

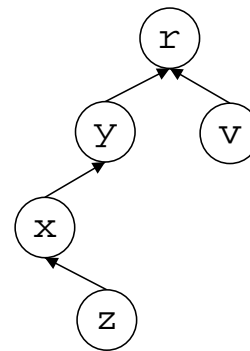
# Analysing the CHR Implementation of Union-Find

Tom Schrijvers

K.U.Leuven, Belgium

Thom Frühwirth

Universität Ulm, Germany



# Overview of the Talk

- 1. CHR
- 1. Disjoint Set Problem
- 2. Union-Find Algorithms
- 4. Union-Find in CHR
- 5. Properties
- 6. Conclusion & Future Work

# 1. CHR [Thom Frühwirth]

- Originally intended for **Constraint Solvers**

# 1. CHR [Thom Frühwirth]

- Originally intended for **Constraint Solvers**
    - ◆ highlevel semantics
    - ◆ declarative reading
- focus on *what*, not *how*

# 1. CHR [Thom Frühwirth]

- Originally intended for **Constraint Solvers**
  - ◆ highlevel semantics
  - ◆ declarative readingfocus on *what*, not *how*
- Efficient implementation of **algorithms** ?  
focus on *how*

# 1. CHR [Thom Frühwirth]

- Originally intended for **Constraint Solvers**
  - ◆ highlevel semantics
  - ◆ declarative readingfocus on *what*, not *how*
- Efficient implementation of **algorithms** ?  
focus on *how*
  - ◆ use refined operational semantics  
(Duck et al.,2004)

# 1. CHR [Thom Frühwirth]

- Originally intended for **Constraint Solvers**
  - ◆ highlevel semantics
  - ◆ declarative readingfocus on *what*, not *how*
- Efficient implementation of **algorithms** ?  
focus on *how*
  - ◆ use refined operational semantics  
(Duck et al.,2004)
- case study: union-find algorithm  
(no efficient Prolog implementation)

## 2. Disjoint Set Problem

---

Data structure for representing disjoint sets

## 2. Disjoint Set Problem

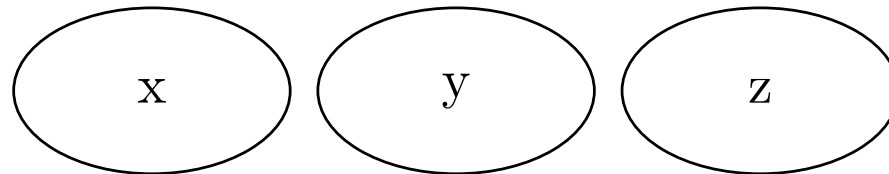
Data structure for representing disjoint sets with 3 operations:

- make: initialize singleton set
- find: return set representative
- union: join sets

## 2. Disjoint Set Problem

Data structure for representing disjoint sets with 3 operations:

- make: initialize singleton set  
 $\text{make}(x)$ ,  $\text{make}(y)$ ,  $\text{make}(z)$

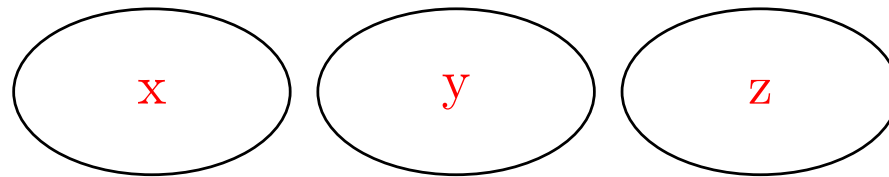


- find: return set representative
- union: join sets

## 2. Disjoint Set Problem

Data structure for representing disjoint sets with 3 operations:

- make: initialize singleton set
- find: return set representative  
 $\text{find}(x)=x, \text{find}(y)=y, \text{find}(z)=z$

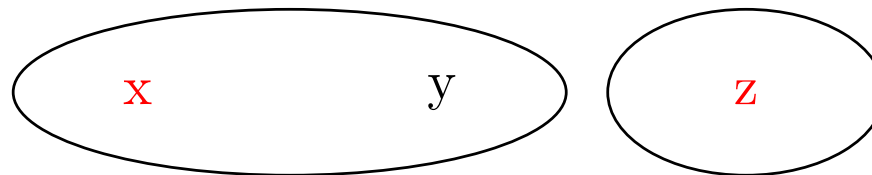


- union: join sets

## 2. Disjoint Set Problem

Data structure for representing disjoint sets with 3 operations:

- make: initialize singleton set
- find: return set representative
- union: join sets  
`union(x, y)`



## 2. Applications

disjoint sets: equivalence relation

## 2. Applications

disjoint sets: equivalence relation

- minimal spanning tree (Kruskal)  
and other graph algorithms

## 2. Applications

disjoint sets: equivalence relation

- minimal spanning tree (Kruskal) and other graph algorithms
- logical variable of Logic Programming (WAM)

# 3. Union-Find Algorithms

- 3.1 General representation
- 3.2 Naive Algorithm
- 3.3 Heuristic: Path Compression
- 3.4 Heuristic: Union-by-Rank
- 3.5 Optimal Complexity

# 3.1 General representation

## Data structure

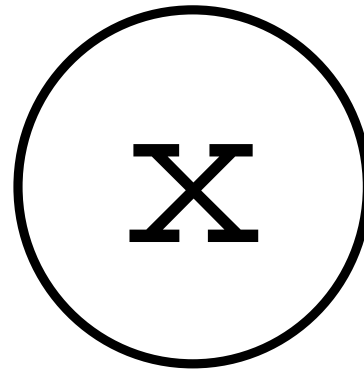
- forest of trees
- tree = set
- tree nodes = elements
- root = set representative (identifier)

## Operations

- $\text{make}(x)$ : new tree with root  $x$
- $\text{find}(x)$ : return root of tree of  $x$
- $\text{union}(x,y)$ : link trees of  $x$  and  $y$

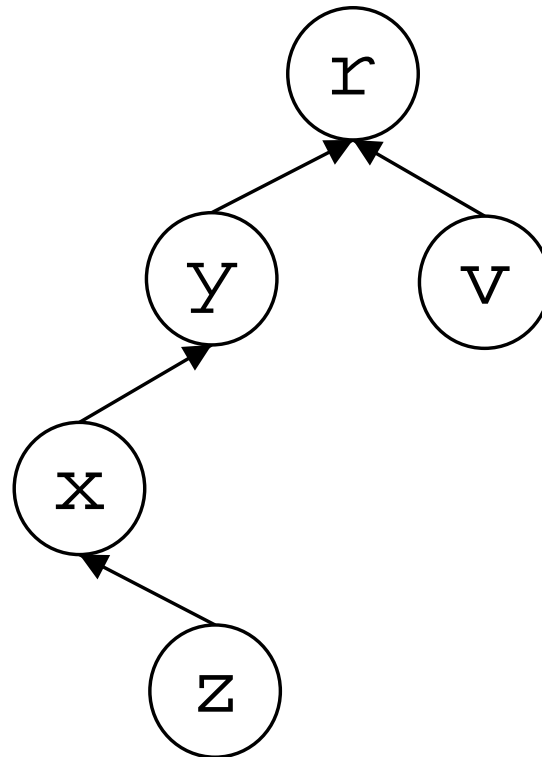
# 3.1 General representation

$\text{make}(x)$



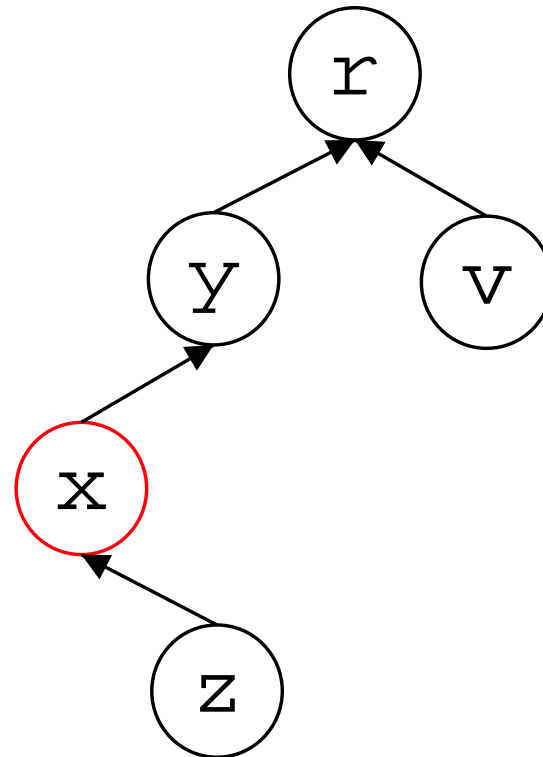
# 3.1 General representation

`find(x) =`



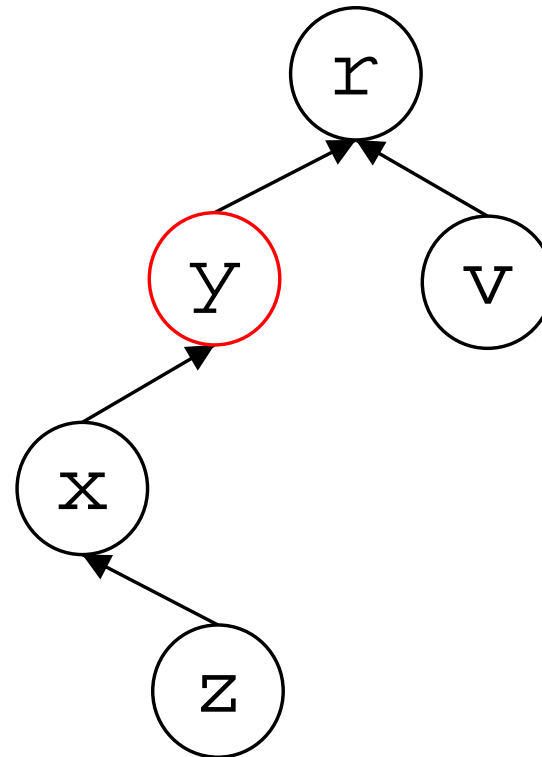
# 3.1 General representation

`find(x) =`



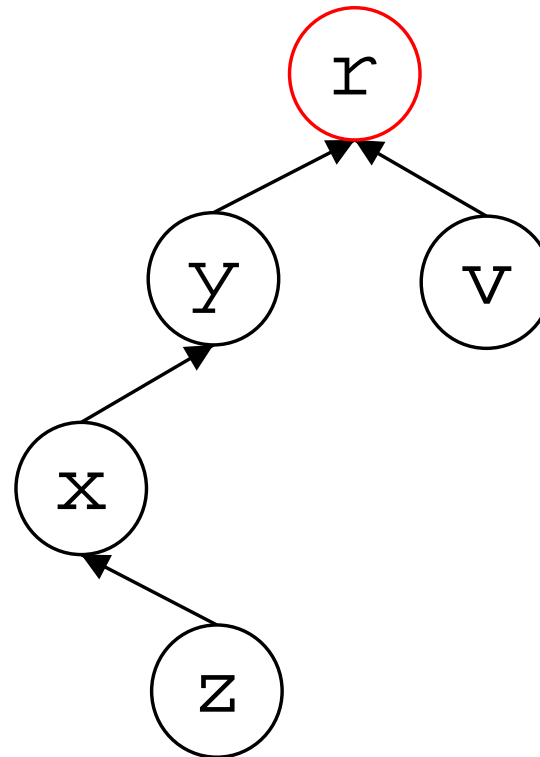
# 3.1 General representation

`find(x) =`



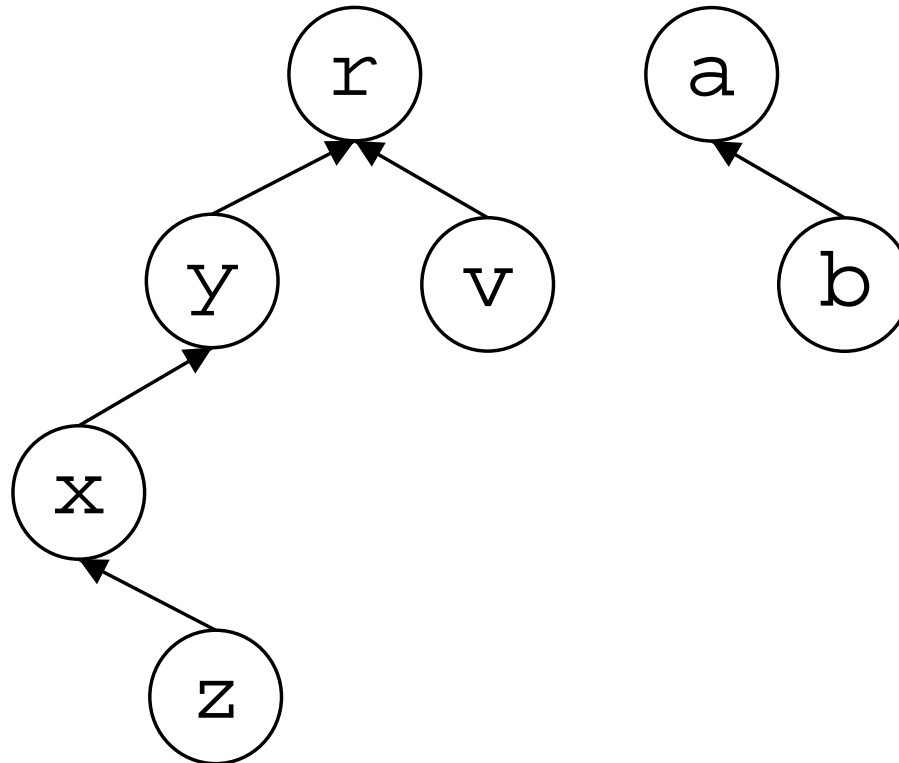
# 3.1 General representation

$\text{find}(x) = r$



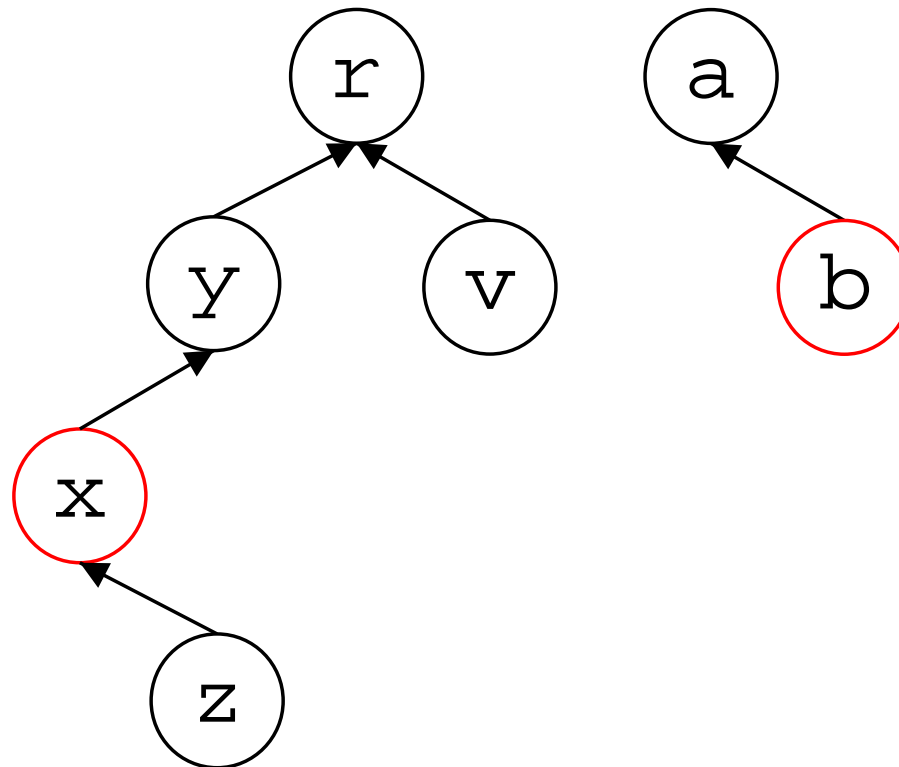
## 3.2 Naive Algorithm

`union(x, b) :`



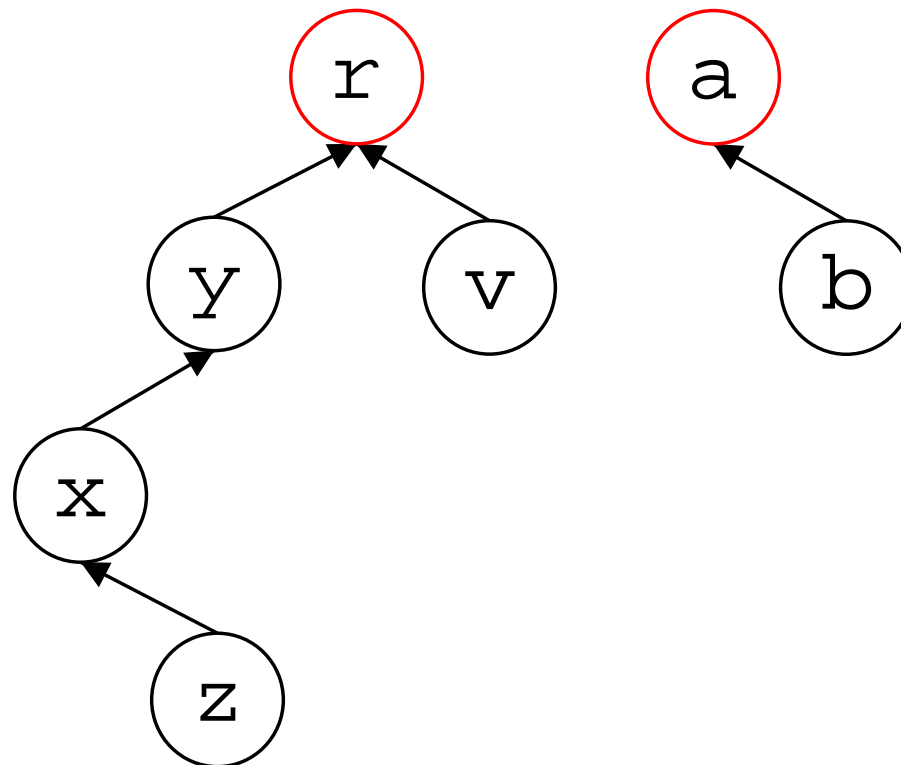
## 3.2 Naive Algorithm

`union(x, b) :`      `find(x) find(b)`



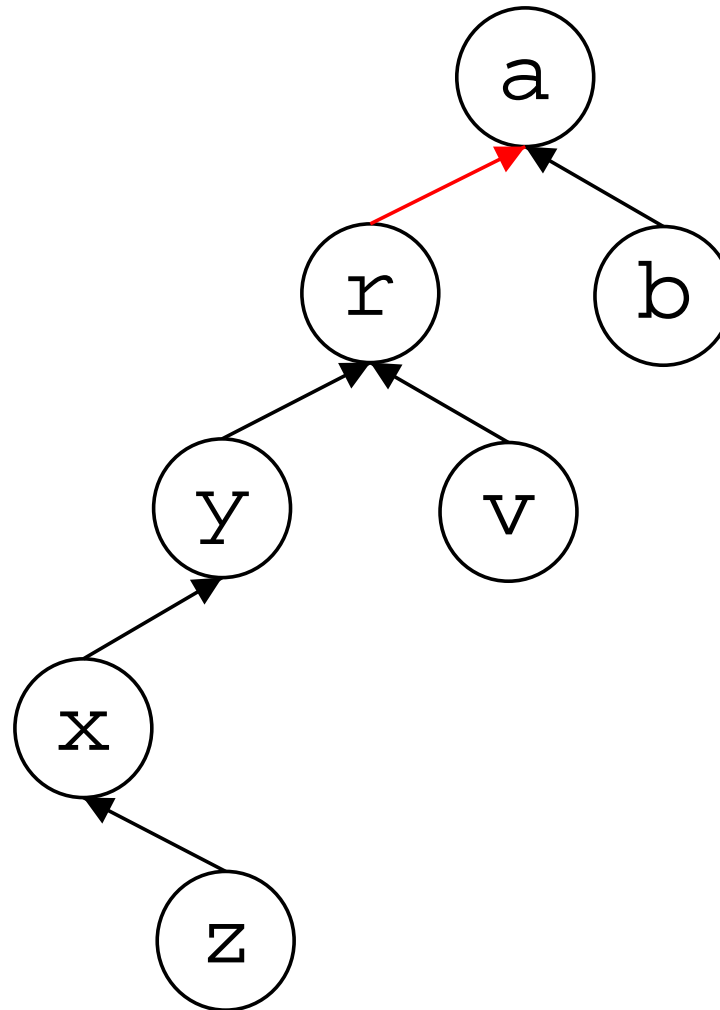
## 3.2 Naive Algorithm

`union(x, b) :`      `find(x) find(b)`



## 3.2 Naive Algorithm

$\text{union}(x, b) : \text{link}(\text{find}(x), \text{find}(b))$

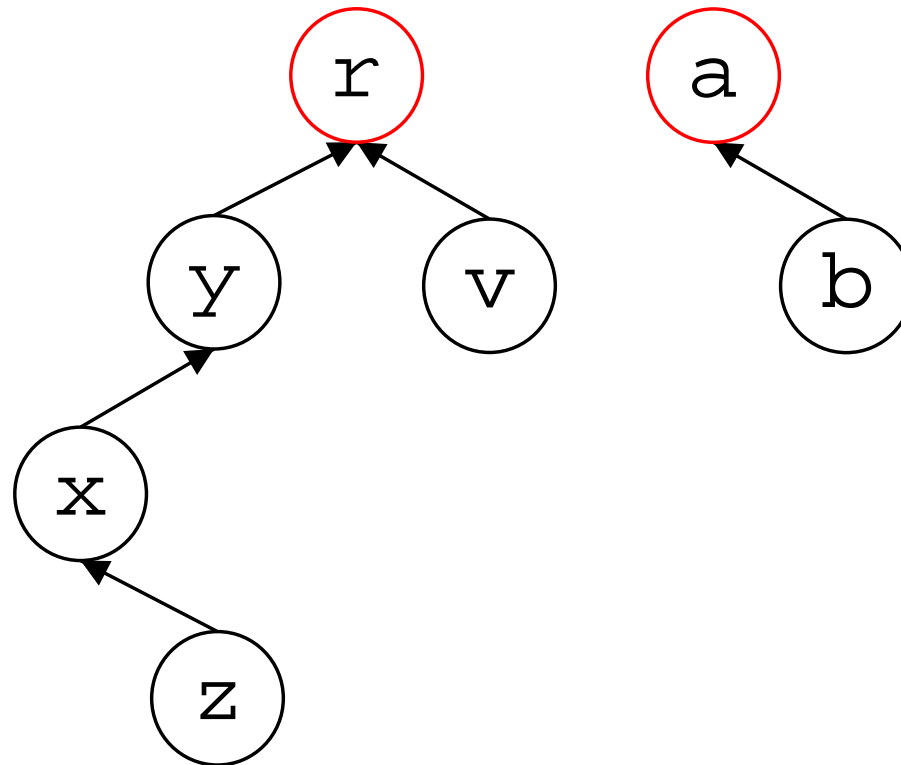


## 3.3 Union-by-rank

rank: depth of tree

$\text{rank}[r] = 3$

$\text{rank}[a] = 1$

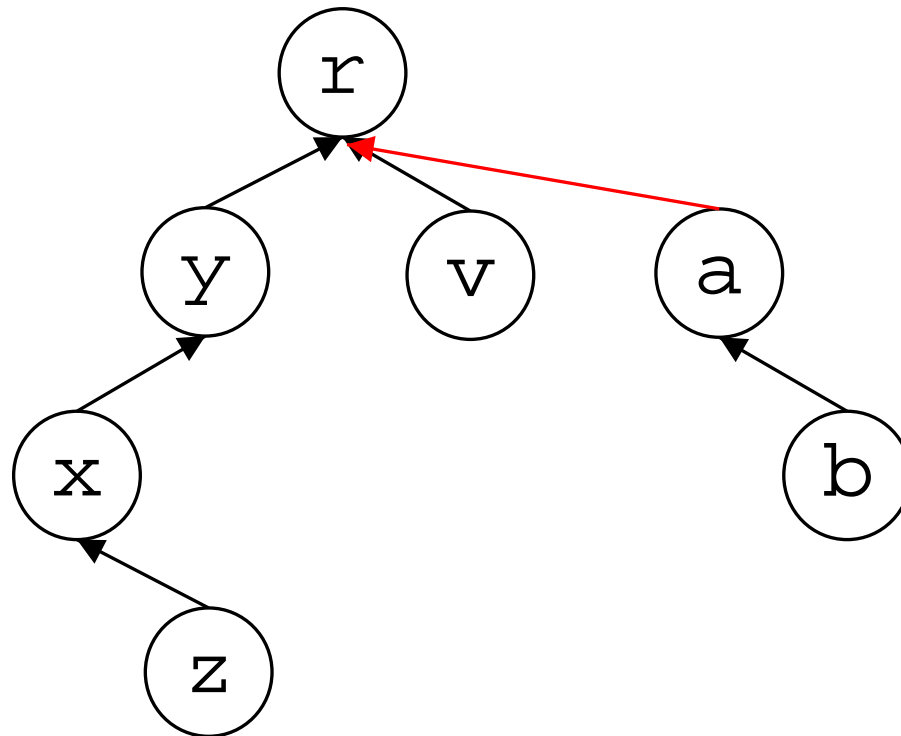


## 3.3 Union-by-rank

rank: depth of tree

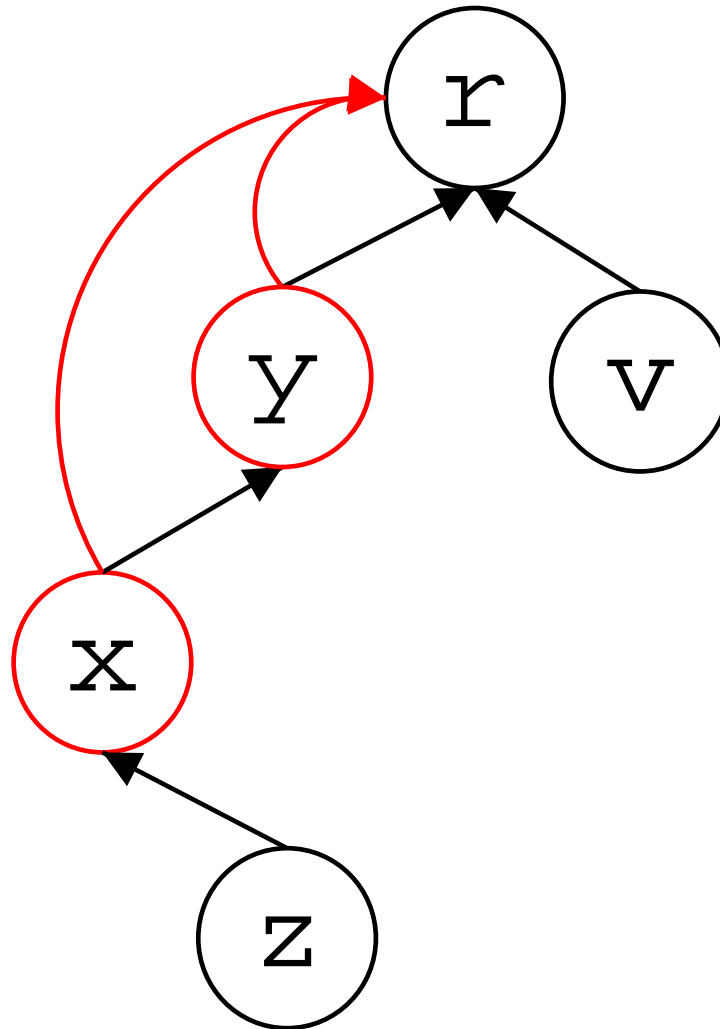
$\text{rank}[r] = 3$

$\text{rank}[a] = 1$



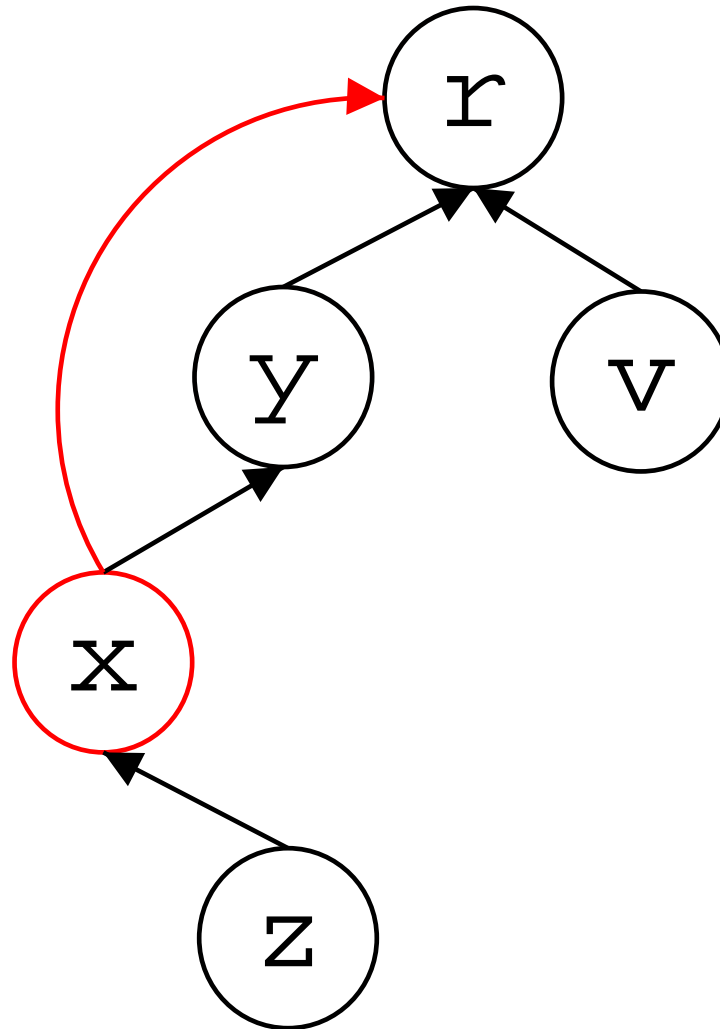
# 3.4 Path Compression

find: *shortcircuit* path to root



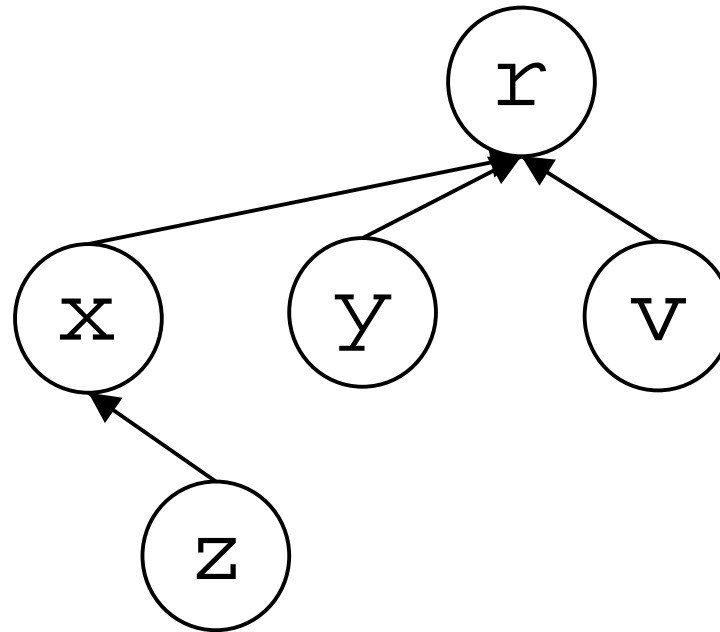
## 3.4 Path Compression

find: *shortcircuit* path to root



## 3.4 Path Compression

find: *shortcircuit* path to root



## 3.4 Optimal Complexity

worst case time complexity find operation:

- naive algorithm:  $\mathcal{O}(n)$

## 3.4 Optimal Complexity

worst case time complexity find operation:

- naive algorithm:  $\mathcal{O}(n)$
- union by rank:  $\mathcal{O}(\log n)$

## 3.4 Optimal Complexity

worst case time complexity find operation:

- naive algorithm:  $\mathcal{O}(n)$
- union by rank:  $\mathcal{O}(\log n)$
- path compression:  $\mathcal{O}(n)$  (average  $\mathcal{O}(\log n)$ )

## 3.4 Optimal Complexity

worst case time complexity find operation:

- naive algorithm:  $\mathcal{O}(n)$
- union by rank:  $\mathcal{O}(\log n)$
- path compression:  $\mathcal{O}(n)$  (average  $\mathcal{O}(\log n)$ )
- both:  $\mathcal{O}(\log n)$

## 3.4 Optimal Complexity

worst case time complexity find operation:

- naive algorithm:  $\mathcal{O}(n)$
- union by rank:  $\mathcal{O}(\log n)$
- path compression:  $\mathcal{O}(n)$  (average  $\mathcal{O}(\log n)$ )
- both:  $\mathcal{O}(\log n)$

amortized time complexity  $n$  finds:

- both:  $\mathcal{O}(n\alpha(n))$   
 $n \leq 2^{65536} \Rightarrow \alpha(n) \leq 5$

# 4. Naive Union-Find in CHR

Operation constraints:

- `make(X)`
- `union(X, Y)`
- `find(X, R)`
- `link(X, Y)`

Data constraints:

- `root(X)`
- `X ~> Y`

## 4. Union-Find in CHR

make @ make(X)  $\Leftrightarrow$  root(X).

union @ union(X,Y)  $\Leftrightarrow$  find(X,A), find(Y,B),  
link(A,B).

findNode @ X  $\sim$ > PX \ find(X,R)  $\Leftrightarrow$  find(PX,R).

findRoot @ root(X) \ find(X,R)  $\Leftrightarrow$  R=X.

linkEq @ link(X,X)  $\Leftrightarrow$  true.

link @ link(X,Y), root(X), root(Y)  $\Leftrightarrow$  Y  $\sim$ > X,  
root(X).

# 4. Union-Find in CHR

`make` @ `make(X) <=> root(X).`

`union` @ `union(X,Y) <=> find(X,A), find(Y,B),  
link(A,B).`

`findNode` @ `X ~> PX \ find(X,R) <=> find(PX,R).`

`findRoot` @ `root(X) \ find(X,R) <=> R=X.`

`linkEq` @ `link(X,X) <=> true.`

`link` @ `link(X,Y), root(X), root(Y) <=> Y ~> X,  
root(X).`

# 4. Union-Find in CHR

make @ make(X)  $\Leftrightarrow$  root(X).

union @ union(X,Y)  $\Leftrightarrow$  find(X,A), find(Y,B),  
link(A,B).

findNode @ X  $\sim$ > PX \ find(X,R)  $\Leftrightarrow$  find(PX,R).

findRoot @ root(X) \ find(X,R)  $\Leftrightarrow$  R=X.

linkEq @ link(X,X)  $\Leftrightarrow$  true.

link @ link(X,Y), root(X), root(Y)  $\Leftrightarrow$  Y  $\sim$ > X,  
root(X).

## 4. Union-Find in CHR

make @ make(X)  $\Leftrightarrow$  root(X).

union @ union(X,Y)  $\Leftrightarrow$  find(X,A), find(Y,B),  
link(A,B).

findNode @ X  $\sim$ > PX \ find(X,R)  $\Leftrightarrow$  find(PX,R).

findRoot @ root(X) \ find(X,R)  $\Leftrightarrow$  R=X.

linkEq @ link(X,X)  $\Leftrightarrow$  true.

link @ link(X,Y), root(X), root(Y)  $\Leftrightarrow$  Y  $\sim$ > X,  
root(X).

## 4. Union-Find in CHR

make @ make(X)  $\Leftrightarrow$  root(X).

union @ union(X,Y)  $\Leftrightarrow$  find(X,A), find(Y,B),  
link(A,B).

findNode @ X  $\sim$ > PX \ find(X,R)  $\Leftrightarrow$  find(PX,R).

findRoot @ root(X) \ find(X,R)  $\Leftrightarrow$  R=X.

linkEq @ link(X,X)  $\Leftrightarrow$  true.

link @ link(X,Y), root(X), root(Y)  $\Leftrightarrow$  Y  $\sim$ > X,  
root(X).

# 4. Union-by-Rank

- `root ( X , Rank )`

- new rules:

```
make      @ make(X) <=> root(X,0) .
```

```
linkEq    @ link(X,X) <=> true.
```

```
linkLeft  @ link(X,Y), root(X,RX) root(Y,RY) <=>
           RX >= RY |
           Y ~> X,
           NRX is max(RX,RY+1), root(X,NRX) .
```

```
linkRight @ link(X,Y), root(Y,RY), root(X,RX) <=>
           RY >= RX |
           X ~> Y,
           NRY is max(RY,RX+1), root(Y,NRY) .
```

# 4. Path Compression

```
findNode @ X ~> PX , find(X,R) <=>
        find(PX,R) , X ~> R .
findRoot @ root(X) \ find(X,R) <=> R=X.
```

# 5. Properties of Implementation

---

- 5.1 Confluence
- 5.2 Logical Meaning
- (5.3 Time Complexity)

# 5.1 Confluence

- what if two rules apply in a state?  
 $a \Rightarrow b$ . or  $a \Rightarrow c$ .
- does rule order matter?

# 5.1 Confluence

- what if two rules apply in a state?  
 $a \Rightarrow b$  or  $a \Rightarrow c$ .
- does rule order matter?
- **No**  $\Rightarrow$  confluent!  
Same result along any derivation

# 5.1 Confluence

- what if two rules apply in a state?  
 $a \Rightarrow b$  or  $a \Rightarrow c$ .
- does rule order matter?
- **No**  $\Rightarrow$  confluent!  
Same result along any derivation
- **Yes**  $\Rightarrow$  not confluent!
  - ◆ different results possible
  - ◆ undesirable
  - ◆ indicates bug?

# 5.1 Confluence

- what if two rules apply in a state?  
 $a \Rightarrow b$  or  $a \Rightarrow c$ .
- does rule order matter?
- **No**  $\Rightarrow$  confluent!  
Same result along any derivation
- **Yes**  $\Rightarrow$  not confluent!
  - ◆ different results possible
  - ◆ undesirable
  - ◆ indicates bug?
- program verification: confluence checking

# 5.1 Union-Find Confluence

Confluence checker found **non-confluence**

# 5.1 Union-Find Confluence

Confluence checker found **non-confluence**  
8 (73) reasons in 3 categories:

# 5.1 Union-Find Confluence

Confluence checker found **non-confluence**  
8 (73) reasons in 3 categories:

- Non-trees: e.g. `root(x), root(x)`  
not relevant for proper use

# 5.1 Union-Find Confluence

Confluence checker found **non-confluence**  
8 (73) reasons in 3 categories:

- Non-trees: e.g. `root(x), root(x)`  
not relevant for proper use
- Pending link: delay treatment of link until  
node is no longer a root  
not relevant in sequential execution

# 5.1 Union-Find Confluence

Confluence checker found **non-confluence**  
8 (73) reasons in 3 categories:

- Non-trees: e.g. `root(x), root(x)`  
not relevant for proper use
- Pending link: delay treatment of link until  
node is no longer a root  
not relevant in sequential execution
- Representative may change after union, i.e.  
order of union and find matters  
inherent **destructive update**

## 5.2 Logical Meaning

- Logical meaning of CHR program = logical theory with axioms derived from program:

## 5.2 Logical Meaning

- Logical meaning of CHR program = logical theory with axioms derived from program:
  - ◆ Replace  $\leq$  with  $\Leftrightarrow$
  - ◆ Replace  $,$  with  $\wedge$

## 5.2 Logical Meaning

- Logical meaning of CHR program = logical theory with axioms derived from program:
  - ◆ Replace  $\leq$  with  $\Leftrightarrow$
  - ◆ Replace  $,$  with  $\wedge$
- Initial and final state are equivalent under logical theory

## 5.2 Logical Meaning

- Logical meaning of CHR program = logical theory with axioms derived from program:
  - ◆ Replace  $\langle = \rangle$  with  $\Leftrightarrow$
  - ◆ Replace  $,$  with  $\wedge$
- Initial and final state are equivalent under logical theory
- Verification: intended theory  $\Rightarrow (\Leftrightarrow)$  program theory

## 5.2. Logical Meaning

$$\textit{make}(A) \Leftrightarrow \textit{root}(A)$$

$$\textit{union}(A, B) \Leftrightarrow \exists XY (\textit{find}(A, X) \wedge \textit{find}(B, Y) \wedge \textit{link}(X, Y))$$

$$\textit{find}(A, X) \wedge A \rightarrow B \Leftrightarrow \textit{find}(B, X) \wedge A \rightarrow B$$

$$\textit{root}(A) \wedge \textit{find}(A, X) \Leftrightarrow \textit{root}(A) \wedge X = A$$

$$\textit{link}(A, A) \Leftrightarrow \textit{true}$$

$$\textit{link}(A, B) \wedge \textit{root}(A) \wedge \textit{root}(B) \Leftrightarrow B \rightarrow A \wedge \textit{root}(A)$$

## 5.2. Logical Meaning

$$\textit{make}(A) \Leftrightarrow \textit{root}(A)$$

$$\textit{union}(A, B) \Leftrightarrow \exists XY (\textit{find}(A, X) \wedge \textit{find}(B, Y) \wedge \textit{link}(X, Y))$$

$$\textit{find}(A, X) \wedge A \rightarrow B \Leftrightarrow \textit{find}(B, X) \wedge A \rightarrow B$$

$$\textit{root}(A) \wedge \textit{find}(A, X) \Leftrightarrow \textit{root}(A) \wedge X = A$$

$$\textit{link}(A, A) \Leftrightarrow \textit{true}$$

$$\textit{link}(A, B) \wedge \textit{root}(A) \wedge \textit{root}(B) \Leftrightarrow B \rightarrow A \wedge \textit{root}(A)$$

## 5.2 Logical meaning

Does this model equivalence (partition)?

## 5.2 Logical meaning

Does this model equivalence (partition)?  
Replace constraints with =  
except make and root: *true*

## 5.2 Logical meaning

Does this model equivalence (partition)?

Replace constraints with =

except make and root: *true*

union  $A=B \Leftrightarrow \exists XY (A=X \wedge B=Y \wedge X=Y)$

findNode  $A=X \wedge A=B \Leftrightarrow B=X \wedge A=B$

findRoot  $A=X \Leftrightarrow X=A$

linkEq  $A=A \Leftrightarrow \text{true}$

link  $A=B \Leftrightarrow B=A$

## 5.3 Optimal Complexity?

Yes! near-linear

## 5.3 Optimal Complexity?

**Yes!** near-linear  
based on

- operational equivalence proof
- efficient constraint stores

## 5.3 Optimal Complexity?

**Yes!** near-linear  
based on

- operational equivalence proof
- efficient constraint stores

See:

Tom Schrijvers and Thom Frühwirth  
*Optimal Union-Find in Constraint Handling Rules*

To appear in TPLP

# 6. Conclusion & Future Work

---

Implementation in CHR

# 6. Conclusion & Future Work

Implementation in CHR

- confluent, except for update

# 6. Conclusion & Future Work

Implementation in CHR

- confluent, except for update
- sound & complete wrt logical equivalence

# 6. Conclusion & Future Work

## Implementation in CHR

- confluent, except for update
- sound & complete wrt logical equivalence
- (optimal complexity)

# 6. Conclusion & Future Work

## Implementation in CHR

- confluent, except for update
- sound & complete wrt logical equivalence
- (optimal complexity)

## Future work:

- variants of the algorithm: heuristics, undo of union

# 6. Conclusion & Future Work

## Implementation in CHR

- confluent, except for update
- sound & complete wrt logical equivalence
- (optimal complexity)

## Future work:

- variants of the algorithm: heuristics, undo of union
- parallel execution?

# 6. Conclusion & Future Work

## Implementation in CHR

- confluent, except for update
- sound & complete wrt logical equivalence
- (optimal complexity)

## Future work:

- variants of the algorithm: heuristics, undo of union
- parallel execution?
- more practical analyses