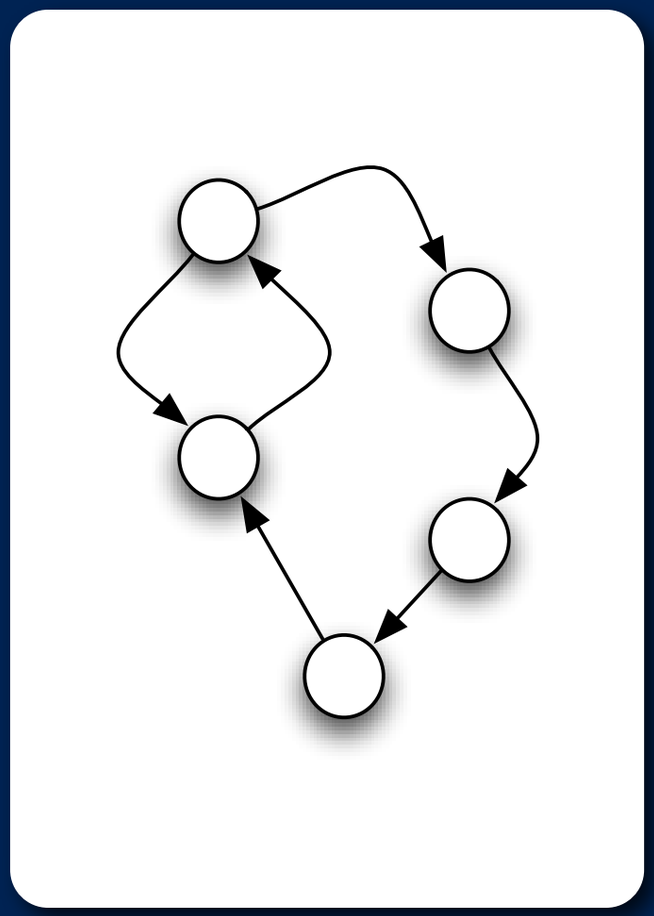
Dependencies

- Control Dependencies - Control Edges
  - Control Flow Graph
- Data Dependencies - Data Flow Edges
  - Reaching (or Use/Def chains)

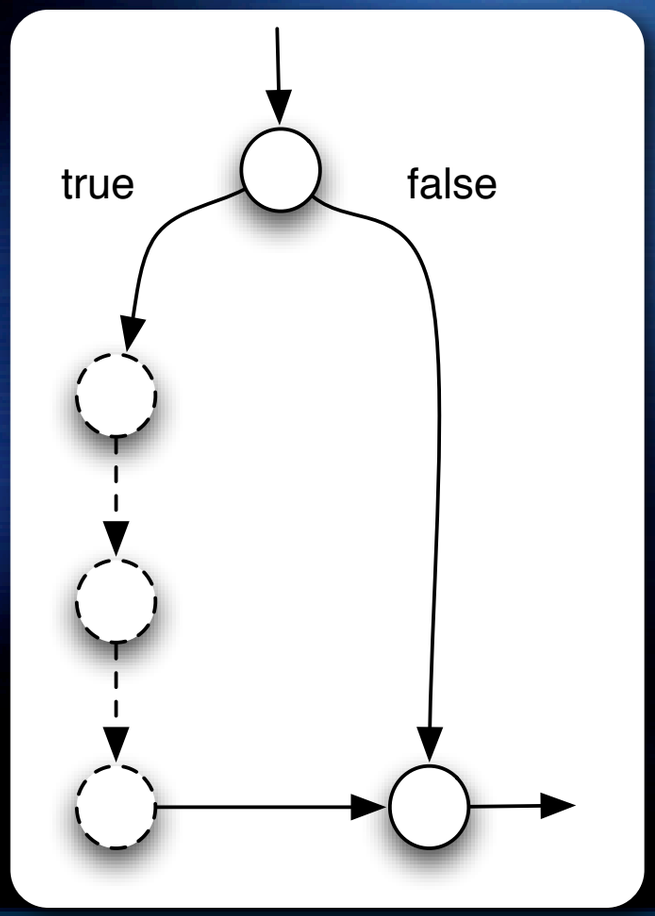
Control-Flow Graph

- Explicit and Implicit Control-Flow

FSA



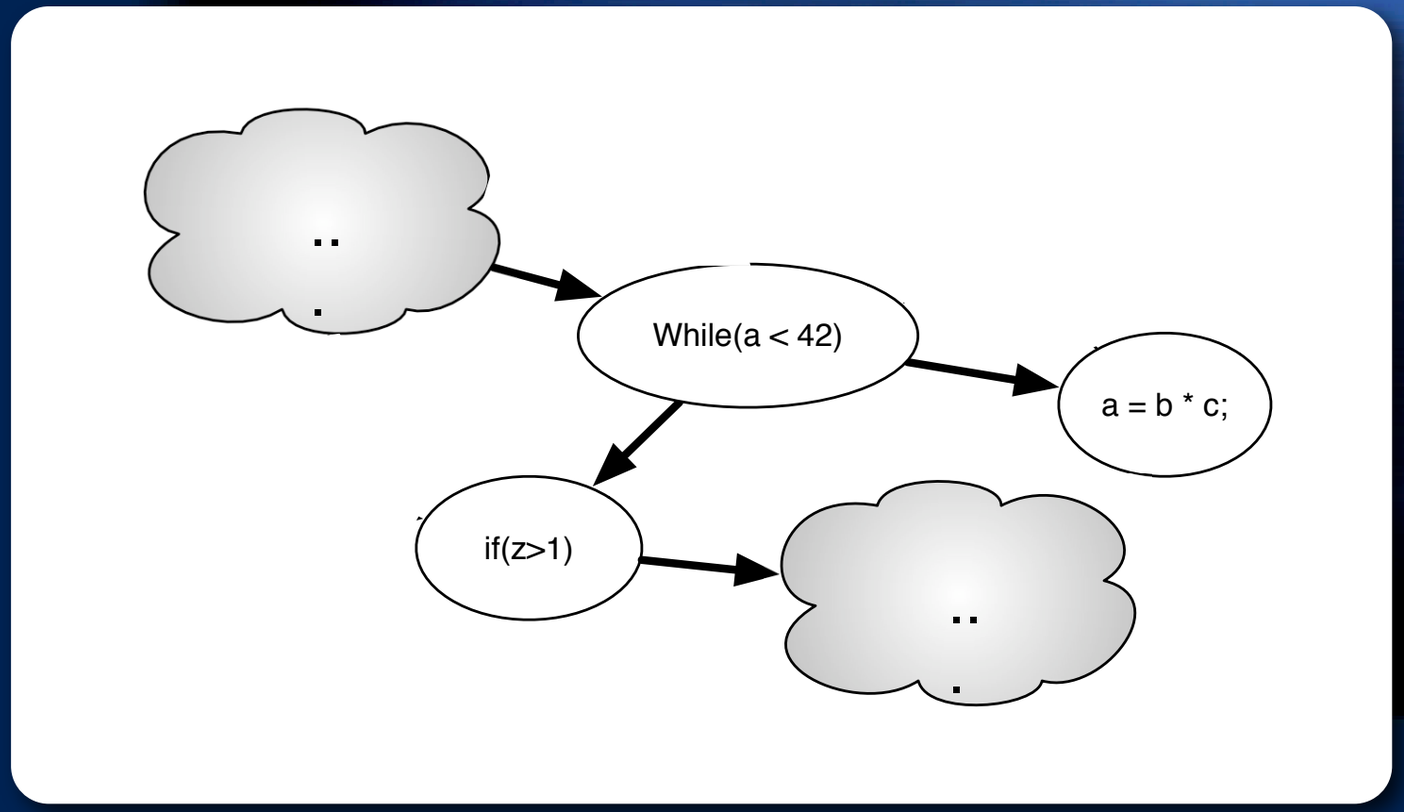
If - statement



Dependency Graphs

- Multi Graph - composed of
  - Control Graph
  - Data Flow Graph

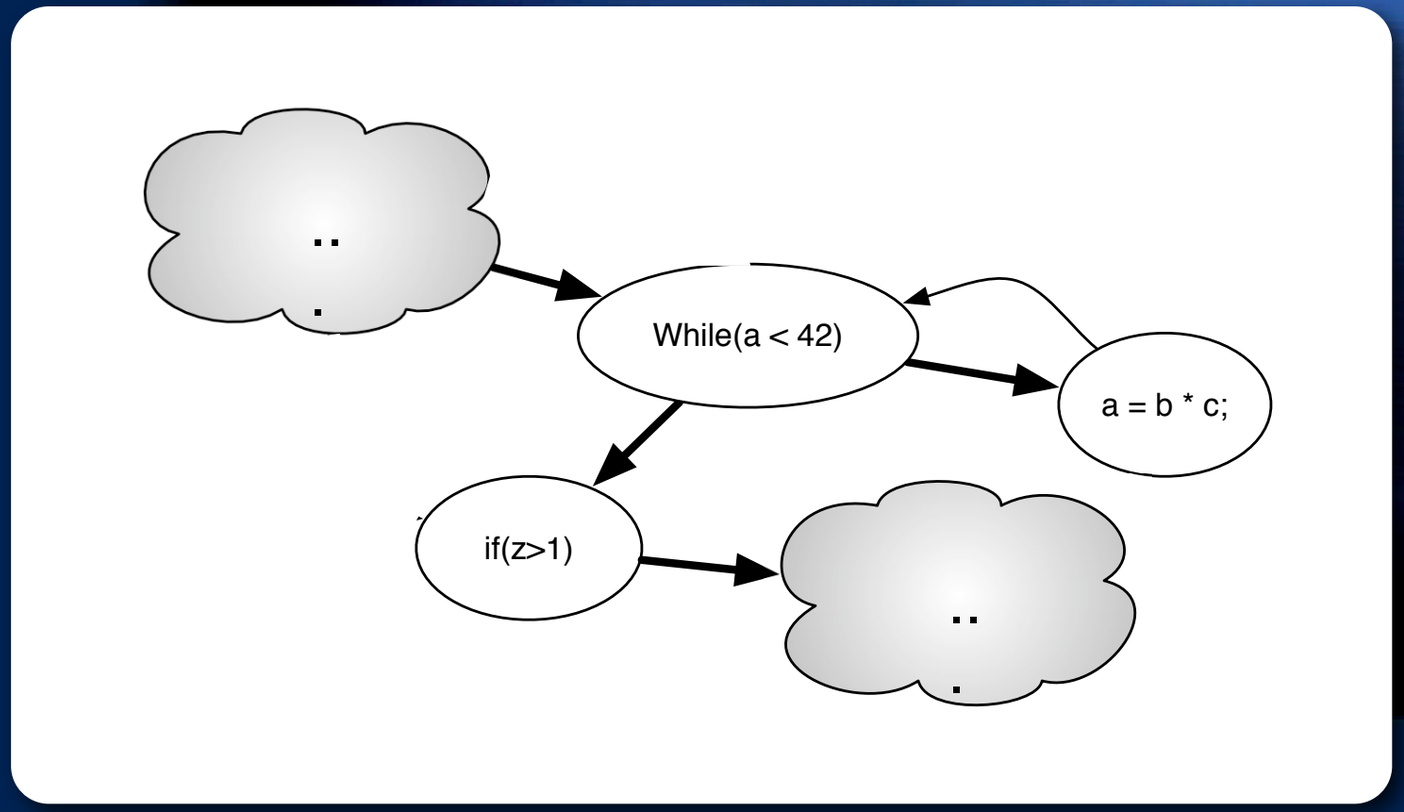
```
...
while (a < 42)
{
    a = b * c;
    if(z > 1)
    {
        ...
    }
}
```



Dependency Graphs

- Multi Graph - composed of
  - Control Graph
  - Data Flow Graph

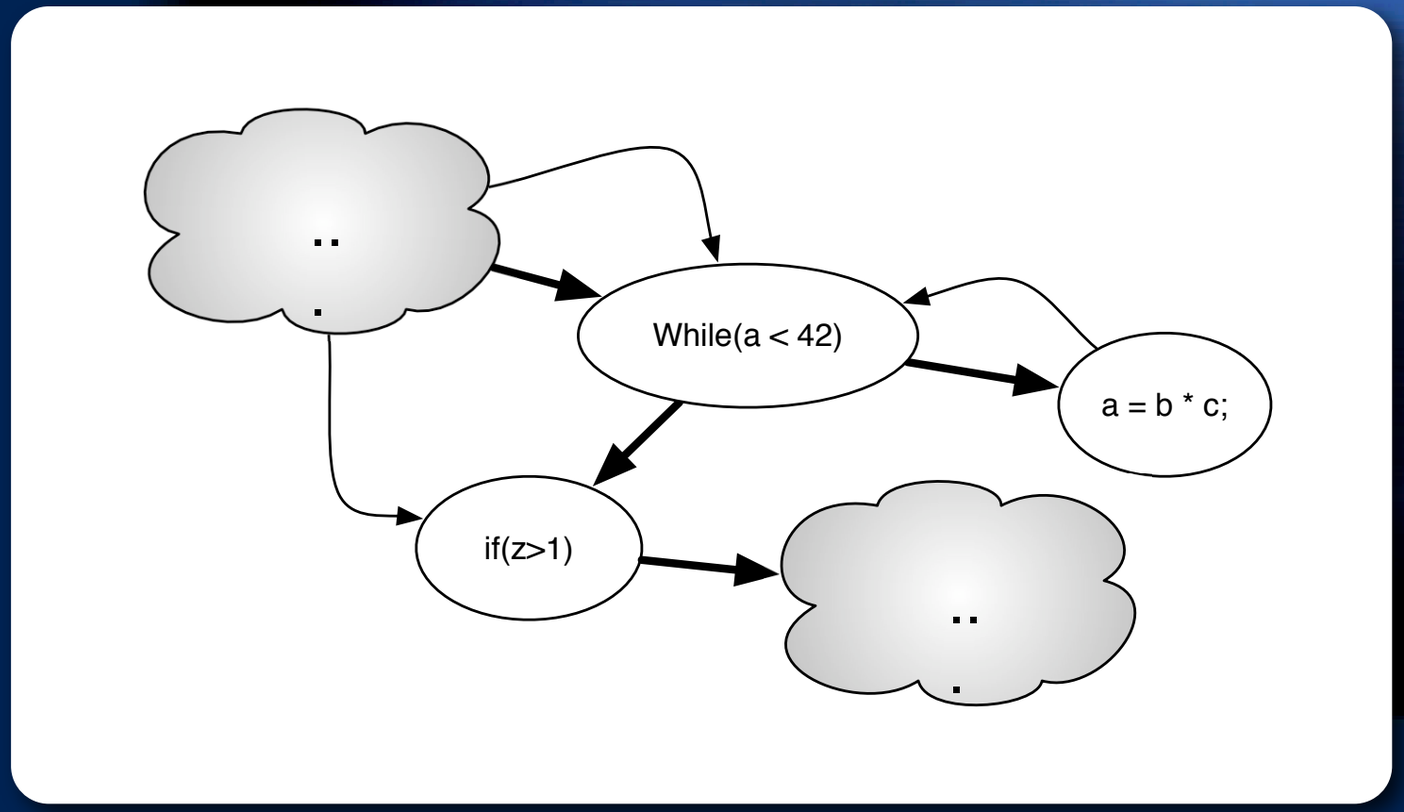
```
...
while (a < 42)
{
    a = b * c;
    if(z > 1)
    {
        ...
    }
}
```



Dependency Graphs

- Multi Graph - composed of
  - Control Graph
  - Data Flow Graph

```
...
while (a < 42)
{
    a = b * c;
    if(z > 1)
    {
        ...
    }
}
```



Example Function Dependency Graph

- Identify place of relevance
- and irrelevant code
- Inter-procedural slicing

[S. Horwitz, T. Reps, and D. Binkley]

```
int x = 0;
int z = 0;

int foo(int a, int b)
{
        while(a < 42)
        {
                a = b * x;
                if(z > 1)
                {
                        z--;
                }
        }
        return a;
}

void bar()
{
        z = 42;
        x = foo(x,z);
}
```

Example Function Dependency Graph

- Identify place of relevance
- and irrelevant code
- Inter-procedural slicing

[S. Horwitz, T. Reps, and D. Binkley]

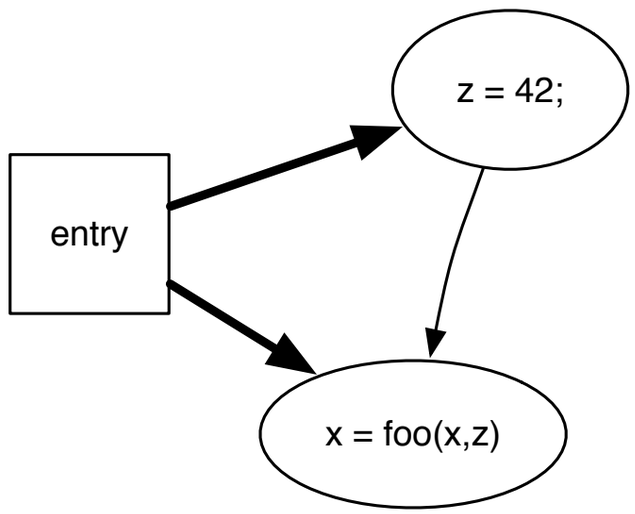
```
int x = 0;
int z = 0;

int foo(int a, int b)
{
        while(a < 42)
        {
                a = b * x;
                if(z > 1)
                {
                        z--;
                }
        }
        return a;
}

void bar()
{
        z = 42;
        x = foo(x,z);
}
```

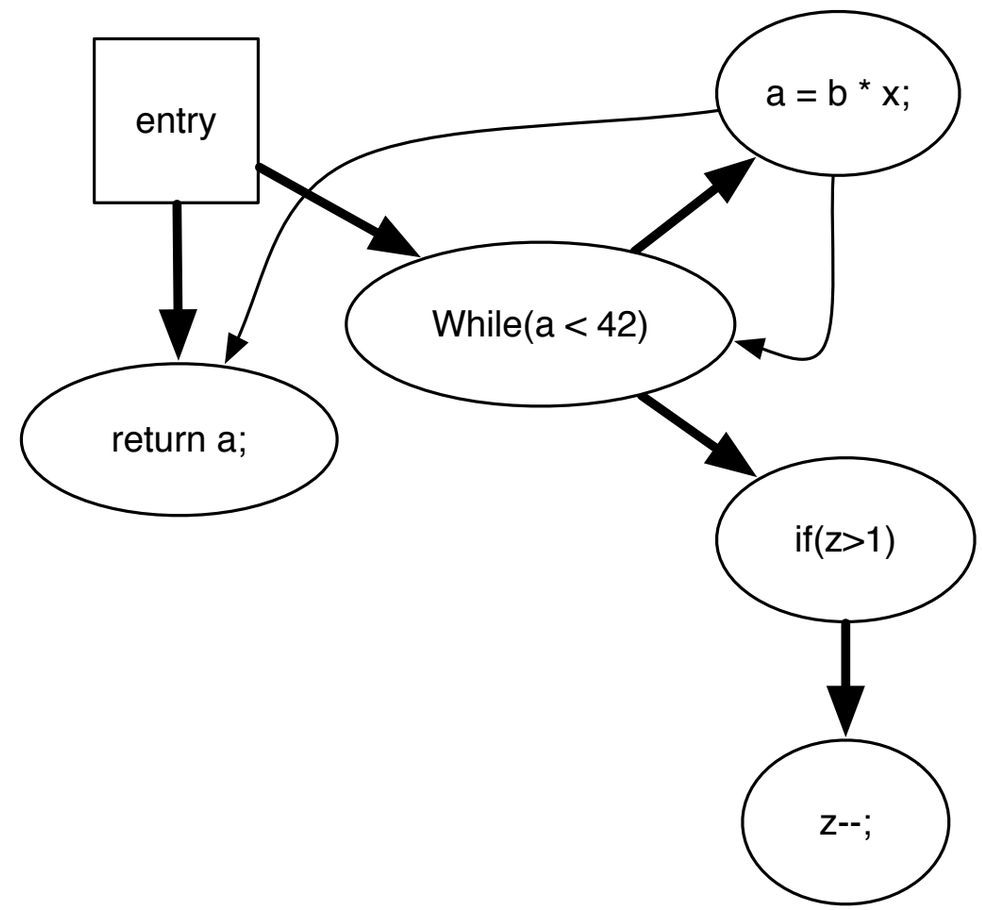
Example Function Dependency Graph

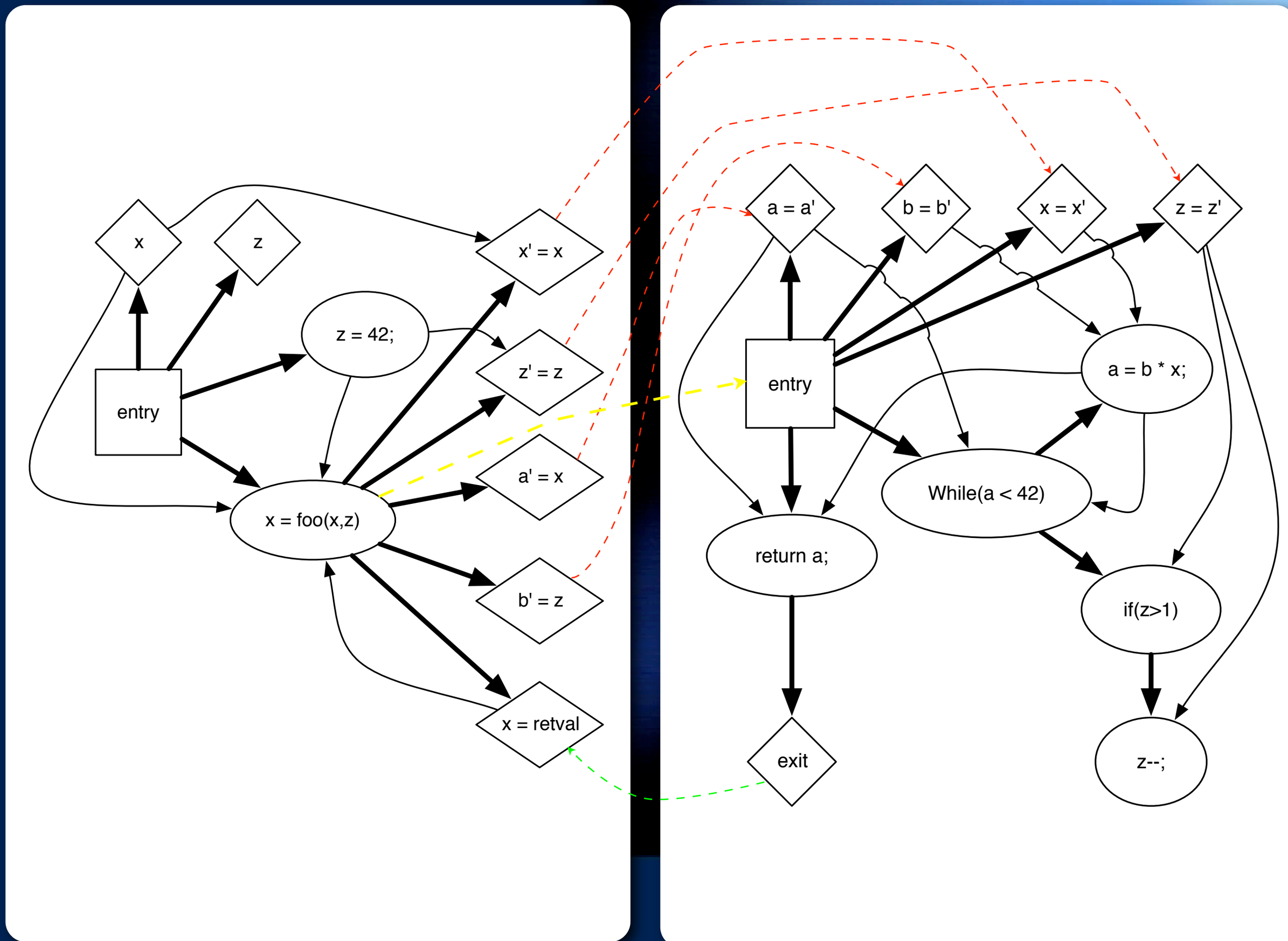
- Identify place of relevance
- and irrelevant code
- Inter-procedural slicing

[S. Horwitz, T. Reps, and D. Binkley]

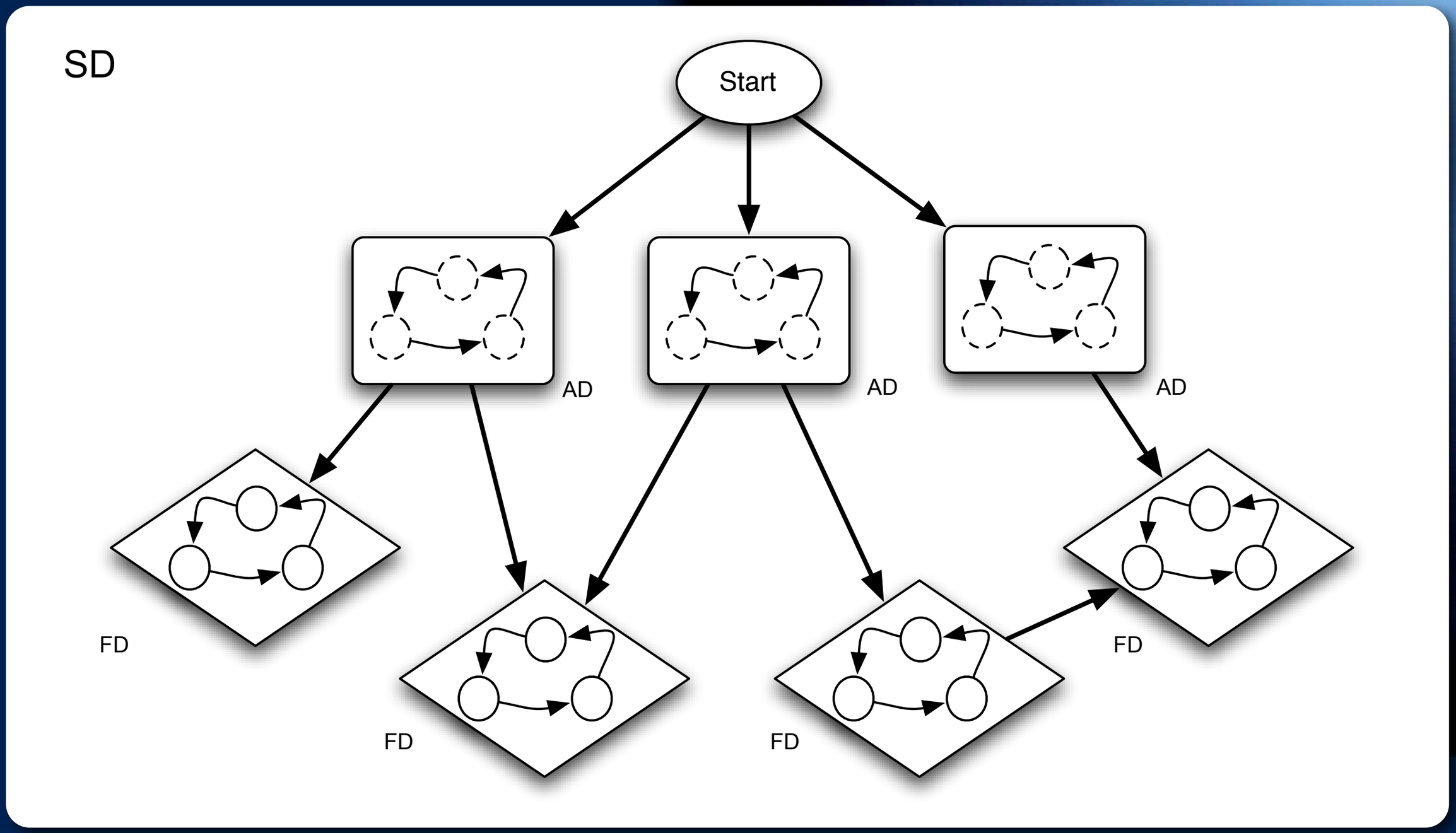
```
int x = 0;
int z = 0;

int foo(int a, int b)
{
        while(a < 42)
        {
                a = b * x;
                if(z > 1)
                {
                        z--;
                }
        }
        return a;
}

void bar()
{
        z = 42;
        x = foo(x,z);
}
```


z = 42;
entry
x = foo(x,z)

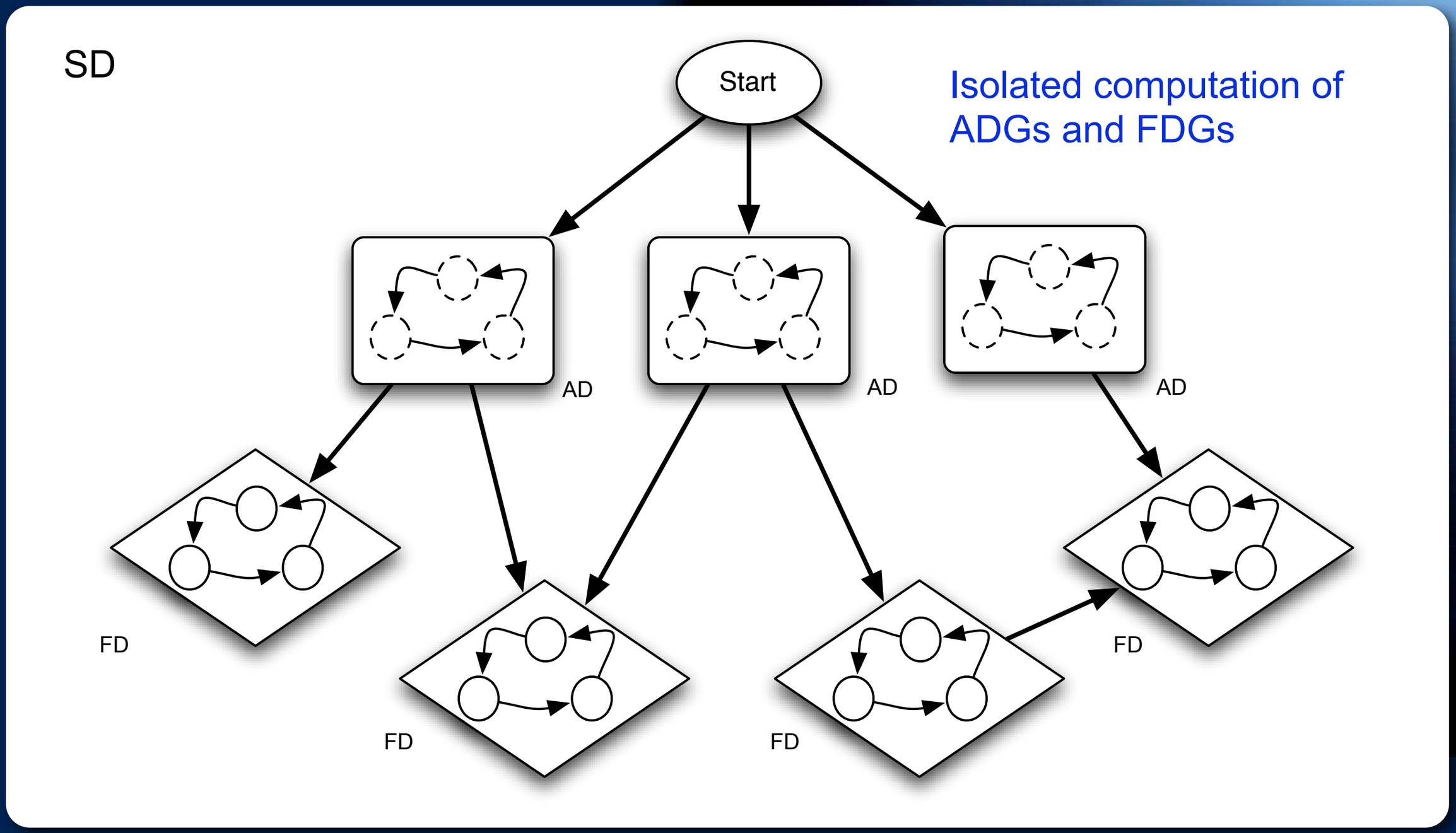

entry
a = b * x;
While(a < 42)
return a;
if(z>1)
z--;


x
z
x' = x
z = 42;
z' = z
entry
a' = x
x = foo(x,z)
b' = z
x = retval
a = a'
b = b'
x = x'
z = z'
entry
a = b * x;
While(a < 42)
return a;
if(z>1)
exit
z--;

System Dependency Graph



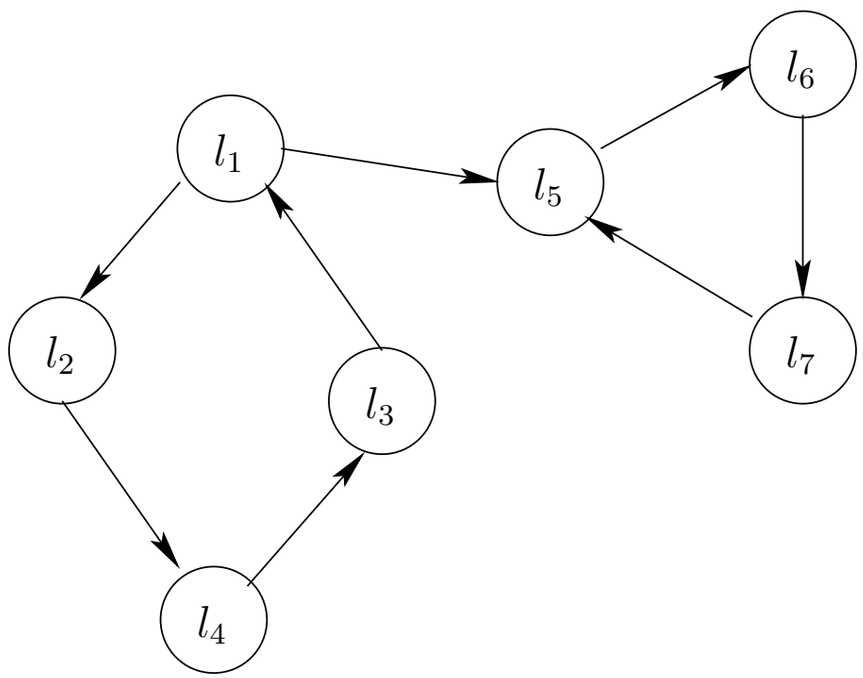
System Dependency Graph



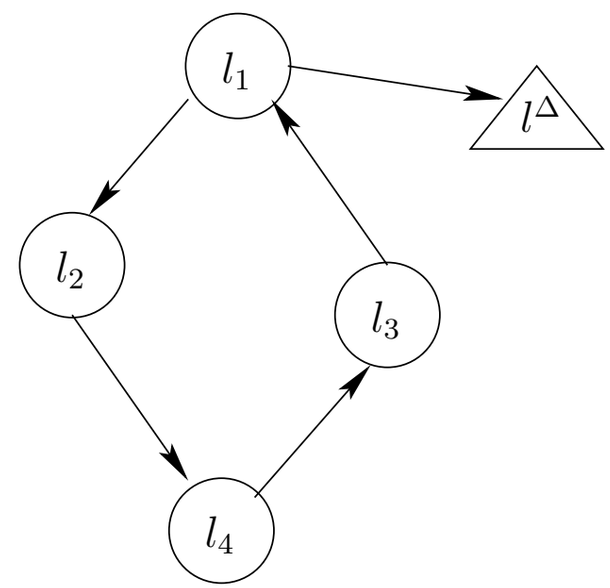
Slicing Extended TA

- Starts at a point of interest e.g. a location in the TA
- Search backwards in the SDG (marking pp)
  - Computes the fixed-point of dependencies in the ADG
  - Searches backward in the called FDGs
- Delete all un-marked statements.

Conservative Automata slicing

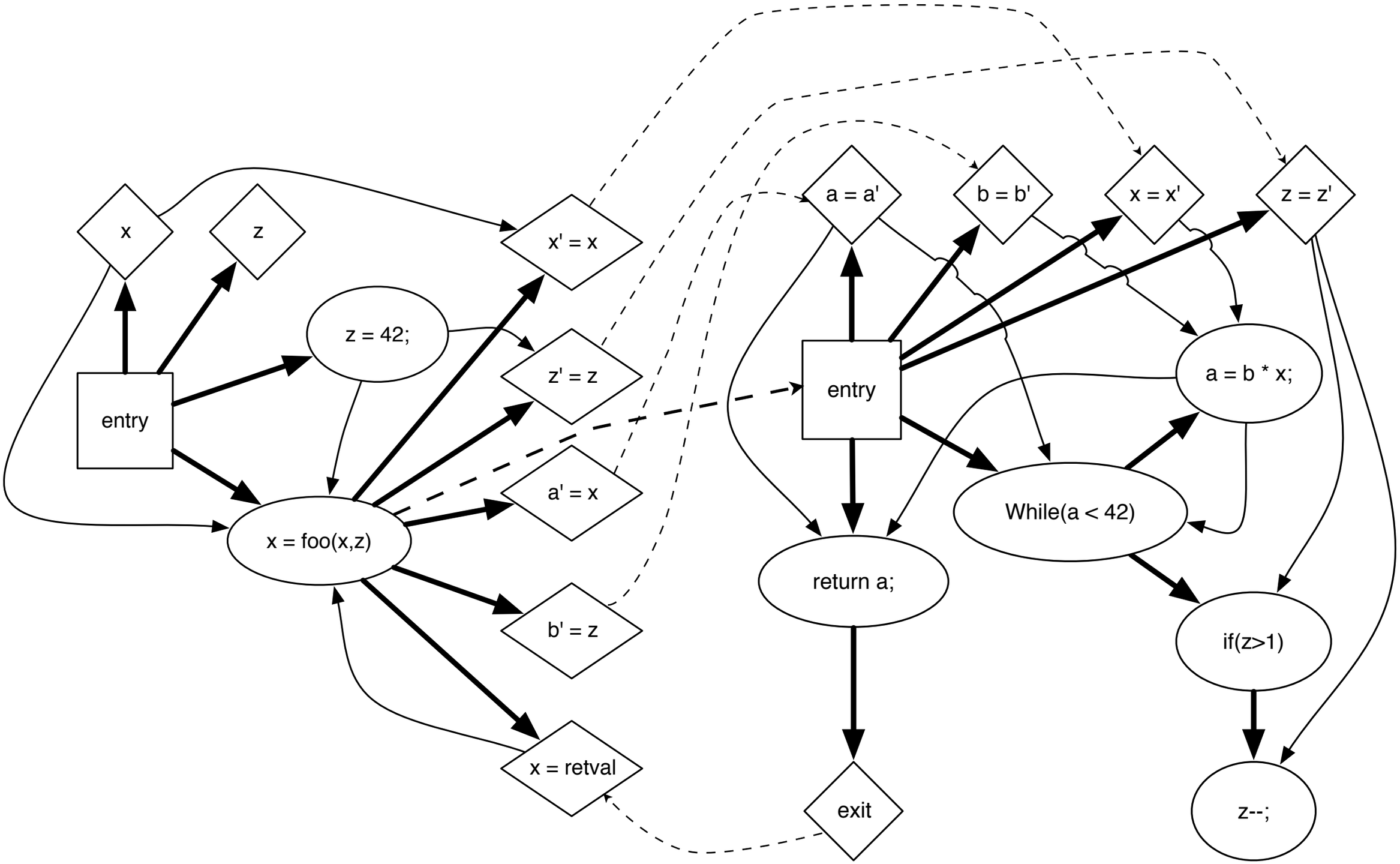


(a) Example Automata

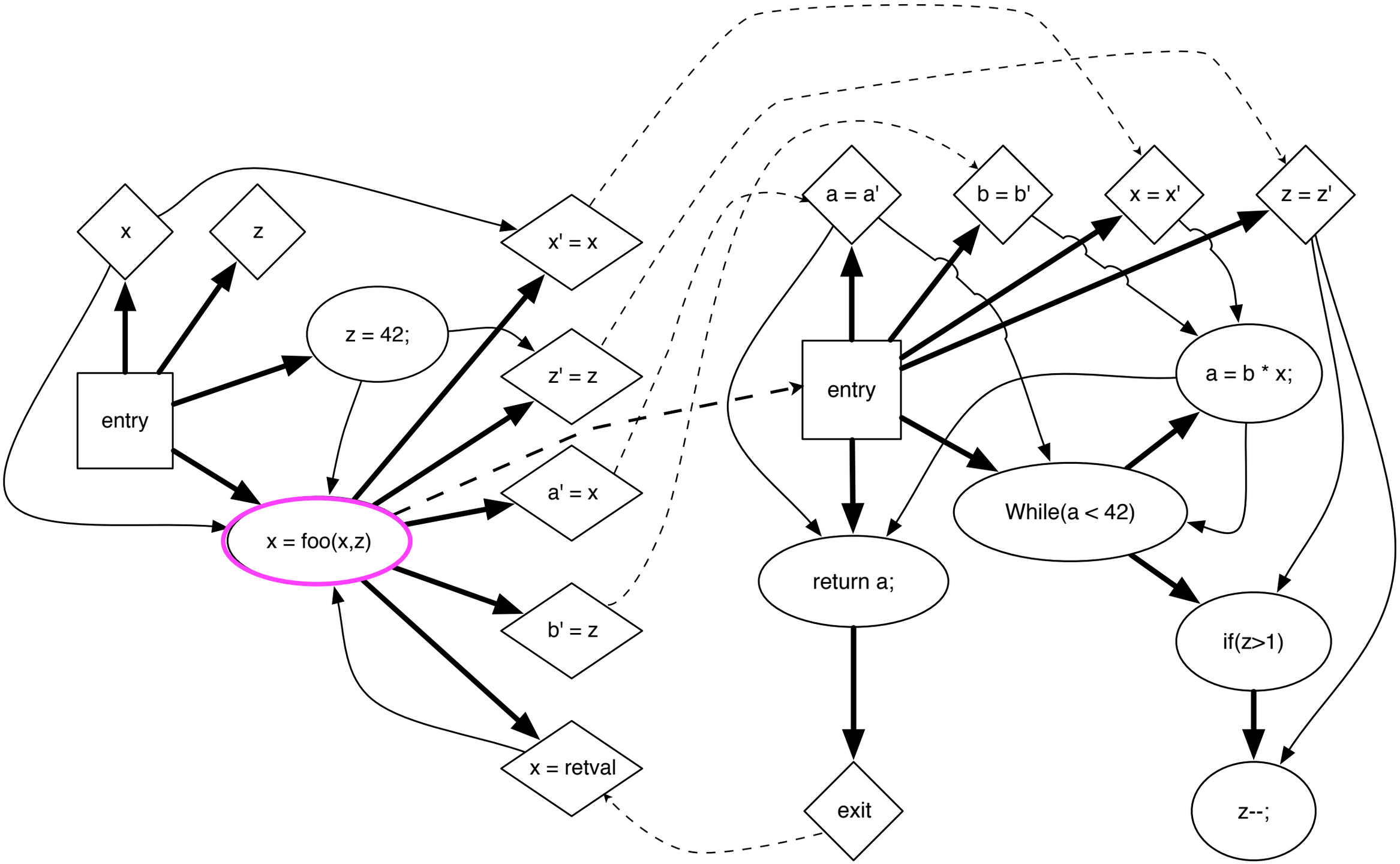


(b) Automata with sink

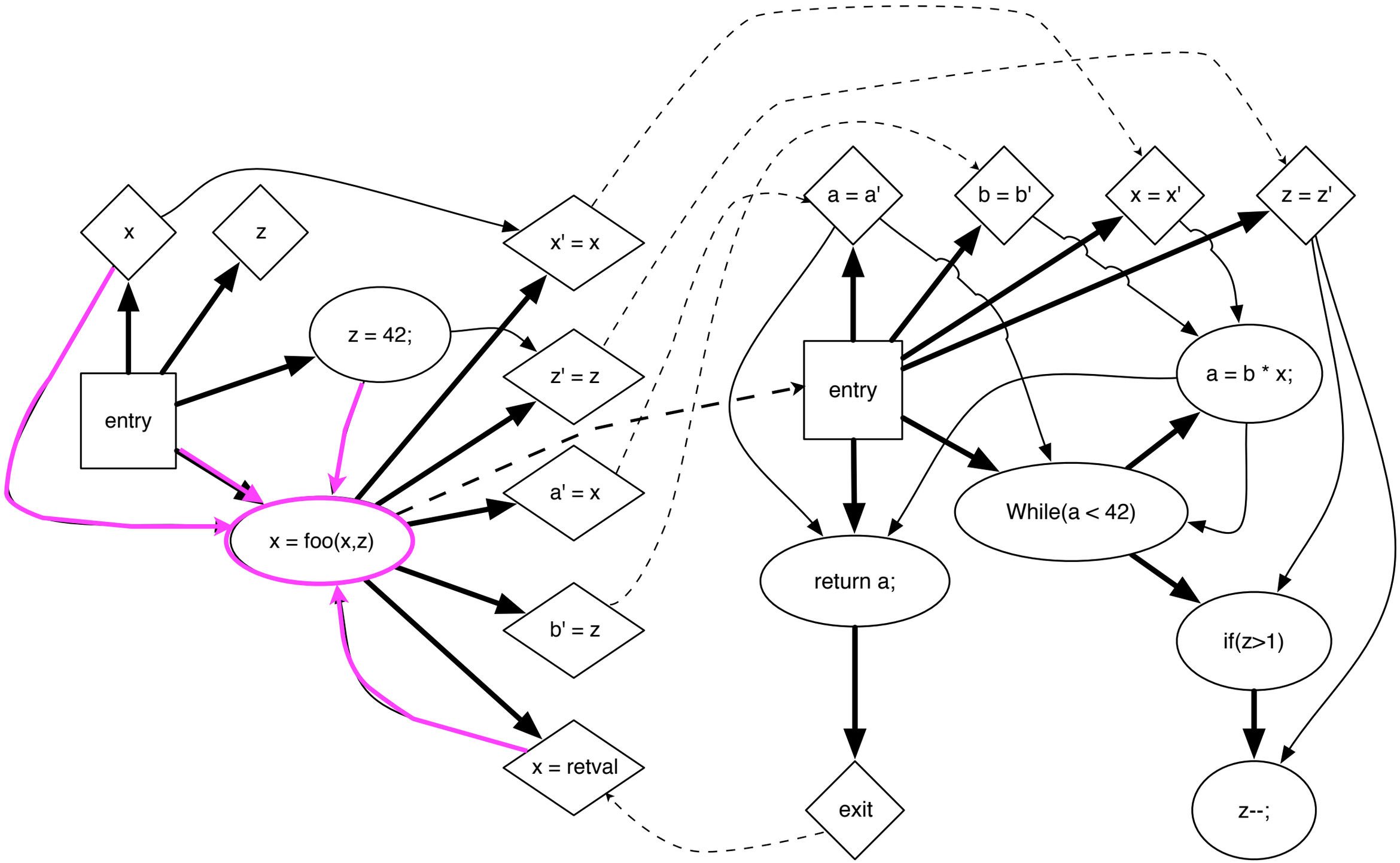
Traversal of imperative code



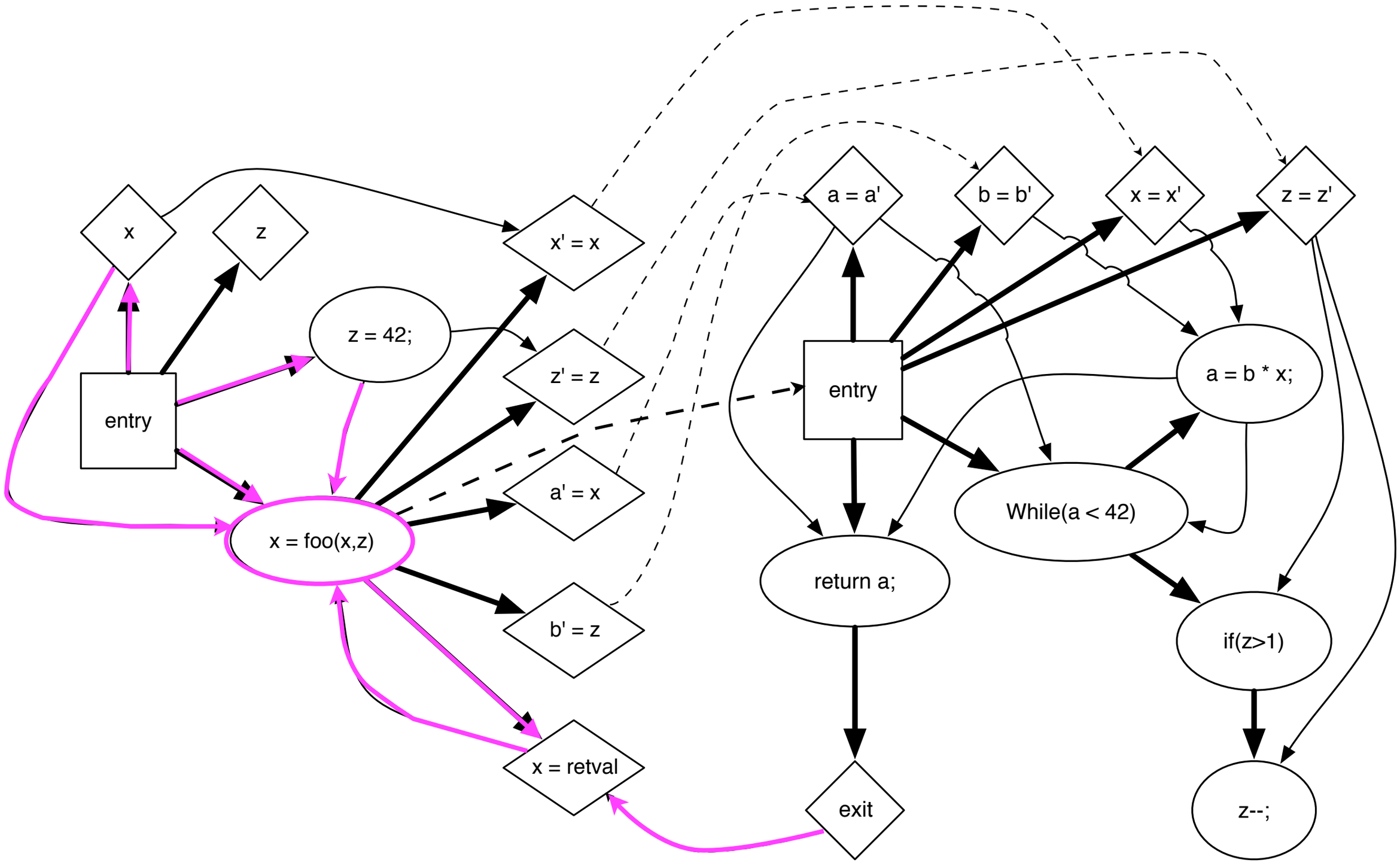
Traversal of imperative code



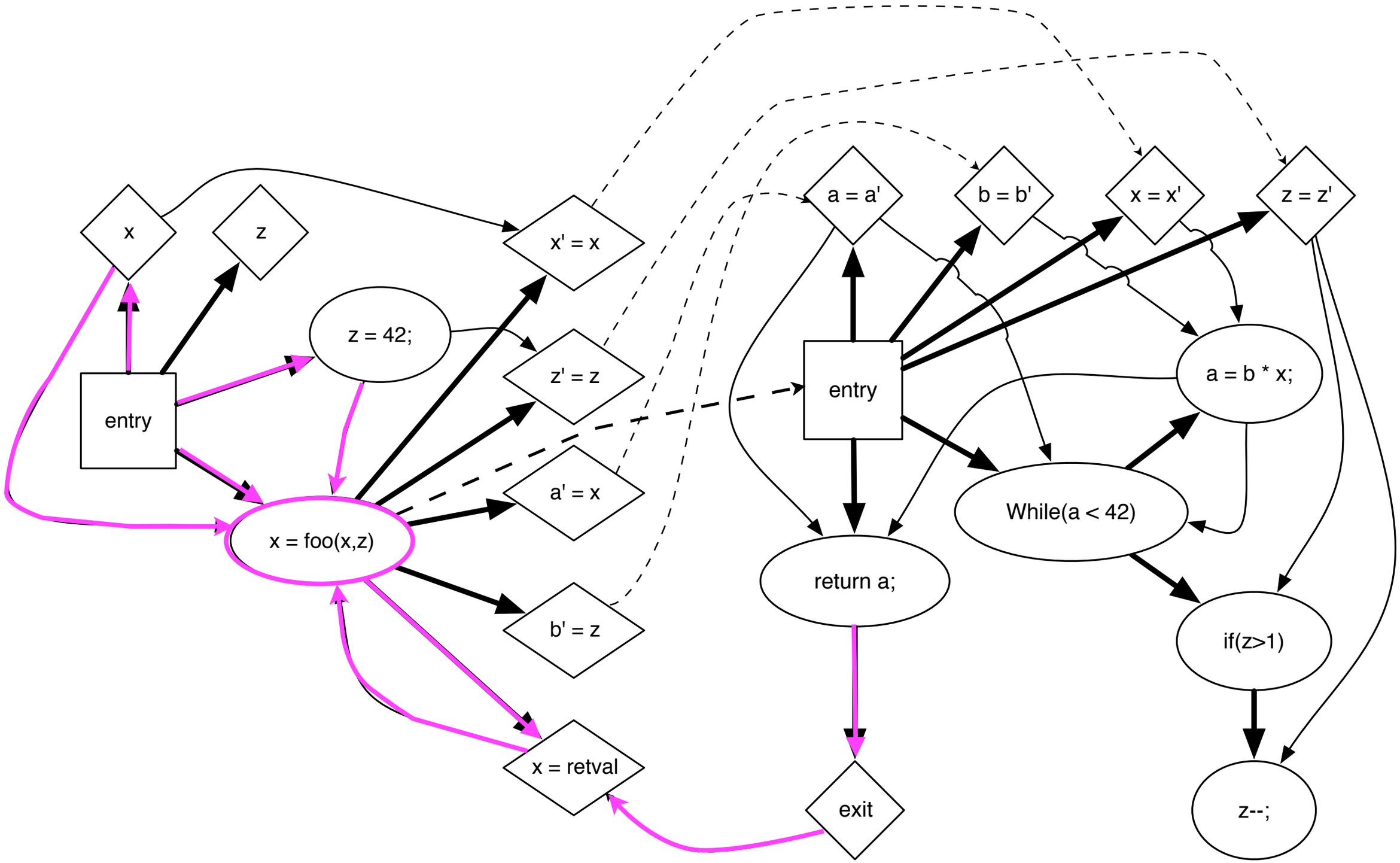
Traversal of imperative code



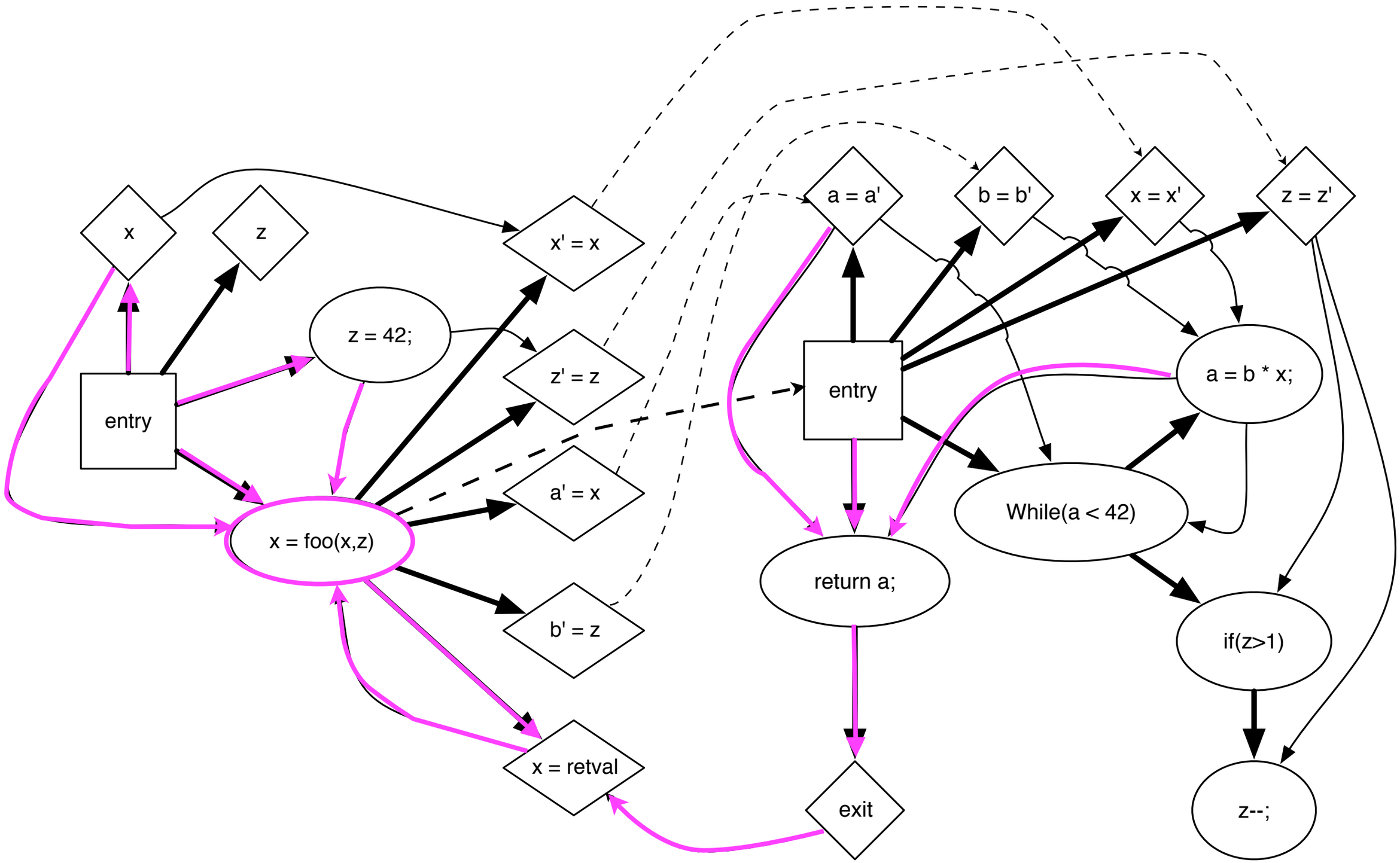
Traversal of imperative code



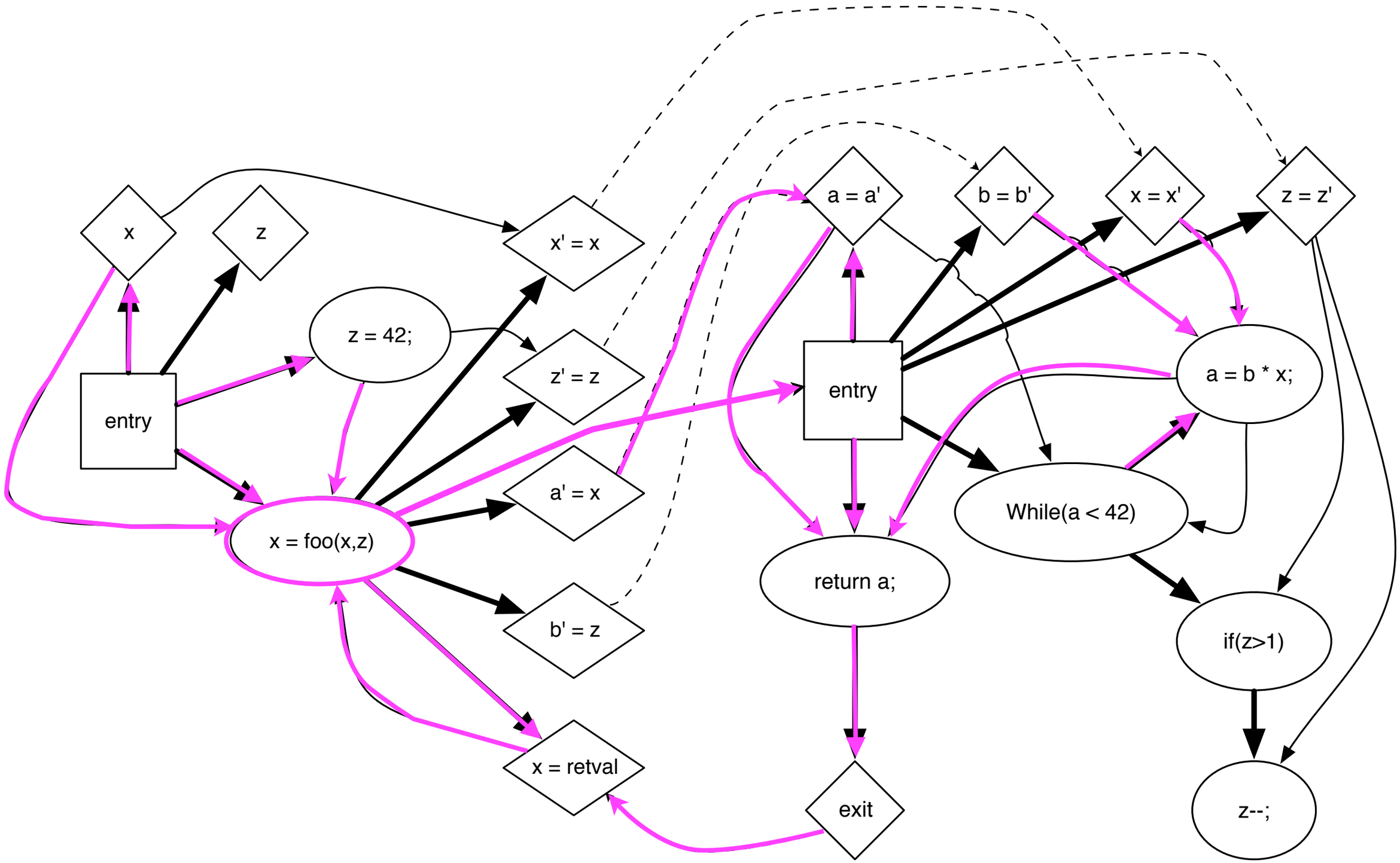
Traversal of imperative code



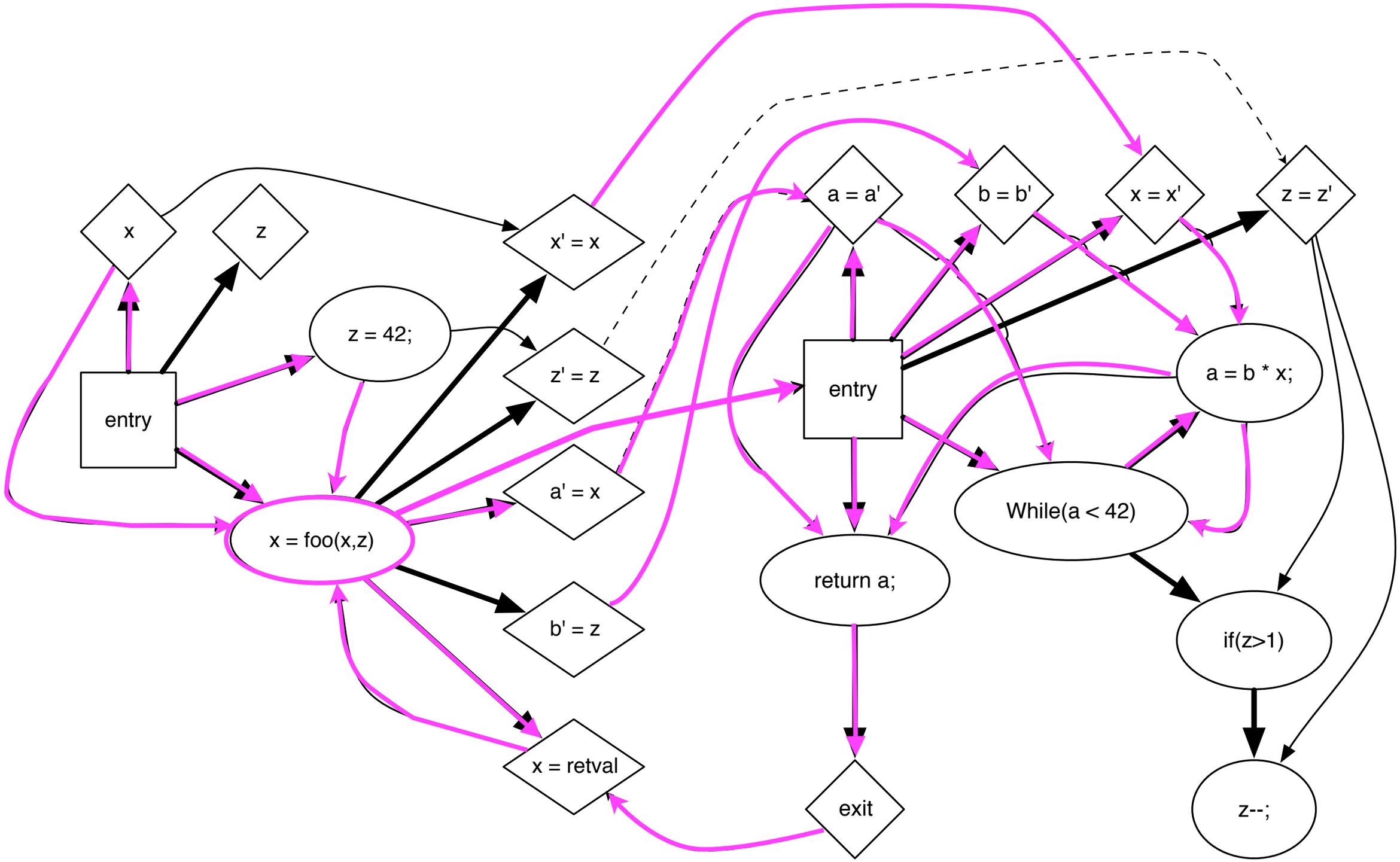
Traversal of imperative code



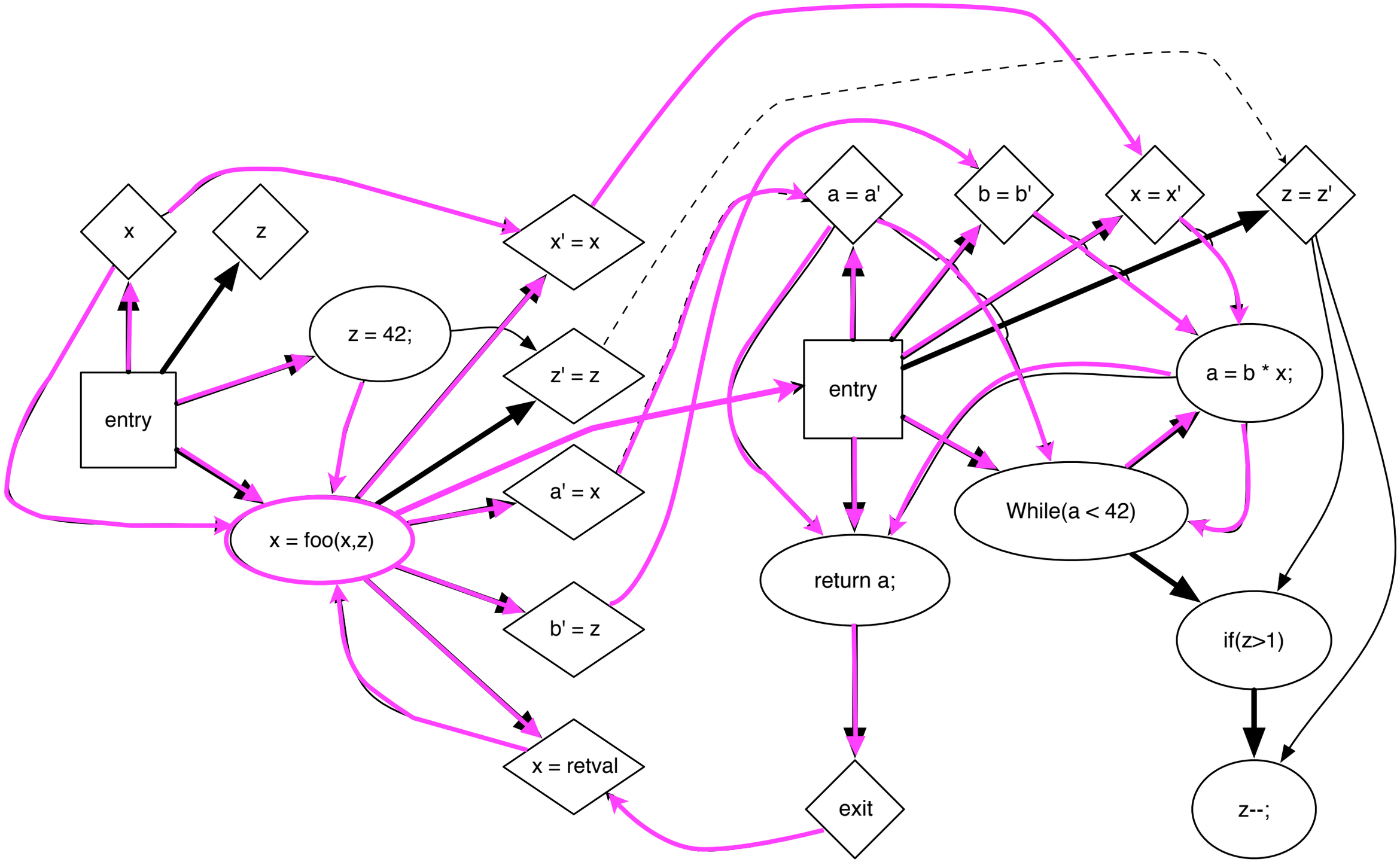
Traversal of imperative code



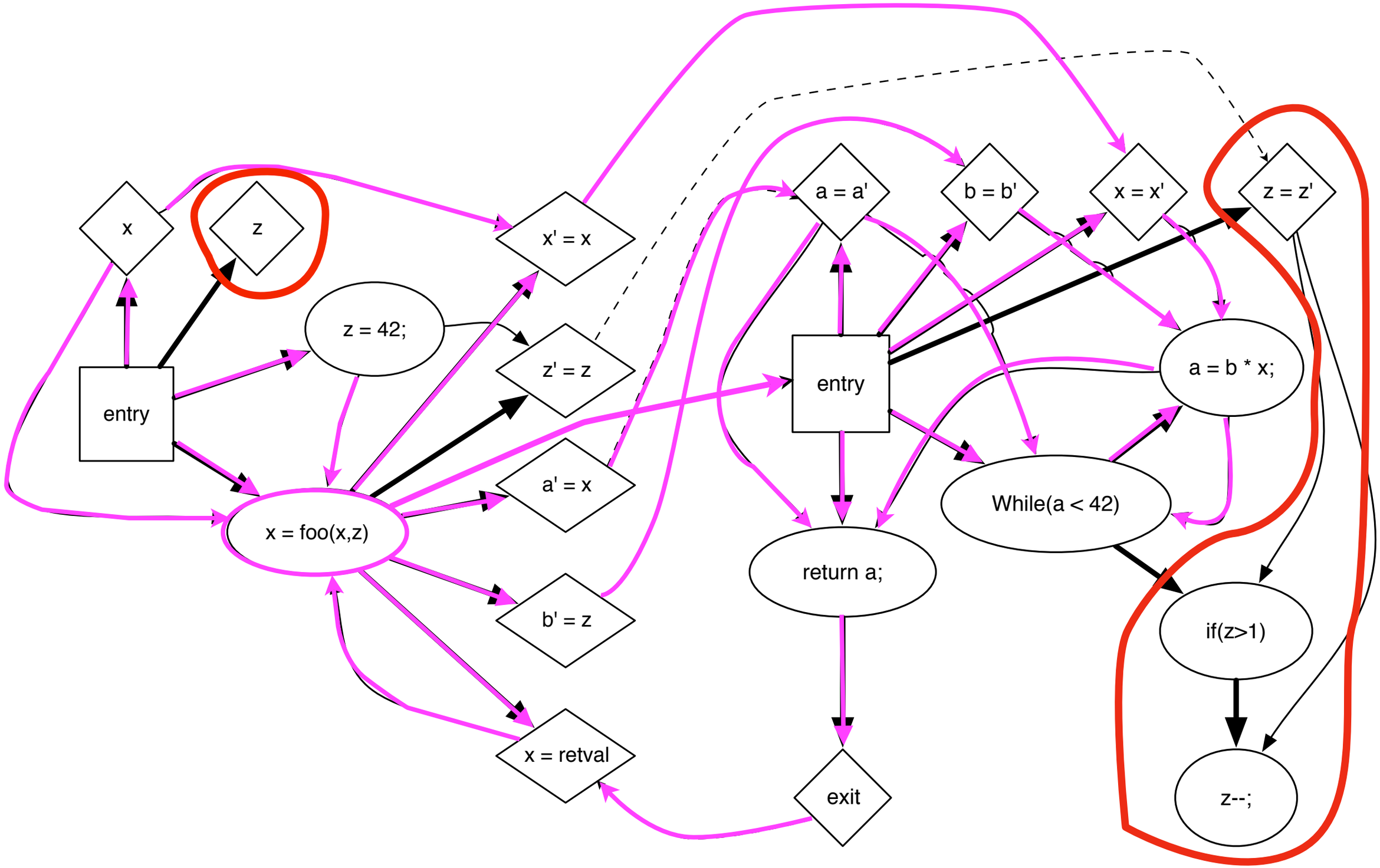
Traversal of imperative code



Traversal of imperative code



Traversal of imperative code



Bisimulation based Correctness

- Reachability preserving slices (CTL)

Lemma 1. *If $s\ R_\varphi\ s'$ for some $E\Diamond\varphi$-Bisimulation then:*

$$s \models E\Diamond\varphi \iff s' \models E\Diamond\varphi$$

Theorem 1. *The relation $\simeq \subseteq S \times S'$ is a $E\Diamond\varphi$-Bisimulation between two structures $M = (S, \mathcal{I})$ and $M' = (S', \mathcal{I}')$.*

Empirical results

Deadlock free	Mapper Example				
	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	85587630+	12	6
After Slicing	11.12sec *	66572KB	786391	6	3
After Fix	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2862KB	2074	12	6
After Slicing	0.10sec	2852KB	199	6	3

Deadlock free	Train-Gate Example				
	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	77636326+	34	14
After Slicing	0.2sec	2848KB	413	28	10
Train may cross	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2856KB	14	34	14
After Slicing	0.10sec	2848KB	14	28	10

VT = Verification Time, MU = Memory Usage, SS = Symbolic States Explored, NS = Number of Statements, NV = Number of Variables

Empirical results

Deadlock free	Mapper Example				
	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	85587630+	12	6
After Slicing	11.12sec *	66572KB	786391	6	3
After Fix	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2862KB	2074	12	6
After Slicing	0.10sec	2852KB	199	6	3

Deadlock free	Train-Gate Example				
	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	77636326+	34	14
After Slicing	0.2sec	2848KB	413	28	10
Train may cross	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2856KB	14	34	14
After Slicing	0.10sec	2848KB	14	28	10

VT = Verification Time, MU = Memory Usage, SS = Symbolic States Explored, NS = Number of Statements, NV = Number of Variables

Empirical results

	Mapper Example				
Deadlock free	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	85587630+	12	6
After Slicing	11.12sec *	66572KB	786391	6	3
After Fix	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2862KB	2074	12	6
After Slicing	0.10sec	2852KB	199	6	3

	Train-Gate Example				
Deadlock free	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	77636326+	34	14
After Slicing	0.2sec	2848KB	413	28	10
Train may cross	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2856KB	14	34	14
After Slicing	0.10sec	2848KB	14	28	10

VT = Verification Time, MU = Memory Usage, SS = Symbolic States Explored, NS = Number of Statements, NV = Number of Variables

Empirical results

	Mapper Example				
Deadlock free	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	85587630+	12	6
After Slicing	11.12sec *	66572KB	786391	6	3
After Fix	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2862KB	2074	12	6
After Slicing	0.10sec	2852KB	199	6	3

	Train-Gate Example				
Deadlock free	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	77636326+	34	14
After Slicing	0.2sec	2848KB	413	28	10
Train may cross	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2856KB	14	34	14
After Slicing	0.10sec	2848KB	14	28	10

VT = Verification Time, MU = Memory Usage, SS = Symbolic States Explored, NS = Number of Statements, NV = Number of Variables

Empirical results

	Mapper Example				
Deadlock free	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	85587630+	12	6
After Slicing	11.12sec *	66572KB	786391	6	3
After Fix	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2862KB	2074	12	6
After Slicing	0.10sec	2852KB	199	6	3

	Train-Gate Example				
Deadlock free	VT	MU	SS	NS	NV
Before Slicing	N/A	4GB+	77636326+	34	14
After Slicing	0.2sec	2848KB	413	28	10
Train may cross	VT	MU	SS	NS	NV
Before Slicing	0.11sec	2856KB	14	34	14
After Slicing	0.10sec	2848KB	14	28	10

VT = Verification Time, MU = Memory Usage, SS = Symbolic States Explored, NS = Number of Statements, NV = Number of Variables

Conclusion

- Slicing is highly beneficial for UPPAAL
- Conservative - reachability preserving algorithm
  - Prototype implementation
- Supports iterative Development / Design
- Side-effect: “encourage” new users

Future

- Further Experiments
- Consolidation of Library (production ready code)
- Complete SSA Form Transformation (C-code)
- Integration in UPPAAL
- Integrate value range propagation (on int vars)

- Visualization of slices in UPPAALs GUI
- Research “Manual” Slicing

Thank you

- Summary:
  - Applied the SDG for imperative code and TA
  - Proof that slicing preserves (CTL) reachability
  - Prototype implementation
    - Generic static analysis library for Uppaal
  - Future work (highlights)
    - Further empirical studies
    - Consolidation of library code