Discrete Mathematics with Applications I

COMPSCI&SFWRENG 2DM3

McMaster University, Fall 2019

Wolfram Kahl

2019-12-04

Counting Challenges

Let A and B be finite sets with $\# A = a$ and $\# B = b$:

- $\# (A \times B) \;=\; ?$ — pairs
- $\# (A \leftrightarrow B) \;=\; \# (\mathbb{P} (A \times B)) \;=\; ?$ — relations
- $\# (A \longrightarrow B) \;=\; ?$ — total functions
- $\# (A \nrightarrow B) \;=\; ?$ — partial functions
- $\# (A \rightarrowtail\!\!\!\twoheadrightarrow A) \;=\; ?$ — homogeneous total bijections
- $\# (A \rightarrowtail\!\!\!\twoheadrightarrow B) \;=\; ?$ — total bijections
- $\# (A \rightarrowtail B) \;=\; ?$ — total injections
- $\# (A \mathrel{\rightarrowtail\!\!\!\!+\!\!\!\twoheadrightarrow} B) \;=\; ?$ — partial bijections
- $\# \{ S \mid S \subseteq B \wedge \# S = a \} \;=\; ?$ — a-combinations of B

- $\# (A \mathrel{\rightarrowtail\!\!\!+} B) \;=\; ?$ — partial injections

- $\# (A \twoheadrightarrow B) \;=\; ?$ — total surjections

Please don't forget to evaluate **all** your courses at `https://evals.mcmaster.ca` !

Plan for Today

- **Combinatorial Analysis — "Counting"** (LADM chapter 16)
  - continued: Combinations
- **Topological Sort** (LADM section 14.4)
  - An example for algorithm development based on discrete math
- **Conclusion**

Review Session

- Saturday, Dec. 7, 13:30, room TBA

2DM3 on Avenue and CalcCheck$_{\text{Web}}$ remain active throughout term 2.

- When a notebook becomes inaccessible, please notify me!

Collected lecture slides will be posted under "General".

r-Combinations — LADM p. 340

- An r-combination of a set is a subset of size r.
- A permutation is a sequence; a combination is a set.
- For example, the 2-permutations of the set consisting of the letters in SOHN are

  SO,SH,SN,OH,ON,OS,HN,HS,HO,NS,NO,NH

  while the 2-combinations are

  {S, O}, {S, H}, {S, N}, {O, H}, {O, N}, {H, N}

(16.9) **Definition:** The binomial coefficient $\binom{n}{r}$, which is read as "n choose r", is defined by:

$$\binom{n}{r} = \frac{n!}{r! \cdot (n-r)!} \qquad \text{(for } 0 \le r \le n\text{)}$$

(16.10) **Theorem:** The number of r-combinations of n elements is $\binom{n}{r}$.

- $\# \{ S \mid S \subseteq B \land \# S = a \} = $ **?** a-combinations of B

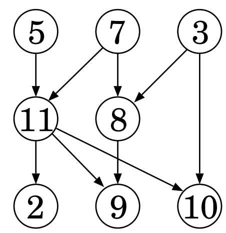
(16.23) **Binomial theorem:** For $n : \mathbb{N}$:

$$(x+y)^n = (\sum k : \mathbb{N} \mid k \le n \bullet \binom{n}{k} \cdot x^k \cdot y^{(n-k)})$$

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1

Topological Sort — Introduction

A topological sort of a acyclic simple directed graph (V, B) is a linear order E containing B, that is, $E \cap E^{\smile} \subseteq \mathrm{Id} \subseteq E \supseteq E \mathbin{⨾} E$ and $E \cup E^{\smile} = V \times V$ and $B \subseteq E$.



Since (V, B) is a DAG, B^* is an order: $B^* \cap B^{*\smile} \subseteq \mathrm{Id} \subseteq B^* \supseteq B^* \mathbin{⨾} B^*$

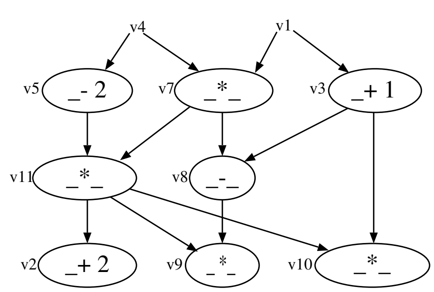
E is normally presented as a sequence in *Seq* V that is sorted with repect to E and contains all elements of V.

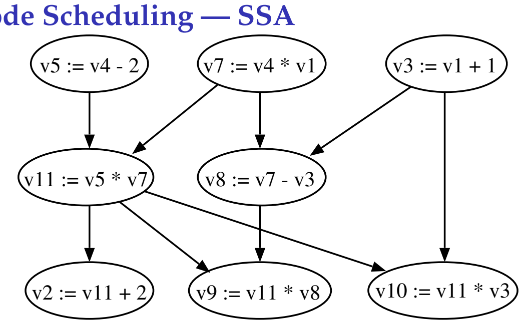
Example: The DAG above has, among others, the following topological sorts:

- [5, 7, 3, 11, 8, 2, 9, 10] — *visual left-to-right, top-to-bottom*
- [3, 5, 7, 8, 11, 2, 9, 10] — *smallest-numbered available vertex first*
- [5, 7, 3, 8, 11, 10, 9, 2] — *fewest edges first*
- [7, 5, 11, 3, 10, 8, 9, 2] — *largest-numbered available vertex first*
- [5, 7, 11, 2, 3, 8, 9, 10] — *attempting top-to-bottom, left-to-right*
- [3, 7, 8, 5, 11, 10, 2, 9] — *(arbitrary)*

$B = \{ \langle 3,8 \rangle, \langle 3,10 \rangle, \langle 5,11 \rangle, \langle 7,8 \rangle, \langle 7,11 \rangle, \langle 8,11 \rangle, \langle 11,2 \rangle, \langle 11,9 \rangle, \langle 11,10 \rangle \}$

Topological Sort — Code Scheduling — SSA





Static single assignment form: Each variable is assigned **once**, and assigned before use.

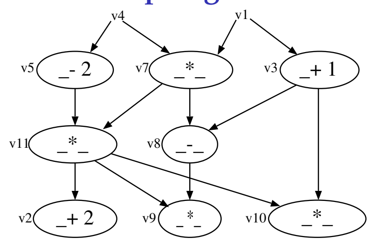
```
v5  := v4 - 2
v7  := v4 * v1
v3  := v1 + 1
v11  := v5 * v7
v8  := v7 - v3
v2  := v11 + 2
v9  := v11 * v8
v10  := v11 * v3
```

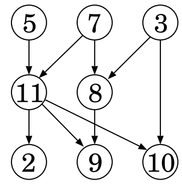
SSA can be considered as **encoding data-flow graphs**.

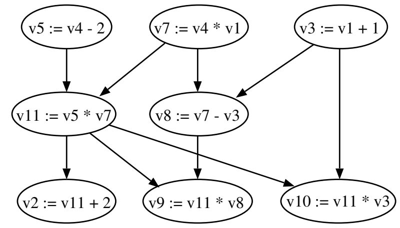
Each admissible re-ordering of an SSA sequence is a different topological sort of that graph.

It is frequently easier to think in terms of that graph than in terms of re-orderings!

Topological Sort — Code Scheduling — SSA — Pipeline Stalls







Static single assignment form: Each variable is assigned **once**, and assigned before use.

```
v7  := v4 * v1
v5  := v4 - 2
v11  := v5 * v7
v3  := v1 + 1
v10  := v11 * v3
v8  := v7 - v3
v9  := v11 * v8
v2  := v11 + 2
```

[7, **5, 11**, **3, 10**, **8, 9**, 2]

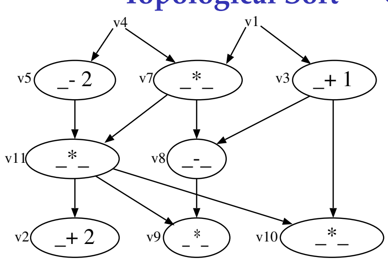
Let E be the topological sort of (V, B);
let $C = E - \mathrm{Id}$ be the associated strict-order.

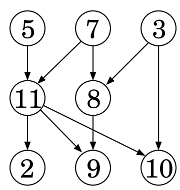
Depth-2 pipelining requires $B \subseteq C ⨾ C$.
Depth-3 pipelining requires $B \subseteq C ⨾ C ⨾ C$.

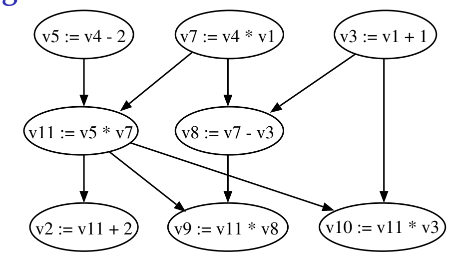
The "next-step" relation: $S = C - C ⨾ C^{+}$

Depth-2 pipelining requires $B \cap S = \{\}$.
Depth-3 pipelining requires $B \cap (S \cup S ⨾ S) = \{\}$.

Topological Sort — Code Scheduling — Different Schedules





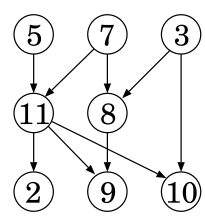


Example: Most of the original example topological sorts induce pipeline stalls:

- [5, 7, 3, 11, 8, 2, 9, 10] — *visual left-to-right, top-to-bottom*
- [3, 5, 7, 8, **11, 2**, 9, 10] — *smallest-numbered available vertex first*
- [5, 7, **3, 8**, 11, 10, 9, 2] — *fewest edges first*
- [7, **5, 11**, **3, 10**, **8, 9**, 2] — *largest-numbered available vertex first*
- [5, **7, 11**, 2, **3, 8, 9**, 10] — *attempting top-to-bottom, left-to-right*
- [3, 7, 8, 5, **11, 10**, 2, 9] — *(arbitrary)*

$B = \{\langle 3,8\rangle, \langle 3,10\rangle, \langle 5,11\rangle, \langle 7,8\rangle, \langle 7,11\rangle, \langle 8,11\rangle, \langle 11,2\rangle, \langle 11,9\rangle, \langle 11,10\rangle\}$

Topological Sort — Simple Algorithm



Given a DAG (V, B),
calculate sequence s encoding a topological sort E.

var $vs : \mathit{Set}\ V$
var $s : \mathit{Seq}\ V$
$vs := V$ ⨾ — **not-yet-used vertices**
$\{ vs = B \}$ — **Precondition**
$s := \epsilon$ ⨾ — **accumulator for result sequence**
$\{$ (vs and $\{v \mid v \in s\}$ partition V) $\wedge$
($\forall v : V \mid v \in s \bullet \forall u : V \mid u \,(\!B\!)\, v \bullet$ u precedes v in s) $\}$ — **Invariant**
while $vs \neq \{\}$ **do**
  Choose a source u of the subgraph $(vs, B \cap (vs \times vs))$ induced by vs ⨾
  $vs, s := vs - \{u\}, s \triangleright u$
od
$\{$ ($\forall u, v : V \mid u \,(\!B\!)\, v \bullet$ u precedes v in s) $\}$ — **Postcondition**

How to "Choose a source u of the subgraph induced by vs" **efficiently?**

Topological Sort — Making Choosing Minimal Elements Easier

To store mappings $V \mathrel{\ooalign{$\rightarrow$\cr\hfil$\circ$\hfil}\!\!\rightarrow} X$ in "**array ... of** X", "assume" $V = \mathsf{Fin}\ k$ where $k = \#\ V$.

$\mathbf{var}\ sources : Seq\ (\mathsf{Fin}\ k)$ — three new variables make vs superfluous
$\mathbf{var}\ preCount : \mathbf{array}\ \mathsf{Fin}\ k\ \mathbf{of}\ \mathbb{Z}$
$\mathbf{var}\ postSet : \mathbf{array}\ \mathsf{Fin}\ k\ \mathbf{of}\ set\ (\mathsf{Fin}\ k)$ — read-only version of $B : V \leftrightarrow V$ as $V \twoheadrightarrow \mathbb{P}V$

Coupling invariant:

$\{u \mid u \in sources\} = vs - (Ran\ B') \wedge$ — *sources* contains sources of $B' = B \cap (vs \times vs)$
$(\forall\ v \mid v \in vs \bullet preCount[v] = \#\ (Dom\ (B' \rhd \{v\}))\ \wedge$
$(\forall\ u \mid u \in vs \bullet postSet[u] = Ran\ (\{u\} \lhd B'))$

Initialisation:

```
for v ∈ Fin k do preCount[v] := # (Dom (B ▷ {v})) od ⨾
for u ∈ Fin k do postSet[u] := Ran ({u} ◁ B)) od ⨾
sources := ε ⨾
for v ∈ Fin k do if preCount[v] = 0 then sources := sources ▷ v fi od
```

Topological Sort — Complete LADM Algorithm

```
for v ∈ Fin k do preCount[v] := # (Dom (B ▷ {v})) od ⨾
for u ∈ Fin k do postSet[u] := Ran ({u} ◁ B)) od ⨾
sources := ε ⨾
for v ∈ Fin k do if preCount[v] = 0 then sources := sources ▷ v fi od
ghost vs := Fin k ⨾
s := ε
while m ≠ ε do
    u := head sources ⨾
    s := s ▷ u ⨾
    sources := tail sources ⨾        — remove u from sources
    ghost vs := vs − {u} ⨾
    for v ∈ postSet[u] do
        preCount[v] := preCount[v] − 1 ⨾
        if preCount[v] = 0 then sources := sources ▷ v fi
    od
od
```

Topological Sort — Complete $O(\#\ B + \#\ V)$ Algorithm

```
for p ∈ B do
    preCount[snd p] := preCount[snd p] + 1
    postSet[fst p] := postSet[fst p] ∪ {v}
od ⨾
sources := ε ⨾ for v ∈ Fin k do if preCount[v] = 0 then sources := sources ▷ v fi od
ghost vs := Fin k ⨾
s := ε
while m ≠ ε do
    u := head sources ⨾
    s := s ▷ u ⨾
    sources := tail sources ⨾        — remove u from sources
    ghost vs := vs − {u} ⨾
    for v ∈ postSet[u] do
        preCount[v] := preCount[v] − 1 ⨾
        if preCount[v] = 0 then sources := sources ▷ v fi
    od
od
```

More . . .

- More about **induction/recursion:** **Read chapter 12!**
- More about **relations:** **Read chapter 14!**
- A lot more about **graphs**: Chapter 19
- Quite a bit more about **integers**: Chapter 15
- **Sequences:** Chapter 13
- **Reasoning about programs:** Chapter 10 and section 12.6
- . . .
- **Combinatorics:** Chapter 16
- **Infinite sets:** Chapter 20

Do You Speak Math?

- You learnt basic vocabulary:
  - Distributivity, Absorption, Idempotency, Identity,. . .
  - transitive, univalent, injective, order, equivalence, mapping,. . .
  - source, root, connected, SCC,. . .
- You learnt **and practiced** a lot of grammar:
  - Sentence-level:
    - Propositional expressions
    - Typed expressions
    - Quantification / predicate-logic expressions
    - Relation-algebraic expressions
  - Text-level:
    - Inference Rules
    - Applying theorems
    - Calculational proofs
    - Structured proofs, Induction proofs
    - Axioms, Lemmata, Theorems, . . .
- You practiced translating:
  - To and from relational expressions

Do You Speak Math?

- Even if you don't read fluently yet —

  you know how to decipher

- Even if you don't speak/write fluently yet —

  you know how to construct a sentence
  you know how to construct a solid argument
  you know how to solidify a sloppy argument
  you know how to recognise a bogus argument

- **Building on these foundations will be much easier!**

Continued Use of Logics and Discrete Mathematics

- **2FA3 Discrete Mathematics II**
  — Predicate logic, formal languages, grammars, finite automata, transition relations, acceptance predicates, ...
- **2C03 Data Structures and Algorithms**
  — Sets, relations, functions as data; functions/relations on data; datatype invariants
- **CS 2ME3 / SE 2AA4 Introduction to Software Development**
  — Read and write specifications for simple module interfaces
- **SFWRENG 3BB4 Concurrent Systems Design**
  —correctness of concurrent programs
- **3RA3 Software Requirements**
  — Capturing **precisely** what the customer wants, formalisation
- **3DB3 Databases**
  — n-ary relations, relation**al** algebra; functional dependencies
- COMPSCI **3MI3 Principles of Programming Languages**
  — Programming paradigms, including functional programming; mathematical understanding of prog. language constructs
- COMPSCI **3EA3 Software Specification and Correctness**
  — program verification, correct-by-construction programming
- **3FP3 Functional Programming**

Concluding Remarks

- How do I find proofs? — There is no general recipe
- Proving is somewhat like doing puzzles — **practice helps**
- **Proofs** are especially **important for software** — and much care is needed!
- Be aware of **types**, both in programming, and in mathematics
- Be aware of **variable binding** — in quantification, local variables, formal parameters
- Strive to use **abstraction** to **avoid variable binding**
  — e.g., using relation algebra instead of predicate logic
- When designing **data representations**, **think mathematics**: **Subsets, relations, functions, injectivity**, ...
- **Thinking mathematics in programming** is easiest in functional languages, e.g., **Haskell**, OCaml
- **Specify formally!** — **Design for provability!**
- **When doing software, think discrete mathematics!**