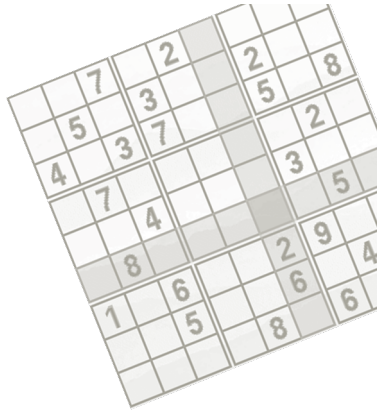


# Constraint Programming

**Roman Barták**
Department of Theoretical Computer Science and Mathematical Logic

Can we achieve GAC **faster than a general GAC algorithm**?

- for example revision of A < B can be done much faster via bounds consistency.

Can we write a **filtering algorithm for a constraint** whose **arity varies**?

- for example all_different constraint

We can exploit **semantics of the constraint** for efficient filtering algorithms that can work with any number of variables.

**☞ global constraints ☜**

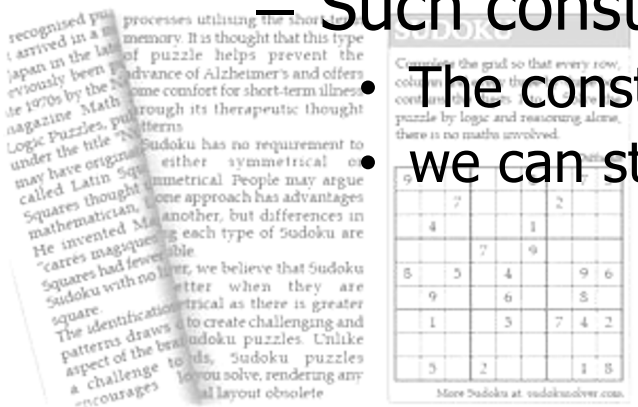
# Recall Sudoku

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 9 | 6 | 3 | 1 | 7 | 4 | 2 | 5 | 8 |
| 1 | 7 | 8 | 3 | 2 | 5 | 6 | 4 | 9 |
| 2 | 5 | 4 | 6 | 8 | 9 | 7 | 3 | 1 |
| 8 | 2 | 1 | 4 | 3 | 7 | 5 | 9 | 6 |
| 4 | 9 | 6 | 8 | 5 | 2 | 3 | 1 | 7 |
| 7 | 3 | 5 | 9 | 6 | 1 | 8 | 2 | 4 |
| 5 | 8 | 9 | 7 | 1 | 3 | 4 | 6 | 2 |
| 3 | 1 | 7 | 2 | 4 | 6 | 9 | 8 | 5 |
| 6 | 4 | 2 | 5 | 9 | 8 | 1 | 7 | 3 |

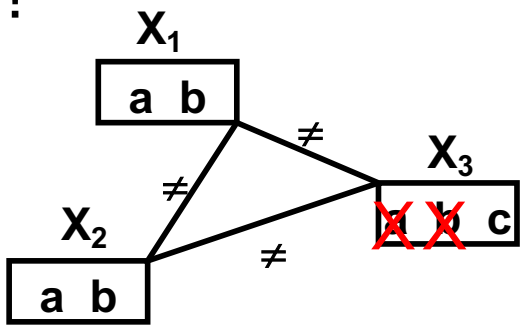
**Logic-based puzzle,** whose goal is to enter digits 1-9 in cells of 9×9 table in such a way, that no digit appears twice or more in every row, column, and 3×3 sub-grid.

**How to model such a problem?**

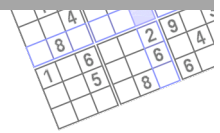
- variables **describe the cells**
- **inequality constraint** connect each pair of variables in each row, column, and sub-grid
- Such constraints do not propagate well!



- The constraint network is AC, but
- we can still remove some values.



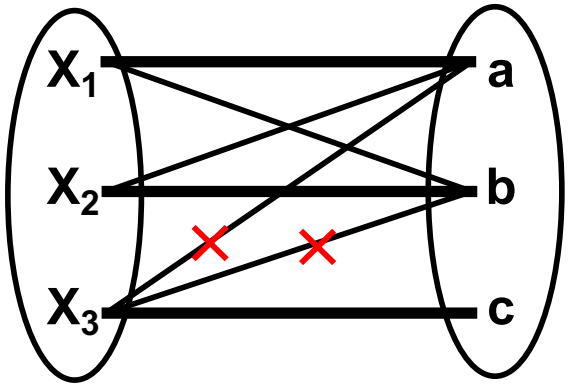
# all-different



This constraint models a complete set of binary inequalities.

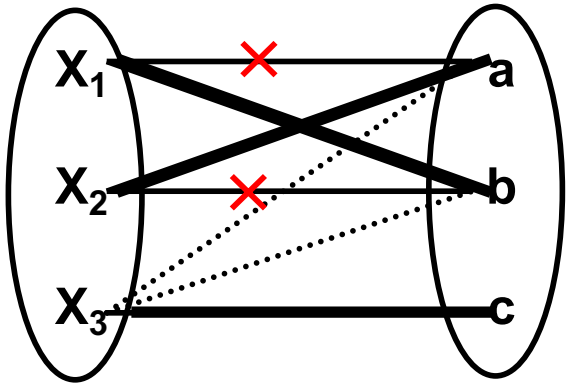
**all_different**($\{X_1,\ldots, X_k\}$) = $\{( d_1,\ldots, d_k) \mid \forall i \;\; d_i \in D_i \;\; \& \;\forall i \neq j \;\; d_i \neq d_j\}$

Domain filtering is based on **matching in bipartite graphs**
(nodes = variables+values, edges = description of domains)




***Initialization:***

1) find a maximum matching
2) remove all edges that do not belong to any maximum matching

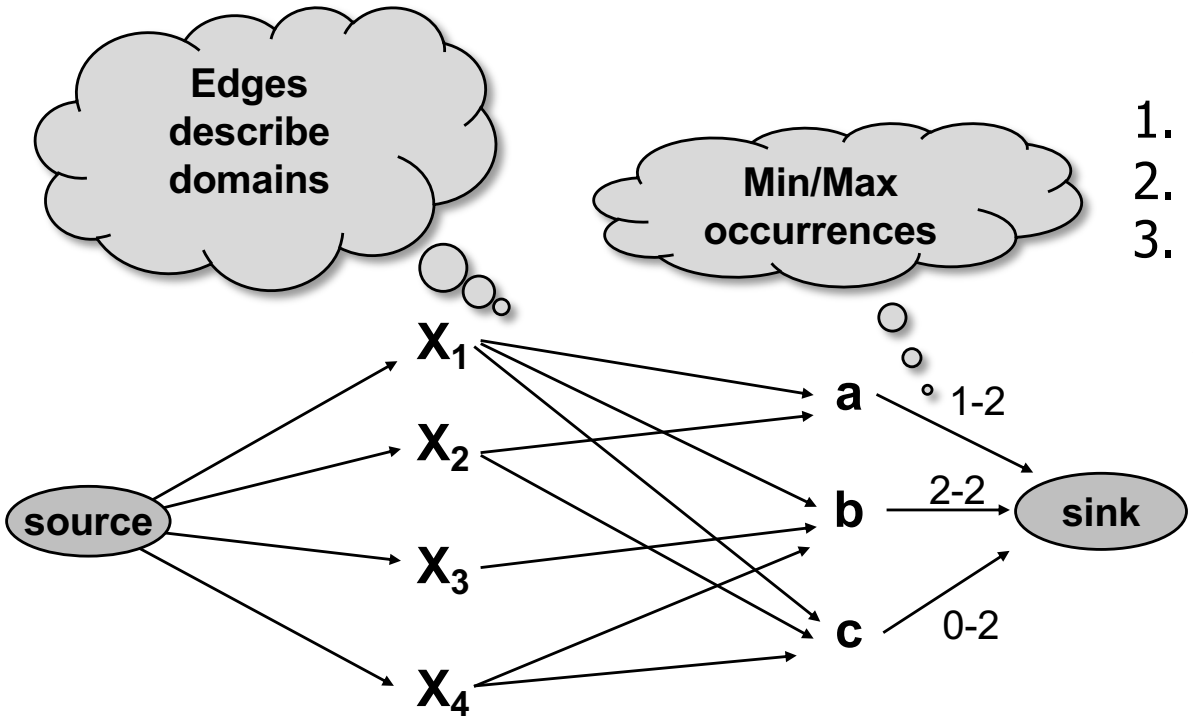



***Incremental propagation ($X_1 \neq a$):***

1) remove "deleted" edges
2) find a new maximum matching
3) remove all edges that do not belong to any maximum matching

# global cardinality

- A generalization of all-different
  - the number of occurrences of a value in a set of variables is restricted by minimal and maximal numbers of occurrences
- Efficient filtering is based on **network flows.**




1. make a value graph
2. add sink and source
3. set upper and lower bounds and edge capacities (0-1 and value occurrences)

**A maximal flow** corresponds to a feasible assignment of variables! We will find edges with zero flow in each maximal flow and then we will remove the corresponding edges.

Existence of **symmetrical solutions** decreases efficiency of constraint satisfaction (symmetrical search spaces are explored).

A classical example with many symmetries – **sports tournament scheduling.**

- there are n teams
- each team plays will all other teams, i.e., (n-1) rounds
- each team plays as a home team or a guest team

**How to model such a problem?**

- Round I is modelled by a sequence of **match codes** $K_i$.
  - $K_{i,j}$ is a code of j-th match at at round i
- We can swap matches at each round – **match symmetry**.
  - match symmetry is removed by constraint $K_{i,j} < K_{i,j+1}$
- We can swap complete rounds – **round symmetry**.
  - round symmetry is removed by constraint $K_i <_{lex} K_{i+1}$.

this constraint models **lexicographic ordering of two vectors**

**lex**$(\{X_1,\ldots, X_n\}, \{Y_1,\ldots, Y_n\}) \equiv (X_1 \leq Y_1) \wedge (X_1 = Y_1 \Rightarrow X_2 \leq Y_2) \wedge \ldots$
$\ldots \wedge (X_1 = Y_1 \wedge \ldots \wedge X_{n-1} = Y_{n-1} \Rightarrow X_n < Y_n)$

**Global filtering procedure** uses two pointers:

$\alpha$: the variables before $\alpha$ are all instantiated and pairwise equal

$\beta$: vectors starting at $\beta$ are lexicographically ordered but "oppositely"
floor($\{X_\beta,\ldots, X_n\}$) $>_{lex}$ ceiling($\{Y_\beta,\ldots, Y_n\}$)

| | |
|---|---|
| X = ⟨ {2}, {1,3,4}, {1,2,3,4,5}, {1,2}, {3,4,5} ⟩ | first set the pointers |
| Y = ⟨ {0,1,2}, {1}, {0,1,2,3,4}, {0,1}, {0,1,2} ⟩ | |
| $\alpha\uparrow$ $\uparrow\beta$ | |
| X = ⟨ {2}, {1,3,4}, {1,2,3,4,5}, {1,2}, {3,4,5} ⟩ | change $Y_1$, so at least $X_1 = Y_1$ |
| Y = ⟨ {2}, {1}, {0,1,2,3,4}, {0,1}, {0,1,2} ⟩ | and shift pointer $\alpha$ |
| $\alpha\uparrow$ $\uparrow\beta$ | |
| X = ⟨ {2}, {1}, {1,2,3,4,5}, {1,2}, {3,4,5} ⟩ | change $X_2$ so at least $X_2 = Y_2$ |
| Y = ⟨ {2}, {1}, {0,1,2,3,4}, {0,1}, {0,1,2} ⟩ | and again shift pointer $\alpha$ |
| $\alpha\uparrow$ $\uparrow\beta$ | |
| X = ⟨ {2}, {1}, {1,2,3}, {1,2}, {3,4,5} ⟩ | because $\alpha = \beta - 1$ |
| Y = ⟨ {2}, {1}, {2,3,4}, {0,1}, {0,1,2} ⟩ | force constraint $X_\alpha < Y_\alpha$ |
| $\alpha\uparrow$ $\uparrow\beta$ | |

## Rostering

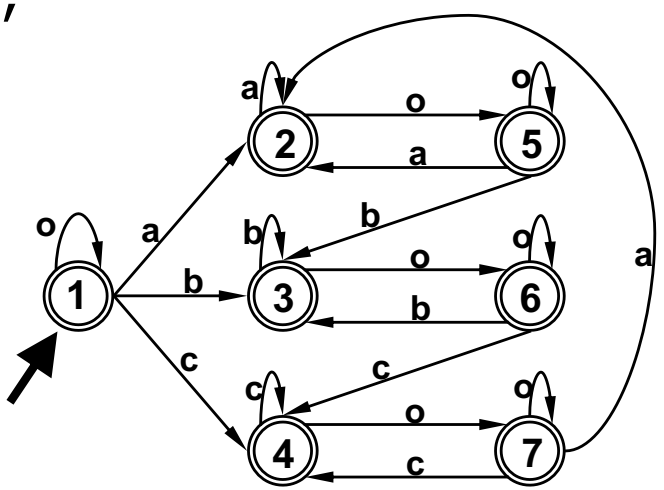
- **scheduling of shifts**, for example in hospitals
- There are typically specific shift sequencing constraints (given by trade unions, law etc.)

## Example:

- shifts: a, b, c, o (o means a free shift)
- constraints:
  - the same shift can repeat each day
  - at least one o shift is between a, b, between b, c, and between c, a
  - a-o*-c, b-o*-a, c-o*-b are not allowed (o* is a sequence of o shifts)
- Any shift can be used the first day, only shifts b, o can be used the second day, shifts a, c, o for the third day, shifts a, b, o for the forth day, and shift a the fifth day.

## How to model such a problem?

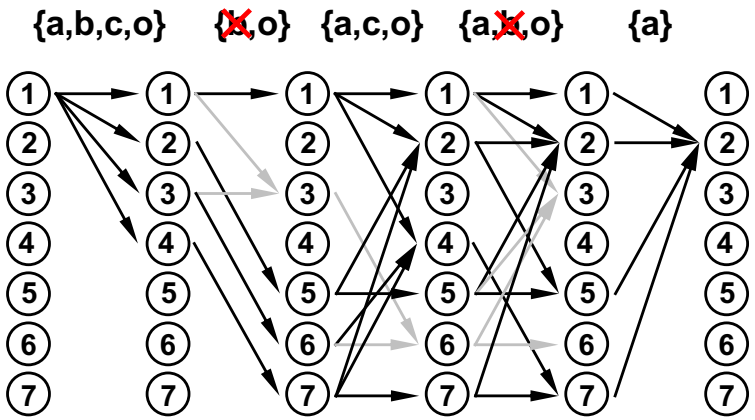
- variables describe shifts in days
- And what about constraints?
  - using a finite state automaton (FSA)



models a sequence of symbols **accepted by a FSA**

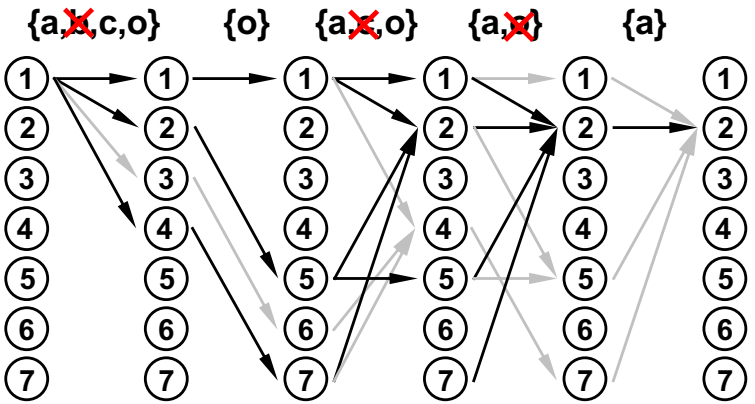
**regular**(A, {$X_1$,..., $X_k$}) = {($d_1$,..., $d_k$) | $\forall i \;\; d_i \in D_i \;\wedge\; d_1 \ldots d_k \in L(A)$}

filtering is based on representing all possible computations of a FSA using a **layered directed graph** (layer=states, arc=transitions)



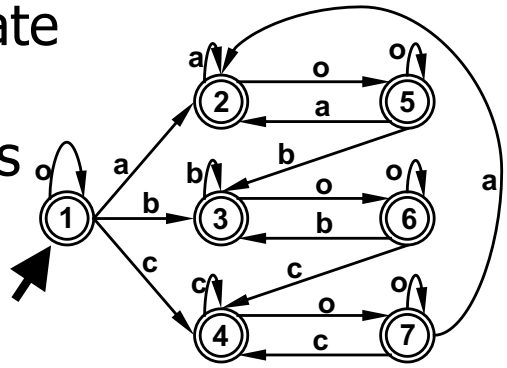

***Initialisation***

1. add arcs going from the initial state based on the symbols in the variables' domains
2. during the backward run, remove the arcs that are not on paths to the final states
3. remove the symbols without any arc




***Incremental filtering ($X_4 \neq o$):***

1. remove arcs for the deleted symbol
2. propagate the update in both directions
3. remove the symbols without any arc



Can we also model a sequence of symbols defined by a **context-free grammar**?

**grammar**(G, $\{X_1,\ldots, X_k\}$) = $\{(d_1,\ldots, d_k) \mid \forall i \;\; d_i \in D_i \;\wedge d_1\ldots d_k \in L(G)\}$

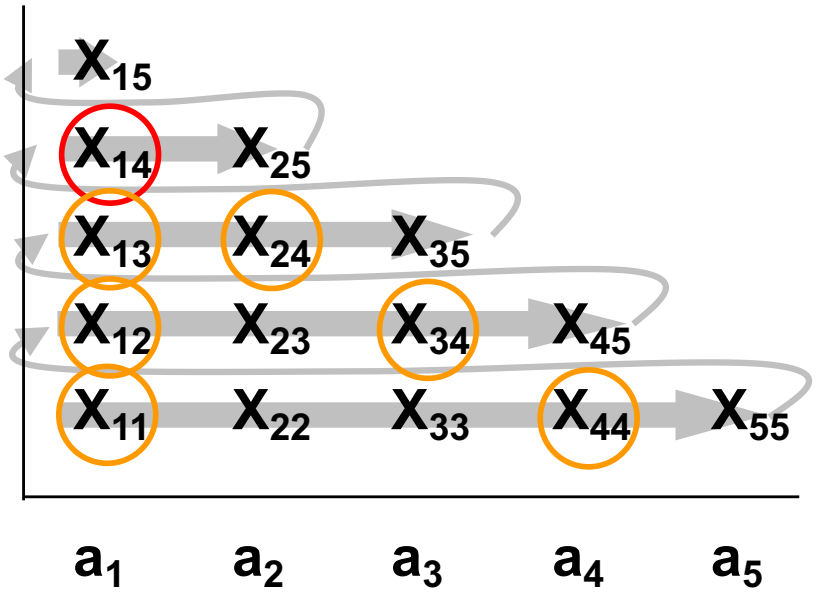
filtering is based on **algorithm CYK** for a Chomsky NF ($A \rightarrow BC \mid x$)

idea: $X_{i,j} = \{A \mid A \Rightarrow^* a_i a_{i+1}\ldots a_j\}$

start with: $X_{i,i} = \{A \mid (A \rightarrow a_i) \in P\}$

continue: $X_{i,j} = \{A \mid \exists k: i \leq k < j \;\; B \in X_{i,k} \wedge C \in X_{k+1,j} \wedge (A \rightarrow BC) \in P\}$

if $S \in X_{1,n}$, the $a_1 a_2 \ldots a_n$ belongs to the language




S → AB | BC
A → BA | a
B → CC | b
C → AB | a

| | | | | |
|---|---|---|---|---|
| {S,A,C} | | | | |
| - | {S,A,C} | | | |
| - | {B} | {B} | | |
| {S,A} | {B} | {S,C} | {S,A} | |
| {B} | {A,C} | {A,C} | {B} | {A,C} |
| b | a | a | b | a |

Let us go back to the **regular** constraint, which behaves like **sliding a special transition constraint over a sequence of variables**.

Such a principle can be generalized!

**slide$_j$**(C, {$X_1$,..., $X_n$}) ≡ ∀i C($X_{ij+1}$,..., $X_{ij+k}$)

- C is a k-ary constraint
- constant j determines the slide length

**Some examples:**

- **regular**(A, {$X_1$,..., $X_n$}) ≡ **slide$_2$**(C, {$Q_0$,$X_1$,$Q_1$, ..., $X_n$, $Q_n$})
  C(P,X,Q) represents a transition δ(P,X) = Q, $Q_0$ = {$q_0$}, $Q_n$ = F
- **lex**({$X_1$,..., $X_n$}, {$Y_1$,..., $Y_n$}) ≡ **slide$_3$**(C, {$B_0$,$X_1$,$Y_1$,$B_1$, ..., $X_n$, $Y_n$, $B_n$})
  C(B,X,Y,C) ≡ B=C=1 or (B=C=0 and X=Y) or (B=0, C=1 and X<Y)
  $B_0$ = 0, $B_n$ = 1 (strict lex), $B_n$ in {0,1} (non lex)
- **stretch**({$X_1$,..., $X_n$}, s, l , t) ≡ **slide$_2$**(C, {$X_1$,$S_1$, ..., $X_n$, $S_n$})
  C($X_i$, $S_i$, $X_{i+1}$, $S_{i+1}$) ≡ $X_i$ = $X_{i+1}$, $S_{i+1}$ = 1+$S_i$, $S_{i+1}$ ≤ l($X_i$),
  or $X_i \neq X_{i+1}$, $S_i \geq s(X_i)$, $S_{i+1}$ = 1, ($X_i$, $X_{i+1}$) ∈ t
  $S_1$ = 1

...

A **scheduling problem** is a problem of allocating a known set of activities to available resources and time.

We will assume a unary resource now.

**Unary resource** allows allocation of at most one activity to the resource at any time.

fixed duration and time window for each activity is given

**How to model such a problem?**

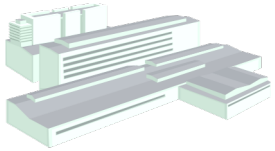
- **variable start(A)** describes the start time of activity A
- **constraints** ensure that no activities overlap in time

  **start(A) + p(A)** $\leq$ **start(B)** $\vee$ **start(B) + p(B)** $\leq$ **start(A)**

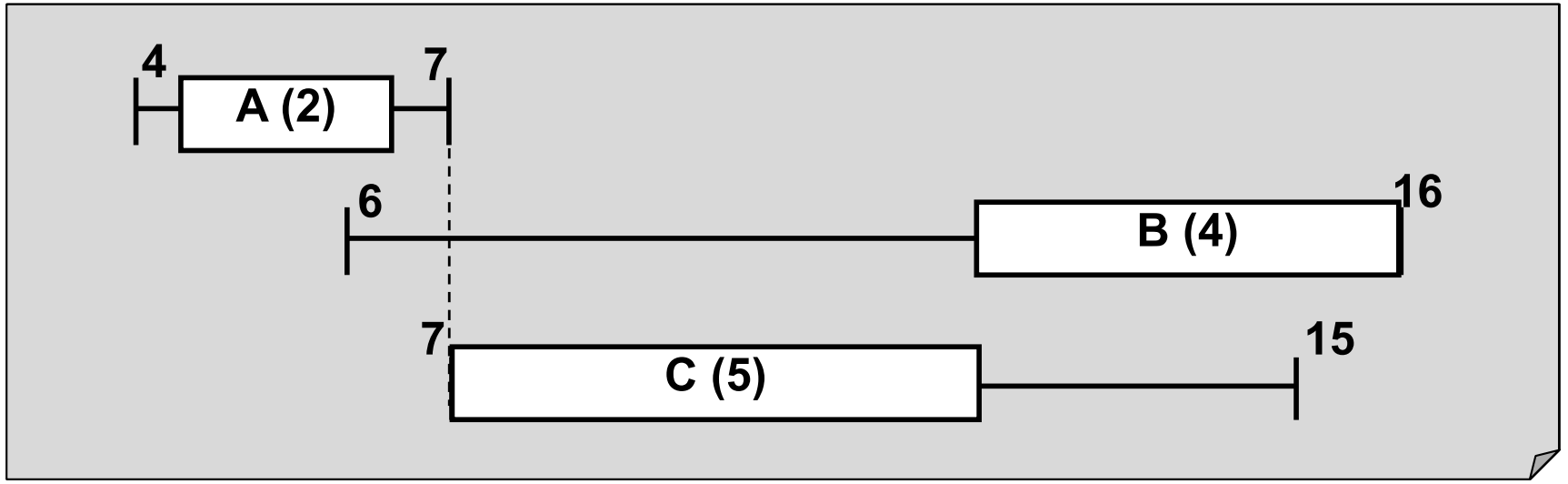
  (hence also the name **disjunctive resource**)
- or other relations such as **precedence** A « B

  **start(A) + p(A)** $\leq$ **start(B)**

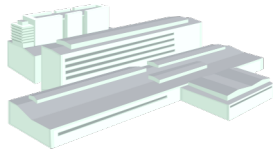
**Recall – disjunctions do almost no filtering!**



**What does happen when A is not processed first?**




**There is not enough time to process A, B,C and A must be first!**



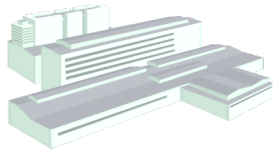
# `edge_finding`: *filtering rules*

## Filtering rules:

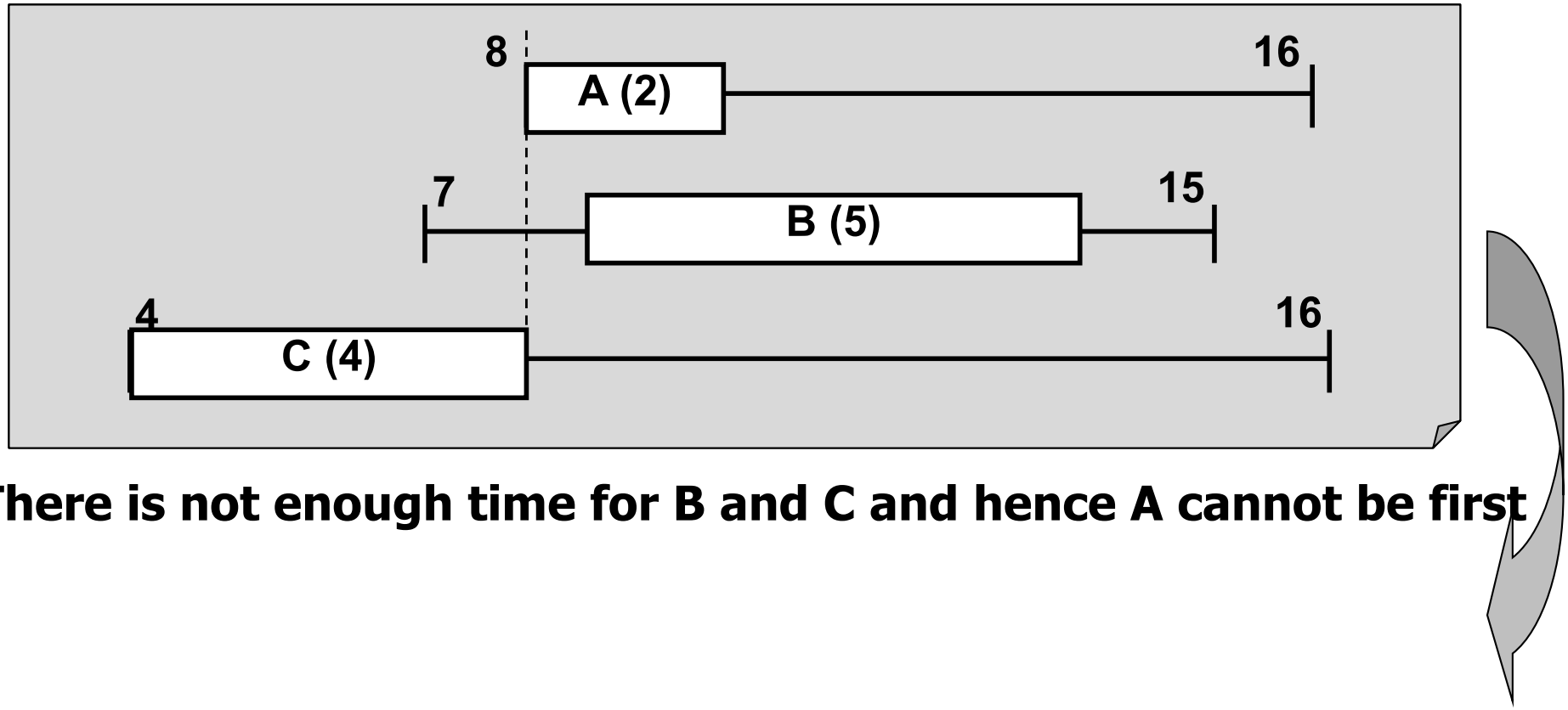
- **p(Ω ∪ {A}) > lct(Ω ∪ {A}) - est(Ω) ⇒ A«Ω**
- **p(Ω ∪ {A}) > lct(Ω) - est(Ω ∪ {A}) ⇒ Ω«A**
- **A«Ω ⇒ end(A) ≤ min{ lct(Ω') - p(Ω') | Ω'⊆Ω }**
- **Ω«A ⇒ start(A) ≥ max{ est(Ω') + p(Ω') | Ω'⊆Ω }**

## In practice:

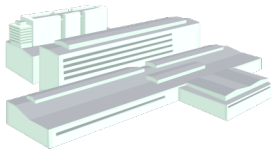
- we need to assume $n.2^n$ pairs (A,Ω) (to many!)
- Instead of sets Ω we can use **task intervals** [X,Y]
  {C | est(X) ≤ est(C) ∧ lct(C) ≤ lct(Y)}
  - ↳ time complexity $O(n^3)$, a frequently used incremental algorithm
- there are also algorithms with time complexity $O(n^2)$ and O(n.log n)



**What does happen if A is processed first?**




**There is not enough time for B and C and hence A cannot be first**



# `not_first`: *filtering rules*

## Rules for `not_first`:

$\mathbf{p(\Omega \cup \{A\}) > lct(\Omega) - est(A) \Rightarrow \neg\, A \ll \Omega}$

$\mathbf{\neg\, A \ll \Omega \Rightarrow start(A) \geq \min\{\, ect(B) \mid B \in \Omega \,\}}$

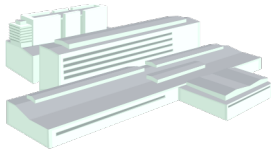
## Rules for `not_last`:

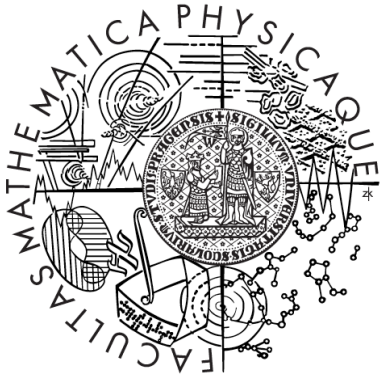
$\mathbf{p(\Omega \cup \{A\}) > lct(A) - est(\Omega) \Rightarrow \neg\, \Omega \ll A}$

$\mathbf{\neg\, \Omega \ll A \Rightarrow end(A) \leq \max\{\, lst(B) \mid B \in \Omega \,\}}$

## In practice:

- can be implemented with time complexity $O(n^2)$ and space complexity $O(n)$








Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz