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

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 1
\end{pmatrix}
\]
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


d - e - f

\[\text{make-set}(a)\]
\[\text{make-set}(b)\]
\[\text{make-set}(c)\]
\[\text{make-set}(d)\]
\[\text{make-set}(e)\]
\[\text{make-set}(f)\]
\[\text{find-set}(f) = f\]
\[\text{union-sets}(f, c)\]
\[\text{find-set}(f) = c\]
\[\text{union-sets}(d, b)\]
\[\text{union-sets}(f, e)\]
\[\text{find-set}(f) = e\]
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`
- `find-set(f) = c`
- `find-set(f) = e`

- `union-sets(f, c)`
- `union-sets(d, b)`
- `union-sets(f, e)`
Disjoint Set Operations

- $\text{make-set}(a)$
- $\text{make-set}(b)$
- $\text{make-set}(c)$
- $\text{make-set}(d)$
- $\text{make-set}(e)$
- $\text{make-set}(f)$

- $\text{find-set}(f) = f$
- $\text{union-sets}(f, c)$
- $\text{find-set}(f) = c$
- $\text{union-sets}(d, b)$
- $\text{union-sets}(f, e)$
- $\text{find-set}(f) = e$
Disjoint Set Operations

- $\text{make-set}(a)$
- $\text{make-set}(b)$
- $\text{make-set}(c)$
- $\text{make-set}(d)$
- $\text{make-set}(e)$
- $\text{make-set}(f)$

- $\text{find-set}(f) = f$
- $\text{union-sets}(f, c)$
  - $\text{find-set}(f) = c$
- $\text{union-sets}(d, b)$
- $\text{union-sets}(f, e)$
  - $\text{find-set}(f) = e$
Disjoint Set Operations

\[
\begin{align*}
\text{make-set}(a) \\
\text{make-set}(b) \\
\text{make-set}(c) \\
\text{make-set}(d) \\
\text{make-set}(e) \\
\text{make-set}(f) \\
\text{find-set}(f) &= f \\
\text{union-sets}(f, c) \\
\text{find-set}(f) &= c \\
\text{union-sets}(d, b) \\
\text{union-sets}(f, e) \\
\text{find-set}(f) &= e
\end{align*}
\]
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

- \(\text{make-set}(a)\)
- \(\text{make-set}(b)\)
- \(\text{make-set}(c)\)
- \(\text{make-set}(d)\)
- \(\text{make-set}(e)\)
- \(\text{make-set}(f)\)

- \(\text{find-set}(f) = f\)
- \(\text{union-sets}(f, c)\)
- \(\text{find-set}(f) = c\)
- \(\text{union-sets}(d, b)\)
- \(\text{union-sets}(f, e)\)
- \(\text{find-set}(f) = e\)
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)

\[ \text{find-set(f)} = f \]

\[ \text{union-sets}(f, c) \]
\[ \text{find-set(f)} = c \]

\[ \text{union-sets}(d, b) \]
\[ \text{union-sets}(f, e) \]
\[ \text{find-set(f)} = e \]
Disjoint Set Forests

![Diagram of Disjoint Set Forests](image)

- **tree** = equivalence class
- **root** = representative

**parent array**

\[
\begin{bmatrix}
a & b & c & d & e & f \\
ad & ad & ad & ad & ad & ad & ad \\
a & b & e & b & e & c
\end{bmatrix}
\]

**parent relation**

\[
\begin{bmatrix}
a & b & c & d & e & f \\
a & 1 & 0 & 0 & 0 & 0 & 0 \\
b & 0 & 1 & 0 & 0 & 0 & 0 \\
c & 0 & 0 & 0 & 0 & 1 & 0 \\
d & 0 & 1 & 0 & 0 & 0 & 0 \\
e & 0 & 0 & 0 & 0 & 1 & 0 \\
f & 0 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
\]
Forest Semantics

**parent relation**

- univalent \( R^T R \subseteq I \)
- total \( I \subseteq RR^T \)
- acyclic \( (R \cap \bar{I})^+ \subseteq \bar{I} \)

**equivalence relation**

- reflexive \( I \subseteq R \)
- transitive \( RR \subseteq R \)
- symmetric \( R = R^T \)

\[ fc(R) = R^* R^{T*} \]

\( fc(R) \) 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 ≠ 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

\[
\text{find-set}(d) = a
\]
Path Compression Implemented

\textbf{path-compression}(p, x, y):
\begin{align*}
w & := x \\
\text{while } y \neq p[w] \text{ do} & \\
& \quad t := w \\
& \quad w := p[w] \\
& \quad p[t] := y \\
\text{return } p
\end{align*}

\textbf{union-sets}(p, x, y):
\begin{align*}
r & := \text{find-set}(p, x) \\
p & := \text{path-compression}(p, x, r) \\
s & := \text{find-set}(p, y) \\
p & := \text{path-compression}(p, y, s) \\
p[r] & := s \\
\text{return } p
\end{align*}
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

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

index \quad P = b

value \quad Q = a

update \quad R[P] := Q

updated array \quad R = \begin{bmatrix} a & b & c \\ \downarrow & \downarrow & \downarrow \\ c & a & b \end{bmatrix}

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

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

point \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}
Associative Array Semantics

update

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

read

\[ R[P] \quad = \quad R^T P \]
Associative Array Read

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

index \quad P = a

read \quad R[P]

value \quad R^\top P = c

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

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

resulting point \quad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}
Associative Array Properties

\[ R[P \mapsto Q] \text{ is } \begin{cases} \text{univalent} \\ \text{total} \\ \text{a mapping} \end{cases} \text{ if } R \text{ is } \begin{cases} \text{univalent} \\ \text{total} \\ \text{a mapping} \end{cases}, \text{ and } P \text{ is a vector.} \]

\[ Q \text{ is } \begin{cases} \text{injective} \\ \text{surjective} \\ \text{bijective} \end{cases} \]
Associative Array Properties

\[ R[P \mapsto Q] \text{ is } \begin{cases} \text{univalent} \\ \text{total} \\ \text{a mapping} \end{cases} \text{ if } R \text{ is } \begin{cases} \text{univalent} \\ \text{total} \\ \text{a mapping} \end{cases}, \ Q \text{ is } \begin{cases} \text{injective} \\ \text{surjective} \\ \text{bijective} \end{cases} \text{ and } P \text{ is a vector.} \]

\[ R[P] \text{ is } \begin{cases} \text{injective} \\ \text{surjective} \\ \text{bijective} \\ \text{a point} \end{cases} \text{ if } R \text{ is } \begin{cases} \text{univalent} \\ \text{total} \\ \text{a mapping} \end{cases} \text{ and } P \text{ is } \begin{cases} \text{injective} \\ \text{surjective} \\ \text{bijective} \\ \text{a point} \end{cases}. \]
Associative Array Properties

\[ R[P \mapsto Q] \] is
\[
\begin{cases}
\text{univalent} \\
\text{total} \\
\text{a mapping}
\end{cases}
\]
if \( R \) is
\[
\begin{cases}
\text{univalent} \\
\text{total} \\
\text{a mapping}
\end{cases}
\]
and \( Q \) is
\[
\begin{cases}
\text{injective} \\
\text{surjective} \\
\text{bijective}
\end{cases}
\]
and \( P \) is a vector.

\[ R[P] \] is
\[
\begin{cases}
\text{injective} \\
\text{surjective} \\
\text{bijective} \\
\text{a point}
\end{cases}
\]
if \( R \) is
\[
\begin{cases}
\text{univalent} \\
\text{total} \\
\text{a mapping}
\end{cases}
\]
and \( P \) is
\[
\begin{cases}
\text{injective} \\
\text{surjective} \\
\text{bijective} \\
\text{a point}
\end{cases}
\].

\[ R[P] = Q \iff P \cap R = P \cap Q^T \] if \( P \) and \( Q \) 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

\[
\text{find-set}(p, x):
\]
\[
\{ p \text{ forest } \land x \text{ point} \}
\]
\[
y := x
\]
\[
\text{while } y \neq p[y] \text{ do}
\]
\[
\{ p \text{ forest } \land x, y \text{ points } \land y \subseteq p^{T^\star}x \}
\]
\[
y := p[y]
\]
\[
\text{return } y
\]
\[
\{ y \text{ point } \land y = \text{root}(p, x) \}\]

\[
\text{root}(p, x) = (p \cap I)p^{T^\star}x
\]
\[
\text{root}(p, x)x^T \subseteq \text{fc}(p) \text{ if } x \text{ injective}
\]
Correctness of Find-Set

\[
\text{find-set}(p, x):
\begin{align*}
\{ & p \text{ forest } \land x \text{ point} \} \\
& y := x \\
& \text{while } y \neq p[y] \text{ do} \\
& \quad \{ p \text{ forest } \land x, y \text{ points } \land y \subseteq p^{T*}x \} \text{ variant } |\{ z \mid z \subseteq p^{T*y}\}| \\
& \quad y := p[y] \\
& \text{return } y \\
& \{ y \text{ point } \land y = \text{root}(p, x) \}
\end{align*}
\]

\[
\text{root}(p, x) = (p \cap I)p^{T*x}
\]

\[
\text{root}(p, x)x^T \subseteq \text{fc}(p) \text{ if } x \text{ injective}
\]
Correctness of Find-Set

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

\[
\text{root}(p, x) = (p \cap 1)p^{T^*}x \\
\text{root}(p, x)x^T \subseteq \text{fc}(p) \text{ if } x \text{ injective} \\
\text{root}(p, x) \text{ point if } p \text{ forest, } x \text{ point} \\
\]

- hidden syntax tree
- operational semantics
- Hoare triples
- total correctness
- determinism
- extract function
- constructive proof
Correctness of Path-Compression

\[
\text{path-compression}(p, x, y): \quad \{ p \text{ forest } \land x, y \text{ points } \land y = \text{root}(p, x) \land p_0 = p \} \\
w := x \\
\text{while } y \neq p[w] \text{ do} \\
\{ \text{postcondition } \land w \text{ point } \land y \subseteq p^{T^w} \land \\
(w \neq x \Rightarrow (y \neq x \land p[x] = y \land p^{T^+}w \subseteq x)) \} \\
t := w \\
w := p[w] \\
p[t] := y \\
\text{return } p \\
\{ p \text{ forest } \land x, y \text{ points } \land y = \text{root}(p, x) \land \\
f_c(p) = f_c(p_0) \land p \cap I = p_0 \cap I \} \\
\]
Correctness of Union-Sets

union-sets\((p, x, y)\):
\[
\begin{align*}
\{ p \text{ forest } \land x, y \text{ points } \land p_0 = p \} \\
r & := \text{find-set}(p, x) \\
p & := \text{path-compression}(p, x, r) \\
s & := \text{find-set}(p, y) \\
p & := \text{path-compression}(p, y, s) \\
p[r] & := s \\
\text{return } p \\
\{ p \text{ forest } \land x, y \text{ points } \land \text{fc}(p) = \text{wcc}(p_0 \cup xy^T) \}
\end{align*}
\]

\(\text{wcc}(x) = (x \cup x^T)^*\) is an equivalence relation
\(\text{wcc}(x) = \text{fc}(x)\) if \(x\) univalent
\(\text{wcc}\) 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