Introduction to Programming

Bertrand Meyer

Last revised 23 January 2004

Lecture 24:
Topological Sort — 1: Background

Un diner en famille.

Surtout ! ne parlez pas de l'affaire Dreyfus!
"Topological sort"

From a given partial order, produce a compatible total order
The problem

From a given partial order, produce a compatible total order

- **Partial order**: ordering constraints between elements of a set, e.g.
  - “Remove dishes before discussing politics”
  - “Walk to Ütliberg before lunch”
  - “Take your medicine before lunch”
  - “Finish lunch before removing dishes”
- **Total order**: sequence including all elements of set
- **Compatible**: the sequence respects all ordering constraints
  - Ütliberg, Medicine, Lunch, Remove, Discuss: OK
  - Medicine, Ütliberg, Lunch, Remove, Discuss: OK
  - Discuss, Medicine, Lunch, Remove, Ütliberg: not OK

Pictured as a graph

- Ütliberg
- Medicine
- Lunch
- Dishes
- Politics

- “Remove dishes before discussing politics”
- “Walk to Ütliberg before lunch”
- “Take your medicine before lunch”
- “Finish lunch before removing dishes”

Sometimes there is no solution

- “Introducing recursion requires that students know about stacks”
- “You must discuss abstract data types before introducing stacks”
- “Abstract data types rely on recursion”

The constraints introduce a cycle
Topological sort: example uses

- From a dictionary, produce a list of definitions such that no word occurs prior to its definition.

- Produce a complete schedule for carrying out a set of tasks, subject to ordering constraints. (This is done for scheduling maintenance tasks for industrial tasks, often with thousands of constraints.)

- Produce a version of a class with the features reordered so that no feature call appears before the feature’s declaration.

Overall structure (1)

Given:
- A type \( G \)
- A set of elements of type \( G \)
- A set of constraints between these elements

Required:
- An enumeration of the elements, in an order compatible with the constraints

```cpp
class ORDERABLE \( [G] \) feature
    elements: LIST \( [G] \)
    constraints: LIST \( [\text{TUPLE} \ [G, G]] \)

top: LIST \( [G] \) is
    ensure
    compatible (Result, constraints)
end
```

Some mathematical background...
**Binary relation on a set**

Any property that either holds or doesn’t hold between two elements of a set

On a set PERSON of persons, example relations are:
- mother: a mother \( a \) holds if and only if \( a \) is the mother of \( b \)
- father
- child
- sister
- sibling (brother or sister)

Notation: \( a \mathbin{r} b \) to express that \( r \) holds of \( a \) and \( b \).

**Example: the before relation**

The set of interest:
\[ \text{Tasks} = \{ \text{Discuss, Lunch, Medicine, Remove, Ütliberg} \} \]

The constraining relation:
\[ \text{Dishes before Politics} \]
\[ \text{Ütliberg before Lunch} \]
\[ \text{Medicine before Lunch} \]
\[ \text{Lunch before Dishes} \]

"Remove dishes before discussing politics"
"Walk to Ütliberg before lunch"
"Take your medicine before lunch"
"Finish lunch before removing dishes"

**Special relations on a set \( X \)**

- universal \( [X] \): holds between any two elements of \( X \)
- id \( [X] \): holds between every element of \( X \) and itself
- empty \( [X] \): holds between no elements of \( X \)
We consider a relation $r$ on a set $P$ as a set of pairs in $P \times P$, containing all the pairs $[x, y]$ such that $x \in r y$.

Then $x \in r y$ simply means that $[x, y] \in r$.

Example:

Example: the before relation

The set of interest:

Tasks = \{Discuss, Lunch, Medicine, Remove, Ütliberg\}

The constraining relation:

constraints = \{[Dishes, Politics], [Ütliberg, Lunch], [Medicine, Lunch], [Lunch, Dishes]\}

Using ordinary set operators

\[ \text{spouse} = \text{wife} \cup \text{husband} \]
\[ \text{sibling} = \text{sister} \cup \text{brother} \cup \text{id} [\text{Person}] \]
\[ \text{sister} \subseteq \text{sibling} \]
\[ \text{father} \subseteq \text{ancestor} \]

universal $[X] = X \times X$ (cartesian product)

empty $[X] = \emptyset$
Possible properties of a relation

(On a set $X$. All definitions must hold for every $a, b, c \ldots \in X$.)

- **Total:** $(a \mathcal{R} b) \lor (b \mathcal{R} a)$
- **Reflexive:** $a \mathcal{R} a$
- **Symmetric:** $a \mathcal{R} b \Rightarrow b \mathcal{R} a$
- **Antisymmetric:** $(a \mathcal{R} b) \land (b \mathcal{R} a) \Rightarrow a = b$
- **Transitive:** $(a \mathcal{R} b) \land (b \mathcal{R} c) \Rightarrow a \mathcal{R} c$

Examples (on a set of persons)

- **sibling** Reflexive, symmetric, transitive
- **sister** Symmetric, transitive
- **family_head** Reflexive, antisymmetric
  
  $(a \text{ family_head } b)$ means $a$ is the head of $b$’s family, with one head per family
- **mother** (None)

- **Total:** $(a \mathcal{R} b) \lor (b \mathcal{R} a)$
- **Reflexive:** $a \mathcal{R} a$
- **Symmetric:** $a \mathcal{R} b \Rightarrow b \mathcal{R} a$
- **Antisymmetric:** $(a \mathcal{R} b) \land (b \mathcal{R} a) \Rightarrow a = b$
- **Transitive:** $(a \mathcal{R} b) \land (b \mathcal{R} c) \Rightarrow a \mathcal{R} c$

Total order relation

Relation is total order if:
- **Total**
- **Reflexive**
- **Transitive**
- **Antisymmetric**

Example: "less than equal" $\leq$ on integers (or reals)
Partial order relation

Relation is a partial order if:
- Reflexive
- Transitive
- Antisymmetric

Example: relation between points in a plane:
$p \preceq q$ if both
- $x_p \leq x_q$
- $y_p \leq y_q$

Example partial order

Here the following hold:
- $a \preceq b$
- $b \preceq d$
- $c \preceq d$
- $a \preceq d$

No link between $a$, $b$, and $c$:
- e.g. neither $a \preceq c$ nor $a \preceq b$

Possible topological sorts

$a \preceq b \preceq c \preceq d$

$a \preceq b \preceq c \preceq d$

$a \preceq d$
Topological sort understood

Here the relation `≤` is:
- \([a, b]\)
- \([a, d]\)
- \([b, d]\)
- \([c, d]\)

One of the solutions is:
- \([a, a]\)
- \([a, b]\)
- \([a, c]\)
- \([a, d]\)
- \([b, b]\)
- \([b, c]\)
- \([b, d]\)
- \([c, c]\)
- \([c, d]\)
- \([d, d]\)

We are looking for a total order relation \(t\) such that `≤` \(\subseteq\) \(t\)

---

Final statement of topological sort problem

From a given partial order, produce a compatible total order

where:

A partial order \(p\) is compatible with a total order \(t\) if and only if

\[ p \subseteq t \]

---

From constraints to partial orders

The relation defined by a set of constraints, such as

\[
\text{constraints} = \{(\text{Dishes, Politics}), (\text{Ütliberg, Lunch}), (\text{Medicine, Lunch}), (\text{Lunch, Dishes})\}
\]

is not by itself a partial order (although it’s easy to infer a partial order from it).
"Powers" and closure of a relation $r$

- $r^0 = \text{id} \left[ X \right]$ where $X$ is the underlying set
- $r^{n+1} = r^n \circ r$ where $\circ$ is composition

Transitive closure
- $r^* = r^0 \cup r^1 \cup r^2 \cup \ldots$ always transitive

Reflexive transitive closure
- $r^* = r^0 \cup r^1 \cup r^2 \cup \ldots$ always reflexive and transitive

No cycle in a relation

$\quad r \cap \text{id} \left[ X \right] = \emptyset$

From constraints to partial orders

The partial order of interest is $\text{constraints}^*$

$\quad \text{constraints} = \{\{Dishes, Politics\}, \{Ütliberg, Lunch\}, \{Medicine, Lunch\}, \{Lunch, Dishes\}\}$
Overall structure (1)

Given:
- A type \( G \)
- A set of elements of type \( G \)

Required:
- An enumeration of the elements, in an order compatible with the constraints

```plaintext
class ORDERABLE [G] feature
    constraints: LIST [TUPLE [G, G]]

    topsort: LIST [G] is
        ensure
            compatible (Result, constraints)
end
```

Non-uniqueness

- In general there are several possible solutions
- In practice topological sort uses an optimization criterion to choose between possible solutions.
Overall structure (2)

topsort: LIST [G] is
  require
    no_cycle (constraints)
  ...
  ensure
    compatible (Result, constraints)

Cycles

- ⊆ must be a partial order: no cycle in the transitive closure of constraints
  - No circular chain of the form \( e_0 \rightarrow e_1 \rightarrow \ldots \rightarrow e_n \rightarrow e_0 \)

- If there are cycles there exists no solution to the topological sort problem!

Overall structure (2)

Assume there are no cycles in the constraints
- Not realistic since input may have errors
Overall structure (3)

Don’t assume anything; find cycles as byproduct of attempt to do topological sort

“Attempt to do the topological sort, accounting for possible cycles in the constraints”

if “Cycles found” then
   “Report presence of one or more cycles in constraints”
end

End of lecture 24