NICTA

From **imagination** to **impact**

Australian Government

Department of Broadband, Communications and the Digital Economy

Australian Research Council

NICTA Funding and Supporting Members and Partners

Australian National University

UNSW THE UNIVERSITY OF NEW SOUTH WALES

NSW GOVERNMENT Trade & Investment

State Government Victoria

THE UNIVERSITY OF MELBOURNE

THE UNIVERSITY OF SYDNEY

Queensland Government

Griffith UNIVERSITY

QUT Queensland University of Technology

THE UNIVERSITY OF QUEENSLAND AUSTRALIA

Algebras for (automatic) Verification of Graph Algorithms

Peter Höfner

NICTA
UNSW THE UNIVERSITY OF NEW SOUTH WALES SYDNEY · CANBERRA · AUSTRALIA

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

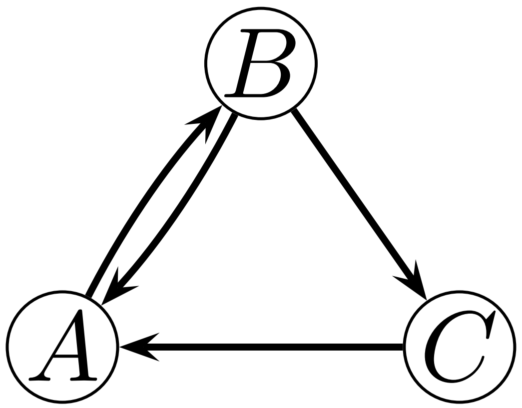
NICTA Funding and Supporting Members and Partners
Australian National University
UNSW
NSW Trade & Investment
Victoria
The University of Melbourne
The University of Sydney
Queensland Government
Griffith University
QUT
The University of Queensland
NICTA Partners

Motivation

- towards more automation in program verification
  - functional correctness
  - use algebra to improve proof automatisation
  - using pre/post conditions (Hoare-style reasoning)

- at the moment
  - look at ‘simple’ and well-known while programs (invariant proofs)
  - find ‘correct’/appropriate algebra
  - limited to algorithms where data structure can be modelled by algebra

Unweighted Graphs



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

$$\{(A, B), (B, A), (B, C), (C, A)\}$$

- edges are relation between nodes
- relation algebra prime candidate
  - elements are sets of relations/Boolean matrices
  - offers operations for
    - sequential composition
    - set operations (union, intersection, complement)
    - transposition
    - finite iteration (Kleene star)
- well known, used for program verification

Warming Up: Reflexive–Transitive Closure

NICTA

input R
$C, v := \mathsf{I}, \mathsf{O};$

while $v \neq R;\mathsf{L}$ **do**
  let $p = point(R;\mathsf{L} \cap \overline{v});$
  $C, v := C \cup C;p;p^{\mathsf{T}};R;C \;,\; v \cup p$

od
return C

Warming Up: Reflexive-Transitive Closure

$\textbf{input } R$
$C, v := \mathsf{I}, \mathsf{O};$

$\textbf{while } v \neq R;\mathsf{L} \textbf{ do}$
$\quad \textbf{let } p = point(R;\mathsf{L} \cap \overline{v});$
$\quad C, v := C \cup C;p;p^{\mathsf{T}};R;C \;,\; v \cup p$

$\textbf{od}$
$\textbf{return } C$

deterministic function returning
a point from R, which was not considered
before

Warming Up: Reflexive-Transitive Closure

$$
\begin{array}{l}
\textbf{input } R \\
C, v := \mathsf{I}, \mathsf{O}; \\
\\
\textbf{while } v \neq R;\mathsf{L} \textbf{ do} \\
\quad \textbf{let } p = point(R;\mathsf{L} \cap \overline{v}\,); \\
\quad C, v := C \cup C;p;p^{\mathsf{T}};R;C\ ,\ v \cup p \\
\\
\textbf{od} \\
\textbf{return } C \\
\{C = R^*\}
\end{array}
$$

deterministic function returning
a point from R, which was not considered
before

Warming Up: Reflexive-Transitive Closure

NICTA

$\{\text{True}\}$
input R
$C, v := \mathsf{I}, \mathsf{O};$

while $v \neq R;\mathsf{L}$ **do**
  let $p = point(R;\mathsf{L} \cap \overline{v});$
  $C, v := C \cup C;p;p^{\mathsf{T}};R;C \;,\; v \cup p$

od
return C
$\{C = R^*\}$

deterministic function returning
a point from R, which was not considered
before

Warming Up: Reflexive-Transitive Closure

$\{\mathrm{True}\}$
input R
$C, v := \mathsf{I}, \mathsf{O};$
$\{C = (R \cap v)^* \ \wedge\ v = v;\mathsf{L}\}$
while $v \neq R;\mathsf{L}$ **do**
  let $p = point(R;\mathsf{L} \cap \overline{v}\,);$
  $C, v := C \cup C;p;p^\top;R;C\ ,\ v \cup p$

od
return C
$\{C = R^*\}$

deterministic function returning
a point from R, which was not considered
before

Warming Up: Reflexive Transitive Closure

$$Inv_0(R, C, v) \;\Leftrightarrow\; C = (R \cap v)^*$$
$$Inv_1(v) \;\Leftrightarrow\; v = v \,;\mathsf{L}$$

- Proof: simple exercise?

Warming Up: Reflexive Transitive Closure

$$Inv_0(R, C, v) \;\Leftrightarrow\; C = (R \cap v)^*$$
$$Inv_1(v) \;\Leftrightarrow\; v = v\,;\mathsf{L}$$

- Proof: simple exercise?
- p is point $\;\Leftrightarrow\; p;\mathsf{L} = p \,\wedge\, \mathsf{L};p = \mathsf{L} \,\wedge\, p;p^\top \subseteq \mathsf{I}$

Warming Up: Reflexive Transitive Closure

$$Inv_0(R,C,v) \;\Leftrightarrow\; C = (R \cap v)^*$$
$$Inv_1(v) \;\Leftrightarrow\; v = v\,;\mathsf{L}$$

- Proof: simple exercise?
- p is point $\;\Leftrightarrow\; p;\mathsf{L} = p \,\wedge\, \mathsf{L};p = \mathsf{L} \,\wedge\, p;p^\top \subseteq \mathsf{I}$

- Proof Automatisation
  (Prover9 or any other automated Theorem Prover)

Establishment	
$Inv_0(R,\mathsf{I},\mathsf{O}) \wedge Inv_1(\mathsf{O})$	0s
Post-Condition	
$v = R\,;\mathsf{L} \wedge Inv_0(R,C,v) \wedge Inv_1(v) \;\Rightarrow\; C = R^*$	0s
Maintainance	
$Inv_1(v) \wedge p$ is point $\wedge\, p \subseteq R\,;L \cap \overline{v} \;\Rightarrow\; Inv_1(v \cup p)$	1s
$Inv_0(R,C,v) \wedge p$ is point $\wedge\, p \subseteq R\,;L \cap \overline{v} \;\Rightarrow\; Inv_0(R, C \cup C\,;p\,;p^\top;R\,;C, v \cup p)$	-

Warming Up: Reflexive Transitive Closure

$$Inv_0(R, C, v) \;\Leftrightarrow\; C = (R \cap v)^*$$
$$Inv_1(v) \;\Leftrightarrow\; v = v\,;\mathsf{L}$$

- Proof: simple exercise?
- p is point $\;\Leftrightarrow\; p;\mathsf{L} = p \,\wedge\, \mathsf{L};p = \mathsf{L} \,\wedge\, p;p^\top \subseteq \mathsf{I}$

- Proof Automatisation
  (Prover9 or any other automated Theorem Prover)

Establishment	
$Inv_0(R, \mathsf{I}, \mathsf{O}) \wedge Inv_1(\mathsf{O})$	0s
Post-Condition	
$v = R\,;\mathsf{L} \wedge Inv_0(R, C, v) \wedge Inv_1(v) \;\Rightarrow\; C = R^*$	0s
Maintenance	
$Inv_1(v) \wedge p \text{ is point} \wedge p \subseteq R\,;L \cap \overline{v} \;\Rightarrow\; Inv_1(v \cup p)$	1s
$Inv_0(R, C, v) \wedge p \text{ is point} \wedge p \subseteq R\,;L \cap \overline{v} \;\Rightarrow\; Inv_0(R, C \cup C\,;p\,;p^\mathsf{T}\,;R\,;C, v \cup p)$	0s

+ 3 properties about Kleene star

More Examples

input R

$S, v := \mathsf{I}, \mathsf{O};$
while $v \neq \mathsf{L}$ **do**

  let $p = point(\,\overline{v} \cap \overline{(R^{\mathsf{T}} \cap \overline{\mathsf{I}}\,) ; \overline{v}}\,);$
  $S, v := S \cup v ; p^{\mathsf{T}}, v \cup p$

od
return S

More Examples

- Topological Sorting

$$
\begin{array}{l}
\textbf{input } R \\
\\
S, v := \mathsf{I}, \mathsf{O}; \\
\textbf{while } v \neq \mathsf{L} \textbf{ do} \\
\\
\qquad \textbf{let } p = point(\,\overline{v} \cap \overline{(R^{\mathsf{T}} \cap \overline{\mathsf{I}}\,);\overline{v}}\,); \\
\qquad S, v := S \cup v;p^{\mathsf{T}}, v \cup p \\
\\
\textbf{od} \\
\textbf{return } S
\end{array}
$$

More Examples

- Topological Sorting

$$
\begin{array}{l}
\textbf{input } R \\
\{R;R^* = \mathsf{O}\} \\
S, v := \mathsf{I}, \mathsf{O}; \\
\textbf{while } v \neq \mathsf{L} \textbf{ do} \\
\quad \textbf{let } p = point(\,\overline{v} \cap \overline{(R^{\mathsf{T}} \cap \overline{\mathsf{I}}\,);\overline{v}}\,); \\
\quad S, v := S \cup v;p^{\mathsf{T}}, v \cup p \\
\textbf{od} \\
\textbf{return } S
\end{array}
$$

More Examples

- Topological Sorting

$$
\begin{array}{l}
\textbf{input } R \\
\{R;R^* = \mathsf{O}\} \\
S, v := \mathsf{I}, \mathsf{O}; \\
\textbf{while } v \neq \mathsf{L} \textbf{ do} \\
\qquad \textbf{let } p = point(\overline{v} \cap \overline{(R^{\mathsf{T}} \cap \overline{\mathsf{I}});\overline{v}}); \\
\qquad S, v := S \cup v;p^{\mathsf{T}}, v \cup p \\
\textbf{od} \\
\textbf{return } S \\
\{R \subseteq S \wedge \mathsf{I} \subseteq S \wedge S;S \subseteq S \wedge S \cap S^{\mathsf{T}} \subseteq \mathsf{I} \wedge S \cup S^{\mathsf{T}} = \mathsf{L}\}
\end{array}
$$

More Examples

- Topological Sorting

$$
\begin{array}{l}
\textbf{input } R \\
\{R;R^* = \mathsf{O}\} \\
S, v := \mathsf{I}, \mathsf{O}; \\
\textbf{while } v \neq \mathsf{L} \textbf{ do} \\
\{\mathsf{I} \subseteq S \ \wedge \ S;S \subseteq S \ \wedge \ S \cap S^{\mathsf{T}} \subseteq S \ \wedge \ S \cup S^{\mathsf{T}} = v;v^{\mathsf{T}} \cup \mathsf{I} \wedge \\
\\
\quad \textbf{let } p = point(\, \overline{v} \ \cap \ \overline{(R^{\mathsf{T}} \cap \overline{\mathsf{I}}\,);\, \overline{v}}\ ); \\
\quad S, v := S \cup v;p^{\mathsf{T}}, v \cup p \\
\\
\textbf{od} \\
\textbf{return } S \\
\{R \subseteq S \ \wedge \ \mathsf{I} \subseteq S \ \wedge \ S;S \subseteq S \ \wedge \ S \cap S^{\mathsf{T}} \subseteq \mathsf{I} \ \wedge \ S \cup S^{\mathsf{T}} = \mathsf{L}\}
\end{array}
$$

More Examples

- Topological Sorting

$$
\begin{array}{l}
\textbf{input } R \\
\{R;R^* = \mathsf{O}\} \\
S, v := \mathsf{I}, \mathsf{O}; \\
\textbf{while } v \neq \mathsf{L} \textbf{ do} \\
\{\mathsf{I} \subseteq S \ \wedge \ S;S \subseteq S \ \wedge \ S \cap S^{\mathsf{T}} \subseteq S \ \wedge \ S \cup S^{\mathsf{T}} = v;v^{\mathsf{T}} \cup \mathsf{I} \wedge \\
\quad v;\mathsf{L} \subseteq v \ \wedge \ S;v \subseteq v \ \wedge \ R \cap v;v^{\mathsf{T}} \subseteq S \ \wedge \ R;v \subseteq v\} \\
\qquad \textbf{let } p = point(\overline{v} \cap \overline{(R^{\mathsf{T}} \cap \overline{\mathsf{I}});\overline{v}}); \\
\qquad S, v := S \cup v;p^{\mathsf{T}}, v \cup p \\
\\
\textbf{od} \\
\textbf{return } S \\
\{R \subseteq S \ \wedge \ \mathsf{I} \subseteq S \ \wedge \ S;S \subseteq S \ \wedge \ S \cap S^{\mathsf{T}} \subseteq \mathsf{I} \ \wedge \ S \cup S^{\mathsf{T}} = \mathsf{L}\}
\end{array}
$$

More Examples

- Topological Sorting

Proof is fully automatic (incl. termination)

$$
\begin{array}{l}
\textbf{input } R \\
\{R;R^* = \mathsf{O}\} \\
S, v := \mathsf{I}, \mathsf{O}; \\
\textbf{while } v \neq \mathsf{L} \textbf{ do} \\
\{\mathsf{I} \subseteq S \wedge S;S \subseteq S \wedge S \cap S^{\mathsf{T}} \subseteq S \wedge S \cup S^{\mathsf{T}} = v;v^{\mathsf{T}} \cup \mathsf{I} \wedge \\
\quad v;\mathsf{L} \subseteq v \wedge S;v \subseteq v \wedge R \cap v;v^{\mathsf{T}} \subseteq S \wedge R;v \subseteq v\} \\
\qquad \textbf{let } p = point(\overline{v} \cap \overline{(R^{\mathsf{T}} \cap \overline{\mathsf{I}});\overline{v}}); \\
\qquad S, v := S \cup v;p^{\mathsf{T}}, v \cup p \\
\\
\textbf{od} \\
\textbf{return } S \\
\{R \subseteq S \wedge \mathsf{I} \subseteq S \wedge S;S \subseteq S \wedge S \cap S^{\mathsf{T}} \subseteq \mathsf{I} \wedge S \cup S^{\mathsf{T}} = \mathsf{L}\}
\end{array}
$$

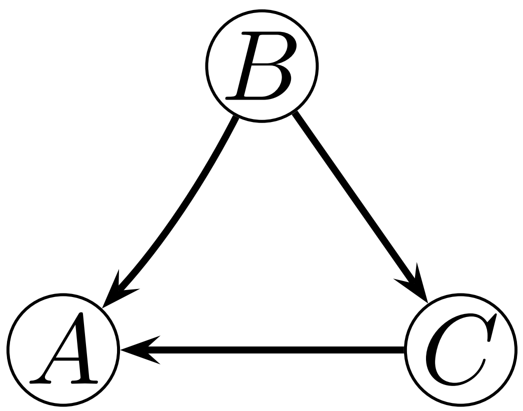
More Examples

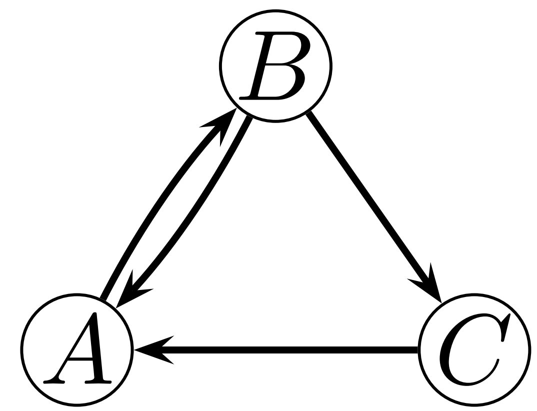
- Matching Algorithm
- Node Colouring
- …

- **Relation algebra seems to be well suited for most (all?) graph problems**

Subtree

- natural order: $\subseteq$

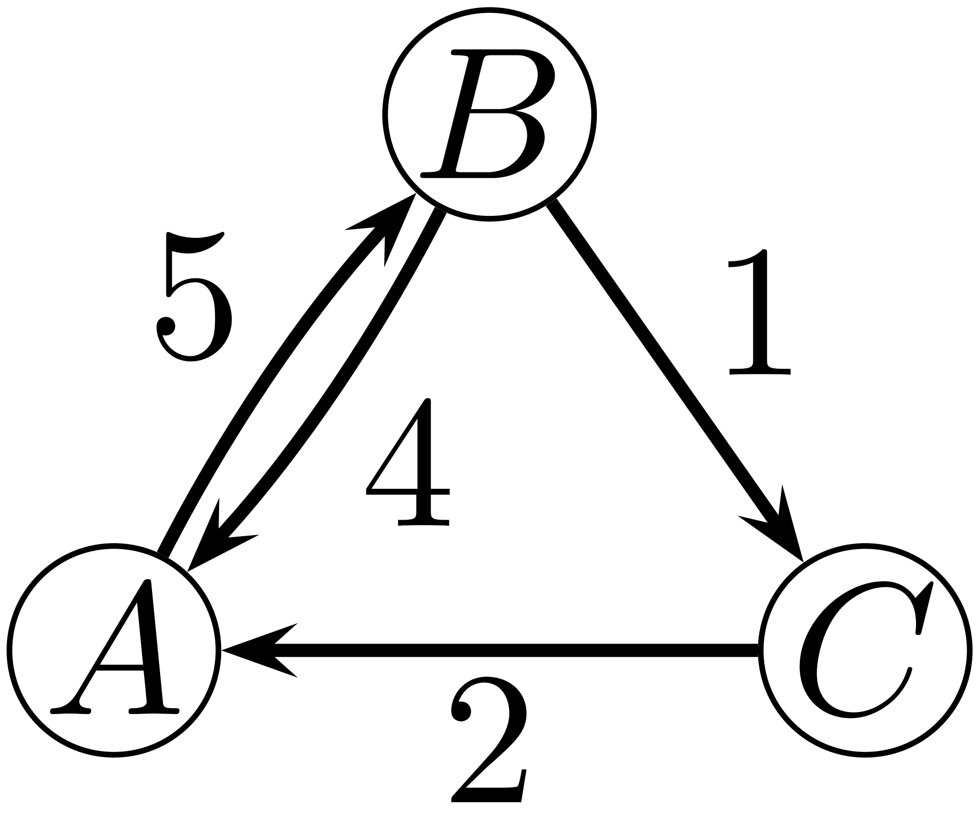




$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \subseteq \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

$$\{ (B,A), (B,C), (C,A)\} \subseteq \{(A,B), (B,A), (B,C), (C,A)\}$$

Weighted Graphs



Algebras for Weighted Graphs

NICTA

- Matrices over Min-Plus-Algebra (and variants)
  - algorithms such as Dijkstra and Floyd-Warshall
- Routing Algebra
  - developed for Mesh Protocols
    (see IFIP 2.1 Reisensburg)
- Other algebras: Max-Plus, Max-Min, Min-Max, …

Min-Plus Algebra

- Choice: Take path with smaller weight
- Path Composition: Addition
- Kleene star: $n^* = \min_{i \geq 0}(\sum_{j=0}^{i} n) = \min(0, n, 2n, \dots) = 0$
- $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0, ^*)$ forms a Kleene algebra
    - no intersection, no complement
    - no transposition
    - natural order defined as usual

$$m \sqsubseteq n \Leftrightarrow \min(m, n) = n \Leftrightarrow n \leq m$$

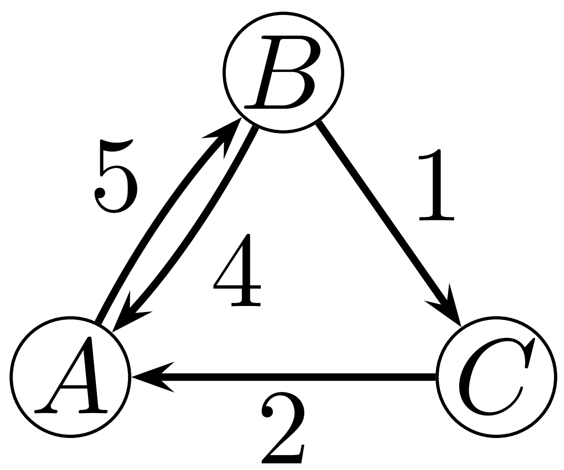
- Theorem:
  Matrices over Kleene algebras are Kleene algebras
    - natural order is defined point-wise

Problems

- Is this algebra as suitable and flexible as relation algebra?

Reflexive-Transitive Closure

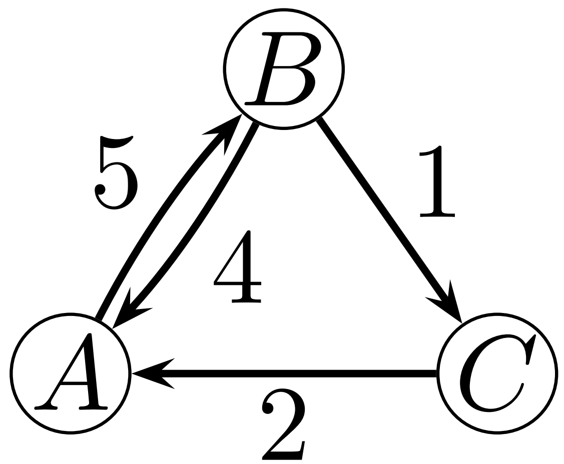
- all-shortest paths



$$G = \begin{pmatrix} \infty & 5 & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix}$$

Reflexive-Transitive Closure

- all-shortest paths

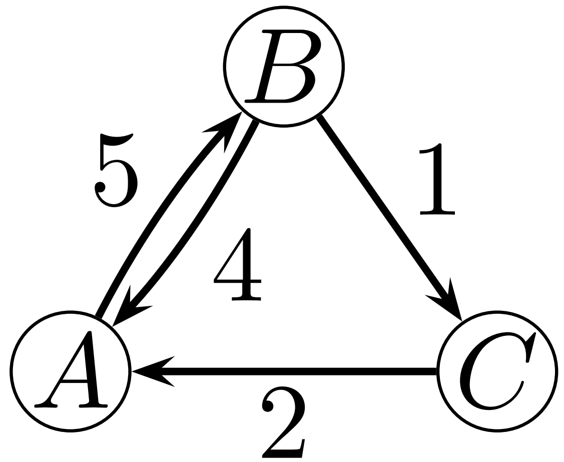


$$G = \begin{pmatrix} \infty & 5 & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix}$$

$$G^* = \begin{pmatrix} 0 & 5 & 6 \\ 3 & 0 & 1 \\ 2 & 7 & 0 \end{pmatrix}$$

Reflexive-Transitive Closure

- all-shortest paths



$$G = \begin{pmatrix} \infty & 5 & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix}$$

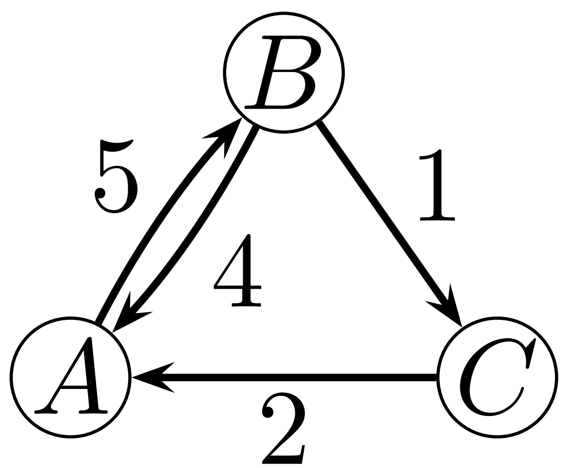
- How to calculate the star
  - classical matrix decomposition (cf. Kozen)
  - algorithm from above ?

Reflexive Transitive Closure

$$
\begin{array}{l}
\textbf{input } R \\
C, v := \mathsf{I}, \mathsf{O}; \\
\\
\textbf{while } v \neq R;\mathsf{L} \textbf{ do} \\
\quad \textbf{let } p = point(R;\mathsf{L} \cap \overline{v}); \\
\quad C, v := C \cup C;p;p^{\mathsf{T}};R;C \;,\; v \cup p \\
\\
\textbf{od} \\
\textbf{return } C
\end{array}
$$

Reflexive-Transitive Closure

- all-shortest paths



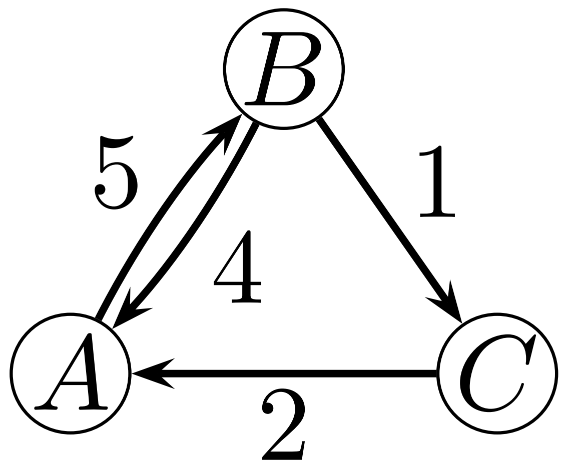
$$G = \begin{pmatrix} \infty & 5 & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix}$$

- How to calculate the star
  - classical matrix decomposition (cf. Kozen)
  - algorithm from above
    - problem: what is a point

$$p;\mathsf{L} = p \;\wedge\; \mathsf{L};p = \mathsf{L} \;\wedge\; p;p^\top \subseteq \mathsf{I}$$

Reflexive-Transitive Closure

- all-shortest paths



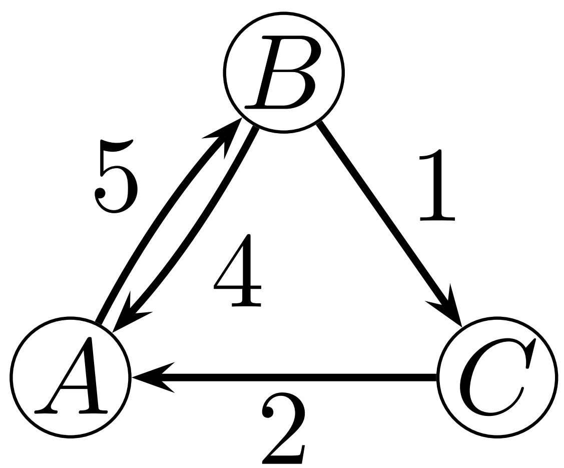
$$G = \begin{pmatrix} \infty & 5 & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix}$$

- How to calculate the star
  - classical matrix decomposition (cf. Kozen)
  - algorithm from above
    - problem: what is a point

$$p \cdot \top = p \;\wedge\; \top \cdot p = \top \;\wedge\; p \cdot p^{\top} \sqsubseteq \mathsf{Id}$$

Reflexive-Transitive Closure

- all-shortest paths



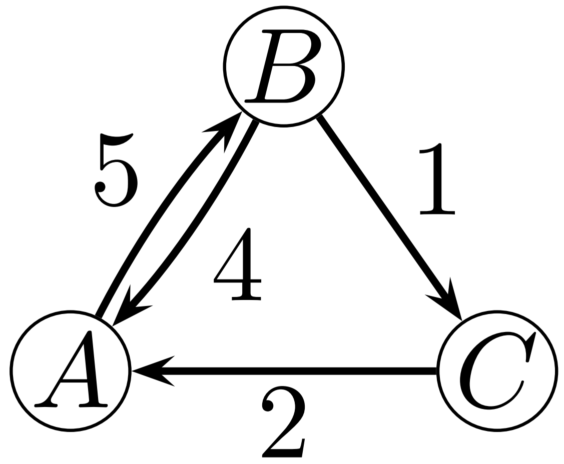
$$G = \begin{pmatrix} \infty & 5 & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix}$$

- How to calculate the star
  - classical matrix decomposition (cf. Kozen)
  - algorithm from above
    - problem: what is a point

$$p \cdot \top = p \wedge \top \cdot p = \top \wedge \cancel{p \cdot p^\top} \sqsubseteq \mathsf{Id}$$

Reflexive-Transitive Closure

- all-shortest paths



$$G = \begin{pmatrix} \infty & 5 & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix}$$

- How to calculate the star
  - classical matrix decomposition (cf. Kozen)
  - algorithm from above
    - points can be characterised via atomic test elements (every Kleene algebra can be equipped with a test algebra — no details in this talk)

Example: Prim's algorithm

input G, v
$\{G \text{ symmetric}\}$
$U, T := v, 0;$
while $U \neq \mathsf{Id}$ **do**
$\{T \text{ is minimal spanning tree in } U \cdot G \cdot U\}$
  let e edge with minimal weight from U to $\neg U$
  $U, T := U + \text{source of } e \ , \ T + e$
od
return T
$\{T \text{ is minimal spanning tree}\}$

Example: algorithm to compute Spanning Tree

input G, v

$\{G \text{ symmetric}\}$

$U, T := v, 0;$

while $U \neq \mathsf{Id}$ **do**

$\{T \text{ is spanning tree in } U \cdot G \cdot U\}$

    let e edge from U to $\neg U$

    $U, T := U + \text{source of } e\ ,\ T + e$

od

return T

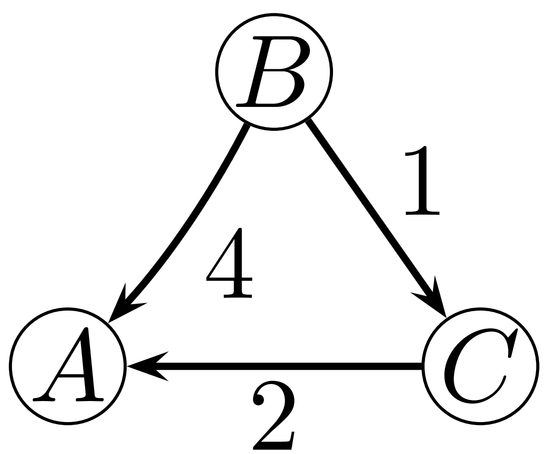
$\{T \text{ is spanning tree}\}$

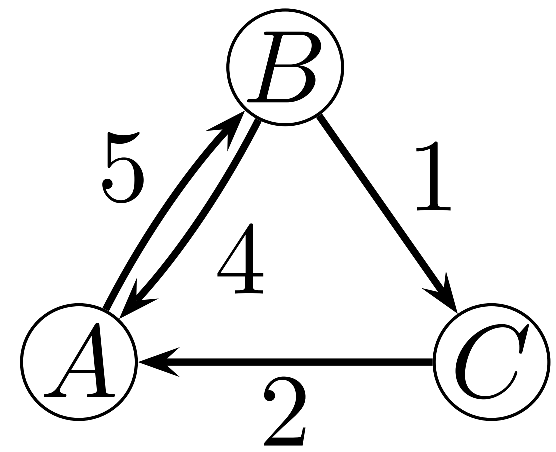
Spanning Tree

- T is spanning tree of G
  - T is tree (injective, reaches everything)
  - T is *subtree* of G

Subtree

- natural order: $\sqsubseteq$

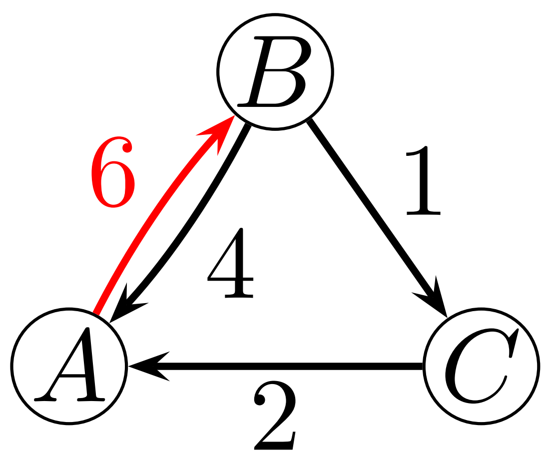


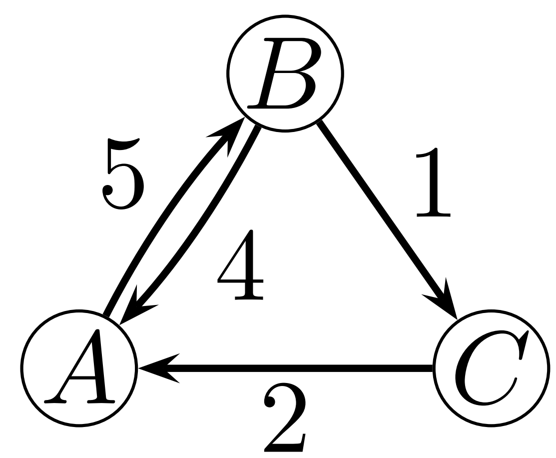


$$\begin{pmatrix} \infty & \infty & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix} \sqsubseteq \begin{pmatrix} \infty & 5 & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix}$$

Subtree

- natural order: $\sqsubseteq$





$$\begin{pmatrix} \infty & \color{red}{6} & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix} \sqsubseteq \begin{pmatrix} \infty & 5 & \infty \\ 4 & \infty & 1 \\ 2 & \infty & \infty \end{pmatrix}$$

`

- Relation algebra (set model): $(\wp(V \times V), \cup, \cap, \ldots)$
- "pseudo" multigraphs (Matrices with sets as entries)
  - $(\wp(V \times \mathbb{N} \times V), \cup, +_{\text{join}}, \emptyset, V \times \{0\} \times V, {}^*)$
    forms Kleene algebra, where $+_{\text{join}}$ is point-wise operation

$$(u, m, v) +_{\text{join}} (w, n, x) = \begin{cases} (u, m+n, x) & \text{if } v = w \\ \text{undefined} & \text{otherwise} \end{cases}$$

  - $(\wp(V \times \mathbb{Z} \times V), \cup, +_{\text{join}}, \emptyset, V \times \{0\} \times V, {}^*)$
    can be turned into a Relation algebra

$$(u, m, v)^{\top} = (v, -m, u)$$

- why not real multi graphs?
  no natural order

Back to Spanning tree

input G, v
$\{G \text{ symmetric}\}$
$U, T := v, 0;$
while $U \neq \mathsf{Id}$ **do**
$\{T \text{ is spanning tree in } U \cdot G \cdot U\}$
    let e edge from U to $\neg U$
    $U, T := U + \text{source of } e \;,\; T + e$
od
return T
$\{T \text{ is spanning tree}\}$

Back to Spanning tree

$\textbf{input } G, v$
$\{G \text{ symmetric}\}$
$U, T := v, 0;$
$\textbf{while } U \neq \mathsf{Id} \textbf{ do}$
$\{T \leq U \cdot G \cdot U \ \wedge \ \mathsf{range}(v \cdot T^{+}) = U\}$
$\quad \textbf{let } e \text{ edge with } e \leq U \cdot G \cdot \neg U \,, \ldots;$
$\quad U, T := U + \mathsf{source}(e) \ , \ T + e$
$\textbf{od}$
$\textbf{return } T$
$\{T \text{ is spanning tree}\}$

Spanning tree

- Correctness can be shown similar to the above examples
  - in all three models
  - straight-forward (full automatic if isotonicity laws are added)
  - source and range can be defined via algebraic operations (tests, domain, codomain)

- But: How to characterise minimality?

Problem: Minimality

- Easy if additional weight-function on top
  - model dependent, requires specific axioms for functions…
  - could be performed on relations only
  - but seems not to be the best way

- can we integrate minimality into algebra?
  - how to access the weights?
  - in $(\wp(V \times \mathbb{N} \times V), \cup, +_{\mathrm{join}}, \emptyset, V \times \{0\} \times V, ^{*})$
    one can at least compare edges

$$e_1 \text{ preferred over } e_2 \;\Leftrightarrow\; \top \cdot e_1 \cdot \top \leq \top \cdot e_2 \cdot \top$$

Summary

- aim at more automation for program verification
  - "black-box" approach
  - any ATP/ITP system should be fine
- focus on graph algorithms
- suitable algebras
  - unweighted graphs: relation algebra
  - shortest paths: min-plus algebra
    (building graphs)
  - spanning trees: ???
    (subtrees)
  - max-plus algebra, max-min algebra …

- weighted graphs need several algebraic models
  (hopefully all based on same algebra)


NICTA


From **imagination** to **impact**