#30: Synthesis with Abstract Interpretation

Sankha Narayan Guria

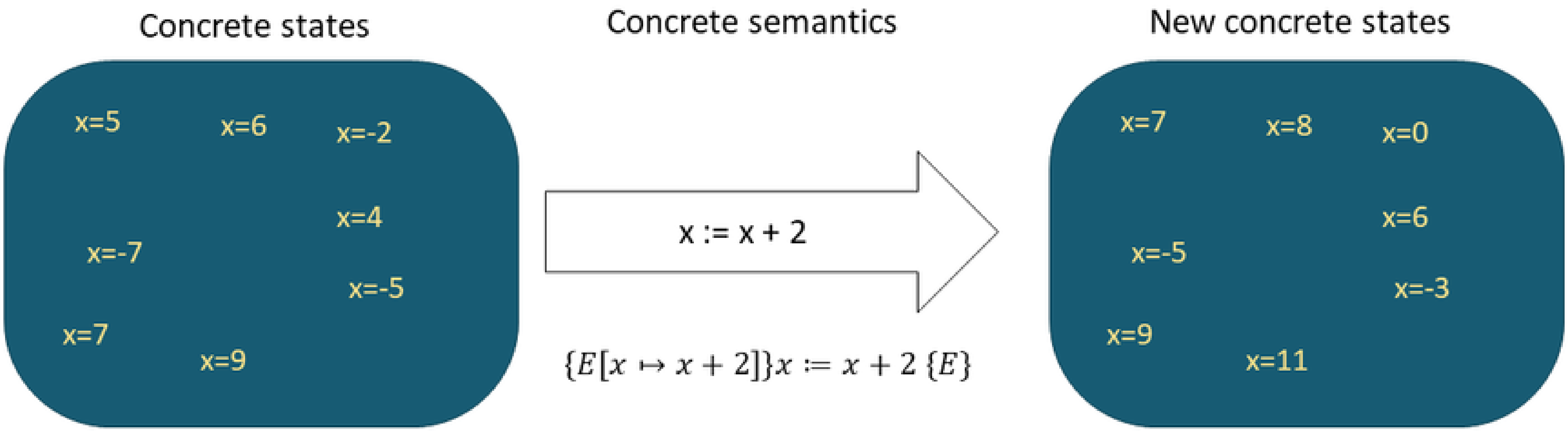
EECS 700: Introduction to Program Synthesis

Today

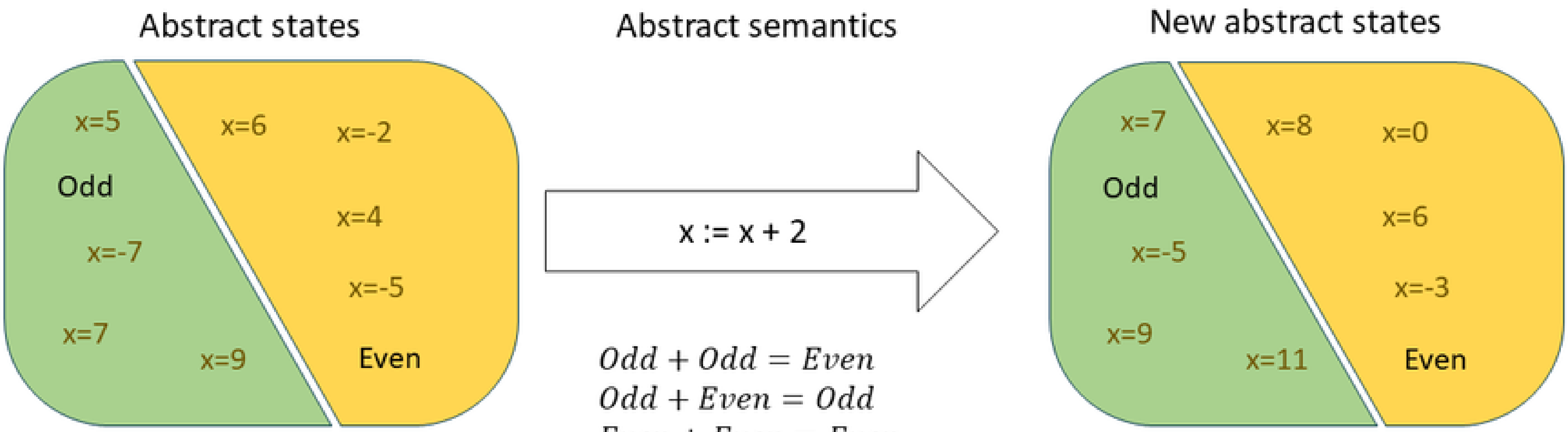
- Synthesizing data-structure manipulation from storyboards
  - Rishabh Singh, Armando Solar-Lezama

- Absynthe: Abstract Interpretation-Guided Synthesis
  - Sankha Narayan Guria, Jeff Foster, David Van Horn

Key idea 1: Abstract domain

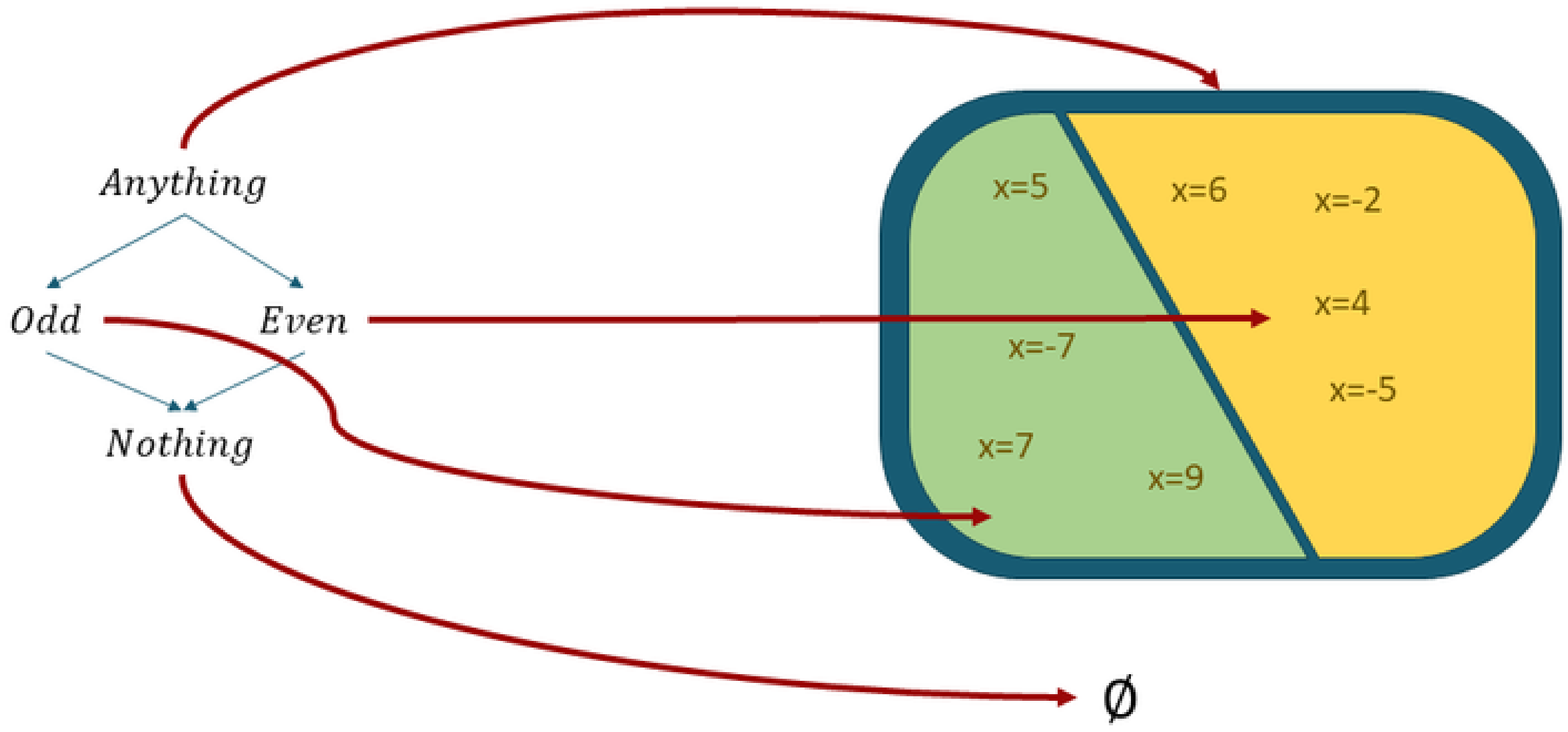


Key idea 1: Abstract domain

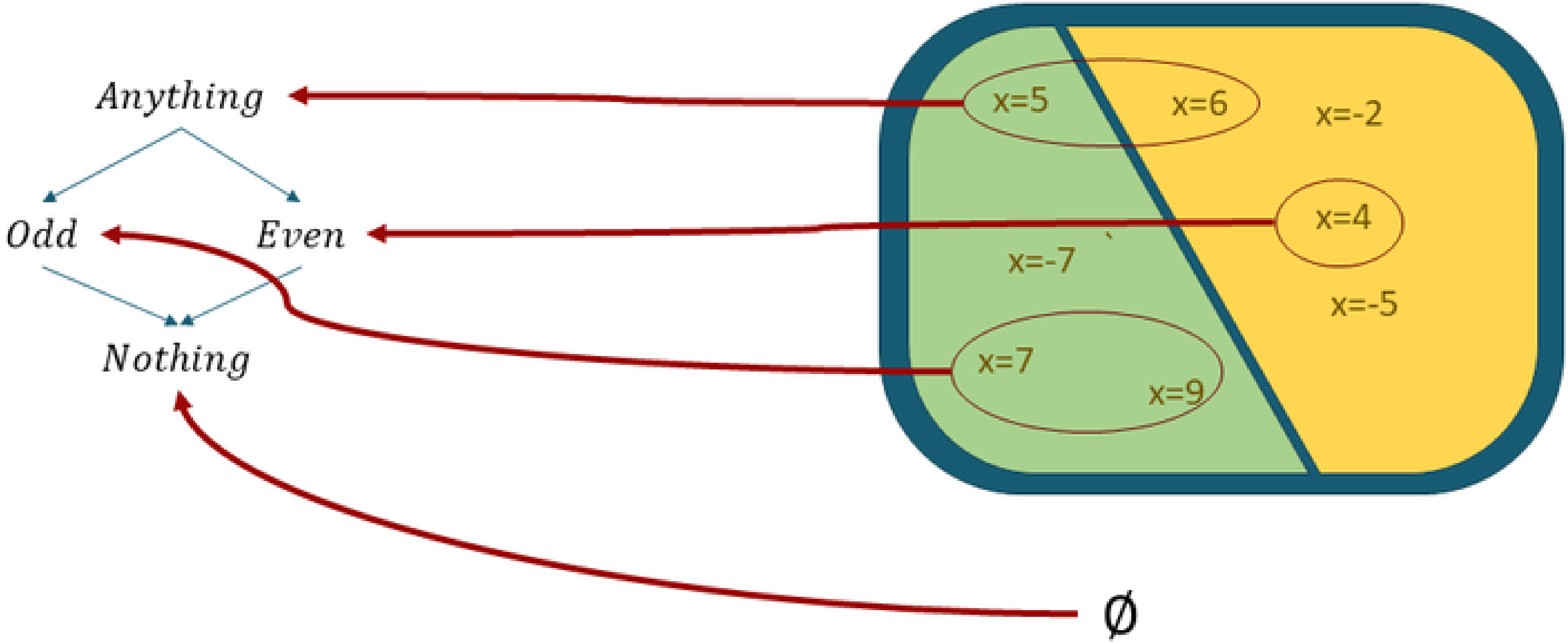


$$Odd + Odd = Even$$
$$Odd + Even = Odd$$
$$Even + Even = Even$$
$$Even + Odd = Odd$$

Concretization



Abstraction



Key idea 2: Abstract Interpretation

Compute an abstract value for every program point

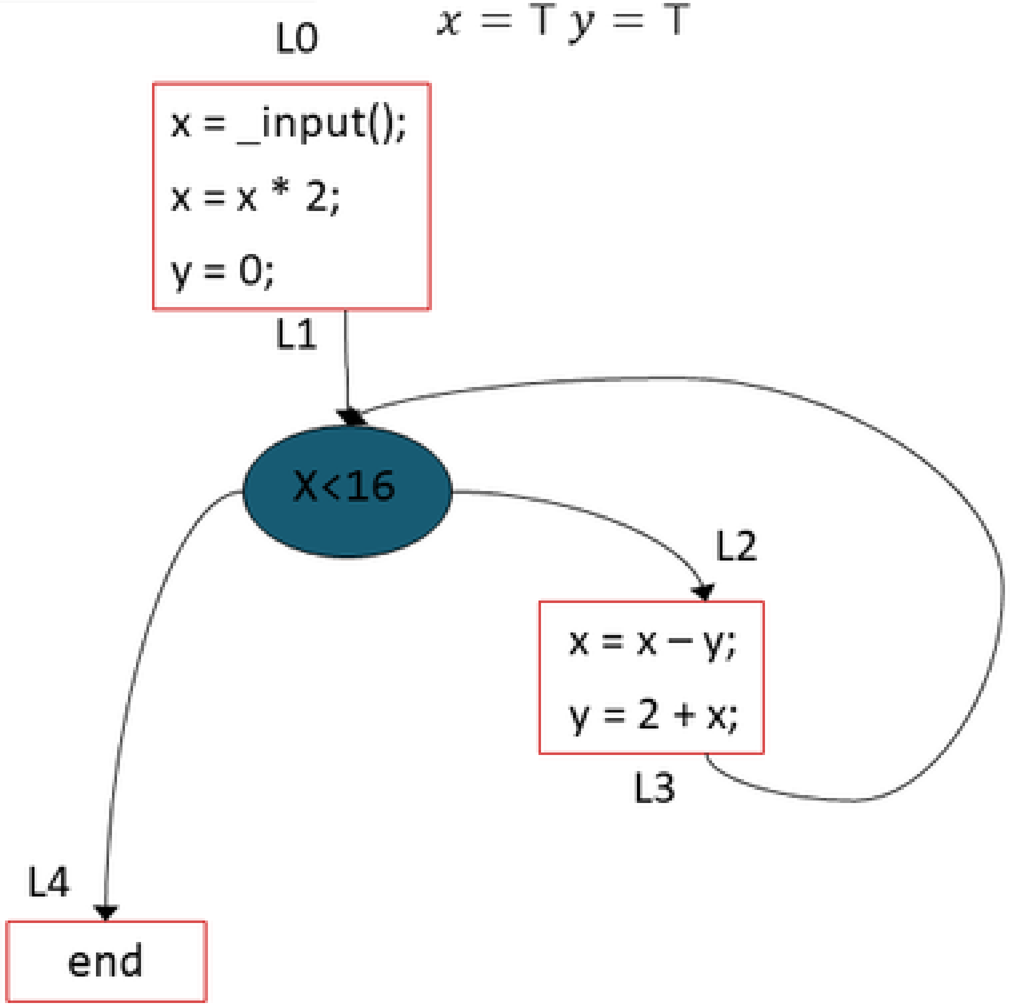
- Abstraction of the set of states possible at that point

Iterate until computation converges

Example

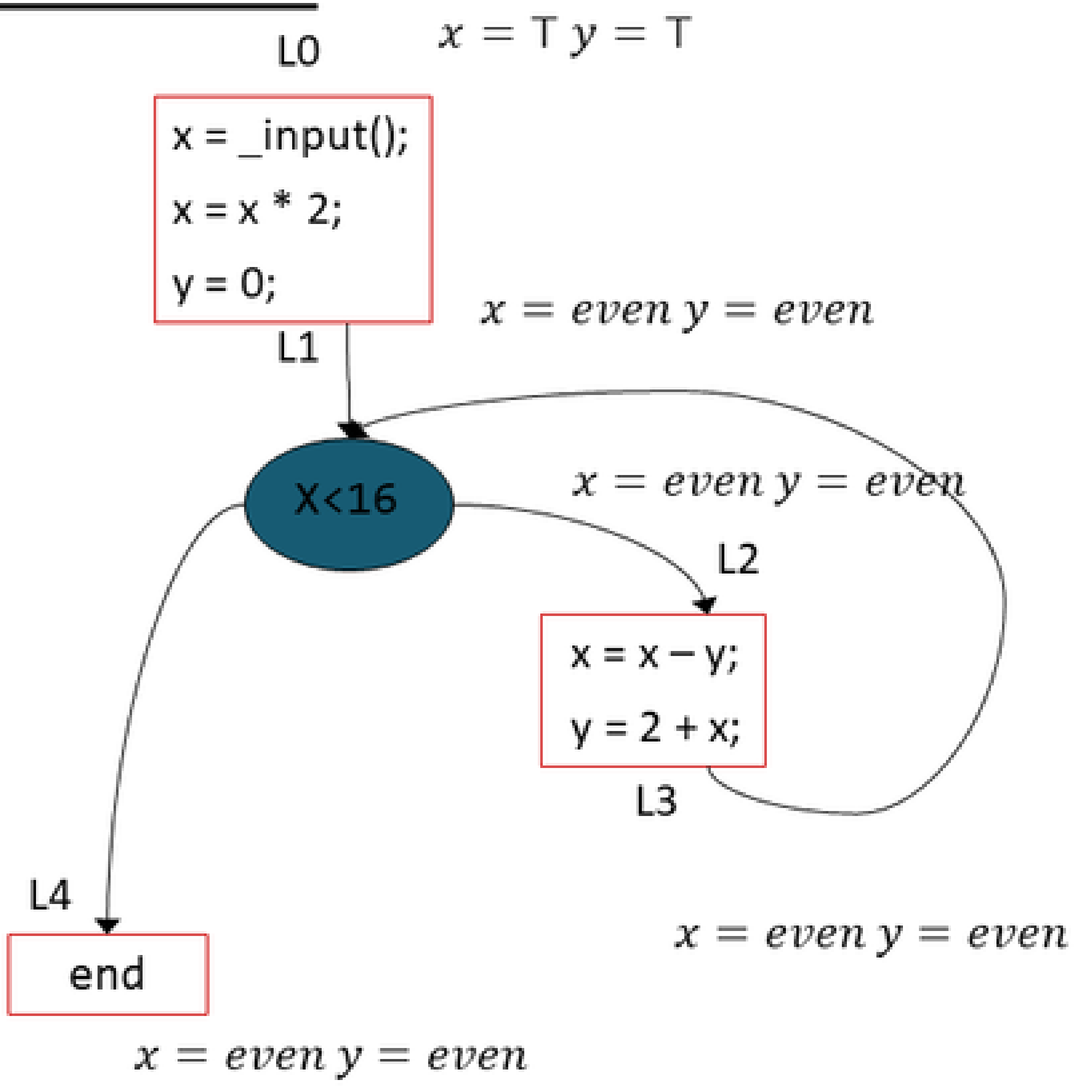
$x = \top \; y = \top$

```
L0
        x = _input();
        x = x * 2;
        y = 0;
L1      while(x < 16){
L2        x = x – y;
          y = 2 + x;
L3      }

L4
```

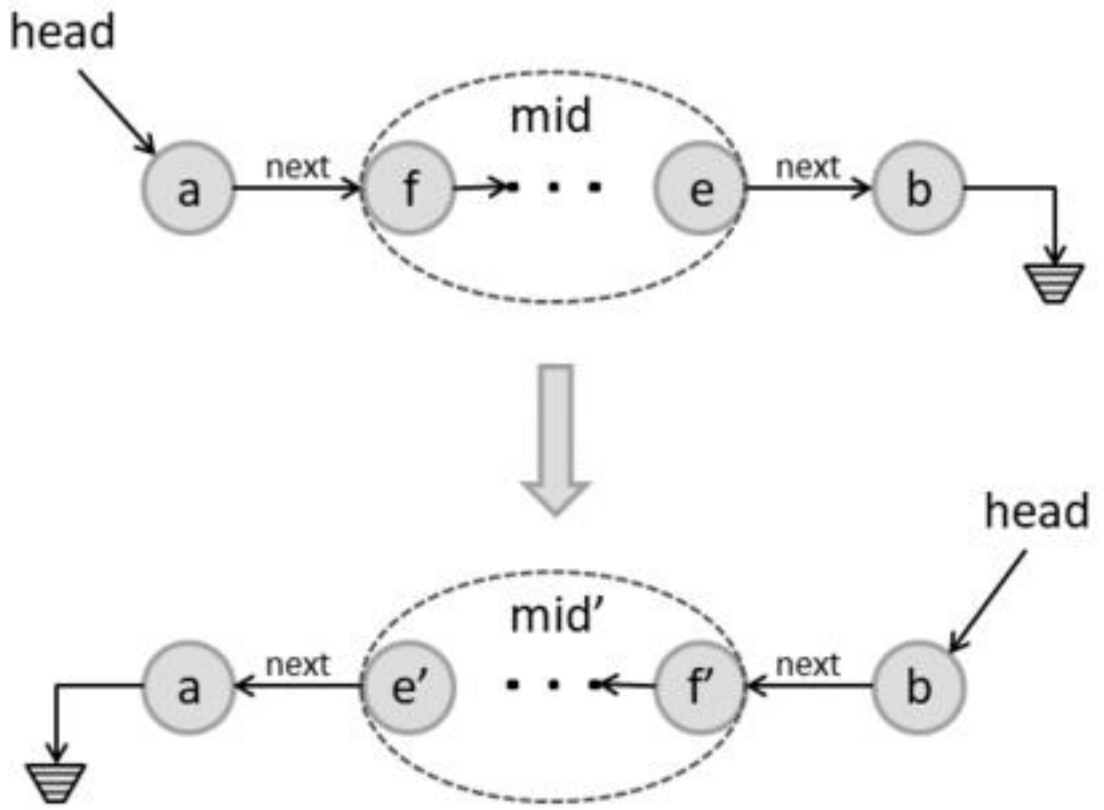


Example

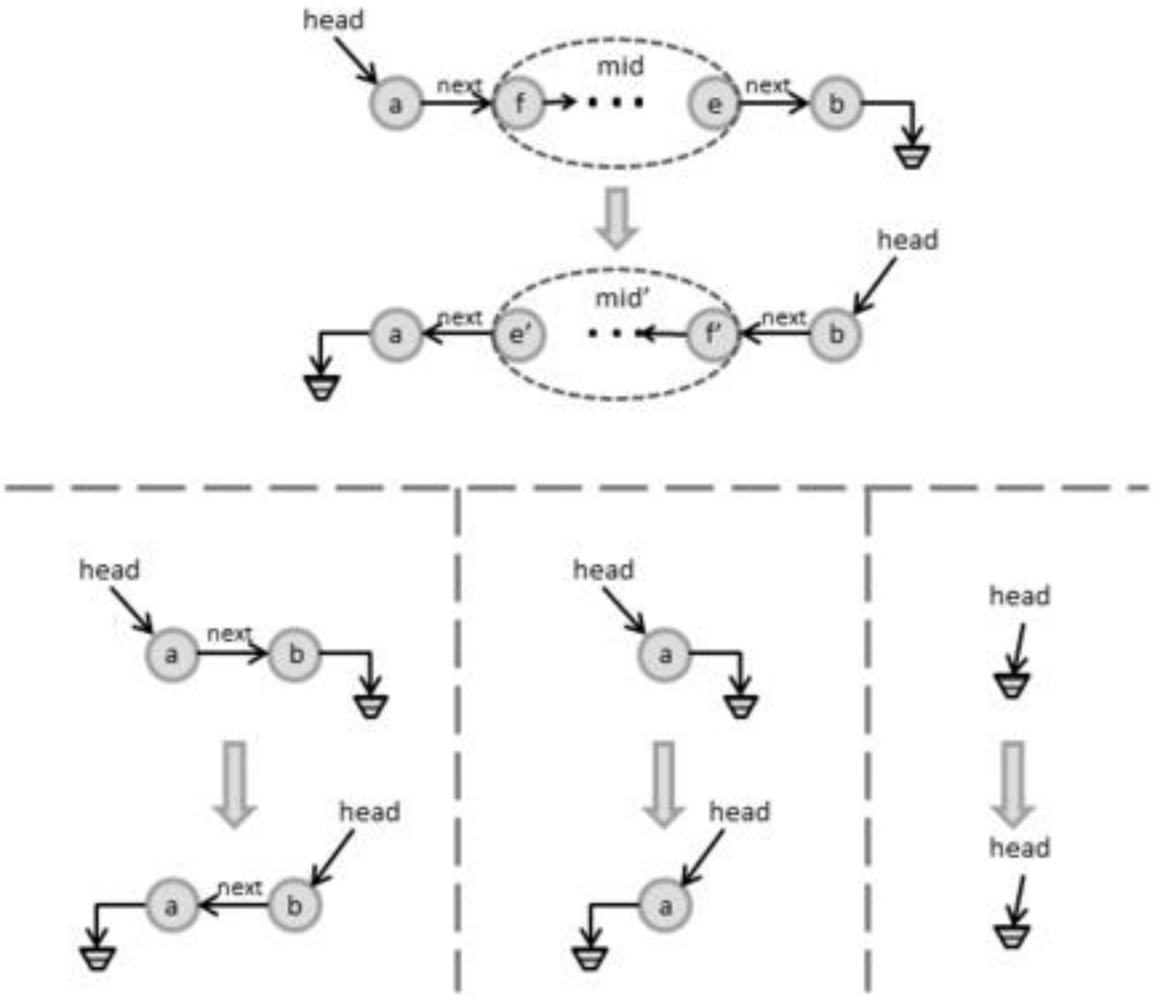
```
L0
        x = _input();
        x = x * 2;
        y = 0;
L1      while(x < 16){
L2        x = x – y;
          y = 2 + x;
L3      }

L4
```



Storyboard Programming

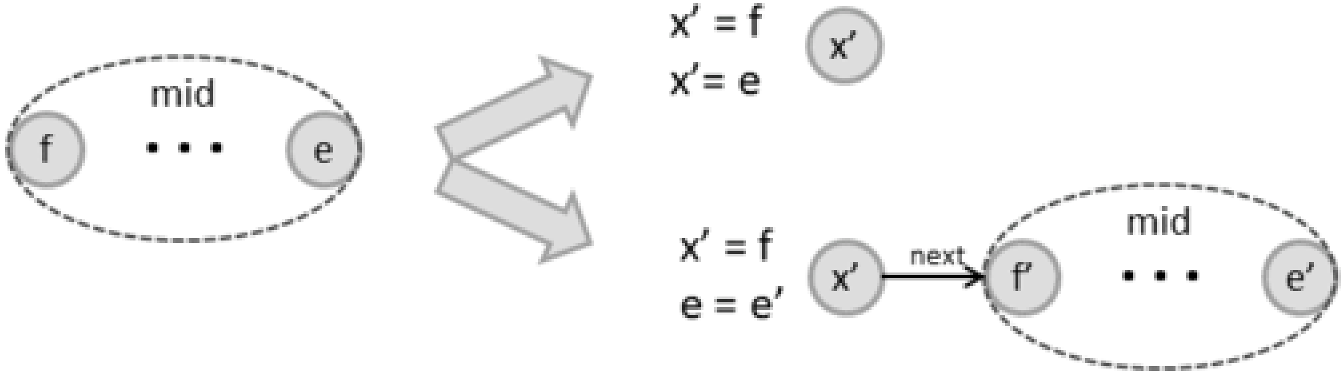


Scenarios for LL-reversal



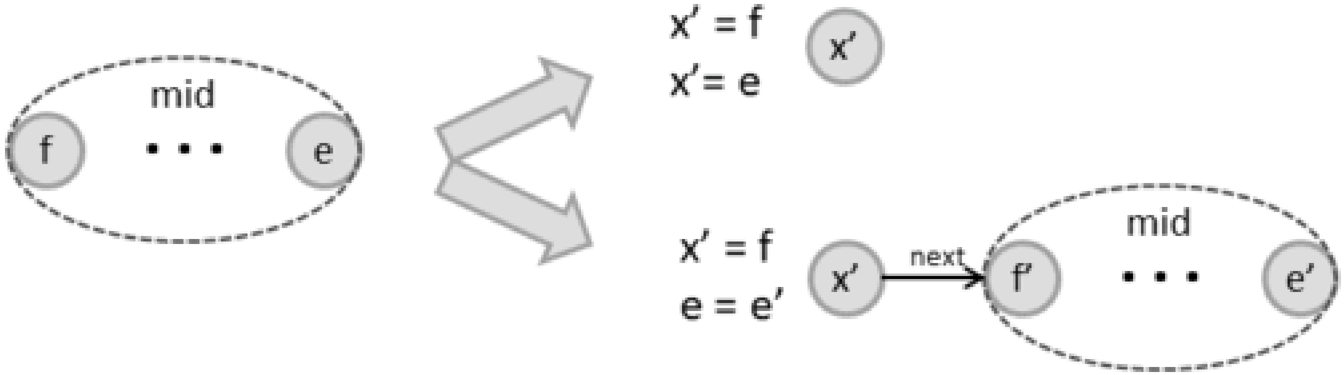
Inductive insights with fold/unfold

Unfold:

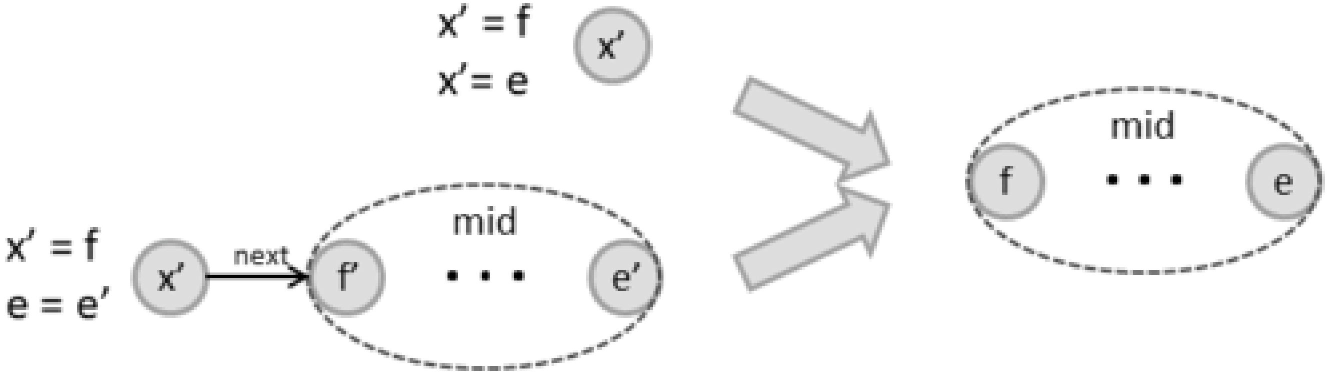


Inductive insights with fold/unfold

Unfold:



Fold:



Concrete Domain

Memory locations: $\mathcal{L}^{\#}$

Variables: $v_0, v_1, \ldots v_k$

Variable predicates: $v_i: \mathcal{L}^{\#} \rightarrow \text{Bool}$ $v_i(l)$ indicates that variable v_i points to loc l

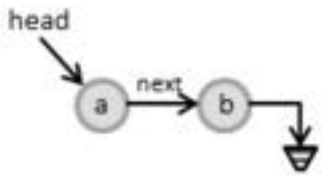
Fields: $sel_0, sel_1, \ldots sel_k$

Field predicates : $sel_0: \mathcal{L}^{\#} \times \mathcal{L}^{\#} \rightarrow Bool$
$sel_i(l_1, l_2)$ indicates that there is a field sel_i from object l_1 to object l_2

$\mathcal{L}^{\#} = \{a, b\}$

$head(a) = true$
$head(b) = false$

$Next$	a	b
a	false	true
b	false	false



Abstract Domain

Abstract memory locations: $\mathcal{L}$

represents a set of concrete locations

Summary location indicator: $sm: \mathcal{L} \rightarrow Tree\ Valued\ Logic\ (TVL)$

indicates if a location represents more than one concrete loc

Attachment Points: $\mathcal{A}: \mathcal{L} \rightarrow \{\mathcal{L}\}$

maps a summary node to a set of locations that serve as attachment points

Variable predicates: $v_i: \mathcal{L} \rightarrow \mathrm{TVL}$ $v_i(l)$ indicates that variable v_i points to loc l

Field predicates : $sel_0: \mathcal{L} \times \mathcal{L} \rightarrow TVL$

$sel_i(l_1, l_2)$ indicates that there is a field sel_i from object l_1 to object l_2

$\mathcal{L} = \{a, f, e, mid, b\}$

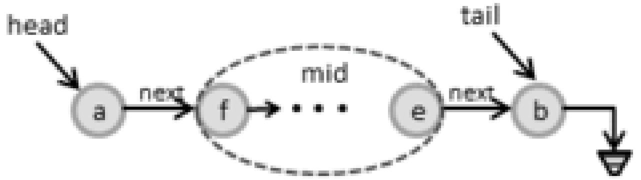
sm	
a	false
f	false
e	false
mid	true
b	false

$\mathcal{A}$	
mid	$\{f, e\}$

$head$	
a	true
f	false
e	false
mid	false
b	false

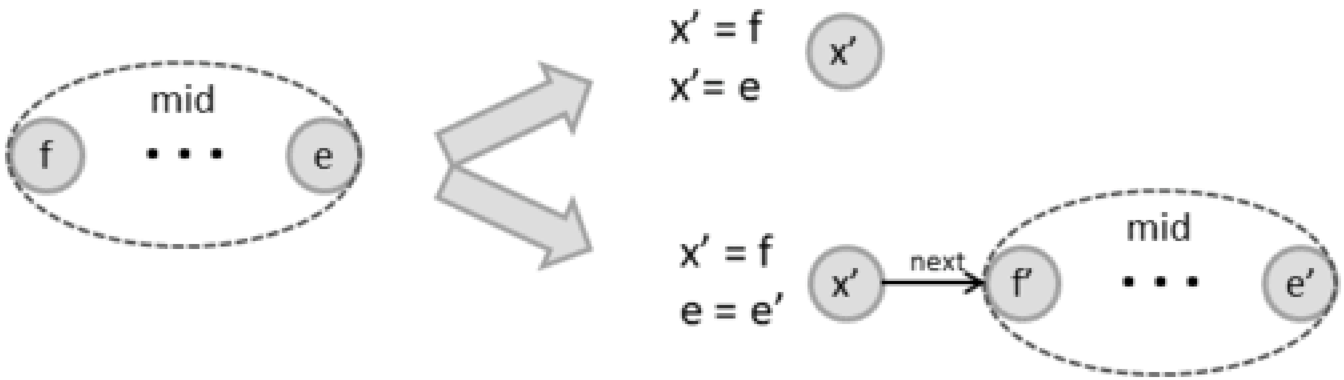
$tail$	
a	false
f	false
e	false
mid	false
b	true

$next$	a	f	e	mi	b
a	F	T	/	/	F
f	F	F	F	/	F
e	F	F	F	F	T
mid	F	F	/	F	/
b	F	F	F	F	F

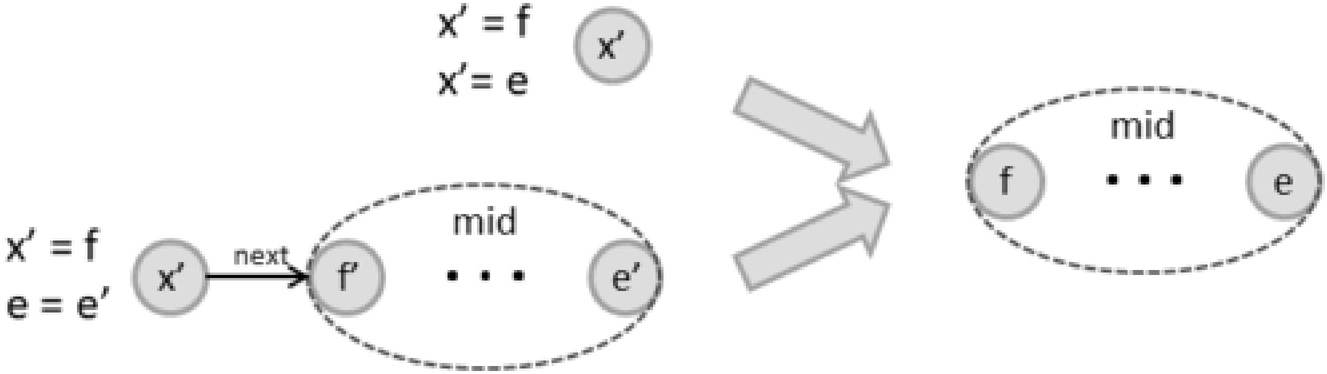


Inductive insights with fold/unfold

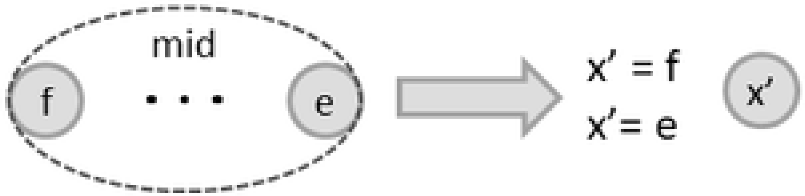
Unfold:



Fold:



Unfold



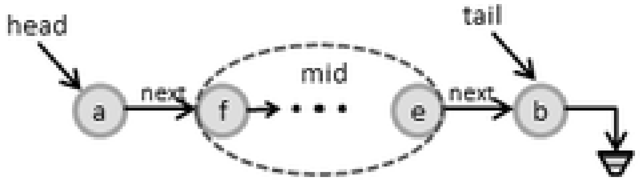
sm	
a	false
f	false
e	false
mid	true
b	false

$\mathcal{A}$	
mid	$\{f, e\}$

$head$	
a	true
f	false
e	false
mid	false
b	false

$tail$	
a	false
f	false
e	false
mid	false
b	true

$next$	a	f	e	$mi($	b
a	F	T	/	/	F
f	F	F	F	/	F
e	F	F	F	F	T
mid	F	F	/	F	/
b	F	F	F	F	F



Unfold(head.next)

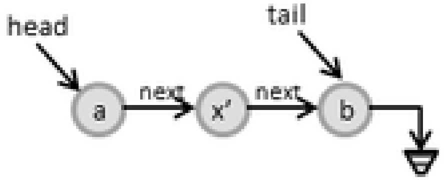
sm	
a	false
x'	false
b	false

$\mathcal{A}$	

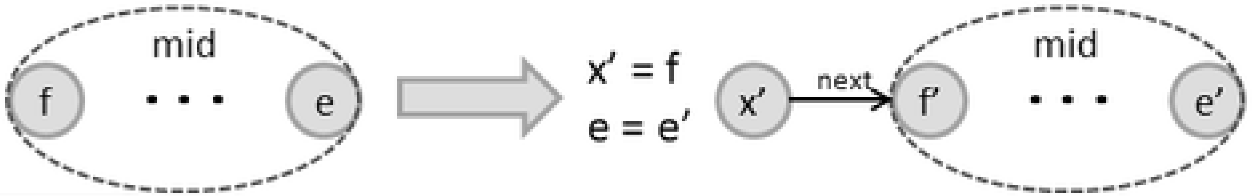
$head$	
a	true
x'	false
b	false

$tail$	
a	false
x'	false
b	true

$next$	a	x'	b
a	F	T	F
x'	F	F	T
b	F	F	F



Unfold



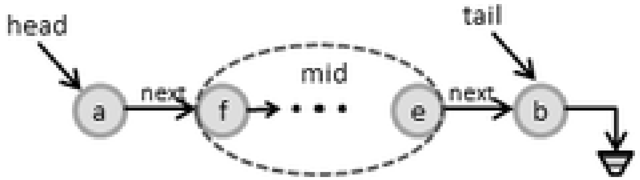
sm	
a	false
f	false
e	false
mid	true
b	false

$\mathcal{A}$	
mid	$\{f,e\}$

$head$	
a	true
f	false
e	false
mid	false
b	false

$tail$	
a	false
f	false
e	false
mid	false
b	true

$next$	a	f	e	mi	b
a	F	T	/	/	F
f	F	F	F	/	F
e	F	F	F	F	T
mid	F	F	/	F	/
b	F	F	F	F	F



Unfold(head.next)

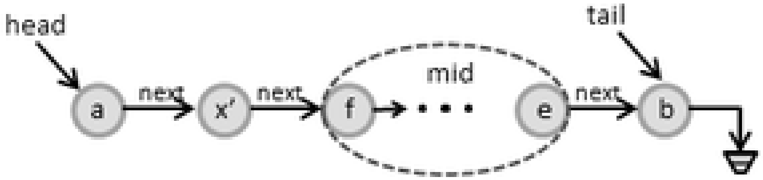
sm	
a	false
f	false
e	false
x'	false
mid	true
b	false

$\mathcal{A}$	
mid	$\{f,e\}$

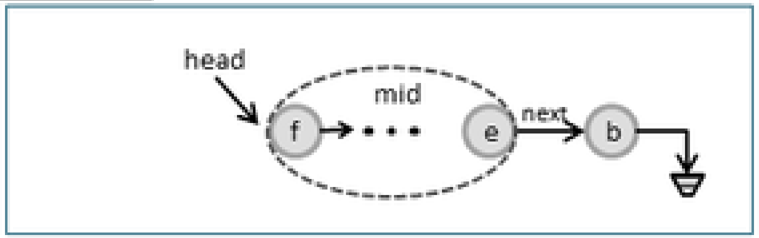
$head$	
a	true
f	false
e	false
x'	false
mid	false
b	false

$tail$	
a	false
f	false
e	false
x'	false
mid	false
b	true

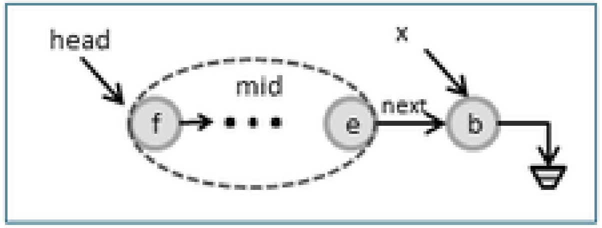
nex	a	f	e	x'	mid	b
a	F	F	F	T	F	F
f	F	F	F	F	/	F
e	F	F	F	F	F	T
x'	F	T	/	F	/	F
mid	F	F	/	F	F	/
b	F	F	F	F	F	F



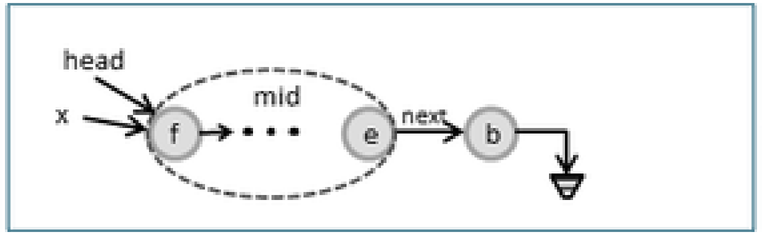
Example

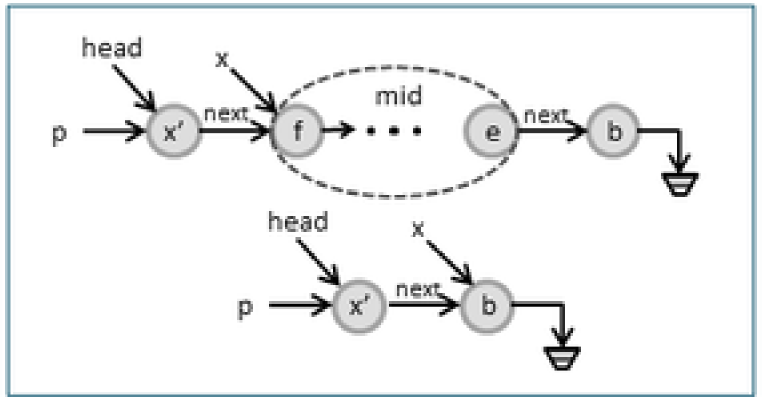


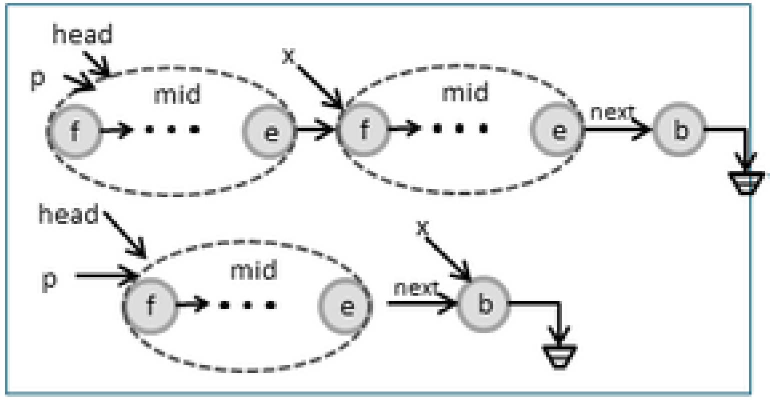

```
x = head;
while (x.next != null) {
   unfold(x);
   x = x.next;
   fold(x);
}
```



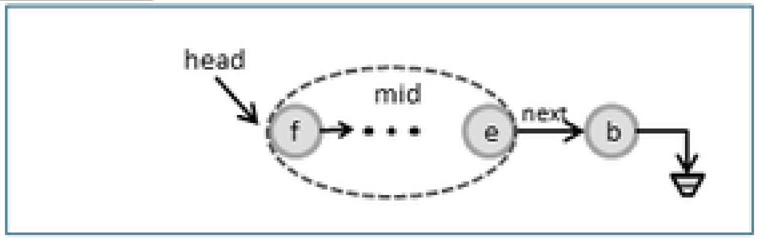
Example

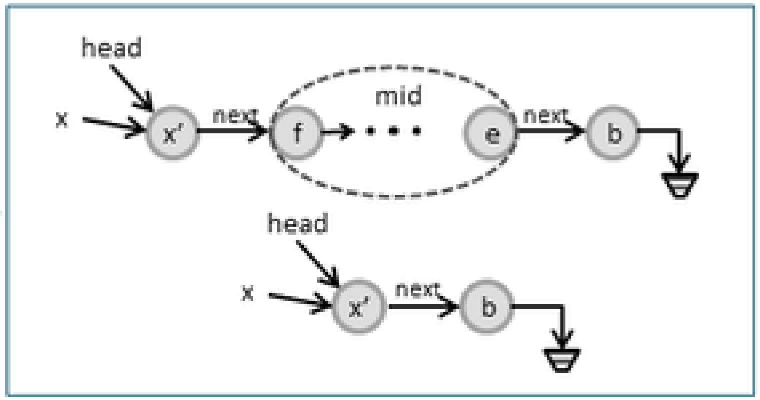


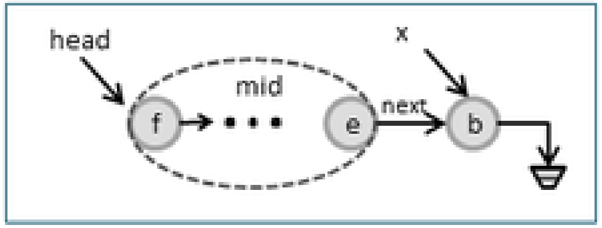




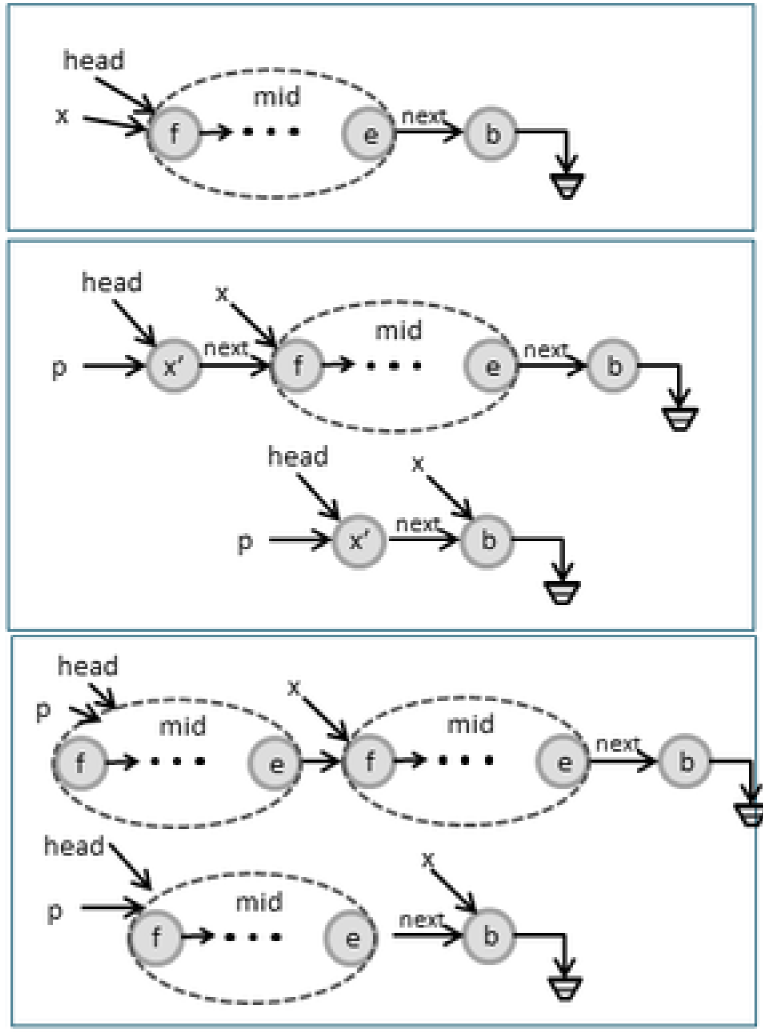

```
x = head;
while (x.next != null) {
   unfold(x);
   p = x;
   x = x.next;
   fold(p);
}
```

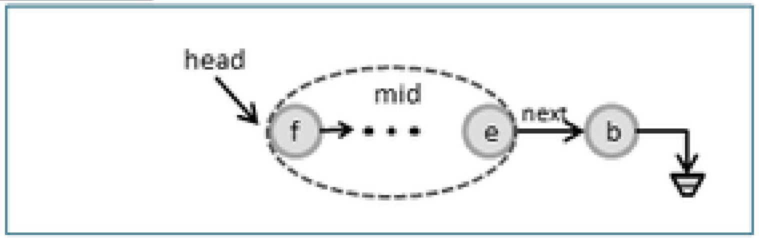




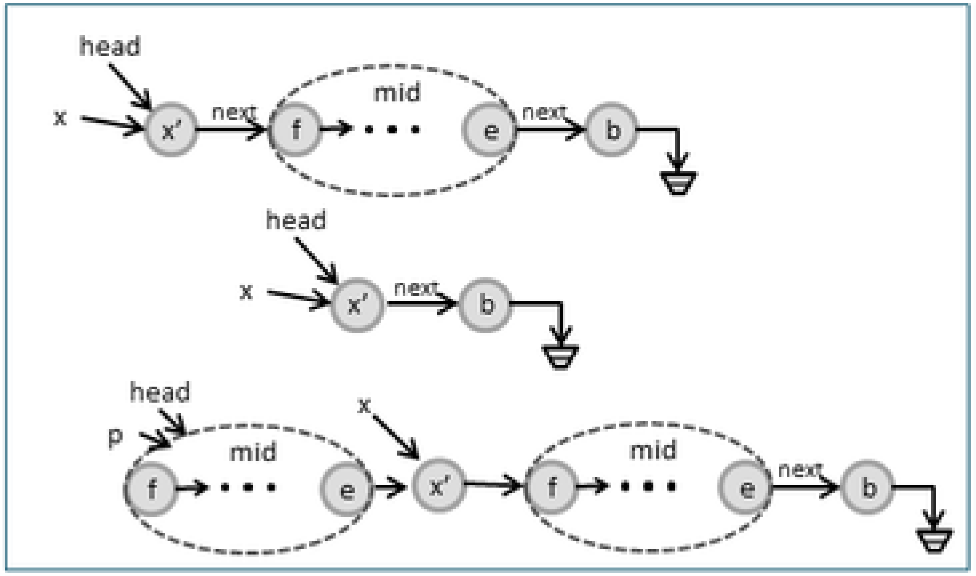


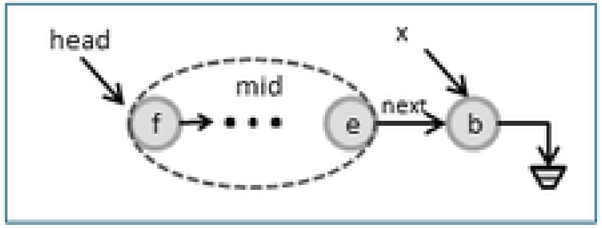
Example



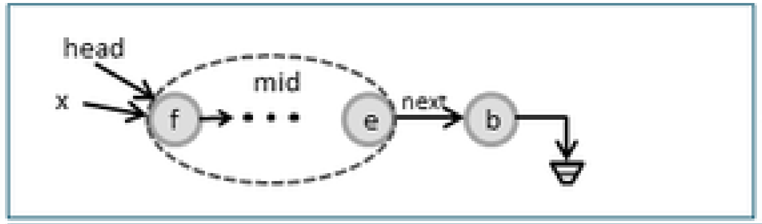


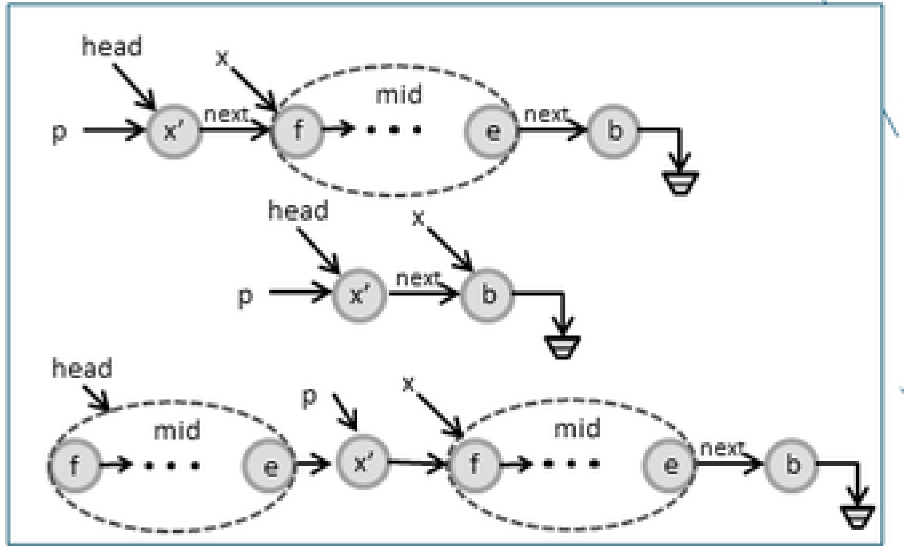

```
x = head;
while (x.next != null) {
   unfold(x);
   p = x;
   x = x.next;
   fold(p);
}
```

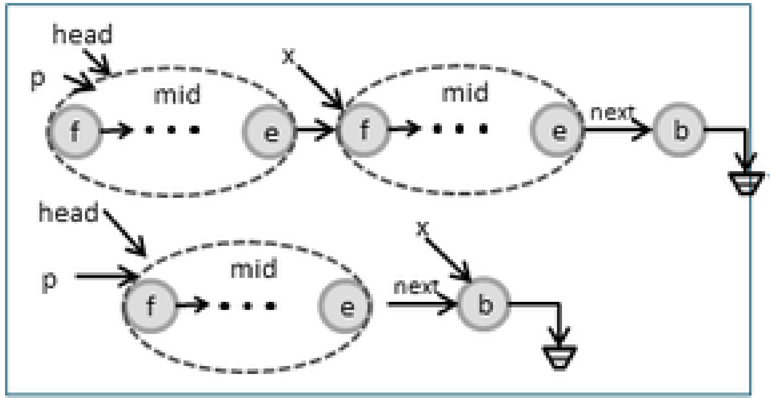




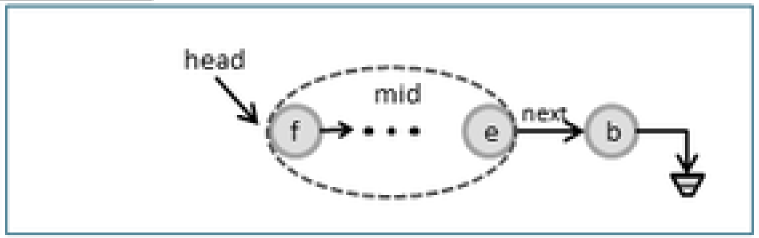
Example

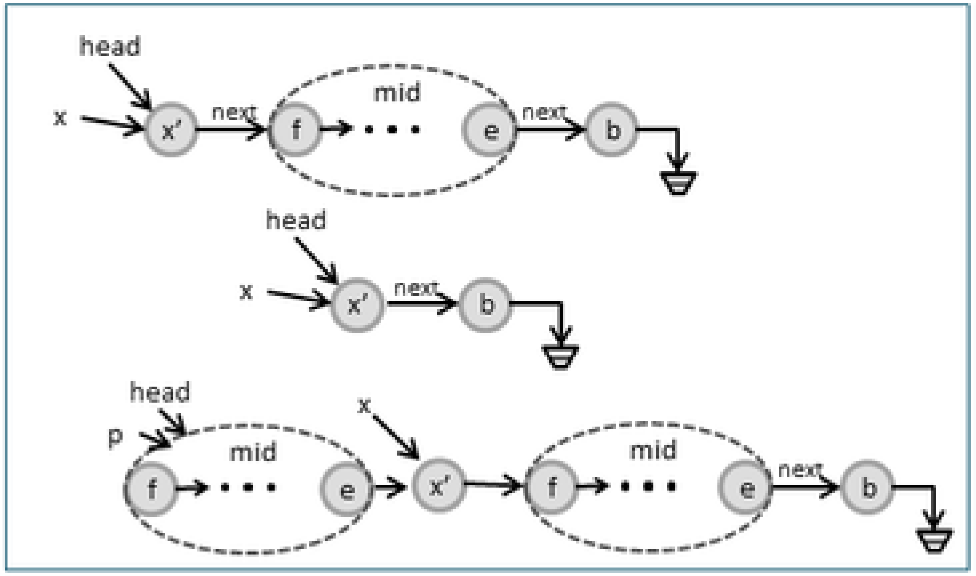


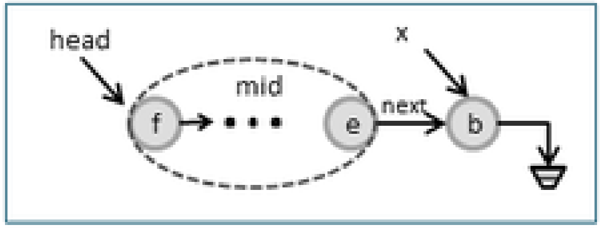





```
x = head;
while (x.next != null) {
   unfold(x);
    p = x;
    x = x.next;
    fold(p);
}
```







Look Sketch

```
void llReverse(Node head)
    {
        ?? /*1*/
       while (?? /*p*/)
       {
           ?? /*2*/
       }
        ?? /*3*/
    }
```

Look Sketch

```
void llReverse(Node head)
    {
        cstmt* /*1*/
        while (cond /*p*/)
        {
          cstmt* /*2*/
        }
        cstmt* /*3*/
    }
```

Conditional Statements

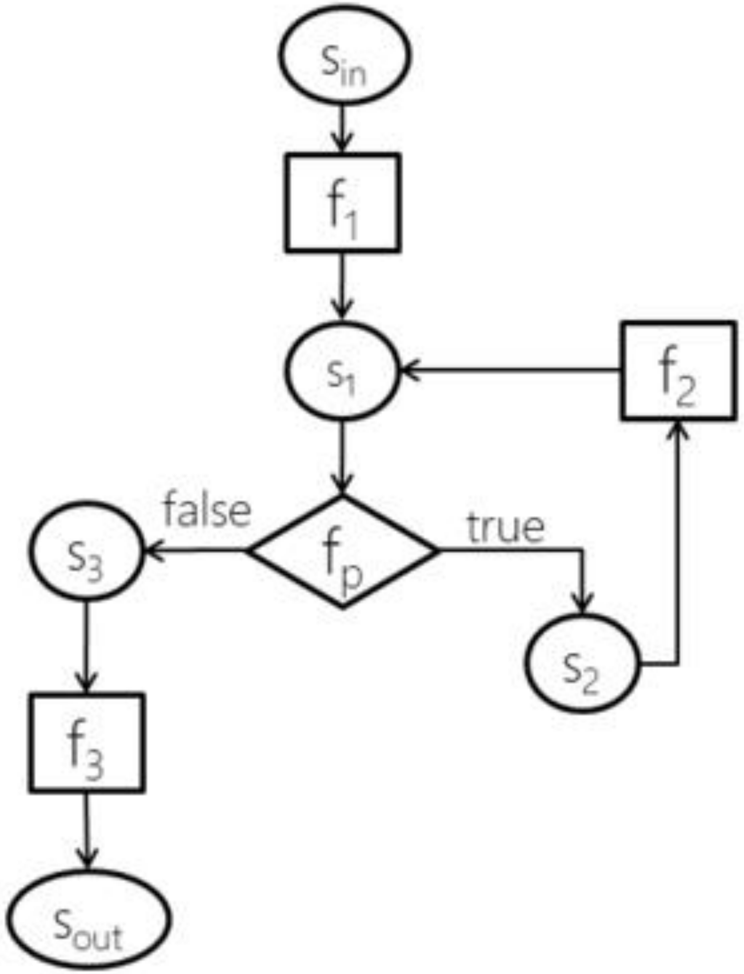
var(.ptr?) op var(.ptr?) | null

cstmt : if(COND) then STMT

var(.ptr?) = var(.ptr?)

unfold/fold var

Data flow equations



$$s_1 = f_1(s_{in}) \cup f_2(s_2)$$

$$s_2 = f_p(s_1)$$

$$s_3 = \overline{f_p}(s_1)$$

$$s_{out} = f_3(s_3)$$

Today

- Synthesizing data-structure manipulation from storyboards
    - Rishabh Singh, Armando Solar-Lezama

- Absynthe: Abstract Interpretation-Guided Synthesis
    - Sankha Narayan Guria, Jeff Foster, David Van Horn

Example

arg0

	id	valueA
0	255	1141
1	91	1130
2	347	830
⋮	⋮	⋮
8	225	638
9	257	616

arg1

	id	valueB
0	255	1231
1	91	1170
2	5247	954
⋮	⋮	⋮
12	211	575
13	25	530

arg2

"valueA ≠ valueB"

	id	valueA	valueB
0	255	1141	1231
1	91	1130	1170
2	347	830	870
5	159	715	734
8	225	638	644

Types and column labels are a potential good abstraction

{"id", "valueA", "valueB"} x DataFrame

Types Abstract Interpreter

Parameter to Absynthe
for a class of problems

```
class PyTypeInterp





end
```

Pandas data frame merge

```
# left.merge(right, opts)
df1.merge(df2, on = [‘id’])
```

Types Abstract Interpreter

```
class PyTypeInterp
  def self.pd_merge(left, right, opt)
    if left  ⊆ DataFrame &&
       right ⊆ DataFrame &&
       opt   ⊆ { on: Array<String>}
       DataFrame
    end
  end




end
```

Pandas data frame query

```
# df.query(pred)
df.query('valueA > 10')
```

Columns Abstract Interpreter

```
class ColNameInterp



end
```

Pandas data frame merge

```
df1.merge(df2, on = ['id’])
```

Final data frame is union of both

Columns Abstract Interpreter

```
class ColNameInterp

  def self.pd_merge(left, right, opt)
    left ∪ right
  end

end
```

Pandas data frame query

`df.query('valueA > 10')`

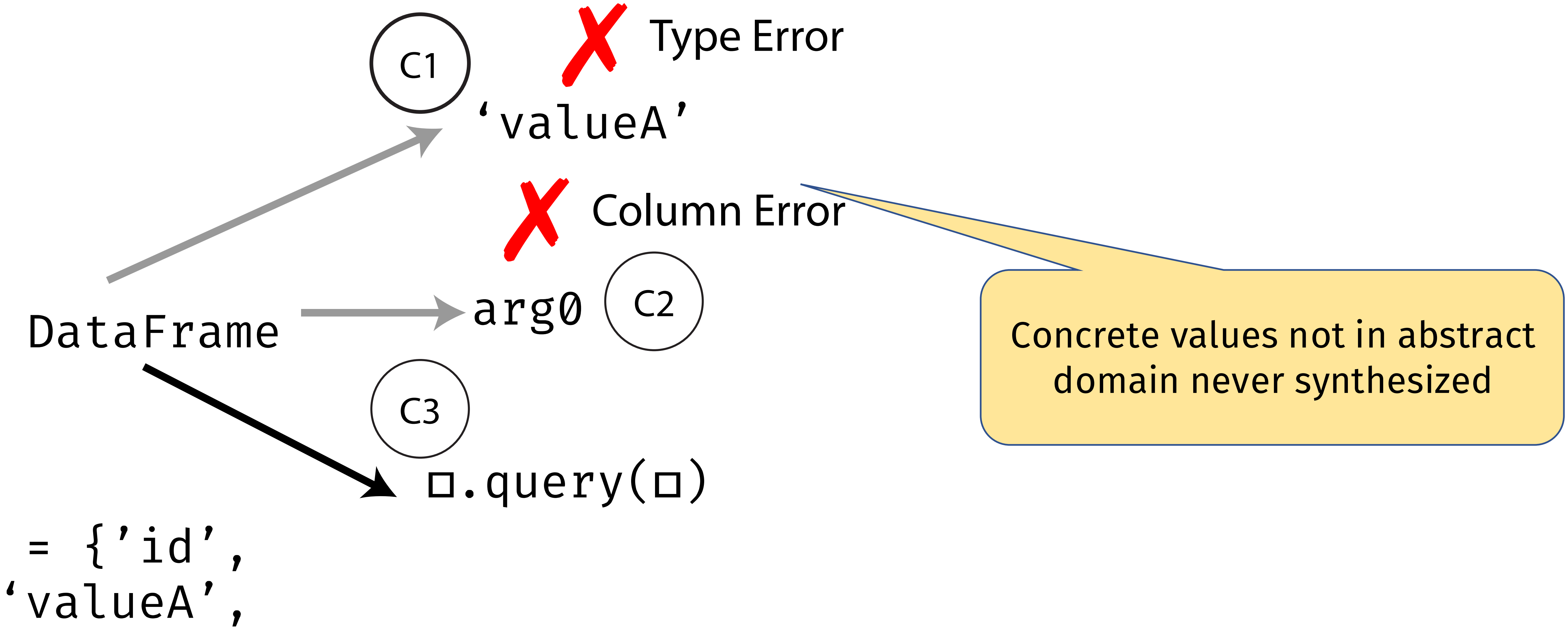
Final data frame has same columns

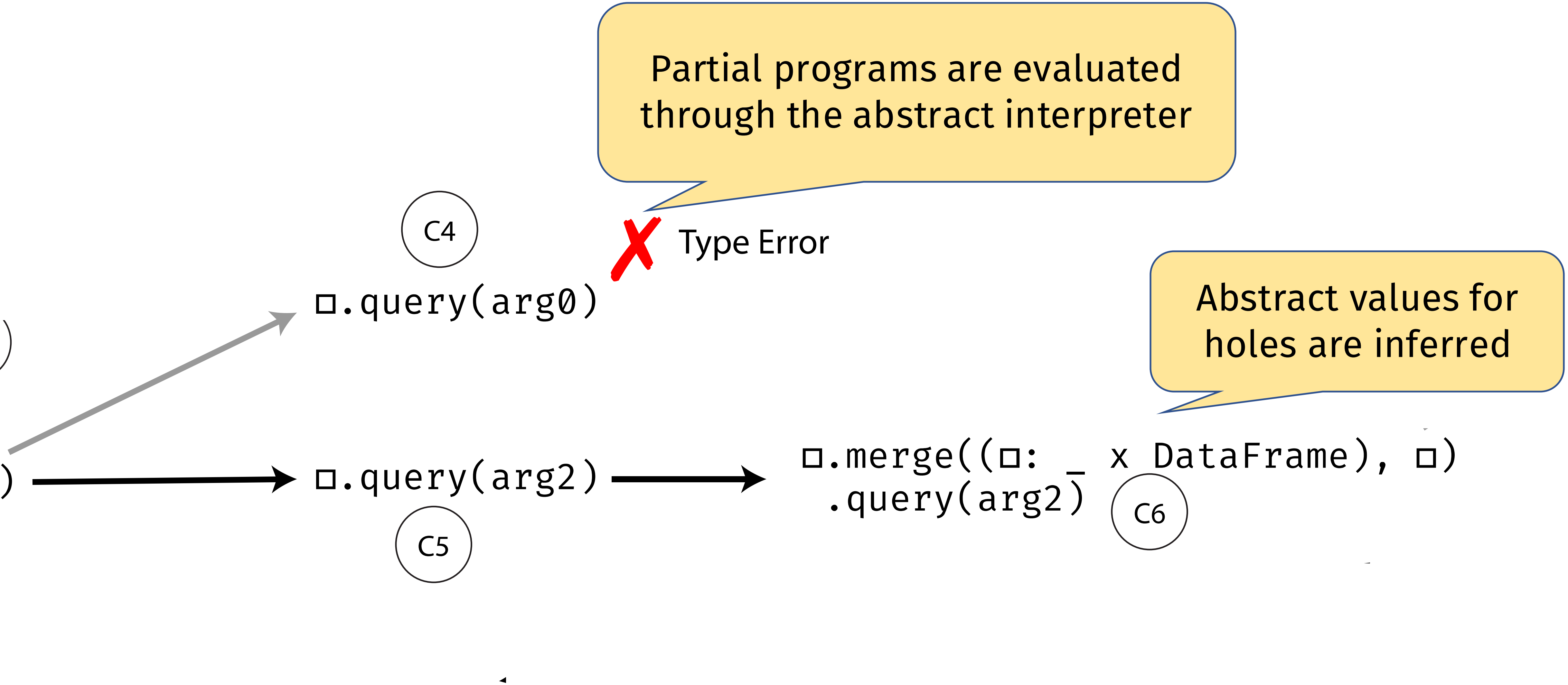
Starting candidate derived from the synthesis goal

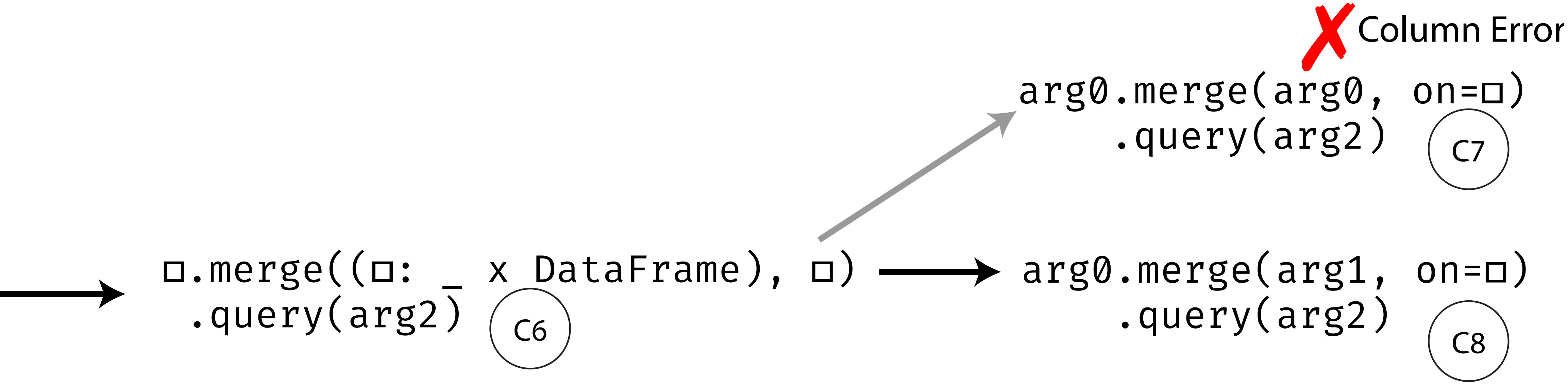
C0

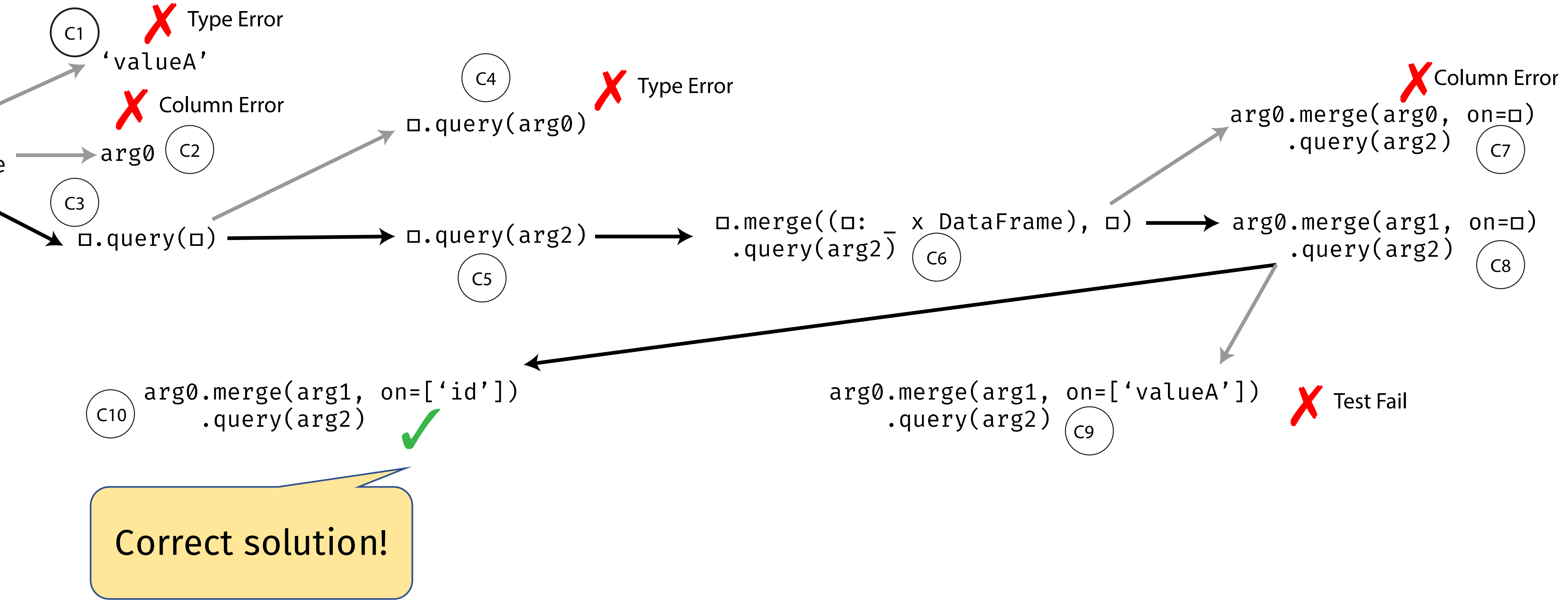
```
□: Col x DataFrame
```

```
Col = {'id',
       'valueA',
       'valueB'}
```


Type Error
C1
'valueA'
Column Error
arg0
C2
DataFrame
C3
□.query(□)
= {'id',
'valueA',
Concrete values not in abstract domain never synthesized


Partial programs are evaluated through the abstract interpreter
C4
Type Error
□.query(arg0)
Abstract values for holes are inferred
□.query(arg2)
C5
□.merge((□: _ x DataFrame), □)
.query(arg2)
C6


Column Error
arg0.merge(arg0, on=□)
.query(arg2)
C7
□.merge((□: _ x DataFrame), □)
.query(arg2)
C6
arg0.merge(arg1, on=□)
.query(arg2)
C8


C1
Type Error
'valueA'
Column Error
arg0
C2
C3
□.query(□)
C4
□.query(arg0)
Type Error
□.query(arg2)
C5
□.merge((□: _ x DataFrame), □)
.query(arg2)
C6
Column Error
arg0.merge(arg0, on=□)
.query(arg2)
C7
arg0.merge(arg1, on=□)
.query(arg2)
C8
C10
arg0.merge(arg1, on=['id'])
.query(arg2)
arg0.merge(arg1, on=['valueA'])
.query(arg2)
C9
Test Fail
Correct solution!

Searching for Programs

Synthesize a term:

$\blacksquare_1$.query($\blacksquare_2$)

such that it satisfies a synthesis goal

Semantics of $\blacksquare_1$.query($\blacksquare_2$)

arg0.query("id != 0")

arg0.query($\blacksquare_3$).query($\blacksquare_2$)

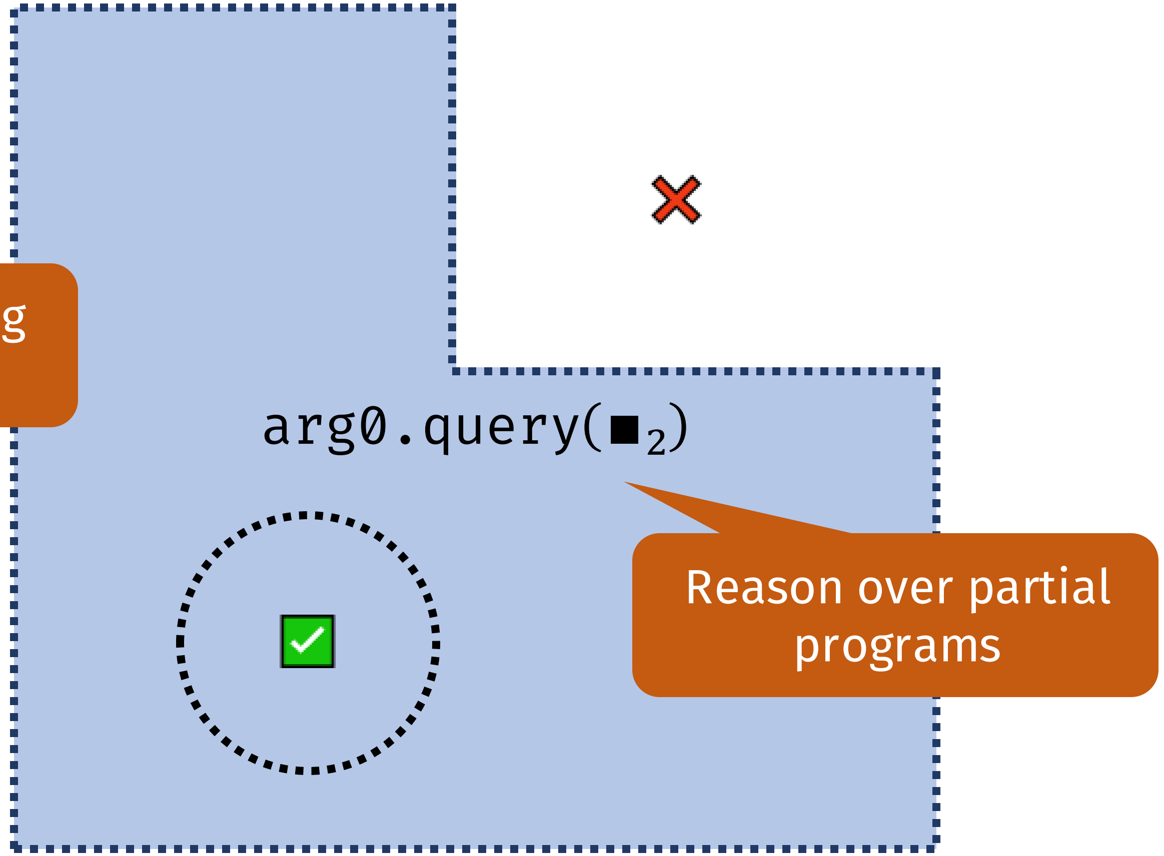
arg0.query($\blacksquare_2$)

Searching for Programs

Synthesize a term:

Assign something to $\blacksquare_2$

arg0.query($\blacksquare_2$)



= Synthesis goal

Inferring abstract values

Finite abstract domains:

Types: `Int`, `Str`, `DataFrame`

Enumerate through valid abstract values

Infinite abstract do

Solver-aided:

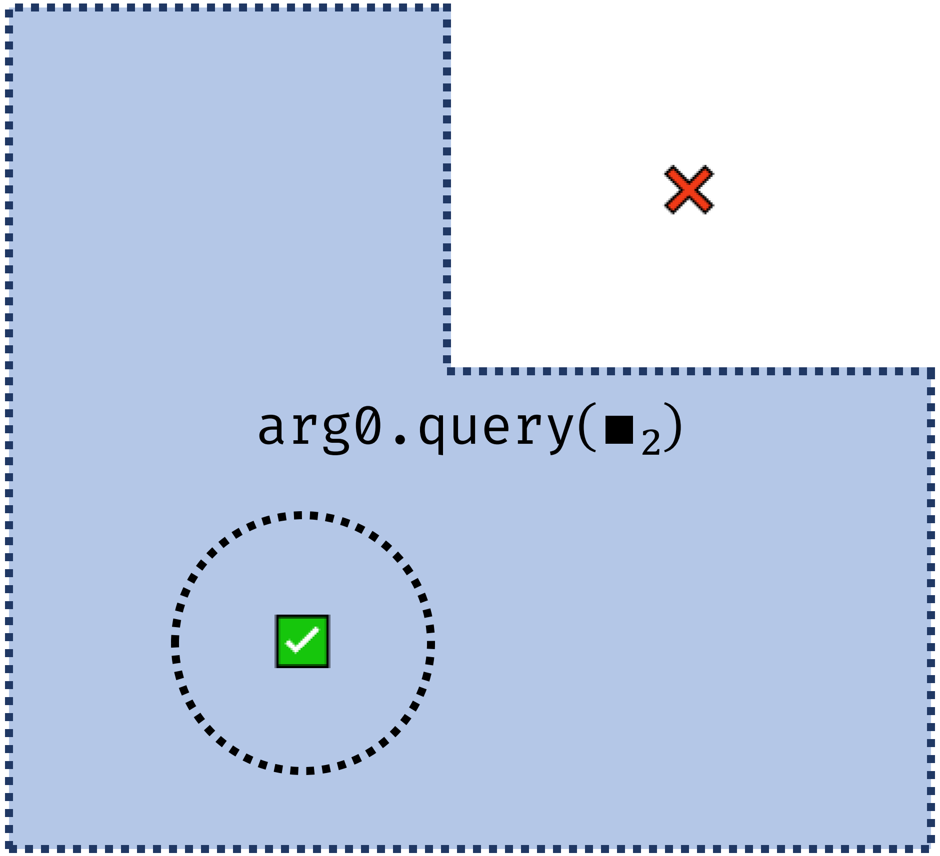
String Length: Linear integer arithmetic

Keep 1 hole symbolic and solve for it

Other:

Data frame columns

Fall back to term enumeration



Absynthe: Abstract Interpretation-Guided Synthesis

- Abstract domains are good at pruning search space
- Framework uses abstract interpreters as a parameter to guide search
- Abstractions for holes are inferred from abstract semantics
- Solves AutoPandas with simple abstract semantics without GPUs

https://github.com/ngsankha/absynthe