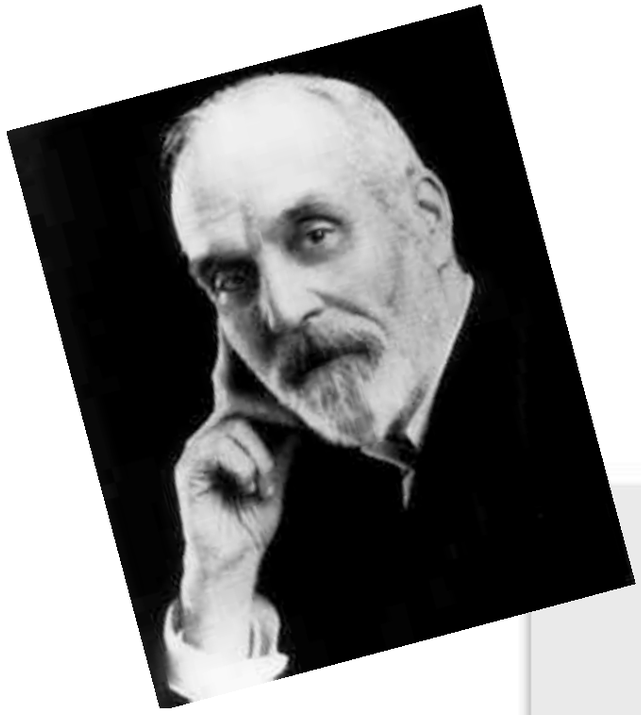
Wrapping up

CP course, lecture 11

Constraint programming
is a problem-solving technique
for combinatorial problems
that works by incorporating constraints
into a programming environment.

(after Apt 2003)



```
  S E N D
+ M O R E
---------
M O N E Y
```

Send More Money

CSPs

- Variables
- Valuations and assignments
- Constraint: set of valuations
- CSP: Variables + Constraints

Solving CSPs

$$\frac{\langle X \cup \{x : \varnothing\} \; ; \; \mathfrak{C} \rangle}{\textbf{fail}}$$

$$\frac{\langle X \cup \{x : D\} \; ; \; \mathfrak{C} \rangle \qquad |D| > 1 \qquad D = D_1 \uplus \cdots \uplus D_k}{\langle X \cup \{x : D_1\} \; ; \; \mathfrak{C} \rangle \mid \cdots \mid \langle X \cup \{x : D_k\} \; ; \; \mathfrak{C} \rangle}$$

$$\frac{\langle X \cup \{x : D\} \; ; \; \mathfrak{C} \cup \{C\} \rangle \qquad d \in D \qquad \mathsf{sat}(C) \cap \{\, \alpha \in \mathsf{ass}(X) \mid \alpha x = d \,\} = \varnothing}{\langle X \cup \{x : D - \{d\}\} \; ; \; \mathfrak{C} \cup \{C\} \rangle}$$

Propagators

$$\frac{\langle X \cup \{x : D\} \,;\, \mathfrak{C} \cup \{C\}\rangle \qquad d \in D \qquad \mathsf{sat}(C) \cap \{\, \alpha \in \mathsf{ass}(X) \mid \alpha x = d \,\} = \varnothing}{\langle X \cup \{x : D - \{d\}\} \,;\, \mathfrak{C} \cup \{C\}\rangle}$$

- Functions S→S (mapping stores to stores)
- Properties:
  - contracting $(p(s) \leq s)$ Domain Reduction
  - correct (maintain solutions)
  - checking (decisive for assignments)

Test

That's not enough!

- Assume a propagator

  p(s) = **if** s(x) = {1,2,3} **then** {x:{1}}

  else s

  and

  s_1={x:{1,2,3}} s_2={x:{1,2}}

- Then $s_2 \leq s_1$ but $p(s_1) \leq p(s_2)$

- Propagators must be monotonic:

  $s_1 \leq s_1 \Rightarrow p(s_1) \leq p(s_2)$

How much can we propagate?

- Idea: Propagate as much as possible, only then resort to branching
- This means:

Compute the largest simultaneous fixpoint of all propagators!

(exists + is unique)

Linear equations

- Propagator for

$$\sum a_i x_i = c$$

- How can bounds information be propagated efficiently?
- Example:

$$ax + by = c$$

All-distinct

- Naive:
  - check that no two determined variables have the same value
    - remove values of determined variables from domains of undetermined variables
- Advantage: simple implementation, avoid $O(n^2)$ propagators
- Disadvantage: not very strong

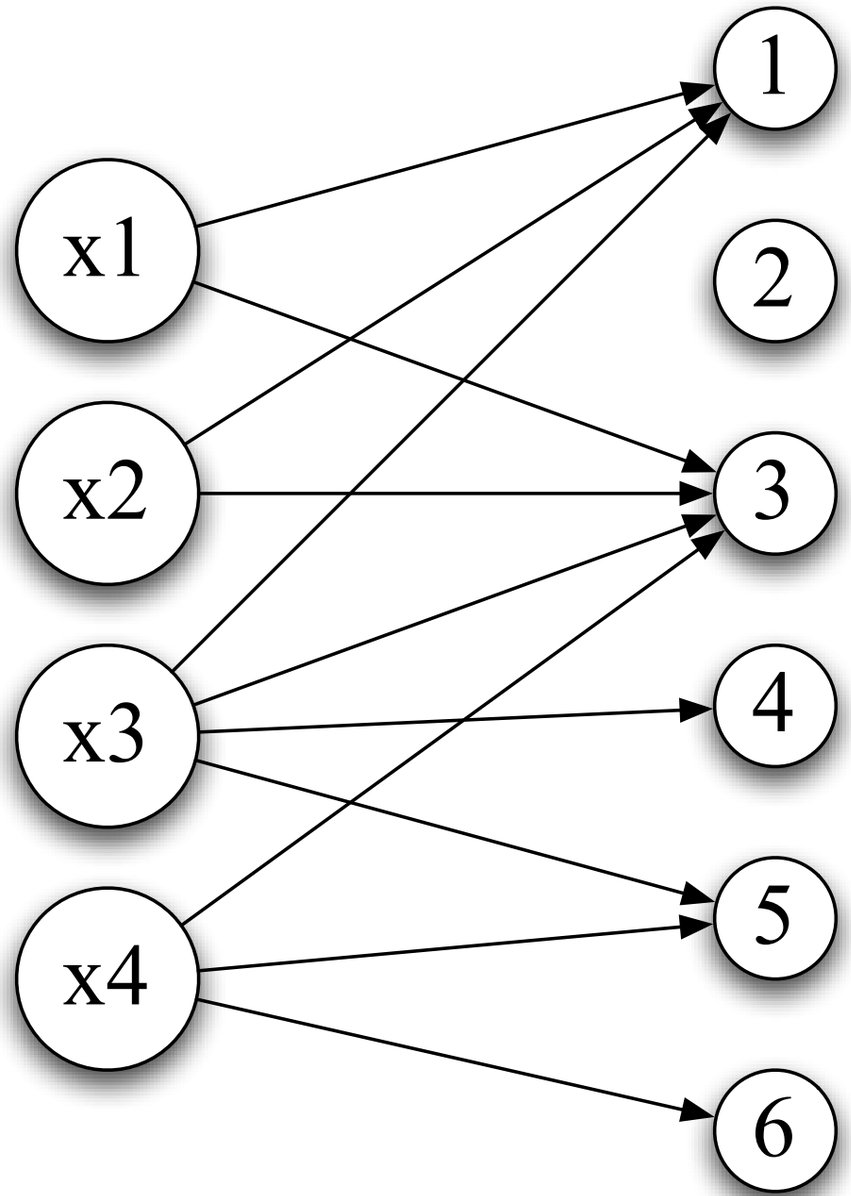
Variable-value Graph

$x1 \in \{1,3\}$

$x2 \in \{1,3\}$

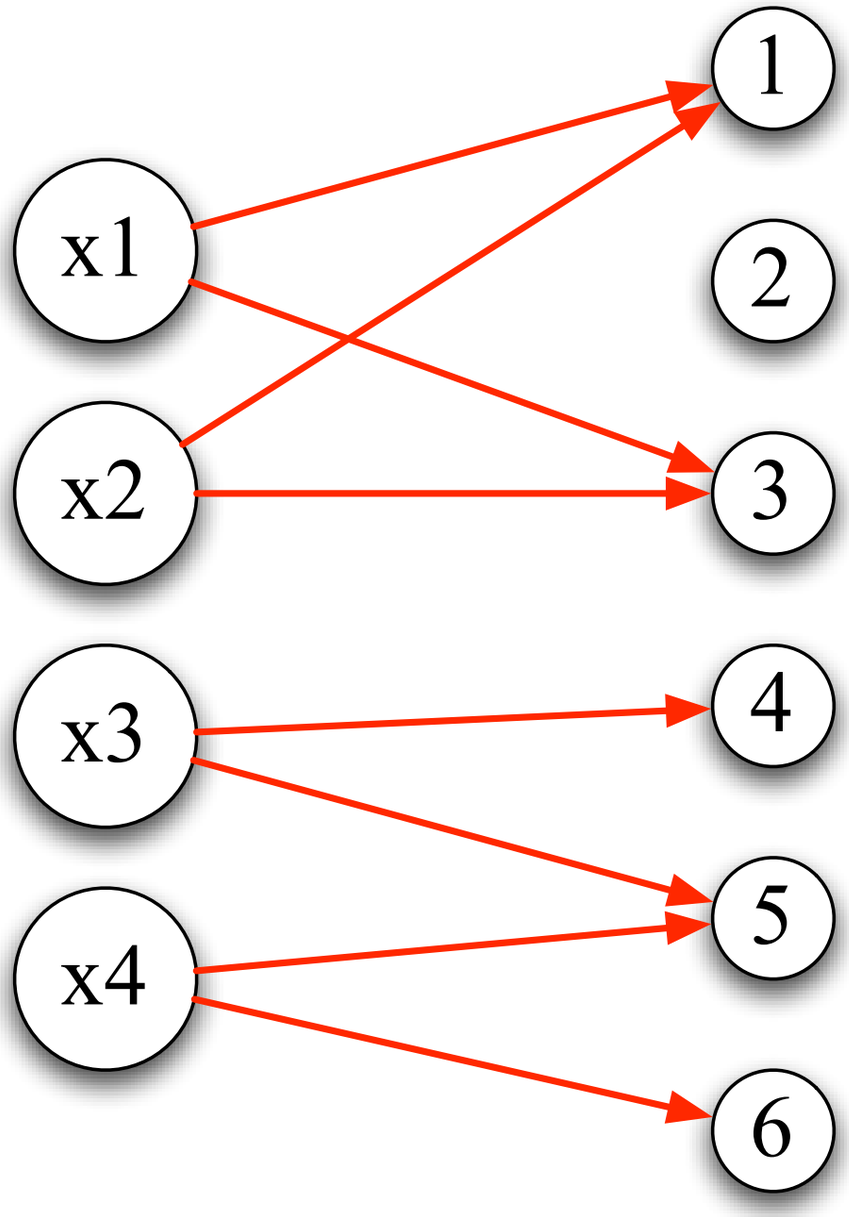
$x3 \in \{1,3,4,5\}$

$x4 \in \{3,5,6\}$



Matching

- Compute union of all maximal matchings
- Delete unmatched edges



Two questions

- *How to branch?*
  - branching strategy (naive, first-fail, …)
    - determines the *shape* of the search tree
- *How to make the choice operation deterministic?*
  - search strategy (depth-first, b & b, …)
  - determines the *order* in which
    the nodes of the search tree are visited

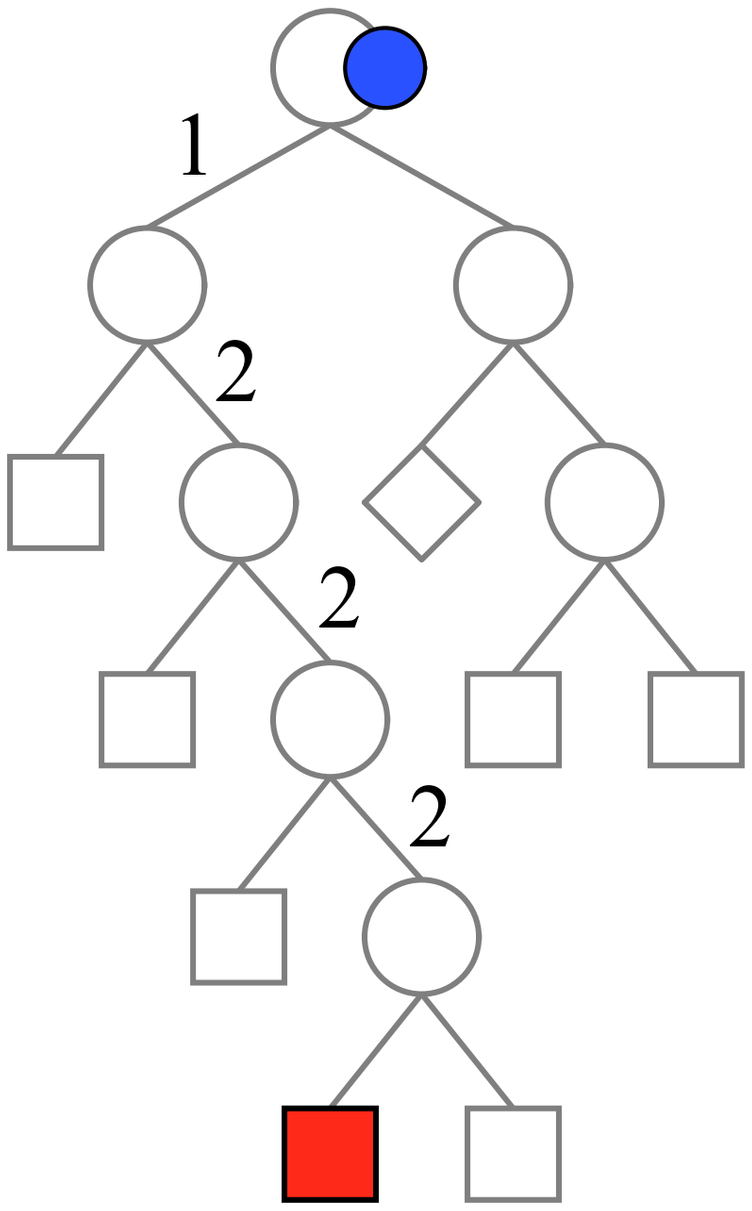
Backtracking strategies

- *copying:*
  backup the state of the system
  before making a choice

- *trailing:*
  remember an undo action for the choice

- *recomputation:*
  recompute the state of the system
  before the choice was made

Recomputation




```
fun recompute s nil = clone s
  | recompute s (i::ir) =
    let
        val s’ = recompute s ir
    in
       commit(s’, i)
       s’
    end
```

Recomputation

- fundamental idea:

  trade space for time

- full recomputation:

  pro: no backup copies, constant space

  con: time overhead

- can we do better?

Finite-set variables

- let Δ be a finite interval of integers
- finite domain variables take values in Δ
- finite set variables take values in $P(\Delta)$
- *domain*: fixed subset of Δ

Non-basic constraints

just as with FD

- express non-basic constraints in terms of basic constraints, using sets of inference rules
- monotonicity: more specific premises yield more specific conclusions
- express inferences in terms of the currently entailed lower and upper bounds

$$\lfloor S \rfloor = \cup\{\, D \mid D \subseteq S \,\}$$
$$\lceil S \rceil = \cap\{\, D \mid S \subseteq D \,\}$$

Union and intersection

$$\boldsymbol{S_1 \cup S_2 = S_3} \quad \equiv \quad S_3 \subseteq \lceil S_1 \rceil \cup \lceil S_2 \rceil \ \wedge \ \lfloor S_1 \rfloor \cup \lfloor S_2 \rfloor \subseteq S_3$$

$$\boldsymbol{S_1 \cap S_2 = S_3} \quad \equiv \quad \lfloor S_1 \rfloor \cap \lfloor S_2 \rfloor \subseteq S_3 \ \wedge \ S_3 \subseteq \lceil S_1 \rceil \cap \lceil S_2 \rceil$$

Binary Relations

- define the notion of the ‘relational image’

$$Rx = \{\, y \in \mathcal{C} \mid Rxy \,\}$$

- understand binary relations as total functions from the carrier to subsets of the carrier

$$f_R = \{\, x \mapsto Rx \mid x \in \mathcal{C} \,\}$$

- represent these functions as vectors on finite set variables

Selection constraints

- generalisation of binary set operations
- participating elements are variable, too
- example: union with selection

$$S = \bigcup_{i \in S'} S_i$$

- propagation in all directions

Composition

$$R_1 \circ R_2 \quad = \quad \{\, (x, z) \in \mathcal{C}^2 \mid \exists y \in \mathcal{C} \colon R_1 x y \wedge R_2 y z \,\}$$

$$\boldsymbol{R_1 \circ R_2 = R_3} \quad \equiv \quad R_3 = \langle \cup R_2[R_1[1]], \ldots, \cup R_2[R_1[n]] \rangle$$

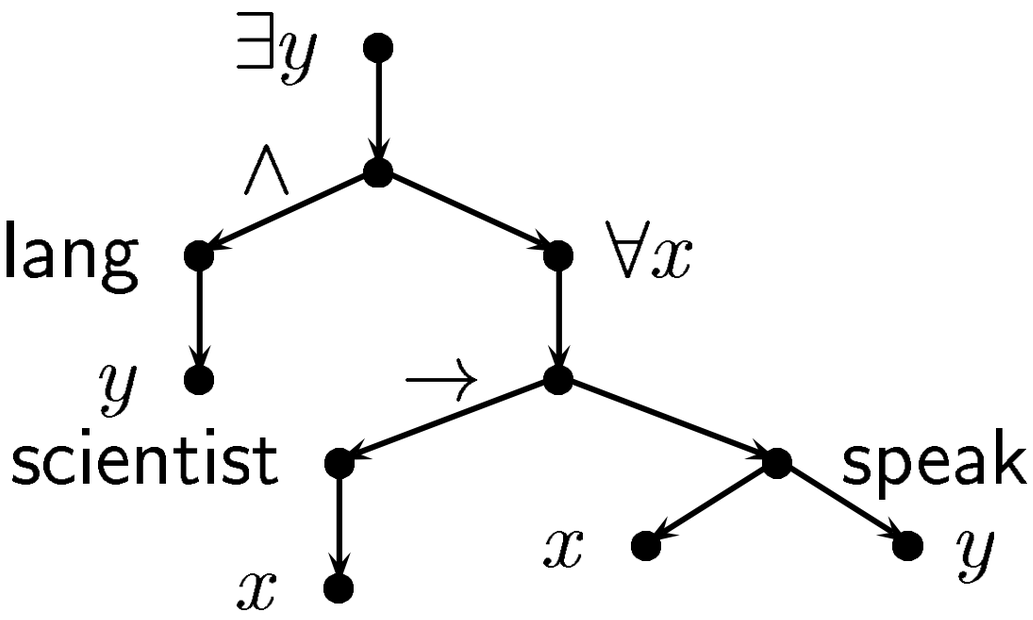
Rooted tree constraint

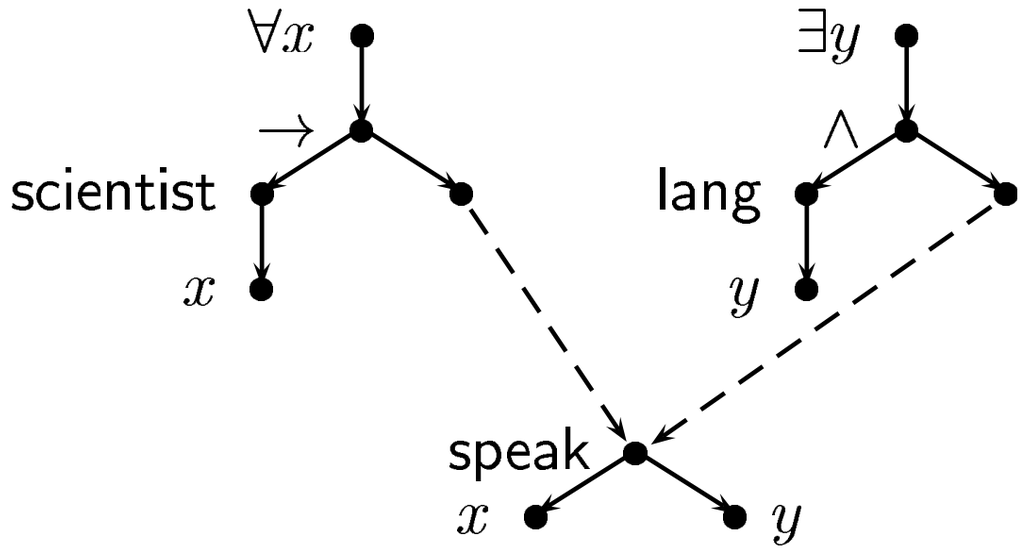
$$\mathbf{rootedTree}(v_1, \ldots, v_n, \mathcal{R})$$

$$\begin{aligned}
&\implies \mathbf{succ} = \mathbf{pred}^{-1} \\
&\implies \mathbf{succ}_* = \mathrm{Id} \cup \mathbf{succ}_+ \\
&\implies \mathbf{succ}_+ = \mathbf{succ} \circ \mathbf{succ}_* \\
&\implies |\mathcal{R}| = 1 \\
&\implies \mathcal{V} = \mathcal{R} \uplus \mathbf{succ}(v_1) \uplus \ldots \mathbf{succ}(v_n) \\
v \in \mathcal{V} &\implies v \in \mathcal{R} \iff \mathbf{pred}(x) = \emptyset
\end{aligned}$$

Dominance constraints

T_2:

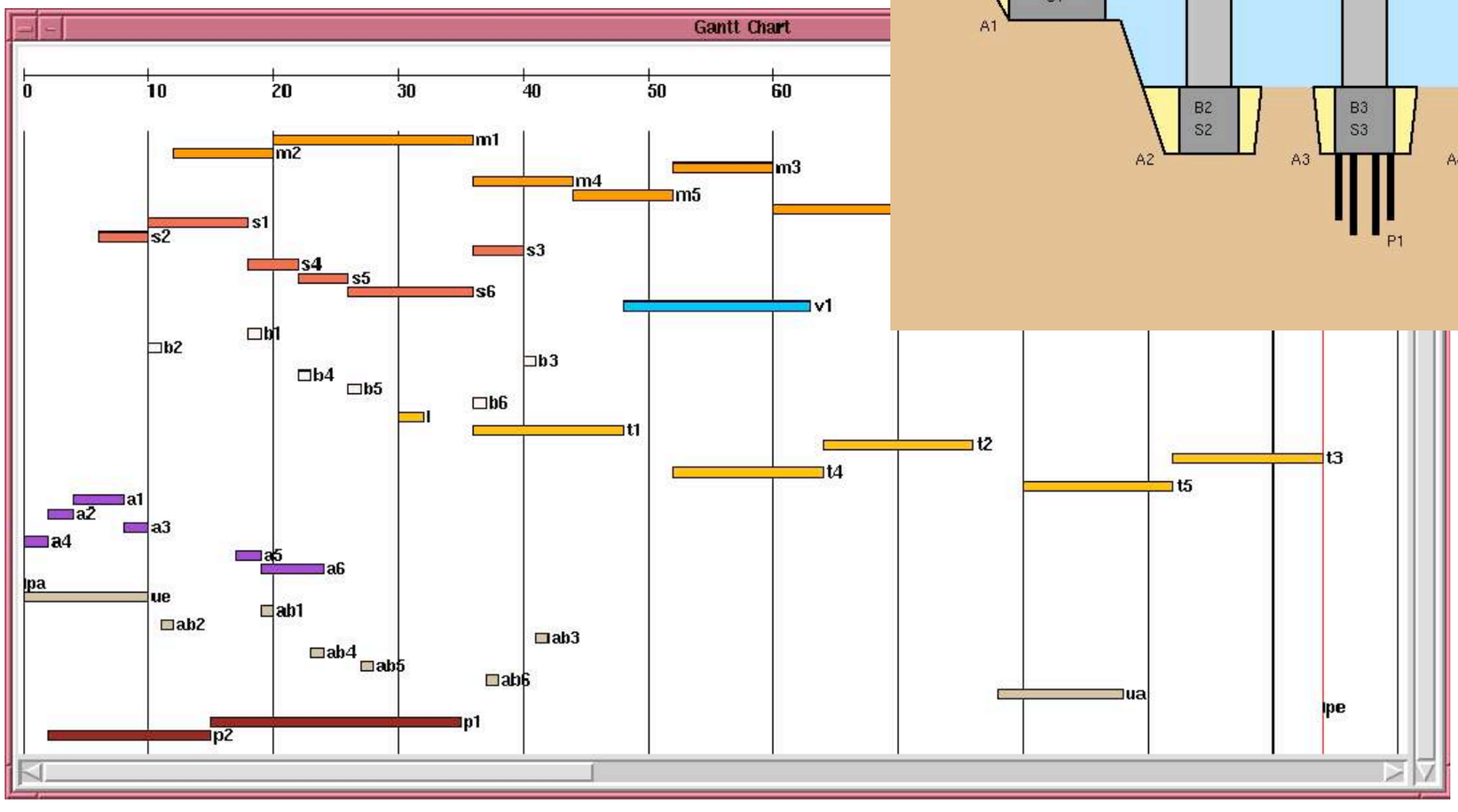


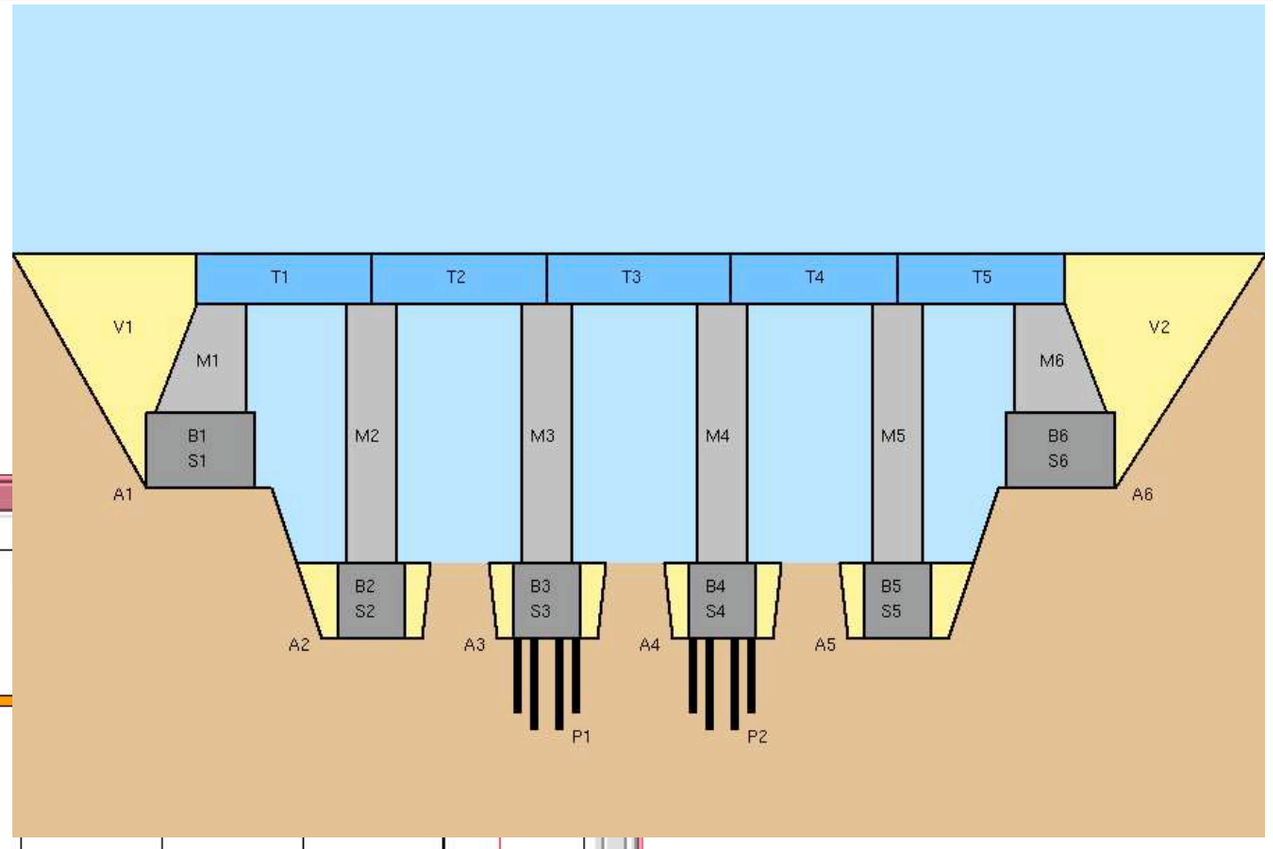


Scheduling

- Tasks *a* (aka activities)
  - duration *dur(a)*
  - resource *res(a)*
- Precedence constraints
  - determine order among two tasks
- Resource constraints
  - e.g. at most one task per resource

Building a Bridge





Model in CP

- Variable for start-time of task *a*
- Precedence constraints:

$$a \textbf{ before } b$$

- Resource constraints:

$$a \textbf{ before } b \ \vee \ b \textbf{ before } a$$

similar to temporal relations

Think global

- Model employs local view:
  - constraints on pairs of tasks
    - $O(n^2)$ propagators for n tasks
- Global view:
  - order all tasks on one resource
  - employ smart global propagator

Ordering Tasks: Serialization

- Consider all tasks on one resource
- Deduce their order as much as possible
- Propagators:
  - Timetabling: look at free/used time slots
  - Edge-finding: which task first/last?
  - Not-first / not-last

Potential Projects

- Are you interested in a Fopra, Bachelor's, Master's, or Diploma thesis?
- Projects:

  Gecode (C++), Gecode/Alice, Alice, CoLi

Grading

Grading scheme

- 4 graded assignments = 120 points
- Exam = 180 points
- Total: 300 points
- You will pass the course with 150 points

Exam

When, where, how...

- Thursday, July 14
- 14:00 (that's 2pm, sine tempore!)
- here! (lecture room III)
- 90 min, 180 points
- just bring a pen and your student id

and... what?

I already told you...

Thank you!