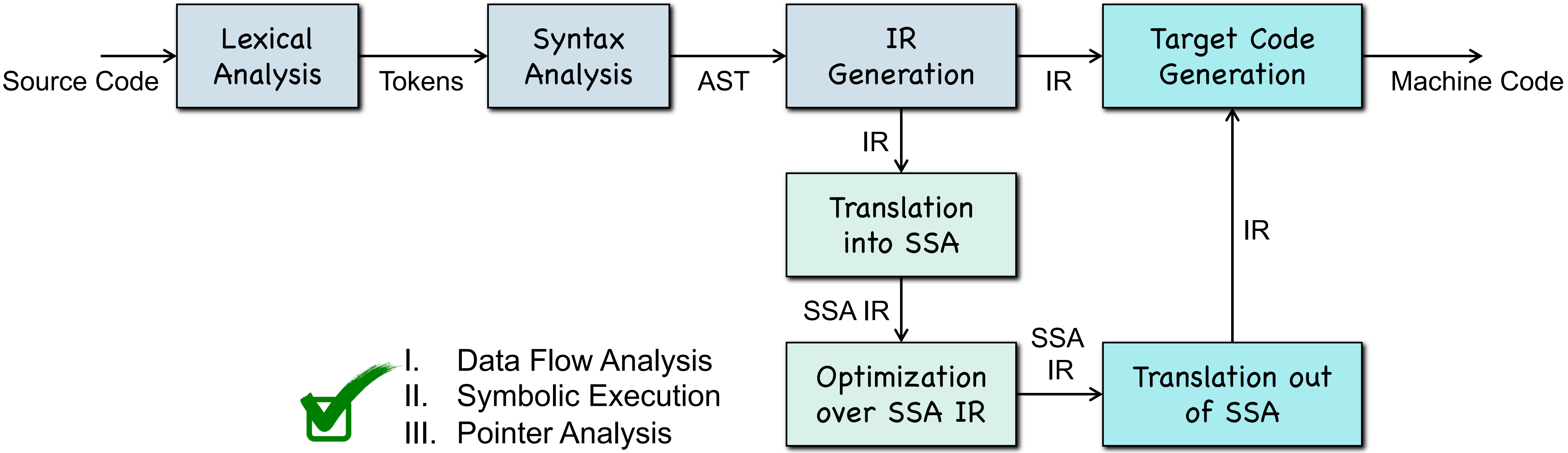
Made by Qingkai Shi
qingkaishi@nju.edu.cn

Recap-2
The Compilers’ Middle End

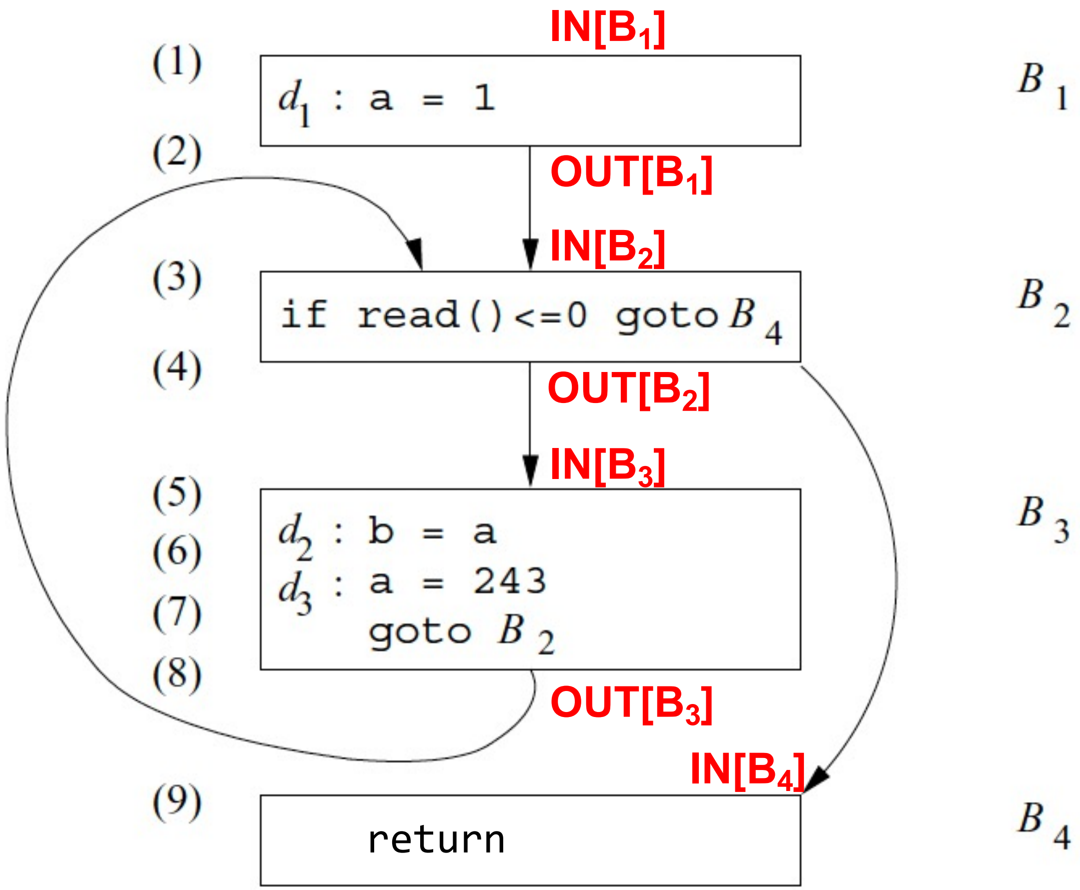
Middle End of Compilers



I. Data Flow Analysis
II. Symbolic Execution
III. Pointer Analysis
IV. Datalog-Based Analysis

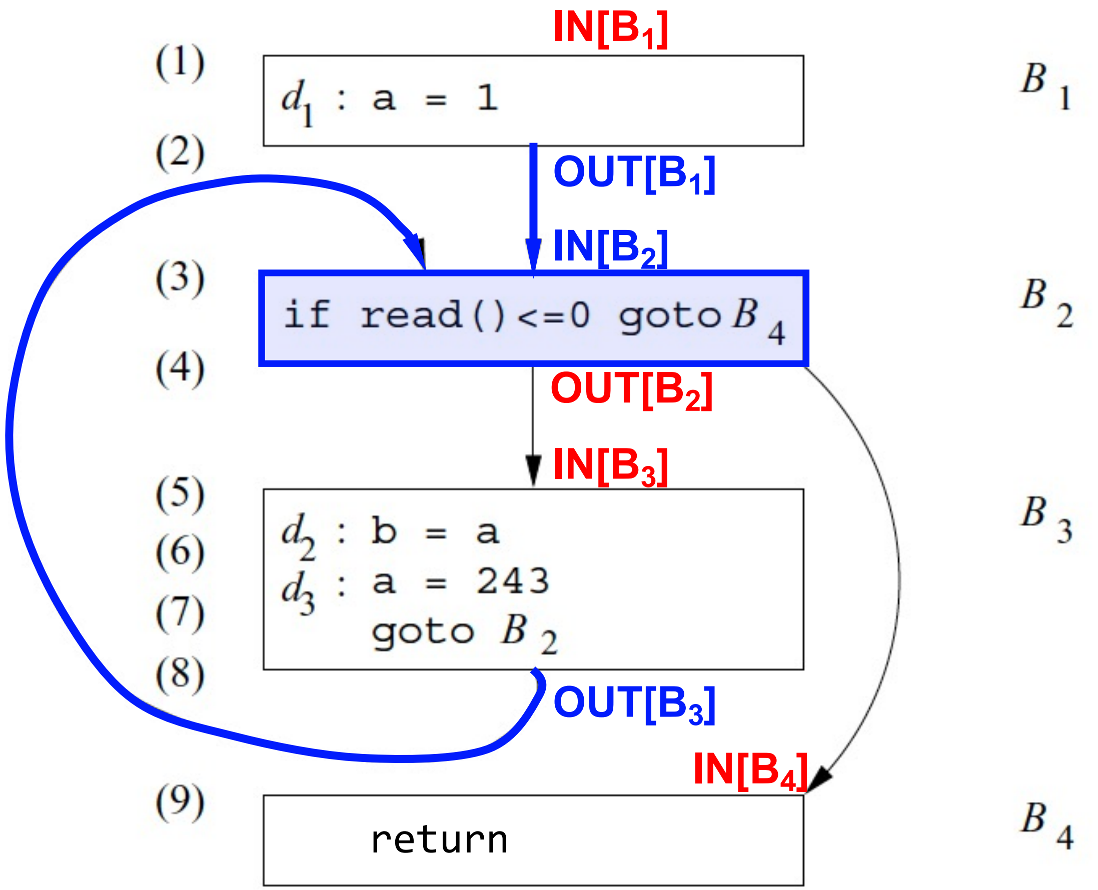
PART I: Data Flow Analysis

Data Flow Analysis



- IN[B]/OUT[B]
  - Set of data flow facts before and after a block
- Constraints
  - OUT[B] = f_B(IN[B])
  - IN[B] = $\wedge_{P \in preds(B)}$ OUT[P]

Data Flow Analysis



- IN[B]/OUT[B]
  - Set of data flow facts before and after a block
- Constraints
  - OUT[B] = f_B(IN[B])
  - IN[B] = $\wedge_{P \in preds(B)}$ OUT[P]

Data Flow Analysis

- To perform data flow analysis, define
  - transfer function f for each statement/block
  - merge function ∧ at the joint point of multiple paths
  - the initial set of data flow facts for each block

Data Flow Analysis

- To perform data flow analysis, define
  - transfer function f for each statement/block
  - merge function ∧ at the joint point of multiple paths
  - the initial set of data flow facts for each block

- Data flow analysis produces
  - IN[B]/OUT[B] that include data flow facts at a program point
  - The resulting data flow facts are always true during program execution

南京大学编译技术
及软件安全研究组

The Worklist Algorithm

- **ForEach** basic block B: initialize IN[B] and OUT[B]; **EndFor**
- worklist = set of all basic blocks;

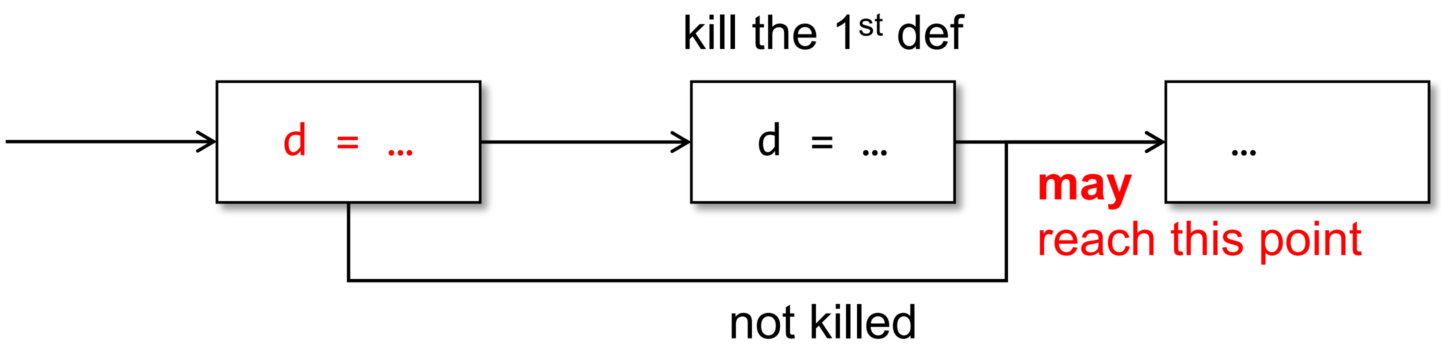
- **While** (!worklist.empty()) **Do**
-     B = worklist.pop();
-     IN[B] = $\wedge_{P\in preds(B)}$ OUT[P];   OUT[B] = f_B(IN[B])
-     **If** OUT[B] changed: worklist.push(all B's successors); **EndIf**
- **EndWhile**

Classic DFA

- Reaching Definition
- Available Expressions
- Live Variables

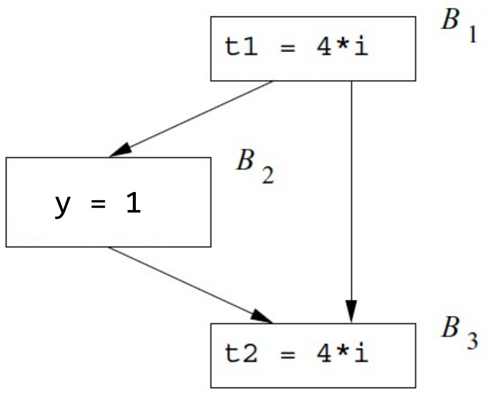
Reaching Definitions

- A definition *d* may reach a program point, if there is a path from *d* to the program point such that *d* is not killed along the path.

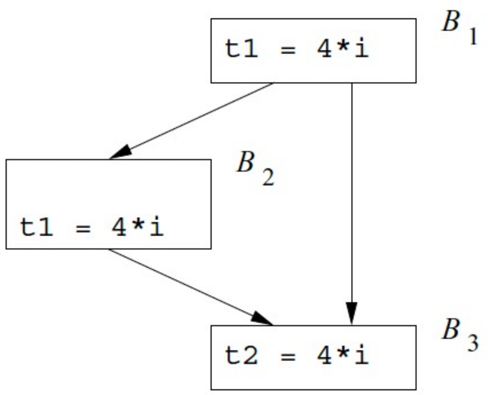


Available Expressions

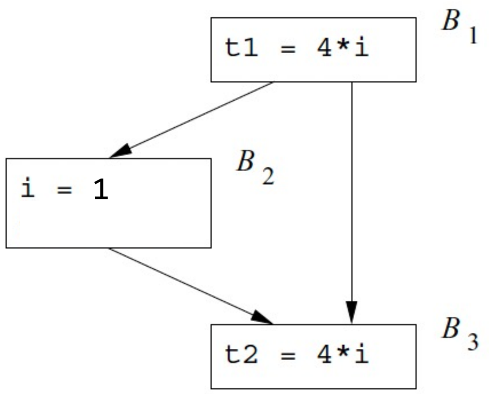
- An expression x + y is available at a point p if
  - every path from the entry node to p evaluates x + y, and
  - after the last such evaluation prior to reaching p, there are no subsequent assignments to x or y.



4 * i is available before B_3



4 * i is available before B_3



4 * i is **NOT** available before B_3

Live Variable Analysis

- We wish to know if the value of a variable x at point p can be used along the path starting from p.

Live Variable Analysis

- We wish to know if the value of a variable x at point p can be used along the path starting from p.

The values of y, z at this point are **live** as we can use them in the next statement.

The value of x at this point is **dead** because x will be redefined and the old value cannot be used any longer.

```
x = y + z
```

Transfer & Merging Functions

- Transfer Functions?
- Merge Functions?
- Forward or Backward Analysis?

PART II: Symbolic Execution

Symbolic Execution

- Path Sensitivity
- Flow Sensitivity
- Context Sensitivity

Symbolic Execution

- Path Explosion
- Constraint Solving
- External Function Call

Solving Path Constraints

- How can we check the satisfiability of a constraint?

$$\alpha_a \neq 0 \land \alpha_b = 0 \land \neg(2(\alpha_a + \alpha_b) - 4 \neq 0)$$

- Constraint solver
  - Satisfiable => A possible solution
  - Unsatisfiable
  - Unknown (due to timeout)

Z3 https://github.com/Z3Prover/z3

CVC5 https://cvc5.github.io

Solving Path Constraints

- Propositional Logic
- Satisfiability (SAT) --- DPLL and CDCL

- First Order Logic
- Satisfiability Modulo Theories (SMT)
- The Bit Vector Theory

The SAT Problem

- Deciding satisfiability of a formula F in PL

The SAT Problem

- Deciding satisfiability of a formula F in PL

- **Step 1**: Transforming F to CNF by Tseitin Transformation
- **Step 2**: Invoke the DPLL algorithm

The SAT Problem

- Deciding satisfiability of a formula F in PL

- **Step 1**: Transforming F to CNF by Tseitin Transformation
- **Step 2**: Invoke the DPLL algorithm

What is CNF?
Why not DNF or NNF?

DPLL

- Davis–Putnam–Logemann–Loveland (DPLL) algorithm
- **Goal**: Find a solution to satisfy the input formula

DPLL

- $(p \vee q \vee \neg r) \wedge (p \vee q \vee r) \wedge (p \vee \neg q) \wedge \neg p$

- **Goal**: Find a solution to satisfy the input formula

- Repeat the following steps:
- (1) Choose a variable, p, assign true or false to the variable
- (2) Propagate the value of p

DPLL

- $(p \vee q \vee \neg r) \wedge (p \vee q \vee r) \wedge (p \vee \neg q) \wedge \neg p$

```
bool DPLL (F, I) {
    if (F == false) return UNSAT;
    if (F == true)  return SAT;

    p = choose(F);
    bool ret = DPLL(F[p→true], I[p→true])
    if (ret == SAT) return SAT;

    return DPLL(F[p→false], I[p→ false]);
}
```

DPLL

- $(p \lor q \lor \neg r) \land (p \lor q \lor r) \land (p \lor \neg q) \land \neg p$

```
bool DPLL (F, I) {
    if (F == false) return UNSAT;
    if (F == true)  return SAT;

    p = choose(F);
    bool ret = DPLL(F[p→true], I[p→true])
    if (ret == SAT) retu
    
    return DPLL(F[p→fal
}
```

Optimizations may be applied!

Theorem [Resolution]

$$(a_0 \lor a_1 \lor \cdots \lor a_n \lor \boldsymbol{c}) \land (b_0 \lor b_1 \lor \cdots \lor b_m \lor \neg \boldsymbol{c})$$

can be simplified into

$$(a_0 \lor a_1 \lor \cdots \lor a_n \lor b_0 \lor b_1 \lor \cdots \lor b_m)$$

DPLL

- $(p \lor q \lor \boxed{\neg r}) \land (p \lor q \lor \boxed{r}) \land (p \lor \neg q) \land \neg p$

DPLL

- $(p \vee q \vee \boxed{\neg r}) \wedge (p \vee q \vee \boxed{r}) \wedge (p \vee \neg q) \wedge \neg p$
- $(p \vee q) \wedge (p \vee \neg q) \wedge \neg p$

DPLL

- $(p \vee q \vee \boxed{\neg r}) \wedge (p \vee q \vee \boxed{r}) \wedge (p \vee \neg q) \wedge \neg p$
- $(p \vee \boxed{q}) \wedge (p \vee \boxed{\neg q}) \wedge \neg p$

DPLL

- $(p \lor q \lor \boxed{\neg r}) \land (p \lor q \lor \boxed{r}) \land (p \lor \neg q) \land \neg p$
- $(p \lor \boxed{q}) \land (p \lor \boxed{\neg q}) \land \neg p$
- $p \land \neg p$

DPLL

- $(p \vee q \vee \boxed{\neg r}) \wedge (p \vee q \vee \boxed{r}) \wedge (p \vee \neg q) \wedge \neg p$
- $(p \vee \boxed{q}) \wedge (p \vee \boxed{\neg q}) \wedge \neg p$
- $p \wedge \neg p$ **False! Unsatisfiable!**

DPLL

- $(p \vee q \vee \boxed{\neg r}) \wedge (p \vee q \vee \boxed{r}) \wedge (p \vee \neg q) \wedge \neg p$
- $(p \vee \boxed{q}) \wedge (p \vee \boxed{\neg q}) \wedge \neg p$
- $p \wedge \neg p$ **False! Unsatisfiable!**

Theorem [Resolution]

$$(a_0 \vee a_1 \vee \cdots \vee a_n \vee \boldsymbol{c}) \wedge (b_0 \vee b_1 \vee \cdots \vee b_m \vee \neg \boldsymbol{c})$$

can be simplified into

$$(a_0 \vee a_1 \vee \cdots \vee a_n \vee b_0 \vee b_1 \vee \cdots \vee b_m)$$

Tseitin Transformation

- $(r \Rightarrow p) \Rightarrow (\neg(q \wedge r) \Rightarrow p)$

On the Complexity of Derivation in Propositional Calculus

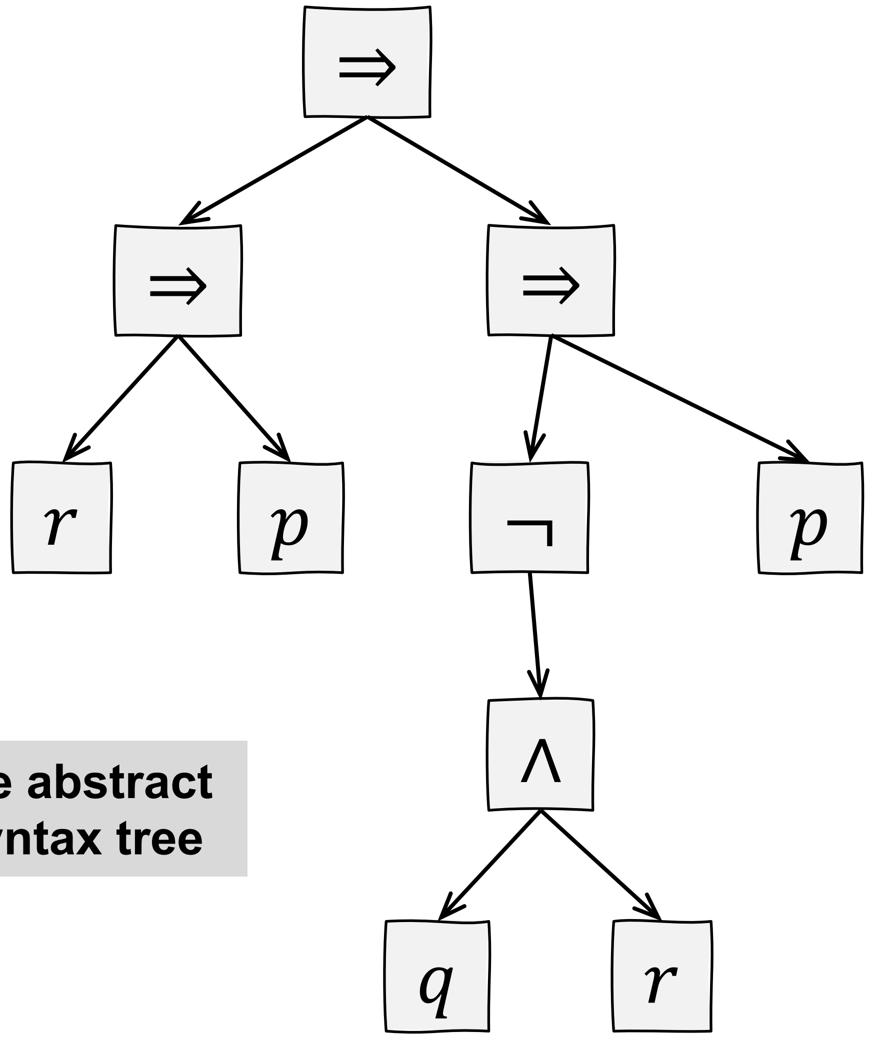
G. S. Tseitin 1966

1. Formulation of the Problem and Principal Results

The question of the minimum complexity of derivation of a given formula in classical propositional calculus is considered in this article and it is proved that estimates of complexity may vary considerably among the various forms of propositional calculus. The forms of propositional calculus used in the present article are somewhat unusual,† but the results obtained for them can, in principle, be extended to the usual forms of propositional calculus.

Tseitin Transformation

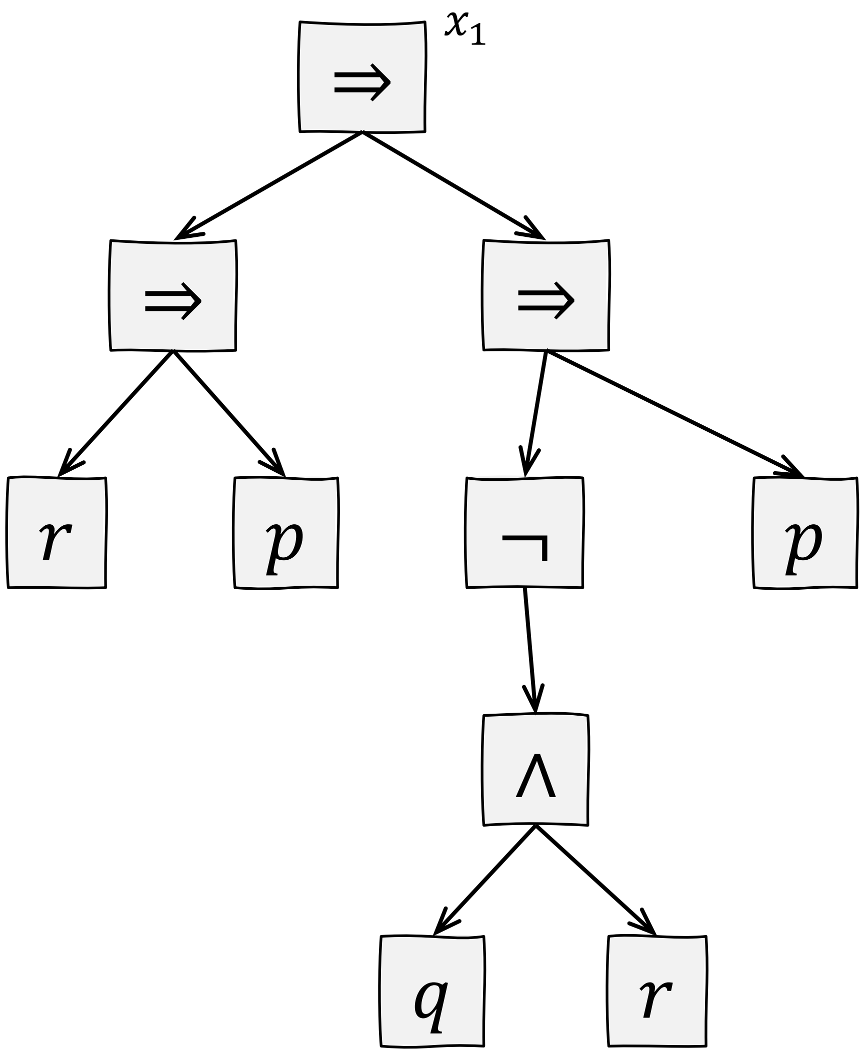
- $(r \Rightarrow p) \Rightarrow (\neg(q \wedge r) \Rightarrow p)$



The abstract syntax tree

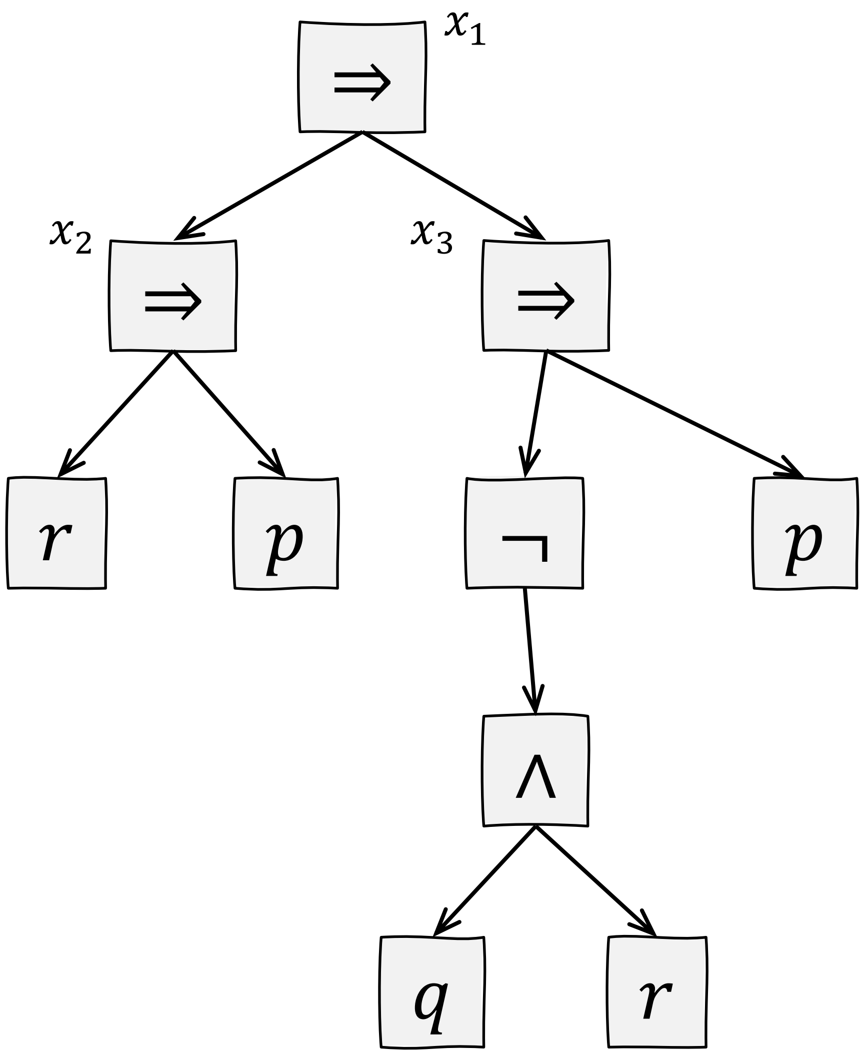
Tseitin Transformation

- $\underbrace{(r \Rightarrow p) \Rightarrow (\neg(q \land r) \Rightarrow p)}_{x_1}$



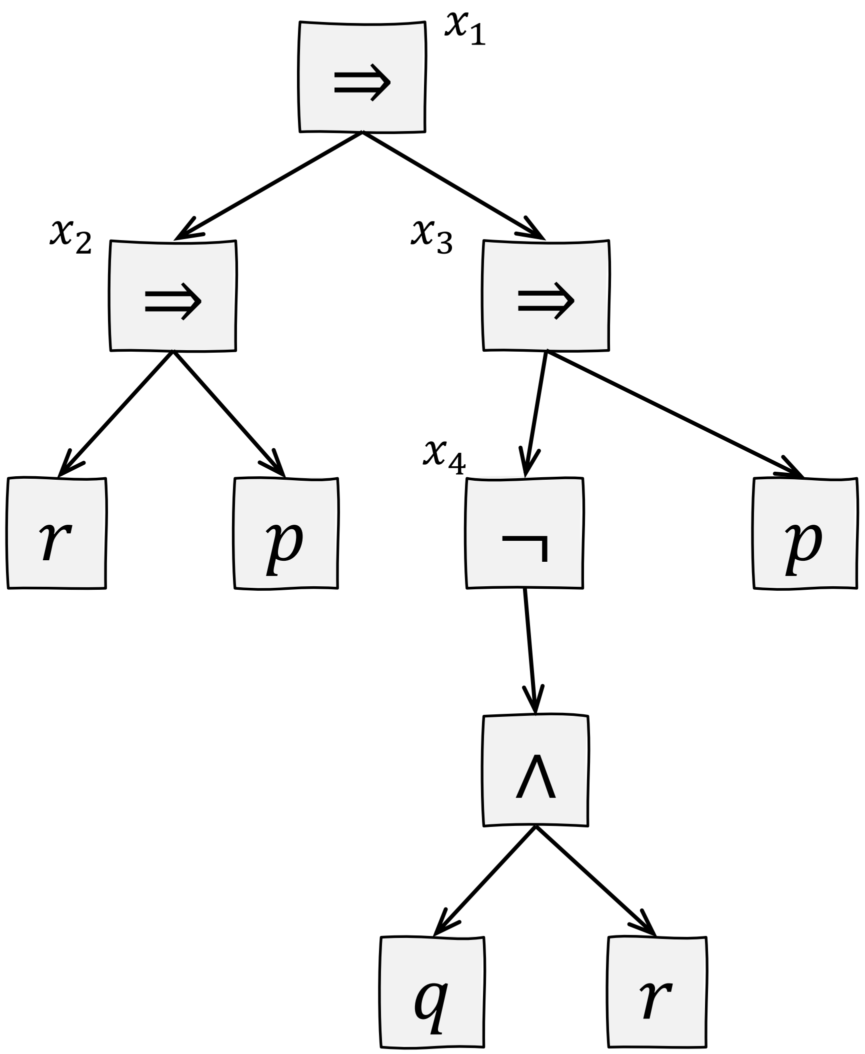
Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\neg(q \wedge r) \Rightarrow p)}^{x_3}}_{x_1}$



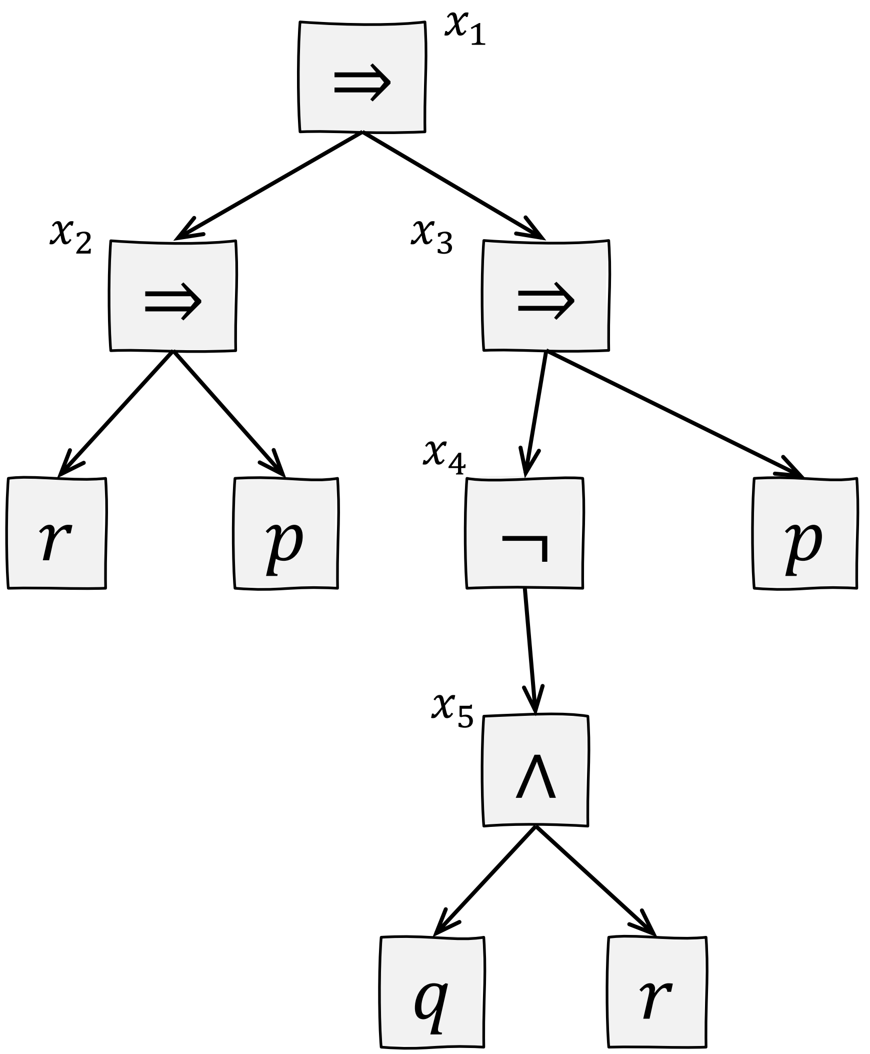
Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\neg\underbrace{(q \wedge r)}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$



Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\underbrace{\neg \overbrace{(q \land r)}^{x_5}}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$



Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\underbrace{\neg\overbrace{(q \wedge r)}^{x_5}}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$

$$x_1 \wedge (x_1 \Leftrightarrow (x_2 \Rightarrow x_3))$$

Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\underbrace{\neg\overbrace{(q \land r)}^{\color{blue}{x_5}}}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$

$$x_1 \ \land (x_1 \Leftrightarrow (x_2 \Rightarrow x_3))$$
$$\land \ x_2 \Leftrightarrow (r \Rightarrow p)$$

Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\underbrace{\neg(\overbrace{q \land r}^{x_5})}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$

$$
\begin{aligned}
& x_1 \land (x_1 \Leftrightarrow (x_2 \Rightarrow x_3)) \\
\land\ & x_2 \Leftrightarrow (r \Rightarrow p) \\
\land\ & x_3 \Leftrightarrow (x_4 \Rightarrow p)
\end{aligned}
$$

Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\underbrace{\neg\overbrace{(q \wedge r)}^{x_5}}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$

$$
\begin{aligned}
& x_1 \wedge (x_1 \Leftrightarrow (x_2 \Rightarrow x_3)) \\
\wedge\ & x_2 \Leftrightarrow (r \Rightarrow p) \\
\wedge\ & x_3 \Leftrightarrow (x_4 \Rightarrow p) \\
\wedge\ & x_4 \Leftrightarrow \neg x_5
\end{aligned}
$$

Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\underbrace{\neg\overbrace{(q \wedge r)}^{x_5}}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$

$$
\begin{aligned}
& x_1 \wedge (x_1 \Leftrightarrow (x_2 \Rightarrow x_3)) \\
\wedge\ & x_2 \Leftrightarrow (r \Rightarrow p) \\
\wedge\ & x_3 \Leftrightarrow (x_4 \Rightarrow p) \\
\wedge\ & x_4 \Leftrightarrow \neg x_5 \\
\wedge\ & x_5 \Leftrightarrow q \wedge r
\end{aligned}
$$

Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\underbrace{\neg\overbrace{(q \wedge r)}^{x_5}}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$

$$
\begin{aligned}
& x_1 \wedge (x_1 \Leftrightarrow (x_2 \Rightarrow x_3)) \\
\wedge\ & x_2 \Leftrightarrow (r \Rightarrow p) \\
\wedge\ & x_3 \Leftrightarrow (x_4 \Rightarrow p) \\
\wedge\ & x_4 \Leftrightarrow \neg x_5 \\
\wedge\ & x_5 \Leftrightarrow q \wedge r
\end{aligned}
$$

Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\underbrace{\neg \overbrace{(q \land r)}^{x_5}}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$

$$
\begin{aligned}
& x_1 \land (x_1 \Leftrightarrow (x_2 \Rightarrow x_3)) \\
\land\ & x_2 \Leftrightarrow (r \Rightarrow p) \\
\land\ & x_3 \Leftrightarrow (x_4 \Rightarrow p) \\
\land\ & x_4 \Leftrightarrow \neg x_5 \\
\land\ & x_5 \Leftrightarrow q \land r
\end{aligned}
$$

$x_4 \Rightarrow \neg x_5 \land \neg x_5 \Rightarrow x_4$

Tseitin Transformation

- $\underbrace{\overbrace{(r \Rightarrow p)}^{x_2} \Rightarrow \overbrace{(\underbrace{\neg \overbrace{(q \wedge r)}^{x_5}}_{x_4} \Rightarrow p)}^{x_3}}_{x_1}$

$$
\begin{aligned}
& x_1 \wedge (x_1 \Leftrightarrow (x_2 \Rightarrow x_3)) \\
\wedge\ & x_2 \Leftrightarrow (r \Rightarrow p) \\
\wedge\ & x_3 \Leftrightarrow (x_4 \Rightarrow p) \\
\wedge\ & x_4 \Leftrightarrow \neg x_5 \\
\wedge\ & x_5 \Leftrightarrow q \wedge r
\end{aligned}
$$

$$x_4 \Rightarrow \neg x_5 \wedge \neg x_5 \Rightarrow x_4$$

$$(\neg x_4 \vee \neg x_5) \wedge (x_5 \vee x_4)$$

Solving Path Constraints

- How can we check the satisfiability of a constraint?

$$\alpha_a \neq 0 \wedge \alpha_b = 0 \wedge \neg(2(\alpha_a + \alpha_b) - 4 \neq 0)$$

- Constraint solver
  - Satisfiable => A possible solution
  - Unsatisfiable
  - Unknown (due to timeout)

Z3 https://github.com/Z3Prover/z3

CVC5 https://cvc5.github.io

Satisfiability Modulo Theories

- For the theory of bit vectors
  - **Step 1**: Word-Level Preprocessing
  - **Step 2**: Bit-Blasting (From FOL to PL)
  - **Step 3**: DPLL/CDCL

Satisfiability Modulo Theories

- For the theory of bit vectors
  - **Step 1**: Word-Level Preprocessing
  - **Step 2**: Bit-Blasting (From FOL to PL)
  - **Step 3**: DPLL/CDCL

Example of Step 2:

$x = [a_7, a_6, \dots, a_0], y = [b_7, b_6, \dots, b_0]$ and $z = [c_7, c_6, \dots, c_0]$ are three 8bit bit vectors

$$x \mid y = z \xrightarrow{\text{Step 2}} \bigwedge_{i=0}^{7} ((a_i \vee b_i) \Leftrightarrow c_i)$$

PART III: Pointer Analysis

Andersen & Steensgaard Algorithm

Inclusion-based pointer analysis (Andersen's Algorithm)

Statement	Constraint	Shorthand
y = &x	$pts(y) \supseteq \{x\}$	
y = x	$pts(y) \supseteq pts(x)$	
*y = x	$\forall v \in pts(y)$: $pts(v) \supseteq pts(x)$	$pts(*y) \supseteq pts(x)$
y = *x	$\forall v \in pts(x)$: $pts(y) \supseteq pts(v)$	$pts(y) \supseteq pts(*x)$

Unification-based pointer analysis (Steensgaard's Algorithm)

Statement	Constraint	Shorthand
y = &x	$pts(y) \supseteq \{x\}$	
y = x	$pts(y) = pts(x)$	
*y = x	$\forall v \in pts(y)$: $pts(v) = pts(x)$	$pts(*y) = pts(x)$
y = *x	$\forall v \in pts(x)$: $pts(y) = pts(v)$	$pts(y) = pts(*x)$

Andersen's Alg. as Graph Closure

- Each graph node x denotes the points-to set $pts(x)$
- Solving the set constraints via a dynamic transitive closure

Andersen’s Alg. as Graph Closure

- Each graph node x denotes the points-to set $pts(x)$
- Solving the set constraints via a dynamic transitive closure

```
p  =  &a;
q  =  &b;
*p =  q;
r  =  &c;
s  =  p;
t  =  *p;
*s =  r;
```

Andersen’s Alg. as Graph Closure

- Each graph node x denotes the points-to set $pts(x)$
- Solving the set constraints via a dynamic transitive closure

```
p  =  &a;
q  =  &b;
*p =  q;
r  =  &c;
s  =  p;
t  =  *p;
*s =  r;
```

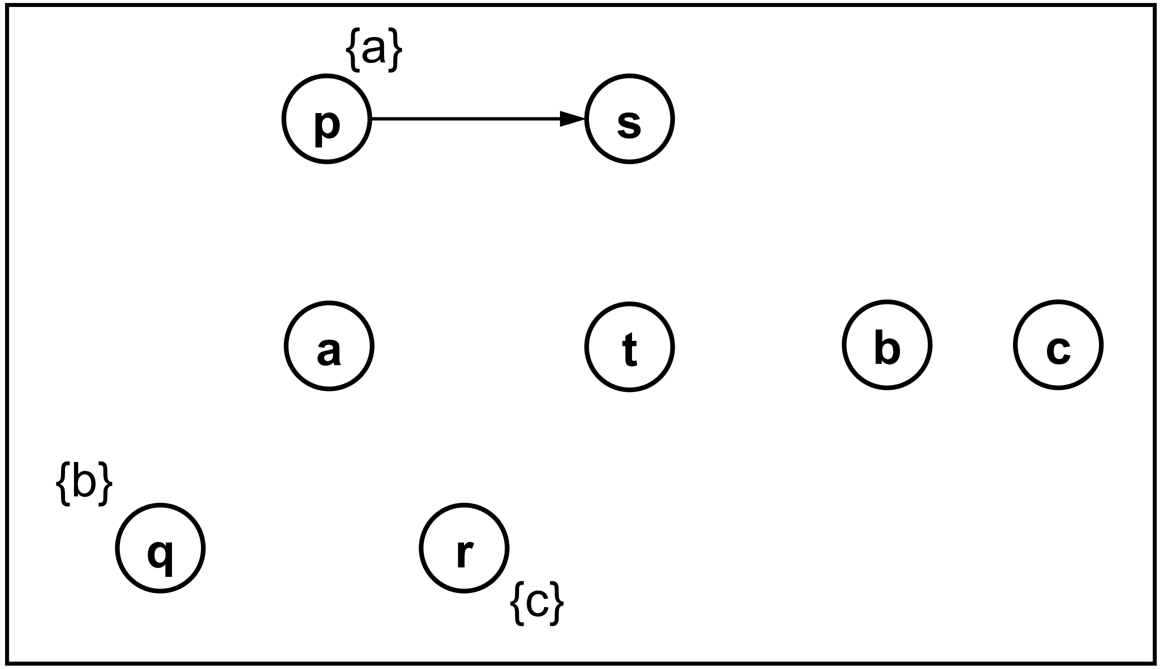
```
pts(p) ⊇ {a};
pts(q) ⊇ {b};
...
pts(r) ⊇ {c};
pts(s) ⊇ pts(p);
...
...
```

Andersen's Alg. as Graph Closure

- Each graph node x denotes the points-to set $pts(x)$
- Solving the set constraints via a dynamic transitive closure

```
p  =  &a;
q  =  &b;
*p =  q;
r  =  &c;
s  =  p;
t  =  *p;
*s =  r;
```

```
pts(p) ⊇ {a};
pts(q) ⊇ {b};
...
pts(r) ⊇ {c};
pts(s) ⊇ pts(p);
...
...
```

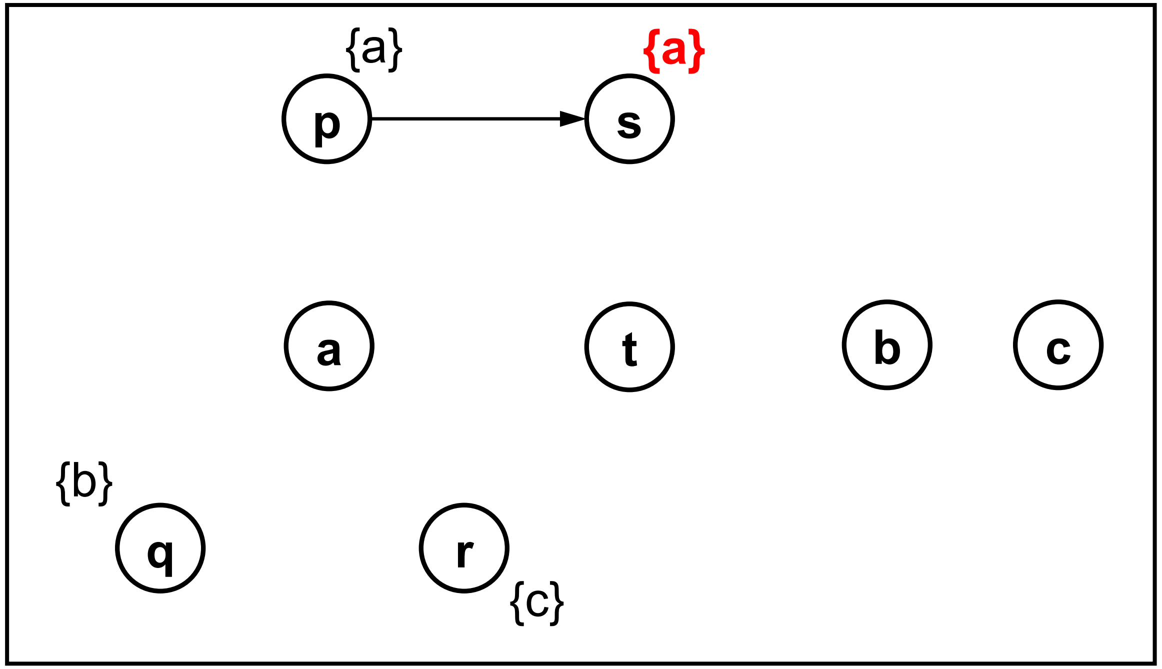


Andersen's Alg. as Graph Closure

- Each graph node x denotes the points-to set $pts(x)$
- Solving the set constraints via a dynamic transitive closure

```
p  =  &a;
q  =  &b;
*p =  q;
r  =  &c;
s  =  p;
t  =  *p;
*s =  r;
```

```
pts(p) ⊇ {a};
pts(q) ⊇ {b};
...
pts(r) ⊇ {c};
pts(s) ⊇ pts(p);
...
...
```

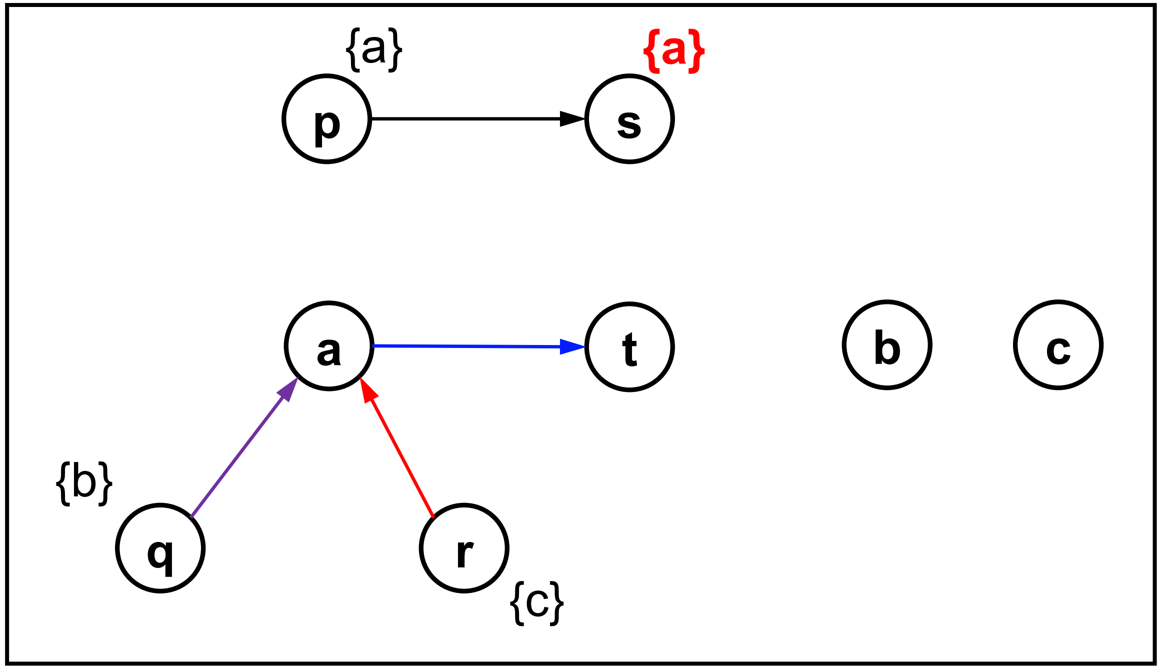


Andersen's Alg. as Graph Closure

- Each graph node x denotes the points-to set $pts(x)$
- Solving the set constraints via a dynamic transitive closure

```
p  = &a;
q  = &b;
*p = q;
r  = &c;
s  = p;
t  = *p;
*s = r;
```

```
...
...
pts(*p) ⊇ pts(q)
...
...
pts(t) ⊇ pts(*p)
pts(*s) ⊇ pts(r)
```

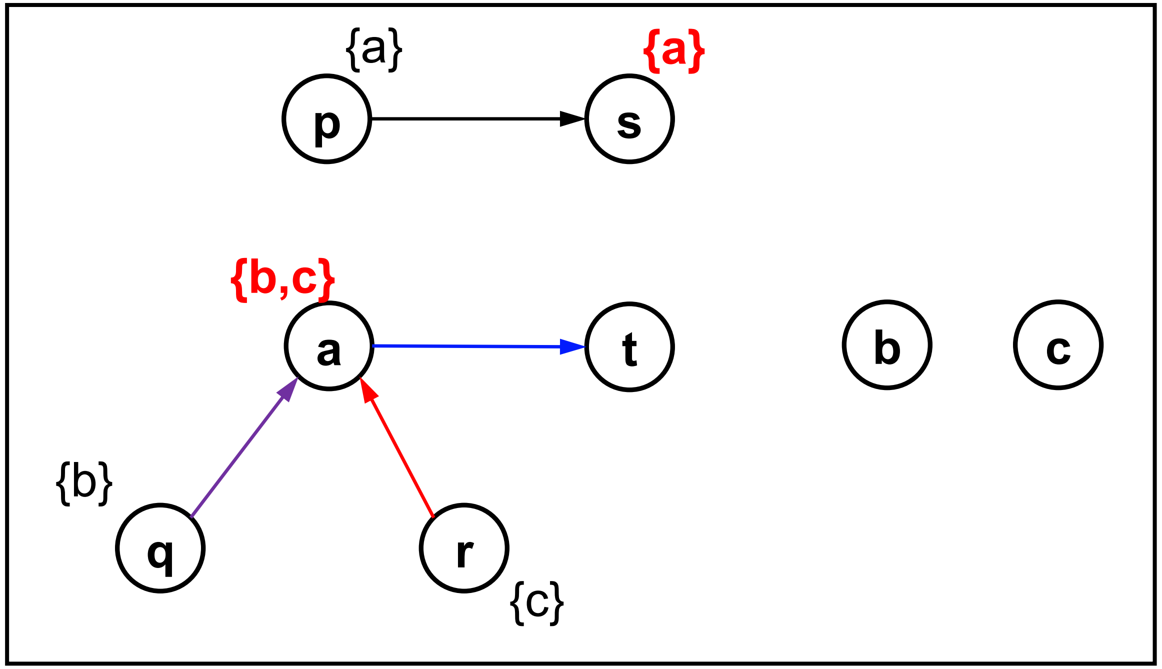


Andersen's Alg. as Graph Closure

- Each graph node x denotes the points-to set $pts(x)$
- Solving the set constraints via a dynamic transitive closure

```
p  =  &a;
q  =  &b;
*p =  q;
r  =  &c;
s  =  p;
t  =  *p;
*s =  r;
```

```
...
...
pts(*p) ⊇ pts(q)
...
...
pts(t) ⊇ pts(*p)
pts(*s) ⊇ pts(r)
```

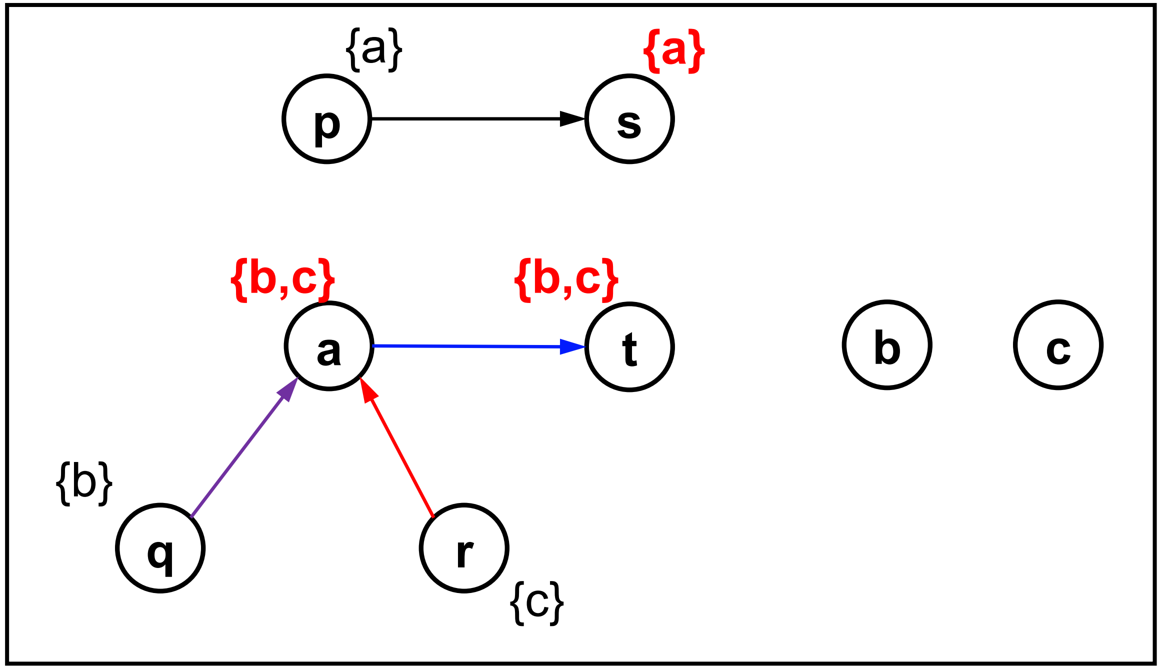


Andersen's Alg. as Graph Closure

- Each graph node x denotes the points-to set $pts(x)$
- Solving the set constraints via a dynamic transitive closure

```
p  =  &a;
q  =  &b;
*p =  q;
r  =  &c;
s  =  p;
t  =  *p;
*s =  r;
```

```
...
...
pts(*p) ⊇ pts(q)
...
...
pts(t) ⊇ pts(*p)
pts(*s) ⊇ pts(r)
```

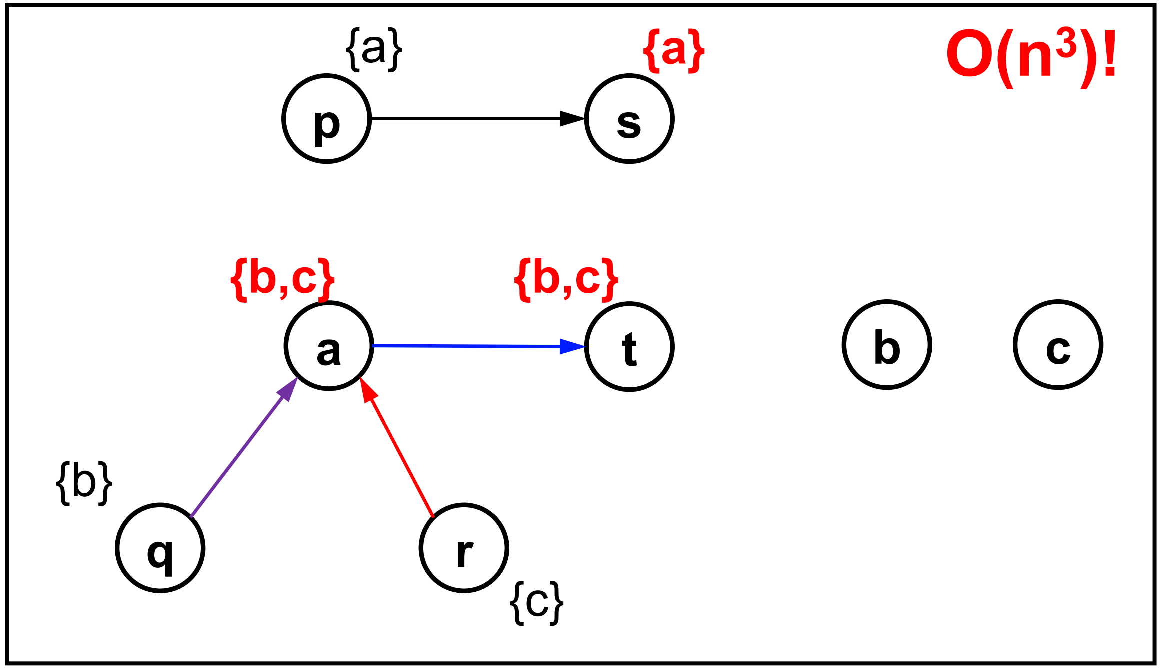


Andersen's Alg. as Graph Closure

- Each graph node x denotes the points-to set $pts(x)$
- Solving the set constraints via a dynamic transitive closure

```
p  =  &a;
q  =  &b;
*p =  q;
r  =  &c;
s  =  p;
t  =  *p;
*s =  r;
```

```
...
...
pts(*p) ⊇ pts(q)
...
...
pts(t) ⊇ pts(*p)
pts(*s) ⊇ pts(r)
```

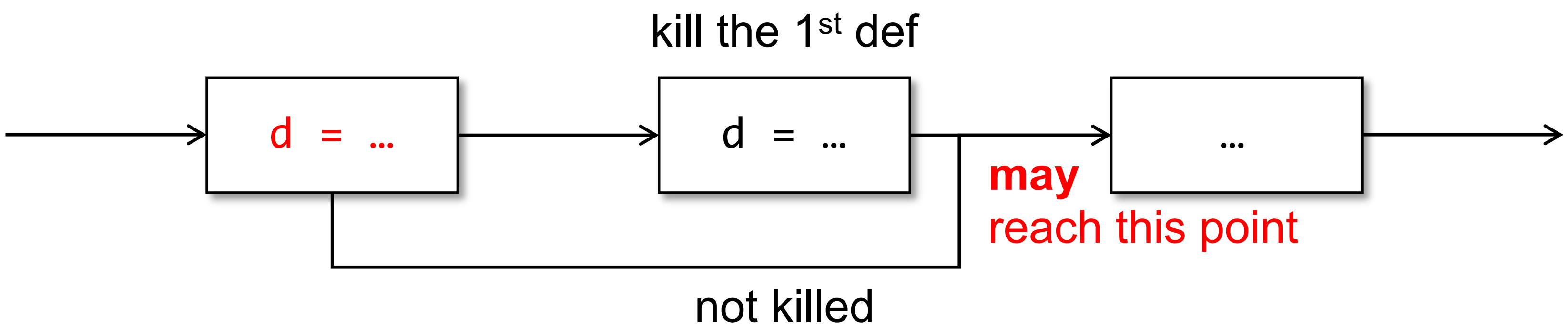


PART IV: Datalog-Based Analysis

Reaching Definitions by Datalog

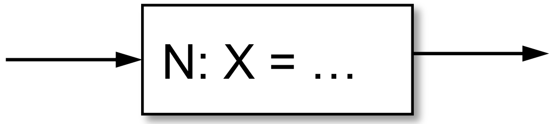
Recap: Reaching Definition

- A definition *d* may reach a program point, if there is a path from *d* to the program point such that *d* is not killed along the path.



Reaching Definitions by Datalog

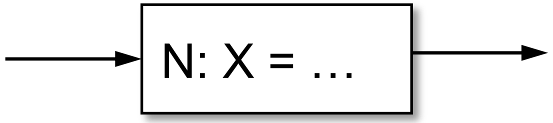
- def(B, N, X)
  - the Nth statement in Block B may define the variable X.



Block B

Reaching Definitions by Datalog

- def(B, N, X)
  - the Nth statement in Block B may define the variable X.

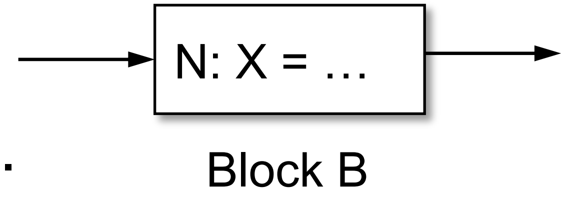


Block B

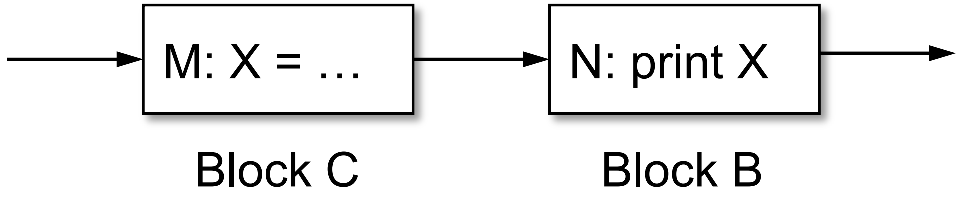
- succ(B, N, C)
  - block C is a successor of block B, and B has N statements.

Reaching Definitions by Datalog

- def(B, N, X)
  - the Nth statement in Block B may define the variable X.



- succ(B, N, C)
  - block C is a successor of block B, and B has N statements.
- rd(B, N, C, M, X)
  - the definition of variable X at the Mth statement of block C reaches the Nth statement in B.

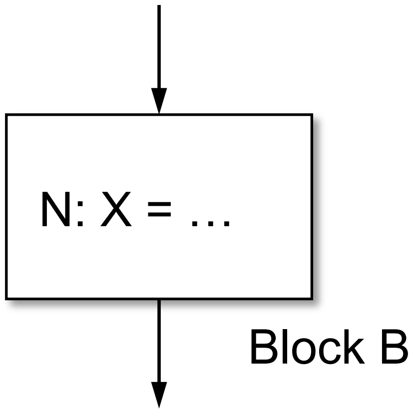


Reaching Definitions by Datalog

- rd(B, N, B, N, X) :- def(B, N, X)
- rd(B, N, C, M, X) :- rd(B, N-1, C, M, X), def(B, N, Y), X≠Y
- rd(B, 0, C, M, X) :- rd(D, N, C, M, X), succ(D, N, B)

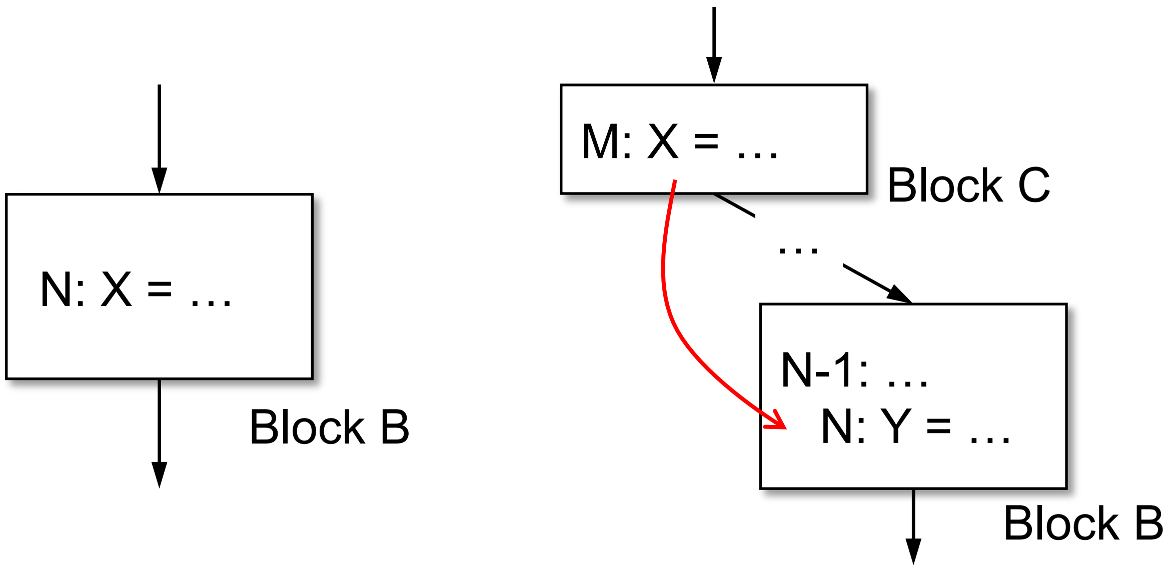
Reaching Definitions by Datalog

- rd(B, N, B, N, X) :- def(B, N, X)
- rd(B, N, C, M, X) :- rd(B, N-1, C, M, X), def(B, N, Y), X≠Y
- rd(B, 0, C, M, X) :- rd(D, N, C, M, X), succ(D, N, B)



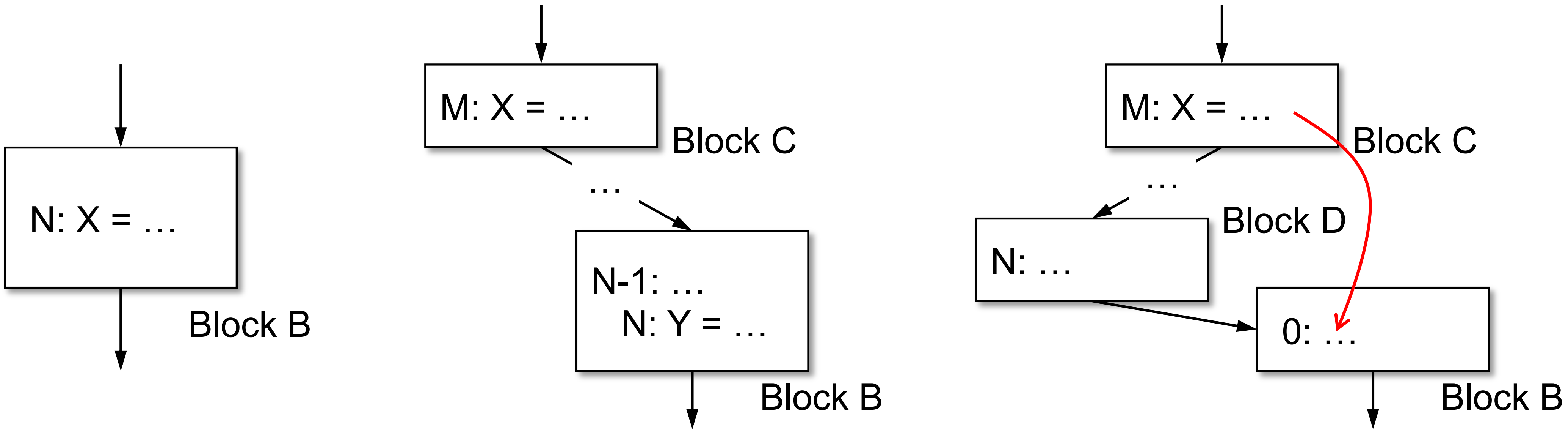
Reaching Definitions by Datalog

- rd(B, N, B, N, X) :- def(B, N, X)
- rd(B, N, C, M, X) :- rd(B, N-1, C, M, X), def(B, N, Y), X≠Y
- rd(B, 0, C, M, X) :- rd(D, N, C, M, X), succ(D, N, B)

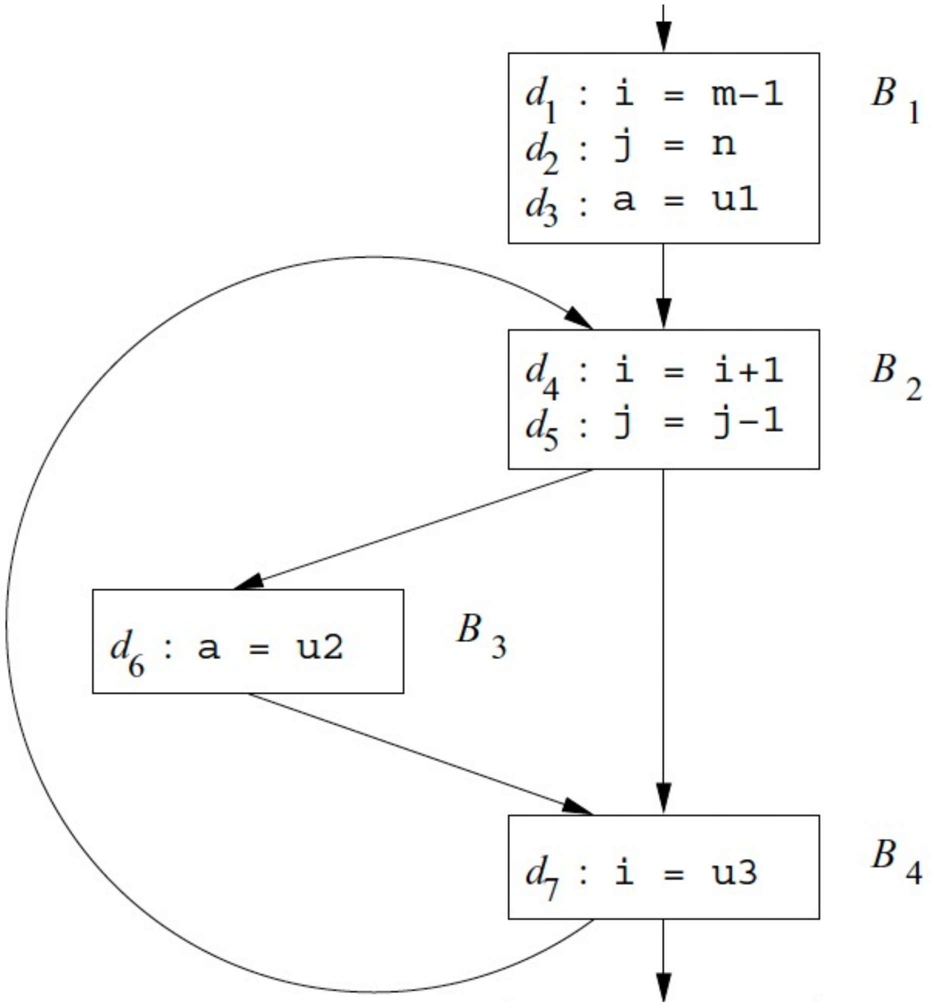


Reaching Definitions by Datalog

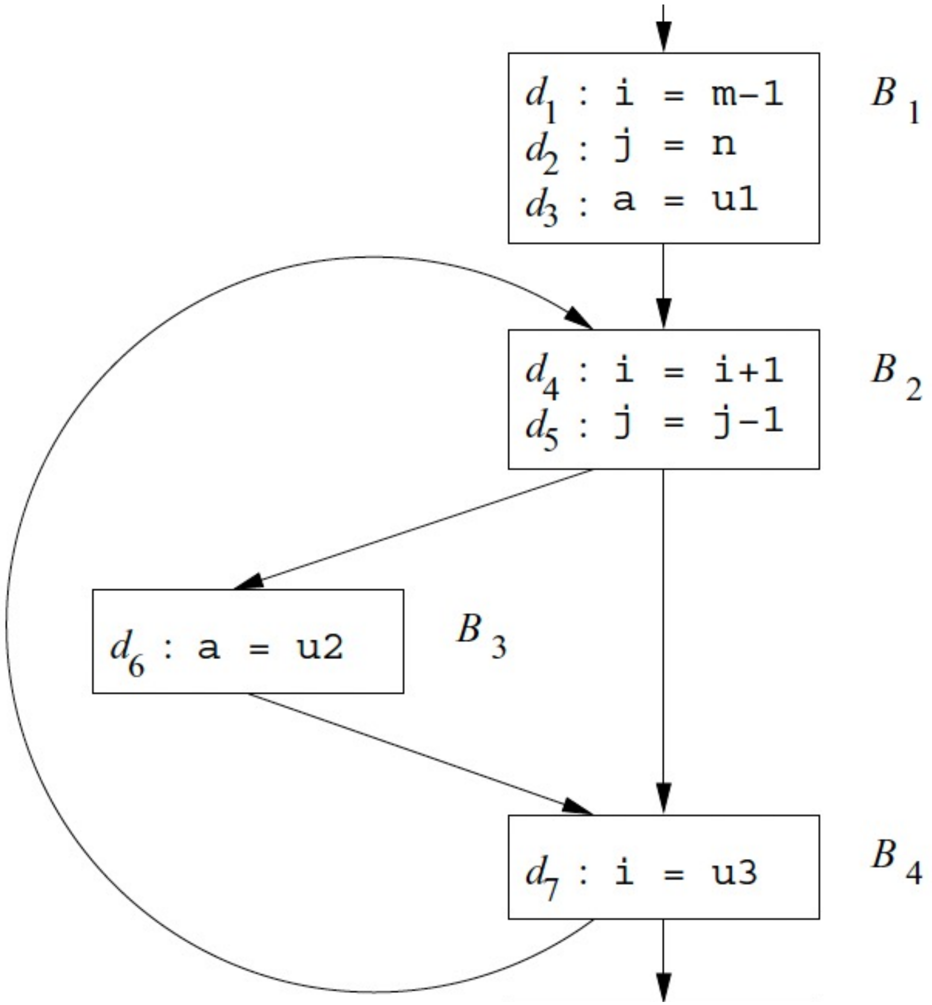
- rd(B, N, B, N, X) :- def(B, N, X)
- rd(B, N, C, M, X) :- rd(B, N-1, C, M, X), def(B, N, Y), X≠Y
- rd(B, 0, C, M, X) :- rd(D, N, C, M, X), succ(D, N, B)



Reaching Definitions by Datalog

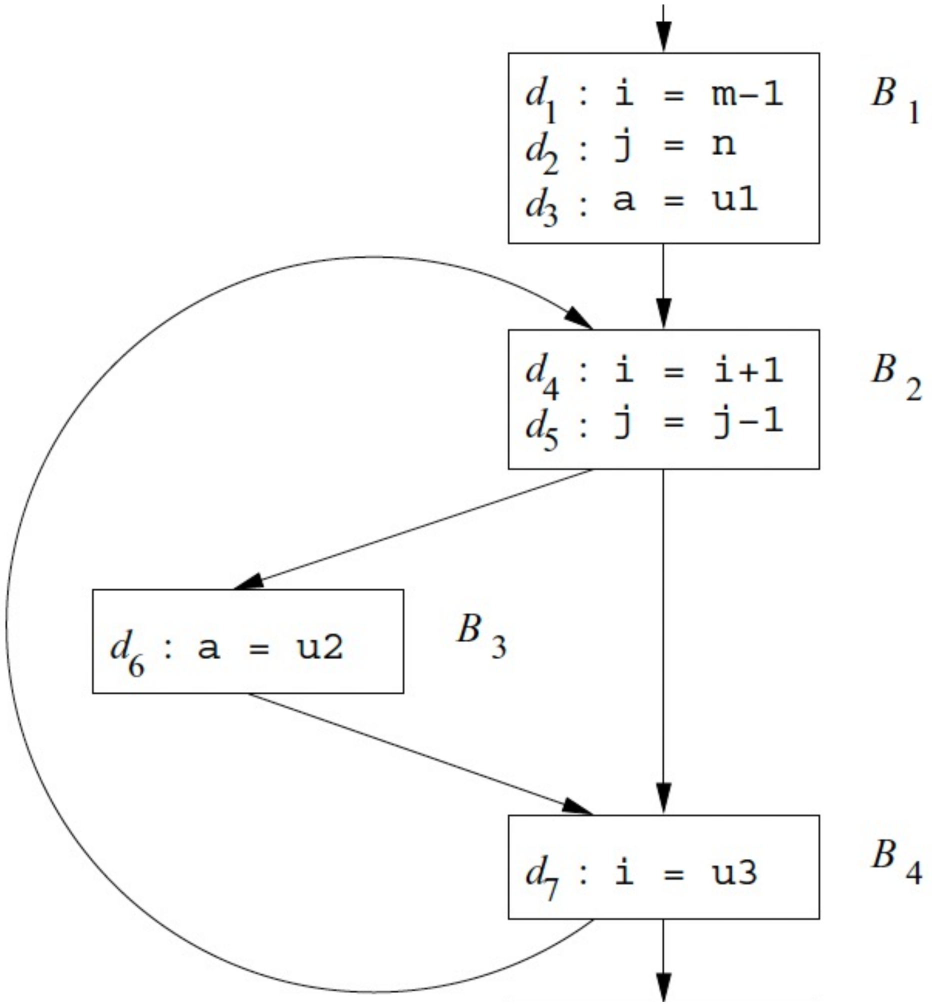


Reaching Definitions by Datalog



- def(B_1, 1, i)
- def(B_1, 2, j)
- def(B_1, 3, a)
- def(B_2, 1, i)
- def(B_2, 2, j)
- def(B_3, 1, a)
- def(B_4, 1, i)

Reaching Definitions by Datalog



- def(B_1, 1, i)
- def(B_1, 2, j)
- def(B_1, 3, a)
- def(B_2, 1, i)
- def(B_2, 2, j)
- def(B_3, 1, a)
- def(B_4, 1, i)

- succ(B_1, 3, B_2)
- succ(B_2, 2, B_3)
- succ(B_2, 2, B_4)
- succ(B_3, 1, B_4)
- succ(B_4, 1, B_2)

Reaching Definitions by Datalog

- rd(B, N, B, N, X) :- def(B, N, X)
- rd(B, N, C, M, X) :- rd(B, N-1, C, M, X), def(B, N, Y), X≠Y
- rd(B, 0, C, M, X) :- rd(D, N, C, M, X), succ(D, N, B)

- def(B_1, 1, i)
- def(B_1, 2, j)
- def(B_1, 3, a)
- def(B_2, 1, i)
- def(B_2, 2, j)
- def(B_3, 1, a)
- def(B_4, 1, i)
- succ(B_1, 3, B_2)
- succ(B_2, 2, B_3)
- succ(B_2, 2, B_4)
- succ(B_3, 1, B_4)
- succ(B_4, 1, B_2)

Reaching Definitions by Datalog

- rd(B, N, B, N, X) :- def(B, N, X)
- rd(B…
- rd(B…

We just define facts and rules.
Analysis is automatically done by Datalog engines!

- def(B_1, 1, i)
- def(B_1, 2, j)
- def(B_1, 3, a)
- def(B_2, 1, i)
- def(B_3, 1, a)
- def(B_4, 1, i)
- succ(B_1, 3, B_2)
- …2, B_3)
- succ(B_2, 2, B_4)
- succ(B_3, 1, B_4)
- succ(B_4, 1, B_2)

Query Example: rd(B_4, 1, B_1, 1, i)

THANKS!