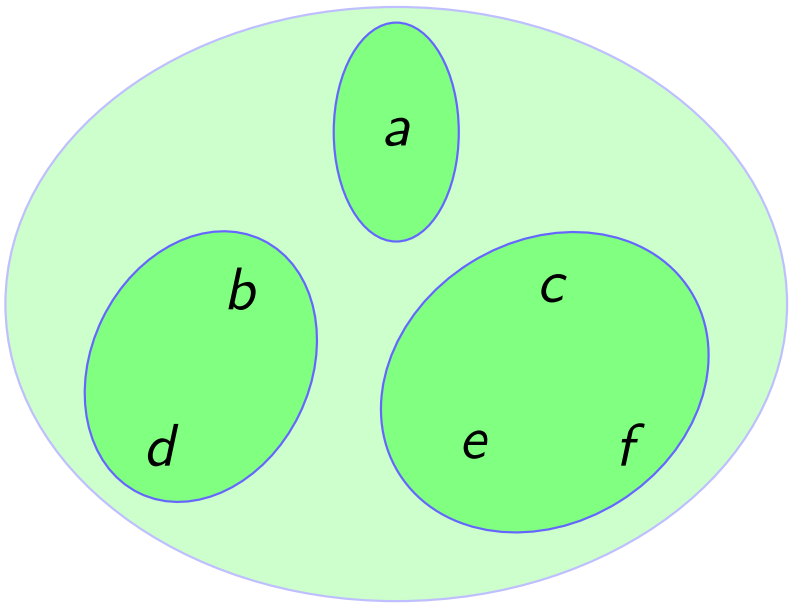
# Verifying the Correctness of Disjoint-Set Forests with Kleene Relation Algebras

Walter Guttmann
University of Canterbury

1. Disjoint Set Forests
2. Implementation
3. Associative Arrays
4. Verification

# Disjoint Sets



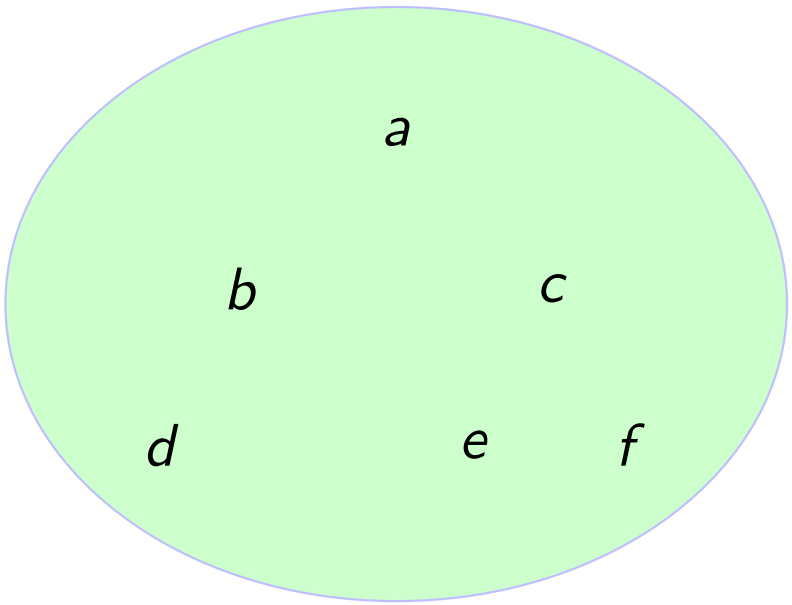

## set of sets

$$\{\{a\},\{b,d\},\{c,e,f\}\}$$

## equivalence relation

$$\begin{array}{c} \\ a \\ b \\ d \\ c \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & d & c & e & f \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

# Disjoint Set Operations




$\longrightarrow$

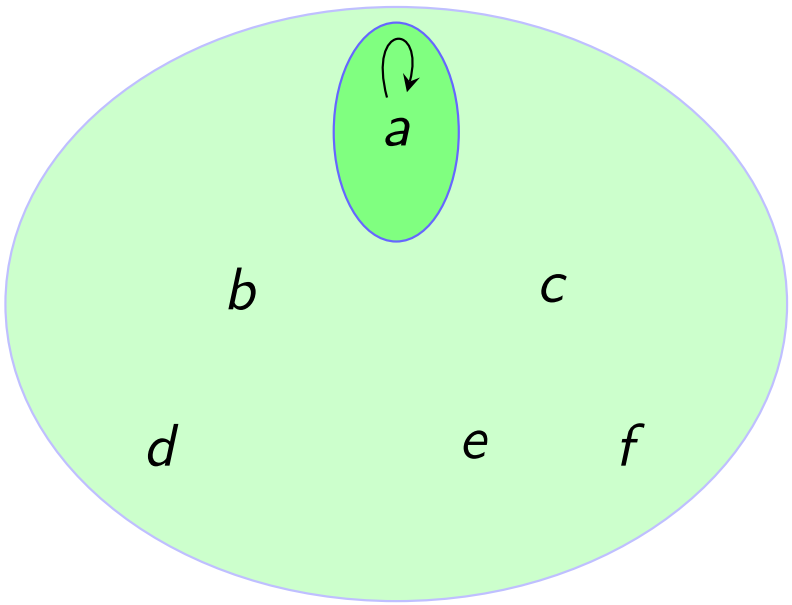
make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations



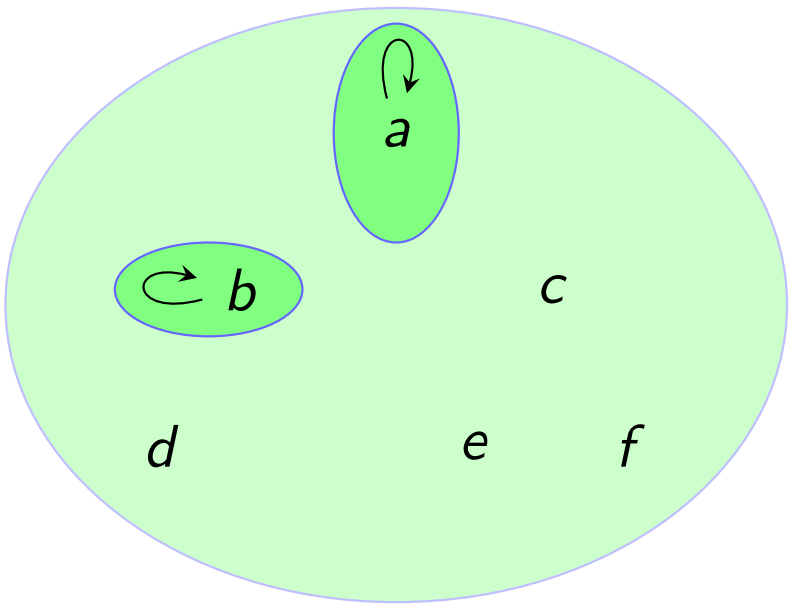

→ make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations



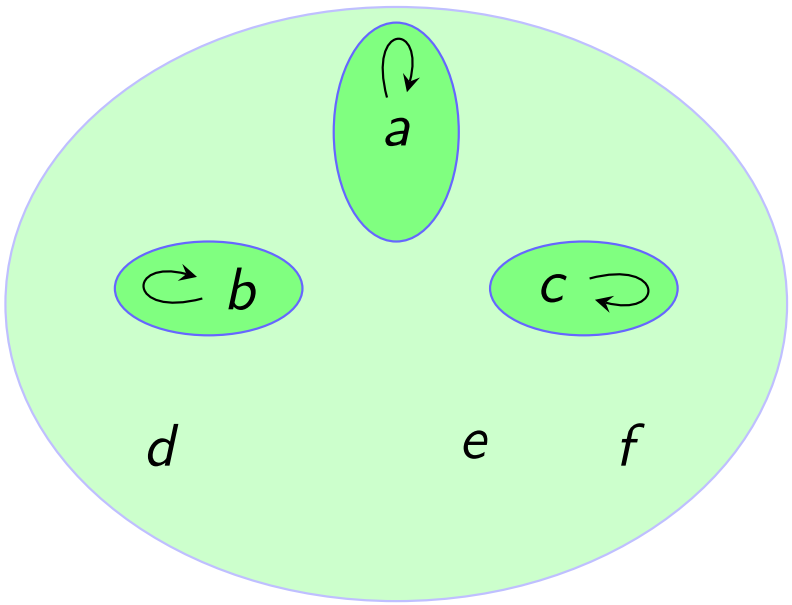

make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations



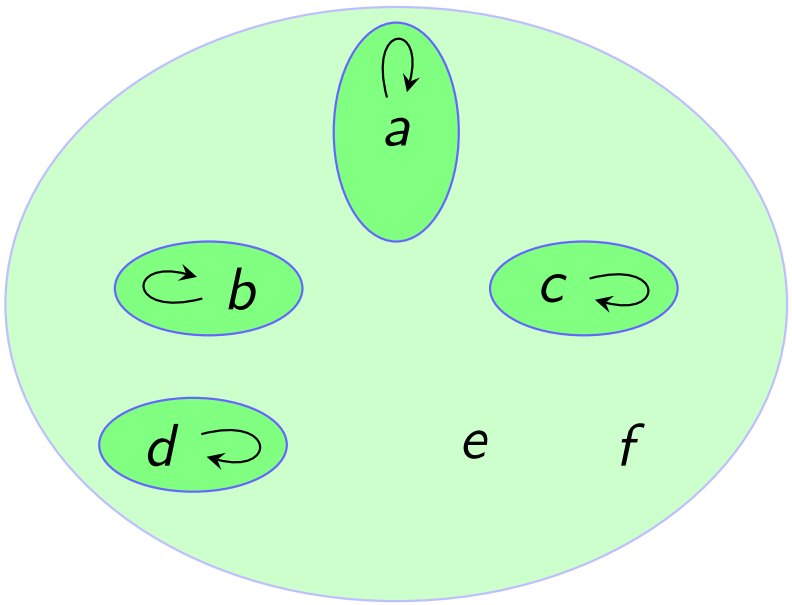

make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations




→

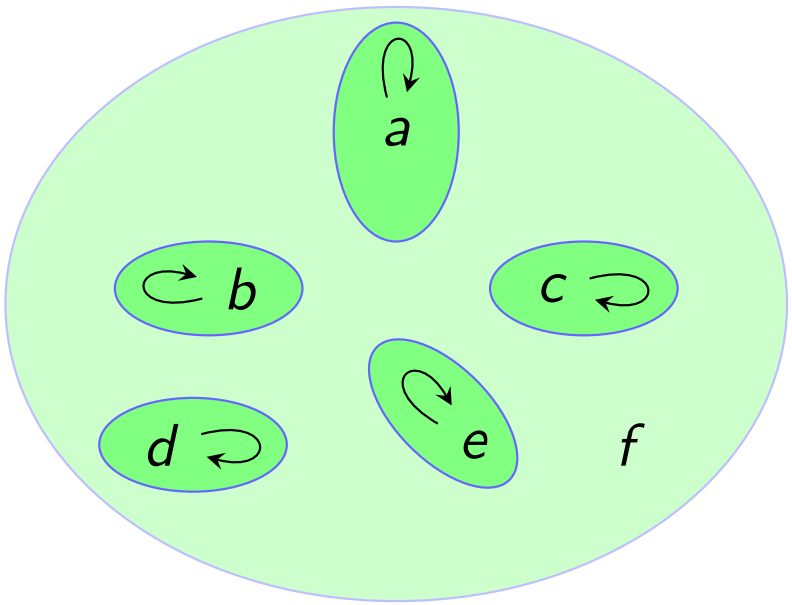
make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations



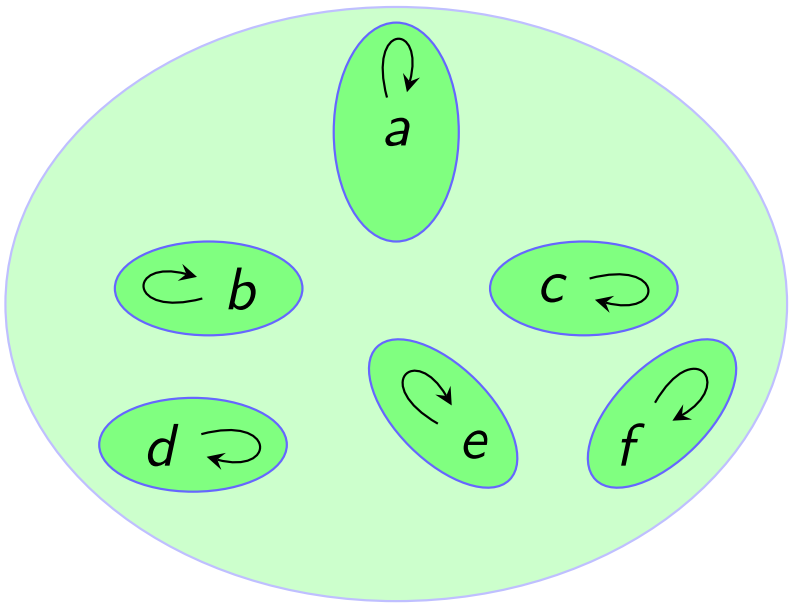

make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations



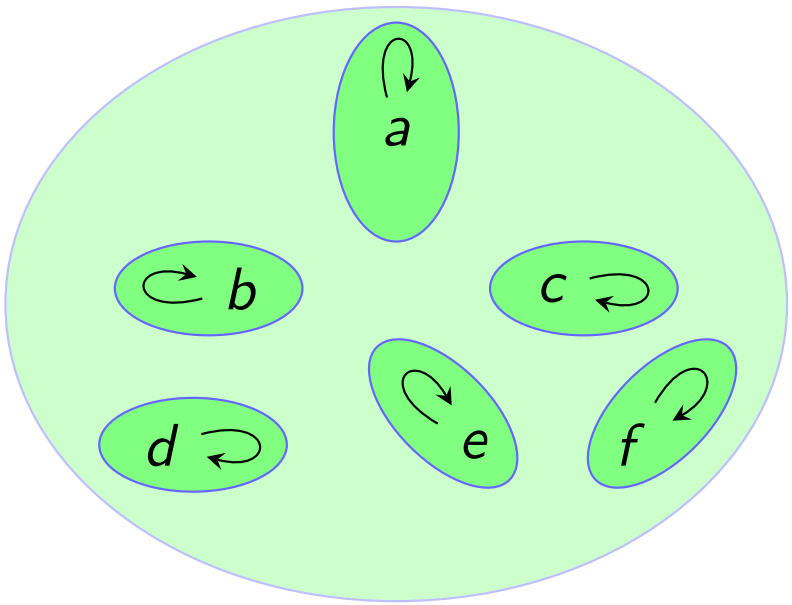

make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

→ find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations




make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

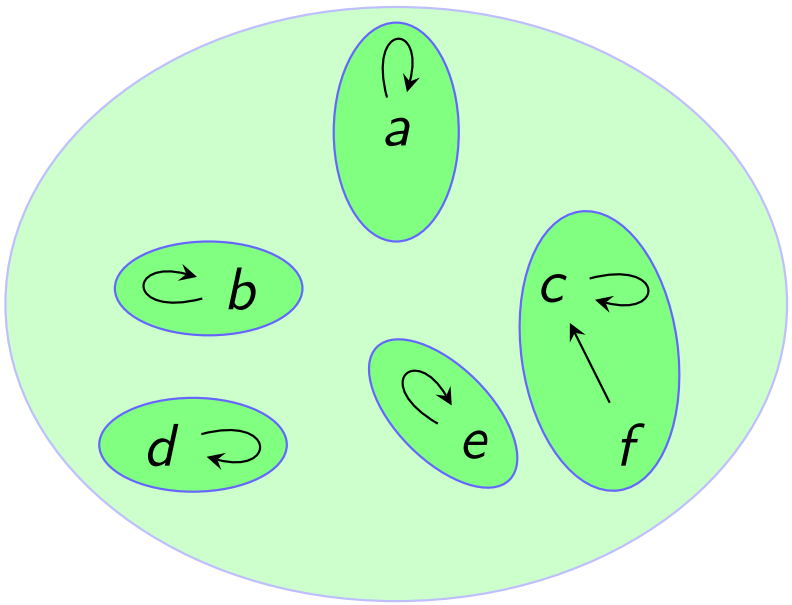
find-set(f) = f

$\longrightarrow$

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations




make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

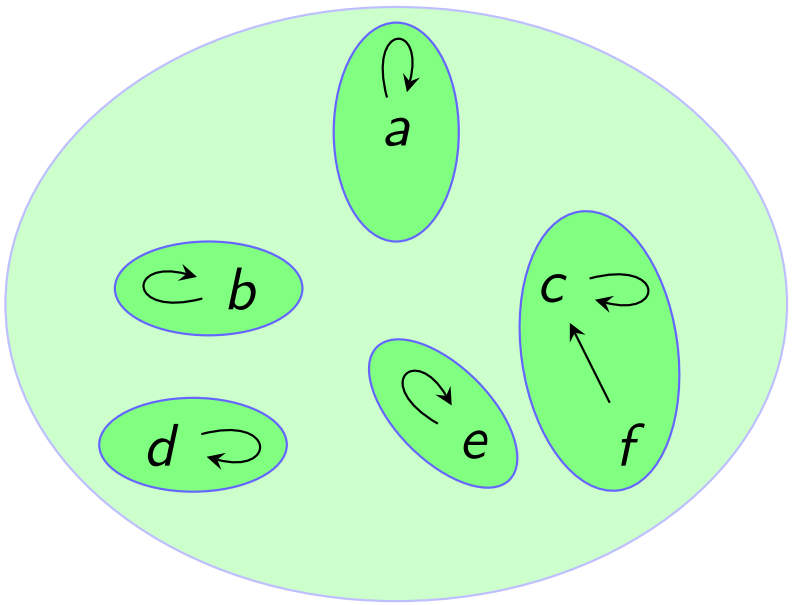
find-set(f) = f

union-sets(f, c)

→ find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations




make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

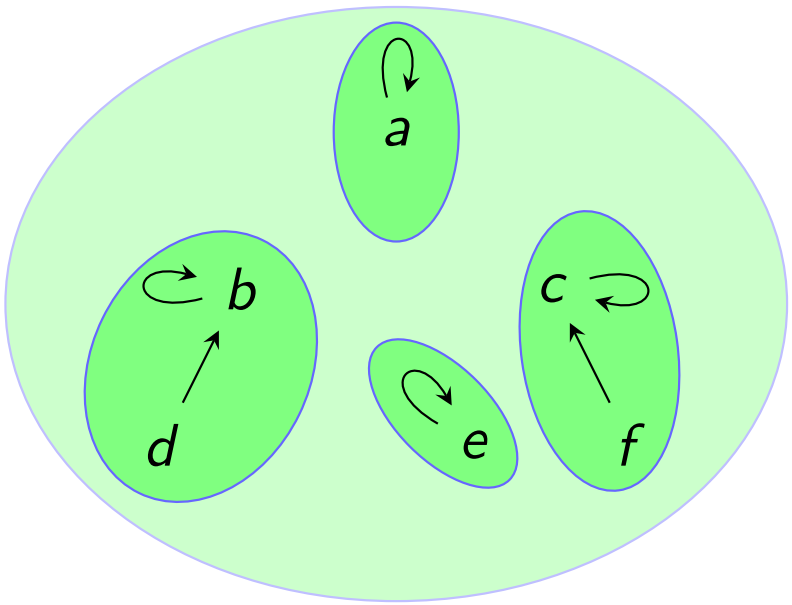
find-set(f) = f

union-sets(f, c)
find-set(f) = c

$\longrightarrow$

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations



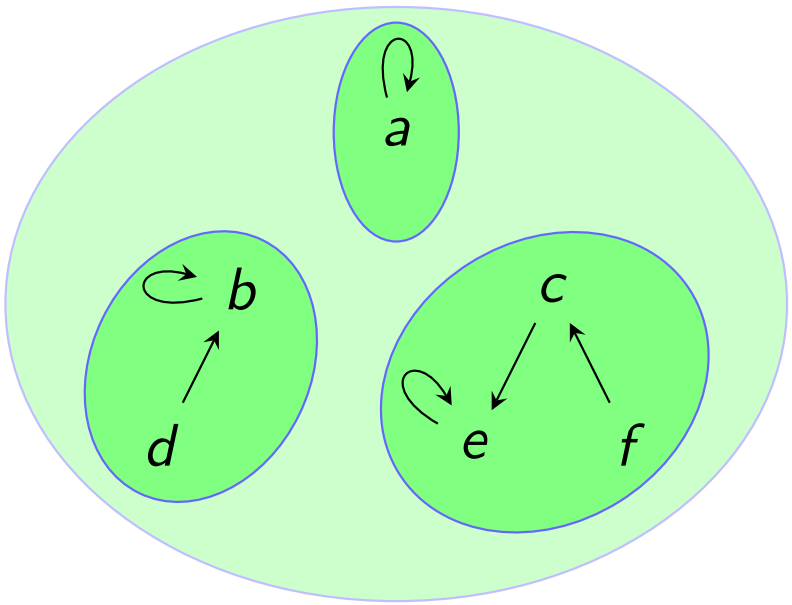

make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Operations



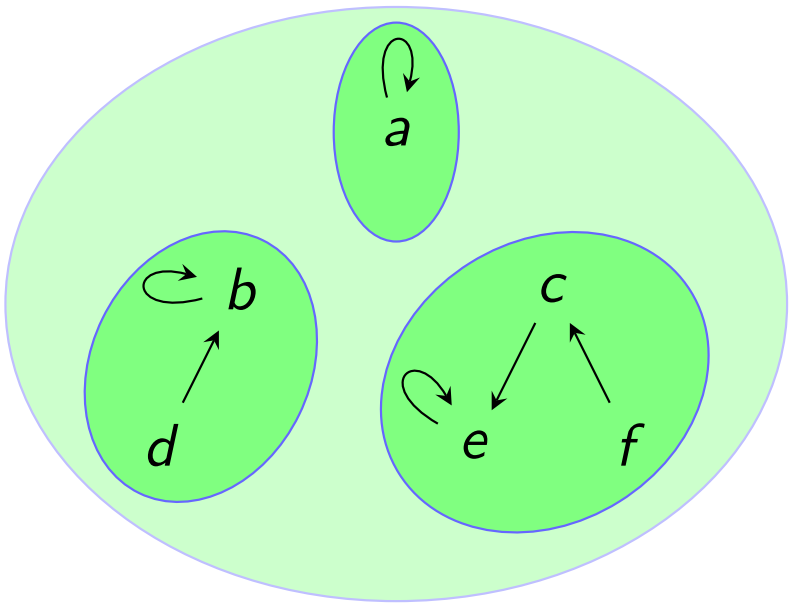

make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
→ find-set(f) = e

# Disjoint Set Operations



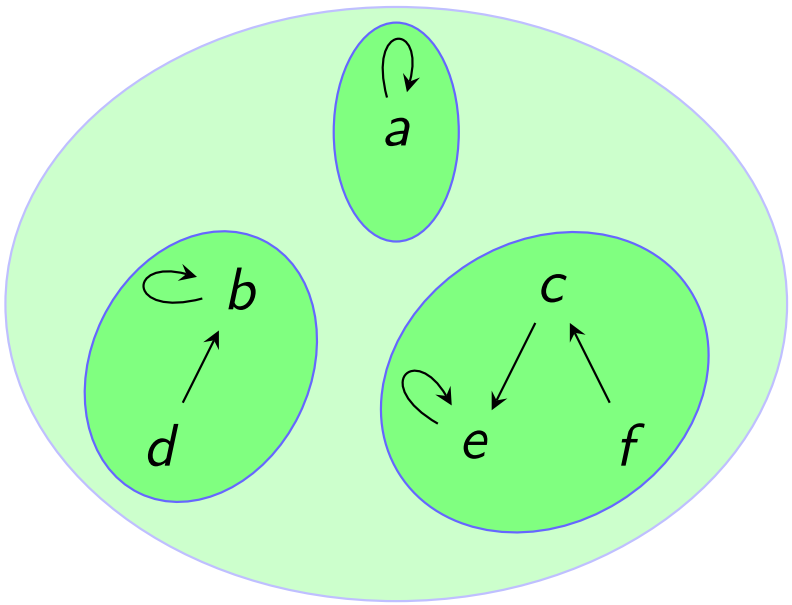

make-set(a)
make-set(b)
make-set(c)
make-set(d)
make-set(e)
make-set(f)

find-set(f) = f

union-sets(f, c)
find-set(f) = c

union-sets(d, b)
union-sets(f, e)
find-set(f) = e

# Disjoint Set Forests




tree = equivalence class

root = representative

## parent array

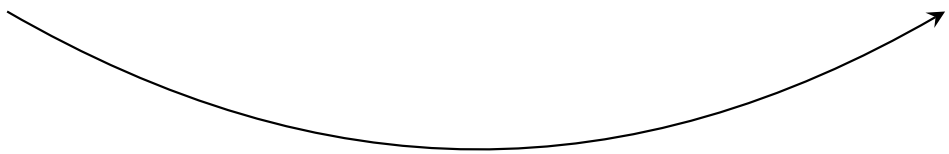
$$\begin{bmatrix} a & b & c & d & e & f \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ a & b & e & b & e & c \end{bmatrix}$$

## parent relation

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

# Forest Semantics

| parent relation | | equivalence relation | |
|---|---|---|---|
| univalent | $R^{\mathsf{T}} R \subseteq \mathsf{I}$ | reflexive | $\mathsf{I} \subseteq R$ |
| total | $\mathsf{I} \subseteq R R^{\mathsf{T}}$ | transitive | $RR \subseteq R$ |
| acyclic | $(R \cap \bar{\mathsf{I}})^+ \subseteq \bar{\mathsf{I}}$ | symmetric | $R = R^{\mathsf{T}}$ |



$$\mathrm{fc}(R) = R^* R^{\mathsf{T}*}$$

$\mathrm{fc}(R)$ is an equivalence relation for univalent $R$

fc is a closure operation on univalent relations

# Verifying the Correctness of Disjoint-Set Forests with Kleene Relation Algebras

Walter Guttmann
University of Canterbury

1. Disjoint Set Forests
2. Implementation
3. Associative Arrays
4. Verification

# Implementation

make-set($p$, $x$):

$p[x] := x$

return $p$

find-set($p$, $x$):

$y := x$

while $y \neq p[y]$ do

$y := p[y]$

return $y$

union-sets($p$, $x$, $y$):

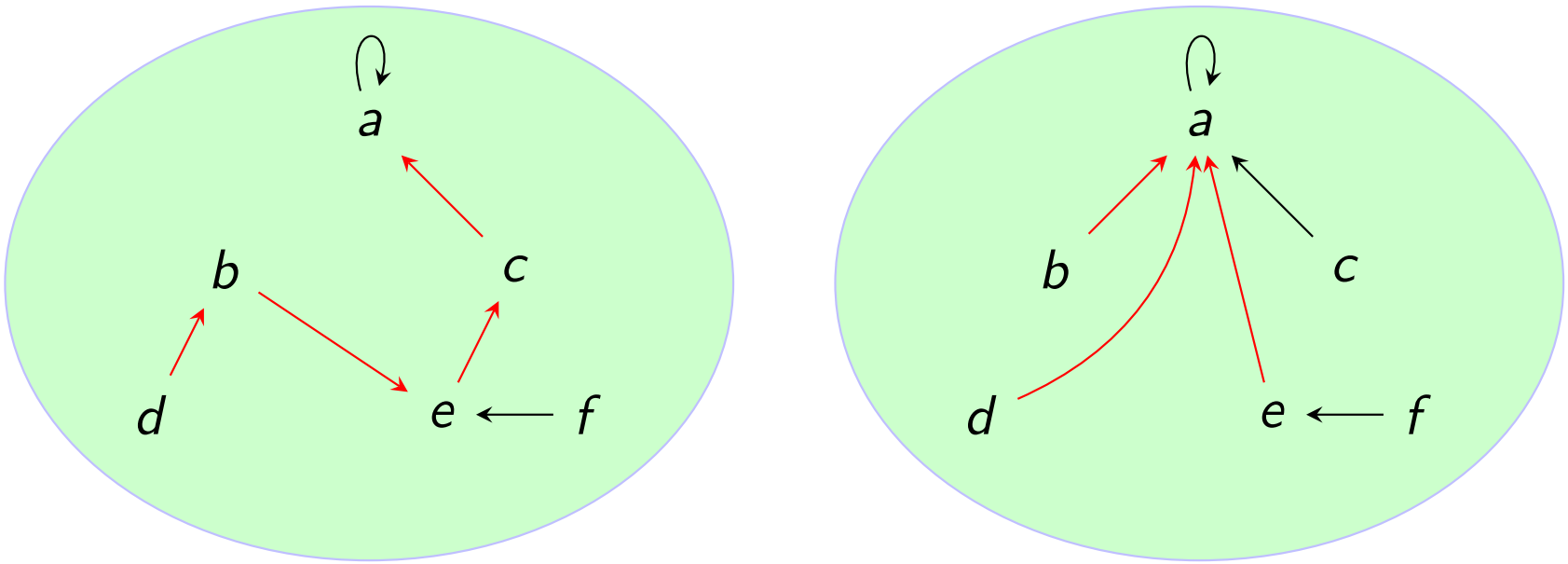
$r :=$ find-set($p$, $x$)

$s :=$ find-set($p$, $y$)

$p[r] := s$

return $p$

# Path Compression




find-set(d) = a

# Path Compression Implemented

path-compression($p$, $x$, $y$):

$w := x$
while $y \neq p[w]$ do
  $t := w$
  $w := p[w]$
  $p[t] := y$
return $p$

union-sets($p$, $x$, $y$):

$r :=$ find-set($p$, $x$)
$p :=$ path-compression($p$, $x$, $r$)
$s :=$ find-set($p$, $y$)
$p :=$ path-compression($p$, $y$, $s$)
$p[r] := s$
return $p$

# Verifying the Correctness of Disjoint-Set Forests with Kleene Relation Algebras

Walter Guttmann
University of Canterbury

1. Disjoint Set Forests
2. Implementation
3. Associative Arrays
4. Verification

# Associative Array Update

$$\text{array} \quad R = \begin{bmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ \downarrow & \downarrow & \downarrow \\ \mathsf{c} & \mathsf{b} & \mathsf{b} \end{bmatrix} \qquad \text{mapping} \quad \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$\text{index} \quad P = b \qquad \text{point} \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\text{value} \quad Q = a \qquad \text{point} \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\text{update} \quad R[P] := Q$$

$$\text{updated array} \quad R = \begin{bmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ \downarrow & \downarrow & \downarrow \\ \mathsf{c} & \mathsf{a} & \mathsf{b} \end{bmatrix} \qquad \text{updated mapping} \quad \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

# Associative Array Semantics

## update

$$R[P] := Q \quad = \quad R := R[P \mapsto Q]$$

$$R[P \mapsto Q] \quad = \quad (P \cap Q^{\mathsf{T}}) \cup (\overline{P} \cap R)$$

## read

$$R[P] \quad = \quad R^{\mathsf{T}} P$$

# Associative Array Read

array $R = \begin{bmatrix} \mathsf{a} & \mathsf{b} & \mathsf{c} \\ \downarrow & \downarrow & \downarrow \\ \mathsf{c} & \mathsf{b} & \mathsf{b} \end{bmatrix}$

mapping $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$

index $P = a$

point $\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

read $R[P]$

value $R^{\mathsf{T}} P = c$

resulting point $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$

# Associative Array Properties

$$R[P \mapsto Q] \text{ is } \left\{\begin{array}{c}\text{univalent}\\ \text{total}\\ \text{a mapping}\end{array}\right\} \text{ if } R \text{ is } \left\{\begin{array}{c}\text{univalent}\\ \text{total}\\ \text{a mapping}\end{array}\right\}, \; Q \text{ is } \left\{\begin{array}{c}\text{injective}\\ \text{surjective}\\ \text{bijective}\end{array}\right\}$$

and $P$ is a vector.

# Associative Array Properties

$$R[P \mapsto Q] \text{ is } \left\{\begin{array}{c}\text{univalent}\\ \text{total}\\ \text{a mapping}\end{array}\right\} \text{ if } R \text{ is } \left\{\begin{array}{c}\text{univalent}\\ \text{total}\\ \text{a mapping}\end{array}\right\}, \; Q \text{ is } \left\{\begin{array}{c}\text{injective}\\ \text{surjective}\\ \text{bijective}\end{array}\right\}$$

$$\text{and } P \text{ is a vector.}$$

$$R[P] \text{ is } \left\{\begin{array}{c}\text{injective}\\ \text{surjective}\\ \text{bijective}\\ \text{a point}\end{array}\right\} \text{ if } R \text{ is } \left\{\begin{array}{c}\text{univalent}\\ \text{total}\\ \text{a mapping}\\ \text{a mapping}\end{array}\right\} \text{ and } P \text{ is } \left\{\begin{array}{c}\text{injective}\\ \text{surjective}\\ \text{bijective}\\ \text{a point}\end{array}\right\}.$$

# Associative Array Properties

$$R[P \mapsto Q] \text{ is } \left\{\begin{array}{c}\text{univalent}\\ \text{total}\\ \text{a mapping}\end{array}\right\} \text{ if } R \text{ is } \left\{\begin{array}{c}\text{univalent}\\ \text{total}\\ \text{a mapping}\end{array}\right\}, Q \text{ is } \left\{\begin{array}{c}\text{injective}\\ \text{surjective}\\ \text{bijective}\end{array}\right\}$$

$$\text{and } P \text{ is a vector.}$$

$$R[P] \text{ is } \left\{\begin{array}{c}\text{injective}\\ \text{surjective}\\ \text{bijective}\\ \text{a point}\end{array}\right\} \text{ if } R \text{ is } \left\{\begin{array}{c}\text{univalent}\\ \text{total}\\ \text{a mapping}\\ \text{a mapping}\end{array}\right\} \text{ and } P \text{ is } \left\{\begin{array}{c}\text{injective}\\ \text{surjective}\\ \text{bijective}\\ \text{a point}\end{array}\right\}.$$

$$R[P] = Q \Leftrightarrow P \cap R = P \cap Q^{\mathsf{T}} \text{ if } P \text{ and } Q \text{ are points.}$$

# Verifying the Correctness of Disjoint-Set Forests with Kleene Relation Algebras

Walter Guttmann
University of Canterbury

1. Disjoint Set Forests
2. Implementation
3. Associative Arrays
4. Verification

# Correctness of Find-Set

find-set($p$, $x$):
  $\{p$ forest $\wedge\ x$ point$\}$
  $y := x$
  while $y \neq p[y]$ do
    $\{p$ forest $\wedge\ x, y$ points $\wedge\ y \subseteq p^{\mathsf{T}*} x\}$
    $y := p[y]$
  return $y$
  $\{y$ point $\wedge\ y = \mathsf{root}(p, x)\}$

$\mathsf{root}(p, x) = (p \cap \mathsf{I}) p^{\mathsf{T}*} x$

$\mathsf{root}(p, x) x^{\mathsf{T}} \subseteq \mathsf{fc}(p)$ if $x$ injective

# Correctness of Find-Set

find-set($p$, $x$):
  $\{p$ forest $\wedge\ x$ point$\}$
  $y := x$
  while $y \neq p[y]$ do
    $\{p$ forest $\wedge\ x, y$ points $\wedge\ y \subseteq p^{\mathsf{T}*}x\}$ variant $|\{z \mid z \subseteq p^{\mathsf{T}*}y\}|$
    $y := p[y]$
  return $y$
  $\{y$ point $\wedge\ y = \mathsf{root}(p, x)\}$

$\mathsf{root}(p, x) = (p \cap \mathsf{I})p^{\mathsf{T}*}x$

$\mathsf{root}(p, x)x^{\mathsf{T}} \subseteq \mathsf{fc}(p)$ if $x$ injective

# Correctness of Find-Set

find-set($p$, $x$):
  $\{p$ forest $\wedge\ x$ point$\}$
  $y := x$
  while $y \neq p[y]$ do
    $\{p$ forest $\wedge\ x, y$ points $\wedge\ y \subseteq p^{\mathsf{T}*}x\}$ variant $|\{z \mid z \subseteq p^{\mathsf{T}*}y\}|$
    $y := p[y]$
  return $y$
  $\{y$ point $\wedge\ y = \mathrm{root}(p, x)\}$

$\mathrm{root}(p, x) = (p \cap \mathsf{I})p^{\mathsf{T}*}x$

$\mathrm{root}(p, x)x^{\mathsf{T}} \subseteq \mathrm{fc}(p)$ if $x$ injective

$\mathrm{root}(p, x)$ point if $p$ forest, $x$ point

- hidden syntax tree
- operational semantics
- Hoare triples
- total correctness
- determinism
- extract function
- constructive proof

# Correctness of Path-Compression

path-compression($p$, $x$, $y$):

$\{p \text{ forest} \wedge x, y \text{ points} \wedge y = \text{root}(p, x) \wedge p_0 = p\}$

$w := x$

while $y \neq p[w]$ do

$$\left\{\begin{array}{l} \text{postcondition} \wedge w \text{ point} \wedge y \subseteq p^{\mathsf{T}*} w \wedge \\ (w \neq x \Rightarrow (y \neq x \wedge p[x] = y \wedge p^{\mathsf{T}+} w \subseteq \overline{x})) \end{array}\right\}$$

$t := w$

$w := p[w]$

$p[t] := y$

return $p$

$$\left\{\begin{array}{l} p \text{ forest} \wedge x, y \text{ points} \wedge y = \text{root}(p, x) \wedge \\ \text{fc}(p) = \text{fc}(p_0) \wedge p \cap \mathsf{I} = p_0 \cap \mathsf{I} \end{array}\right\}$$

# Correctness of Union-Sets

union-sets($p$, $x$, $y$):

$\{p \text{ forest} \wedge x, y \text{ points} \wedge p_0 = p\}$

$r :=$ find-set($p$, $x$)

$p :=$ path-compression($p$, $x$, $r$)

$s :=$ find-set($p$, $y$)

$p :=$ path-compression($p$, $y$, $s$)

$p[r] := s$

return $p$

$\{p \text{ forest} \wedge x, y \text{ points} \wedge \mathrm{fc}(p) = \mathrm{wcc}(p_0 \cup xy^\mathsf{T})\}$

$\mathrm{wcc}(x) = (x \cup x^\mathsf{T})^*$ is an equivalence relation

$\mathrm{wcc}(x) = \mathrm{fc}(x)$ if $x$ univalent

wcc is a closure operation

# Conclusion

- all results proved in Isabelle/HOL
- in Stone-Kleene relation algebras for weighted graphs
- integrate with Kruskal's algorithm
- complexity reasoning