



第十三章 Shape Analysis

冯 洋



为什么要用Shape Analysis



- Sagiv, Mooly, Thomas Reps, and Reinhard Wilhelm. "Shape analysis." In *Proc. Int. Conf. CC' 2000, LNCS 1781*, pp. 1-17. 2000.
- 一种用于分析堆属性的静态程序分析方法



为什么要用Shape Analysis



- **Null pointers:** 在某个程序点上指针表达式是否为null
- **May-Alias:** 两个指针表达式是否指向堆上同一块内存空间
- **Must-alias:** 两个指针表达式始终指向堆上同一块内存空间
- **Sharing:** 是否存在多余一个指针表达式指向堆上同一块内存空间
- **Reachability:** 堆上某一块内存区域是否能够被某一个变量访问（或者是否存在某一个变量能够访问？）
- **Disjointness:** 两个不同的数据结构是否存在任何相同的元素
- **Cyclicity:** 一块堆内存区域是否包含环



Shape Analysis



- Shape Analysis 是一种流敏感分析技术
 - 目标：为每一个程序点，计算一个“有限的，较为保守的栈数据结构表示从入口到程序点的一条路径”
 - “有限的”表示即近似的结果 – 通常会丢失一些精度信息（请思考我们前面课程的内容，为什么要用过程间分析里面inline的例子）



Shape Analysis



- Shape Analysis通过三值逻辑（3-valued logic）实现
 - 谓词（**Predicate**）：看作一系列情况陈述的集合，陈述某事情是不是事实（真假）。如**weight**，表示一些人的体重
 - 此处我们用事实（**fact**）来表示某个程序点谓词的特定值组合
 - Shape Analysis通过确定堆描述谓词（**Predicate**）进行实例化
 - 在实际执行过程中，这些谓词是要么真，要么假
 - 静态分析中，通过三值逻辑完成谓词结果的近似分析
 - True, False, Don't Know



Shape Analysis



- Shape Analysis通过三值逻辑（3-valued logic）实现
 - 什么是逻辑（Logic）？本质上就是一堆规则（Rules）
 - 规则用于逻辑推导，举例
 - $\text{Adult}(\text{person}) \leftarrow \text{Age}(\text{person}, \text{age})$, 如果 $\text{age} \geq 18$



Shape Analysis



- Shape Analysis通过三值逻辑（3-valued logic）实现

And	0	1	\perp
0	0	0	0
1	0	1	\perp
\perp	0	\perp	\perp

OR	0	1	\perp
0	0	1	\perp
1	1	1	1
\perp	\perp	1	\perp



Shape Analysis



- Shape Analysis通过三值逻辑（3-valued logic）实现

And	0	1/2	1
0	0	0	0
1/2	0	1/2	1/2
1	0	1/2	1

OR	0	1/2	1
0	0	1/2	1
1/2	1/2	1/2	1
1	1	1	1



个体与谓词(Individuals and Predicates)



- 个体的全局空间 U (Universe of individuals)
 - $u \in U$ 表示某个抽象的位置
 - 可以用于表示某个, 或某几个, 具体的位置
 - 每个具体的位置只能被一个抽象位置表示
 - 一些谓词
 - `pointed-to-by-variable-x(u)`: 指栈变量 x 指向 u 表示的一个具体位置
 - `pointer-component-f-points-to(u1, u2)`: 指对象 $u1$ 存在一个属性 f , f 指向 $u2$ 表示的一个具体对象
 - `sm(u)`: u 指一个 `summary` 节点, 通常表示多于一个具体位置



谓词的意义



- $\langle U, I \rangle$ 中 I 表示一个三值结构
 - U 表示某个抽象的位置的集合
 - I 表示评估谓词的结果
 - $I : p:\text{Pred} \times U^{\text{arity}(p)} \rightarrow \{0, 1/2, 1\}$
- 一个三值结构通常表示0个或多个具体的状态
 - 如果公式 φ 通过谓词评估之后表达值为1, 则 φ 对 U 表示的所有状态均成立
 - 如果公式 φ 通过谓词评估之后表达值为0, 则 φ 对 U 表示的所有状态均不成立
 - 如果公式 φ 通过谓词评估之后表达值为1/2, 则 φ 对 U 表示的状态下成立结果不确定



谓词的意义



- $\langle U, I \rangle$ 中 I 表示一个三值结构
 - U 表示某个抽象的位置的集合
 - I 表示评估谓词的结果
 - $I : p:\text{Pred} \times U^{\text{arity}(p)} \rightarrow \{0, 1/2, 1\}$
- 一个三值结构通常表示0个或多个具体的状态
 - 如果公式 φ 通过谓词评估之后表达值为1, 则 φ 对 U 表示的所有状态均成立
 - 如果公式 φ 通过谓词评估之后表达值为0, 则 φ 对 U 表示的所有状态均不成立
 - 如果公式 φ 通过谓词评估之后表达值为1/2, 则 φ 对 U 表示的状态下成立结果不确定



Shape Analysis



- 一种以理清内存堆分配形态（布局）的过程内分析方法

An intraprocedural analysis aimed to figure out the shape of an heap-allocated memory

- 为什么需要shape analysis
 - 内存可能变得非常大
 - 我们的分析过程需要一种有限的表示方式
 - 通过抽象表达，来合并很多语义上的重复形态



Shape Analysis



- 一种以理清内存堆分配形态（布局）的过程内分析方法

An intraprocedural analysis aimed to figure out the shape of an heap-allocated memory

- 为什么需要shape analysis
 - 内存可能变得非常大
 - 我们的分析过程需要一种有限的表示方式
 - 通过抽象表达，来合并很多语义上的相似位置与形态（为什么？）



Shape Analysis



- **Shape Graphs** : 用于表示堆及其状态的一种抽象表示，主要由以下部分构成：
 - **S**: 抽象状态 (abstract state)
 - **H**: 抽象堆 (abstract heap)
 - **Is**: 共享信息 (sharing information)

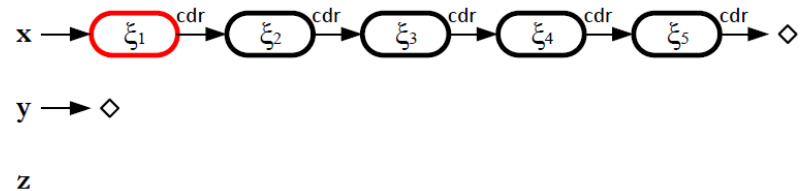


Shape Analysis



■ 一个很小的例子:

```
1:      [y := nil];
2:      while [not is-nil(x)] do
3:          ([z := y];
4:          [y := x];
5:          [x := x.cdr];
6:          [y.cdr := z]);
7:      [z := nil]
```



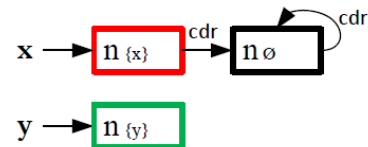
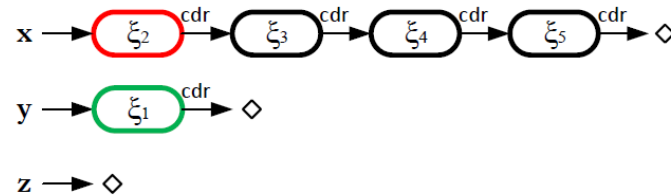


Shape Analysis



■ 一个很小的例子:

```
1:      [y := nil];
2:      while [not is-nil(x)] do
3:          ([z := y];
4:          [y := x];
5:          [x := x.cdr];
6:          [y.cdr := z]);
7:      [z := nil]
```



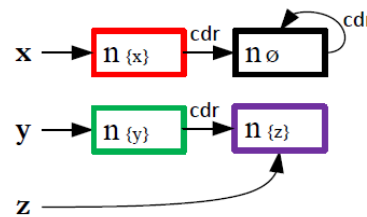
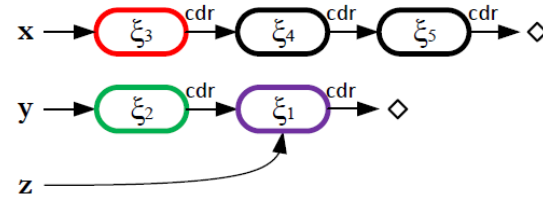


Shape Analysis



■ 一个很小的例子:

```
1:      [y := nil];
2:      while [not is-nil(x)] do
3:          ([z := y];
4:          [y := x];
5:          [x := x.cdr];
6:          [y.cdr := z]);
7:      [z := nil]
```



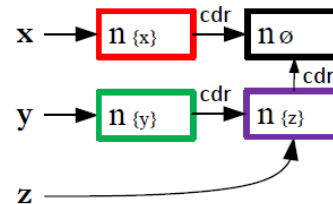
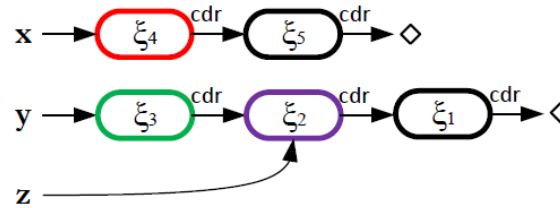


Shape Analysis



■ 一个很小的例子:

```
1:      [y := nil];
2:      while [not is-nil(x)] do
3:          ([z := y];
4:          [y := x];
5:          [x := x.cdr];
6:          [y.cdr := z]);
7:      [z := nil]
```



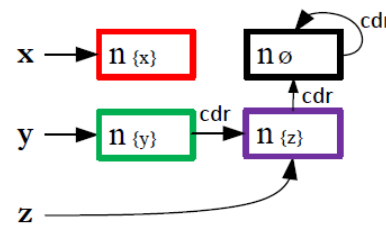
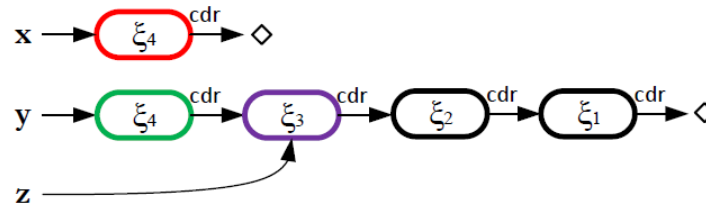


Shape Analysis



■ 一个很小的例子:

```
1:      [y := nil];  
2:      while [not is-nil(x)] do  
3:          ([z := y];  
4:          [y := x];  
5:          [x := x.cdr];  
6:          [y.cdr := z]);  
7:      [z := nil]
```



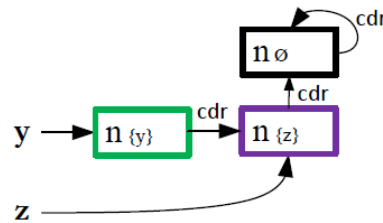
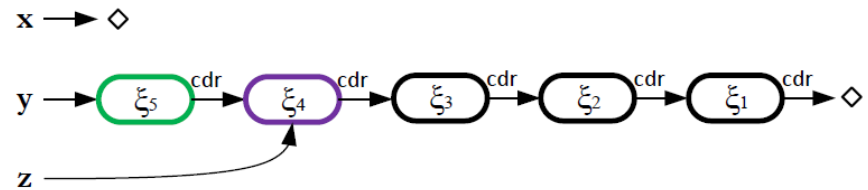


Shape Analysis



■ 一个很小的例子:

```
1:      [y := nil];  
2:      while [not is-nil(x)] do  
3:          ([z := y];  
4:          [y := x];  
5:          [x := x.cdr];  
6:          [y.cdr := z]);  
7:      [z := nil]
```





Shape Analysis的应用



```
x = 3;  
y = 1/(x-3);
```

除以0的异常?

```
x = 3;  
px = &x;  
y = 1/(*px-3);
```

指针的不确定性?

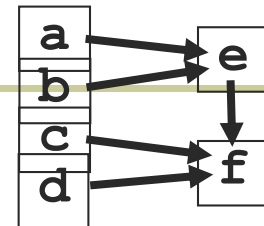
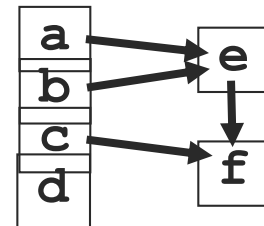
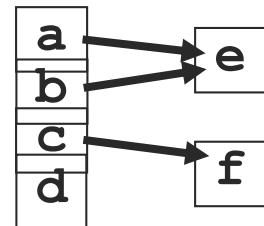
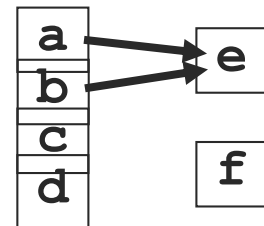
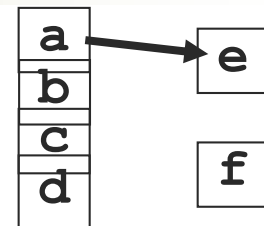
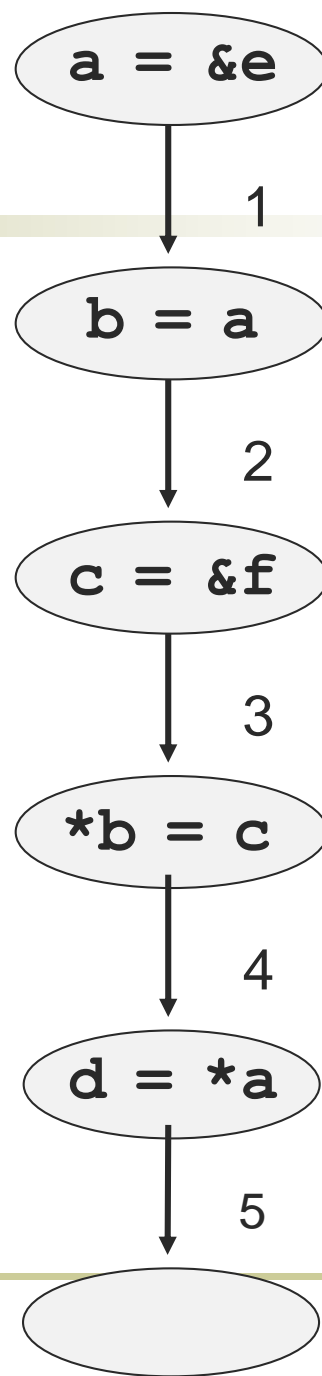
需要追踪堆分配空间

```
x = 3;  
p = (int*)malloc(sizeof int);  
*p = x;  
q = p;  
y = 1/(*q-3);
```



Shape Analysis的应用

■ 右图示例



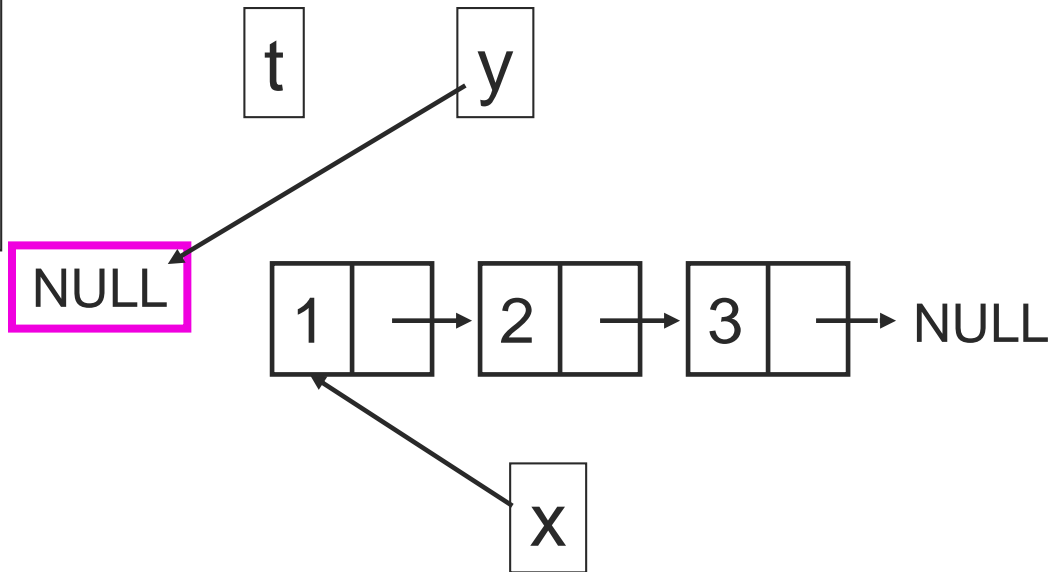


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



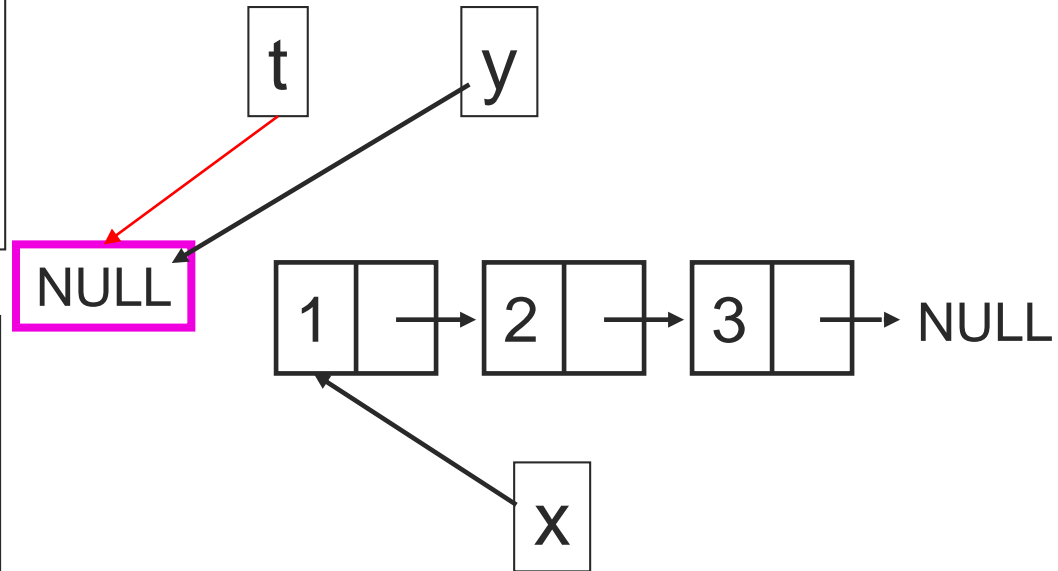


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



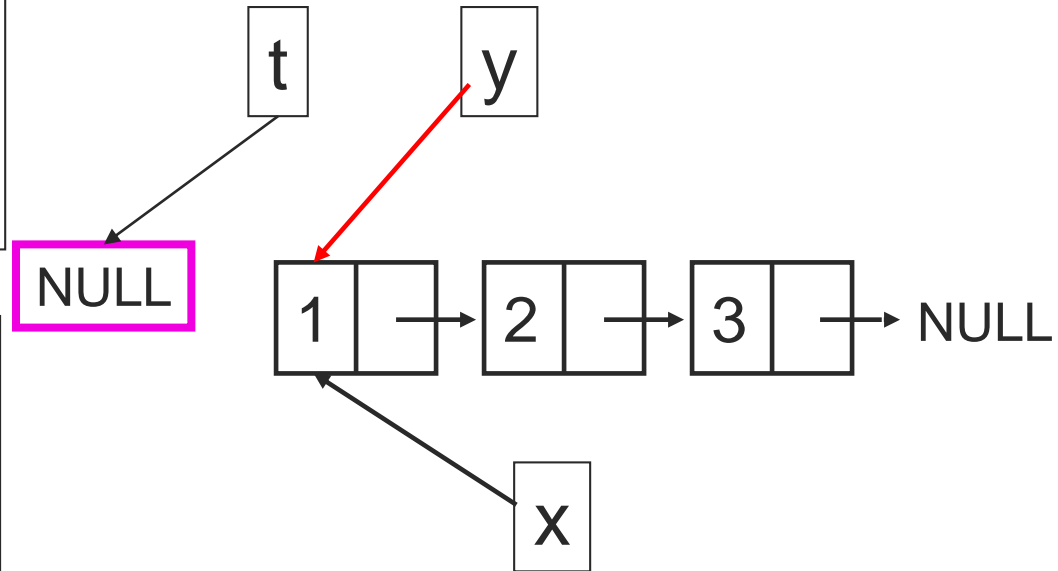


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



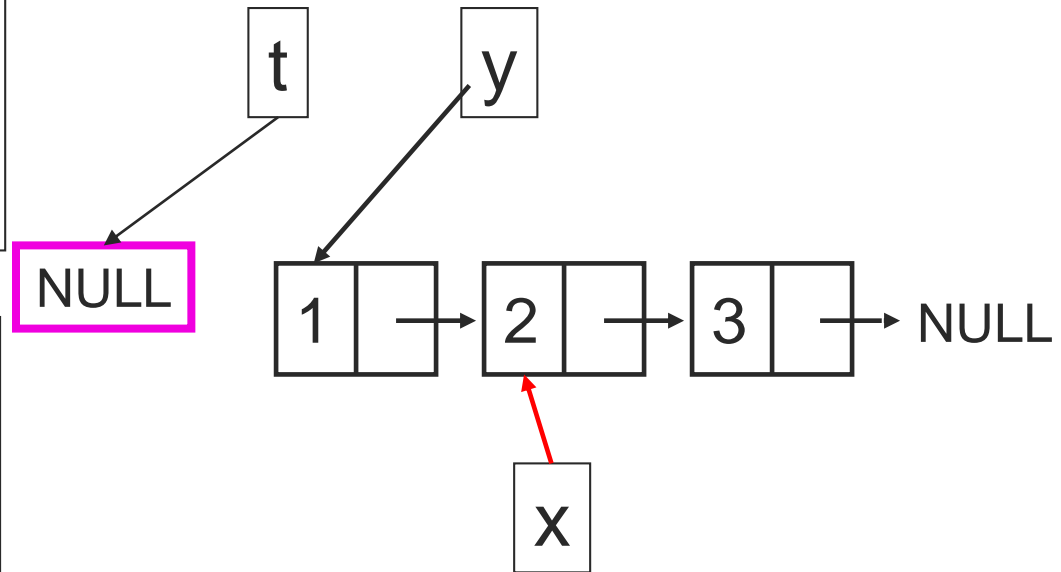


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



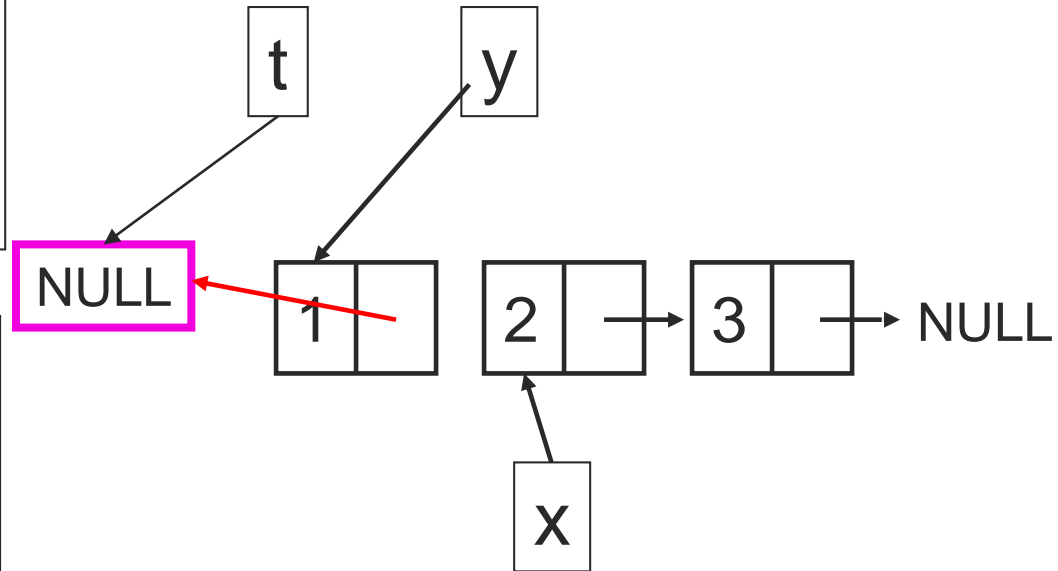


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



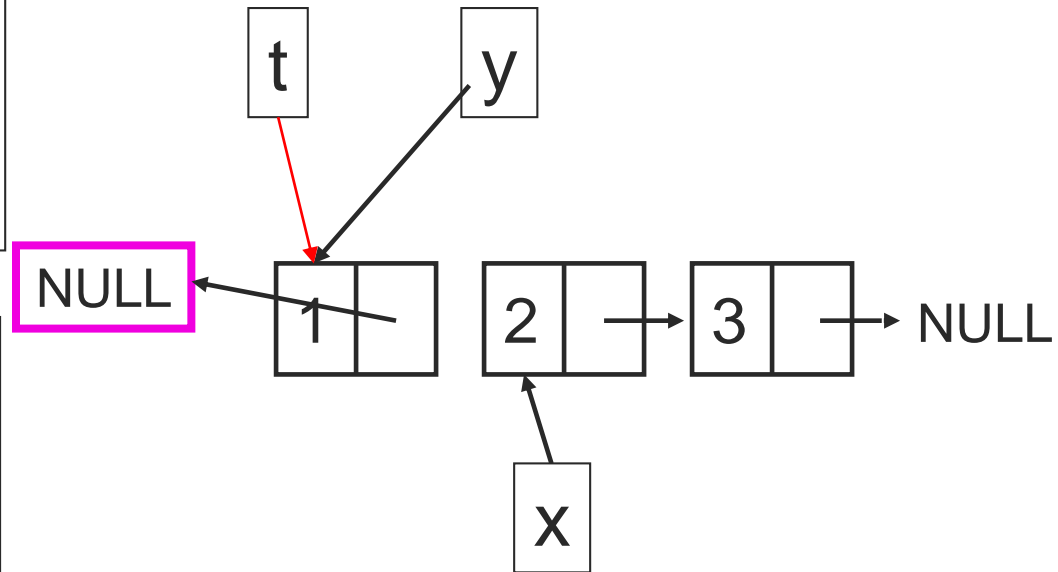


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



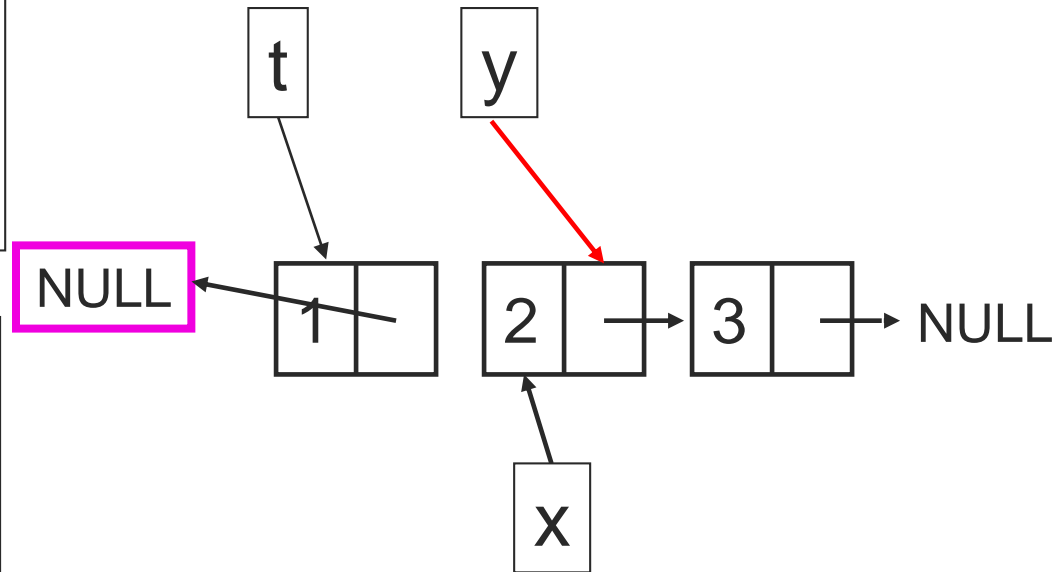


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



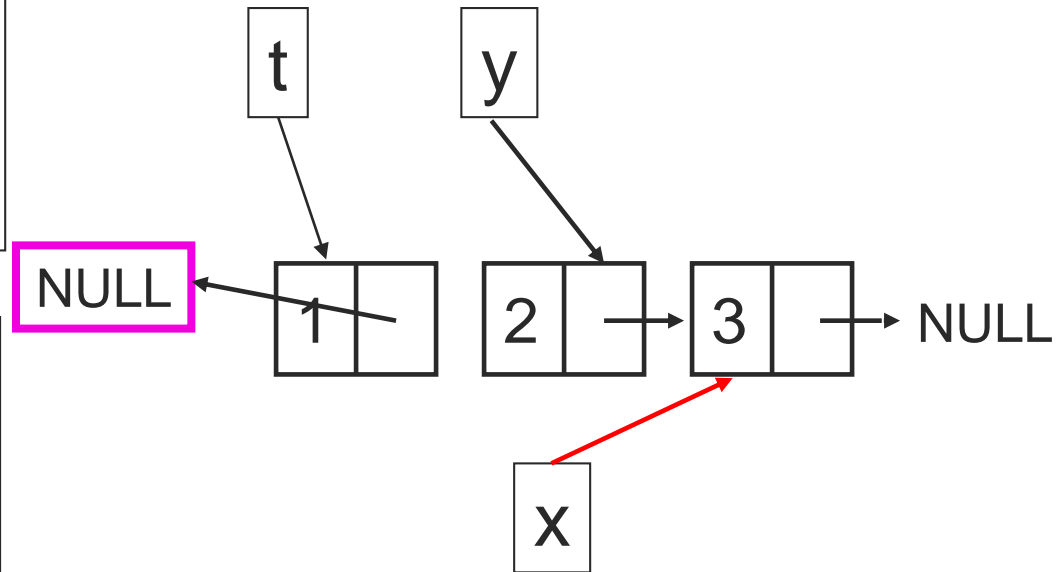


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



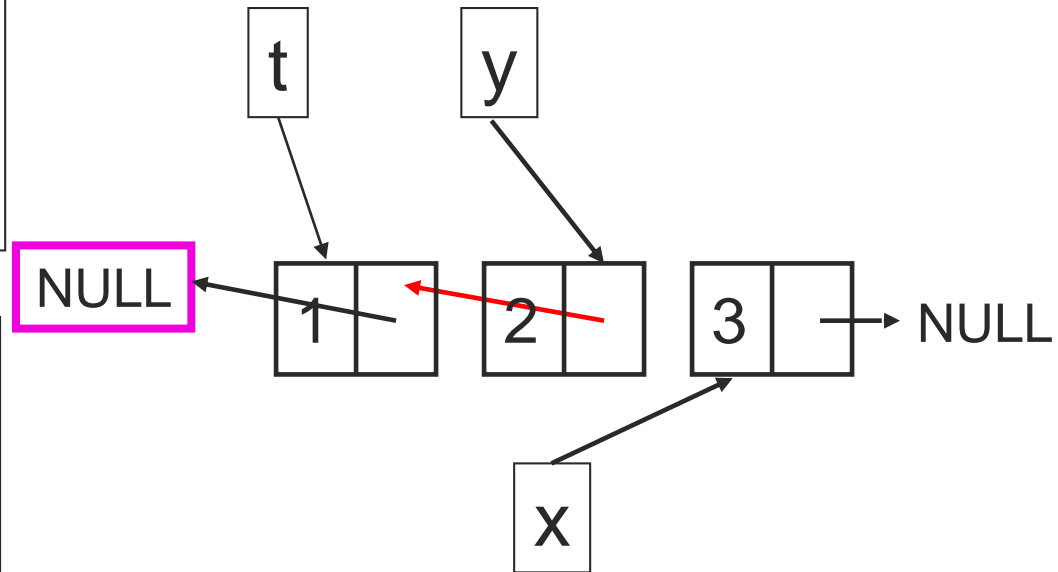


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



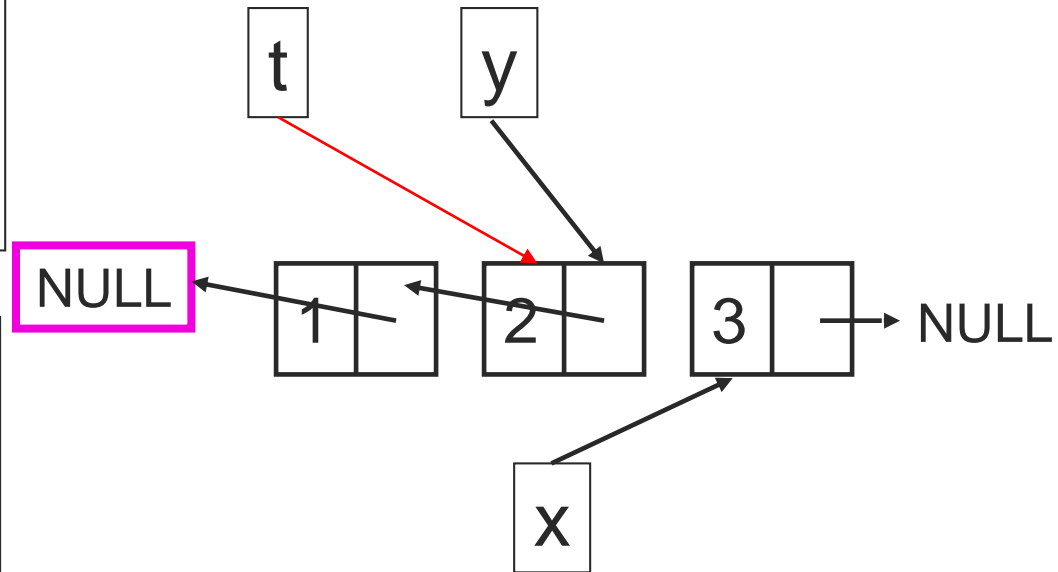


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



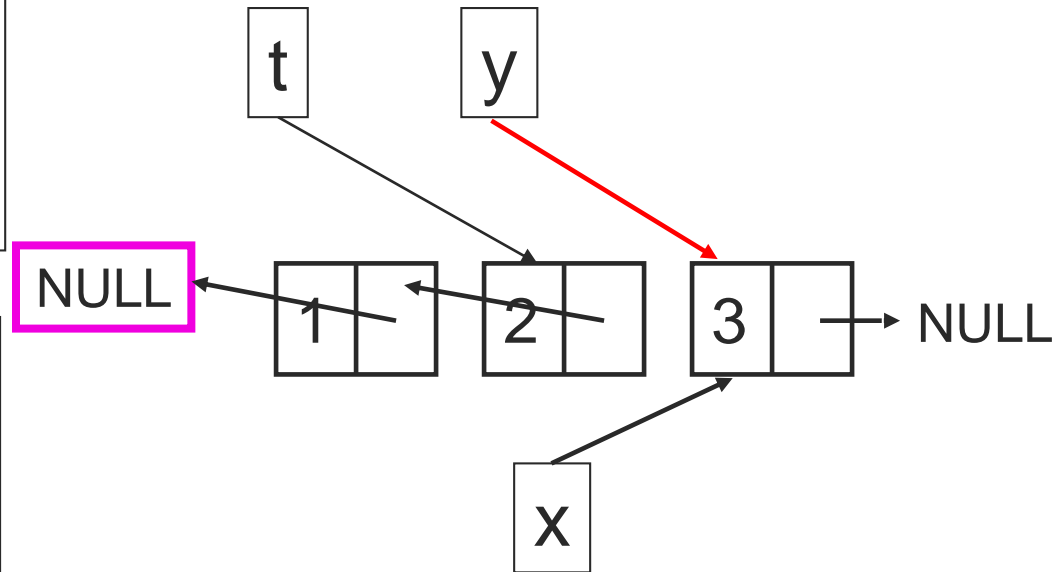


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



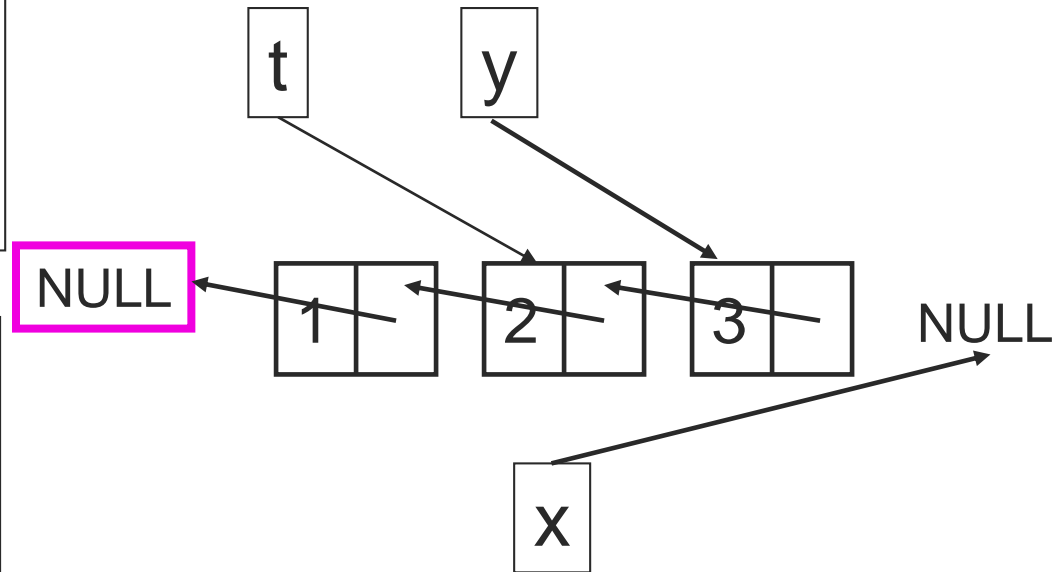


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```



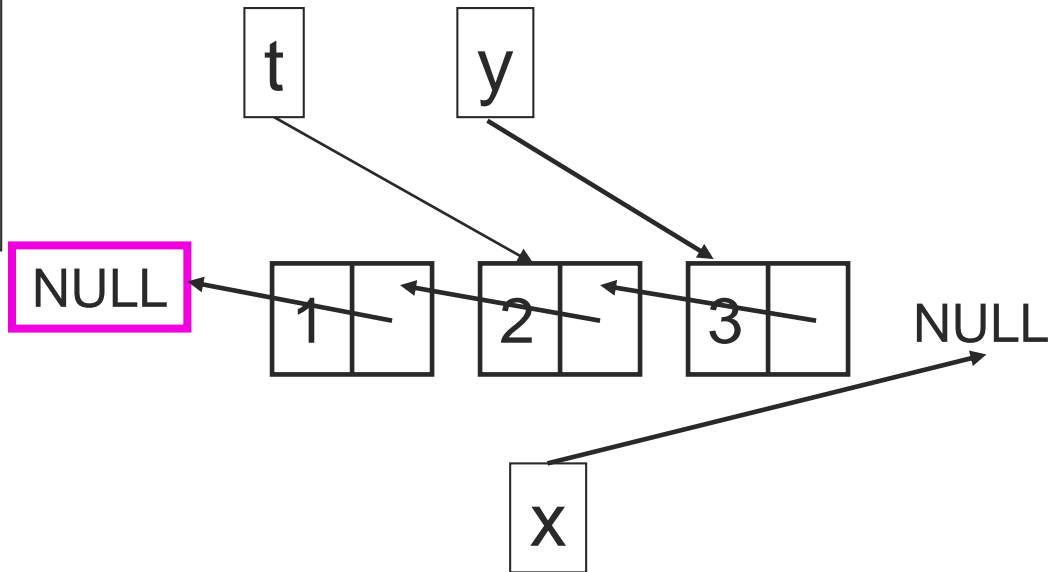


Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```





Shape Analysis的一个运行例子



```
typedef struct list_cell {  
    int val;  
    struct list_cell *next;  
} *List;
```

```
List reverse (List x) {  
    List y, t;  
    y = NULL;  
    while (x != NULL) {  
        t = y;  
        y = x;  
        x = x → next;  
        y → next = t;  
    }  
    return y;  
}
```

t

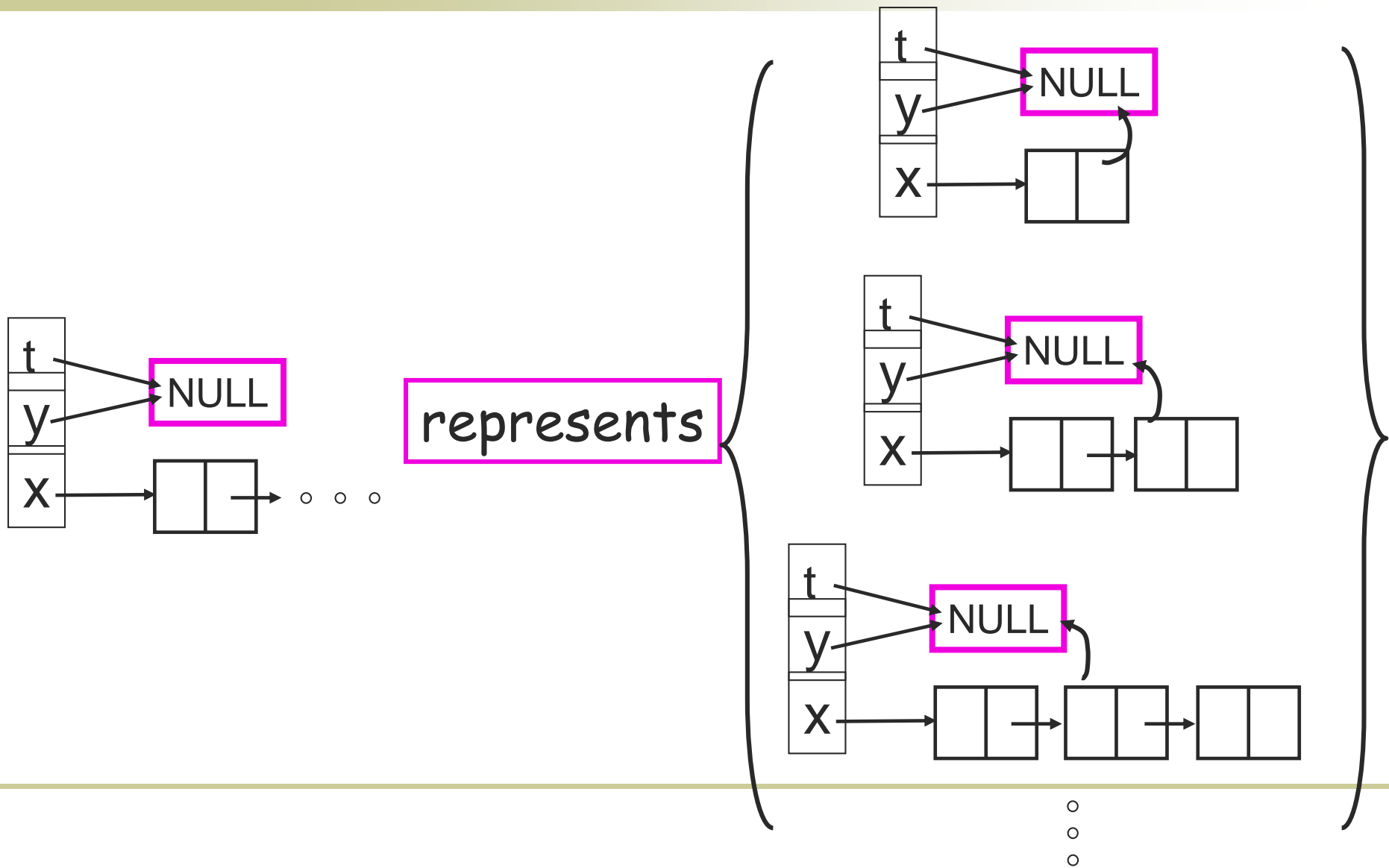
y



x



Shape Analysis的一个运行例子





Shape Analysis的一个运行例子



- **reverse(x)的特点**
 - **x** 指向一个无环链表
 - 每次迭代过程中, **x&y** 指向**disjoint** 的无环链表
 - 所有的指针释放均是安全的
 - 没有**Memory Leak**
 - 退出时, **y** 指向一个无环链表
 - 退出时, **x == NULL**
 - 所有的内存空间均是从**y**可达的



Shape Analysis的一个运行例子




- 常见的别名关系（aliasing relationships）
- Heap → 动态内存分配 → 静态分析场景下无法获得运行时数据结构的大小
- 数据结构可能在运行时改变
- 递归




三值逻辑表达式





二值逻辑

	1	0
1	1	0
0	0	0

	1	0
1	1	1
0	1	0

三值逻辑

	1	{0,1}	{0}
{1}	{1}	{0,1}	{0}
{0,1}	{0,1}	{0,1}	{0}
{0}	{0}	{0}	{0}

	{1}	{0,1}	{0}
{1}	{1}	{1}	{1}
{0,1}	{1}	{0,1}	{0,1}
{0}	{1}	{0,1}	{0}



三值逻辑表达式



\wedge	0	1/2	1
0	0	0	0
1/2	0	1/2	1/2
1	0	1/2	1

\vee	0	1/2	1
0	0	1/2	1
1/2	1/2	1/2	1
1	1	1	1



三值逻辑表达式



- 个体的全局空间 U (Universe of individuals)
 - $u \in U$ 表示某个抽象的位置
 - 可以用于表示某个，或某几个，具体的位置
 - 每个具体的位置只能被一个抽象位置表示
 - 一些谓词
 - `pointed-to-by-variable-x(u)`: $x(u)$ 指栈变量 x 指向 u 表示的一个具体位置
 - `pointer-component-f-points-to(u1, u2)`: $n(u1, u2)$ 指对象 $u1$ 存在一个属性 f , f 指向 $u2$ 表示的一个具体对象
 - `sm(u)`: u 指一个 `summary` 节点，通常表示多于一个具体位置



三值逻辑表达式



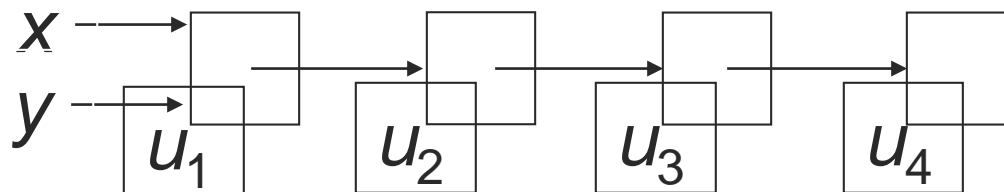
- Are x and y pointer aliases?
 - $\exists v: x(v) \wedge y(v)$



三值逻辑表达式



- Are x and y pointer aliases?
 - If $\exists v: x(v) \wedge y(v) == 1 \rightarrow$ Yes
 - else \rightarrow No



	$x(u)$	$y(u)$	$t(u)$
u_1	1	1	0
u_2	0	0	0
u_3	0	0	0
u_4	0	0	0

n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0



三值逻辑表达式



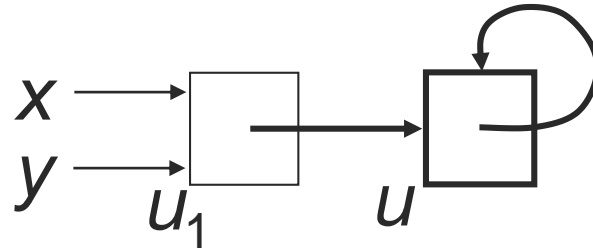
- Are x and y pointer aliases?
 - If $\exists v: x(v) \wedge y(v) == 1 \rightarrow$ Yes
 - else \rightarrow No

	$x(u)$	$y(u)$	$t(u)$
u_1	1	1	0
u_2	0	0	0
u_3	0	0	0
u_4	0	0	0

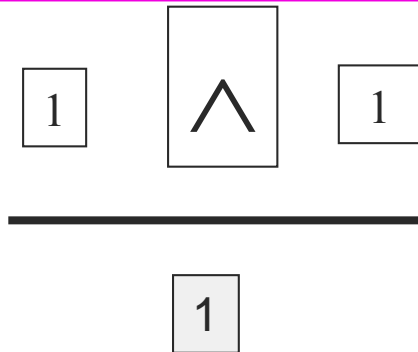
n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0



三值逻辑表达式

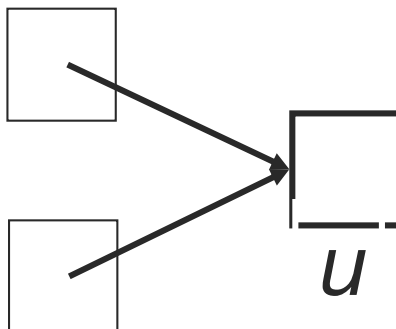


$$\exists v: x(v) \wedge y(v)$$





三值逻辑表达式



$$\exists v_1, v_2: \underline{n(v_1, u)} \wedge \underline{n(v_2, u)} \wedge \underline{v_1 \neq v_2}$$

1

\wedge

1

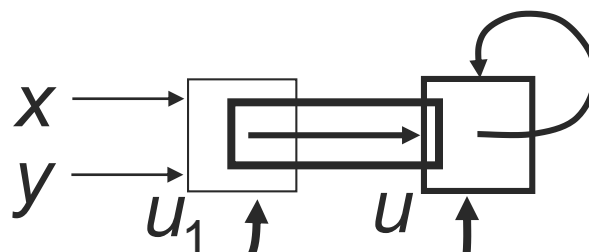
\wedge

1

1



三值逻辑表达式



$$\exists v_1, v_2: \underline{n(v_1, u)} \wedge \underline{n(v_2, u)} \wedge \underline{v_1 \neq v_2}$$

1/2

\wedge

1/2

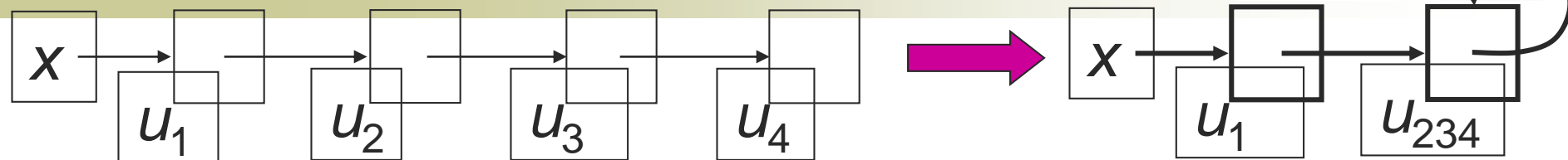
\wedge

1

1/2



三值逻辑表达式



$$v: x(v) \quad 1$$

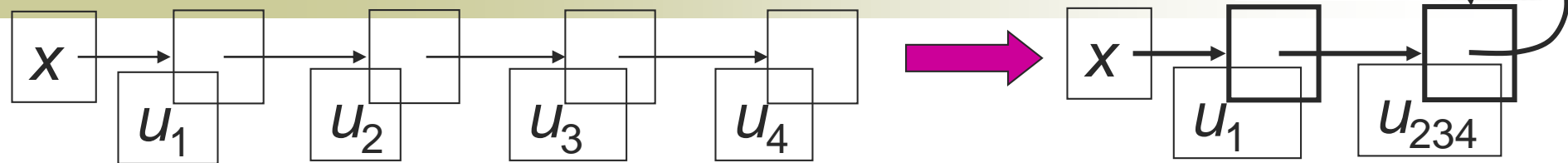
$$v: y(v) \quad 0$$

$$v: x(v) \wedge \quad v: \neg x(v) \quad 1$$

$$v_1, v_2, v: n(v_1, v) \wedge n(v_2, v) \wedge (v_1 \neq v_2) \quad 0$$



三值逻辑表达式

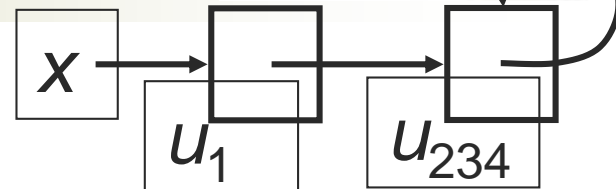
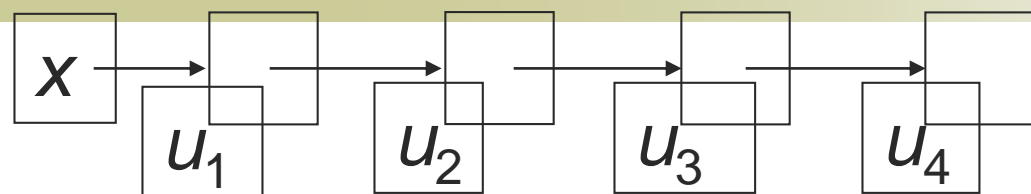


$\langle 1, 0, 1, 0 \rangle$

- 通过谓词抽象，将相似的结构映射到一个 **summary individual** 中
- 仅仅保存很多关键性的操作



三值逻辑表达式



	$x(u)$	$y(u)$
u_1	1	0
u_2	0	0
u_3	0	0
u_4	0	0



	$x(u)$	$y(u)$
u_1	1	0
u_{234}	0	0

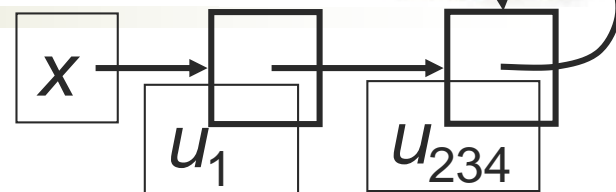
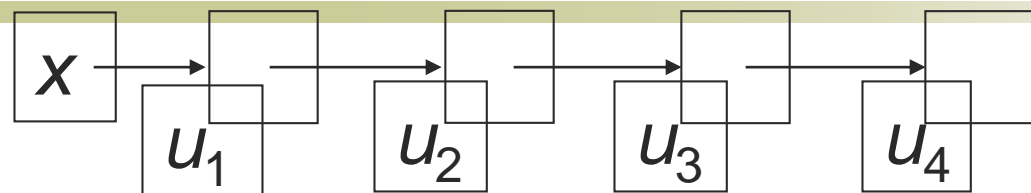
n	u_1	u_2	u_3	u_4
u_1	0	1	0	0
u_2	0	0	1	0
u_3	0	0	0	1
u_4	0	0	0	0



n	u_1	u_{234}
u_1	0	1/2
u_{234}	0	1/2



三值逻辑表达式

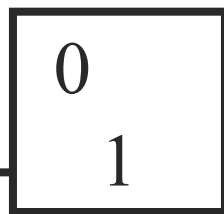


	$x(u)$	$y(u)$
u_1	1	0
u_2	0	0
u_3	0	0
u_4	0	0



	$x(u)$	$y(u)$
u_1	1	0
u_{234}	0	0

$=$	u_1	u_2	u_3	u_4
u_1	1	0	0	0
u_2	0	1	0	0
u_3	0	0	1	0
u_4	0	0	0	1



$=$	u_1	u_{234}
u_1	1	0
u_{234}	0	1/2