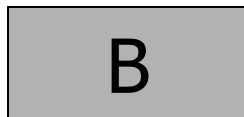
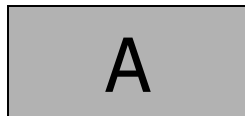


Reasoning with Conditional Time-intervals

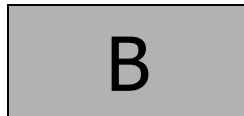
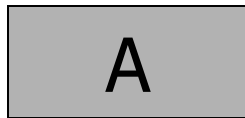
P. Laborie and J. Rogerie
[plaborie,jrogerie]@ilog.fr

- Motivations
- Model
- Constraint Propagation
- Extensions & Conclusion

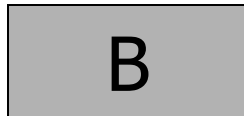
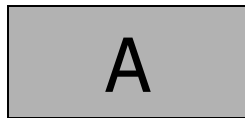
- Optional time-intervals in scheduling:
 - Some activities can be left unperformed



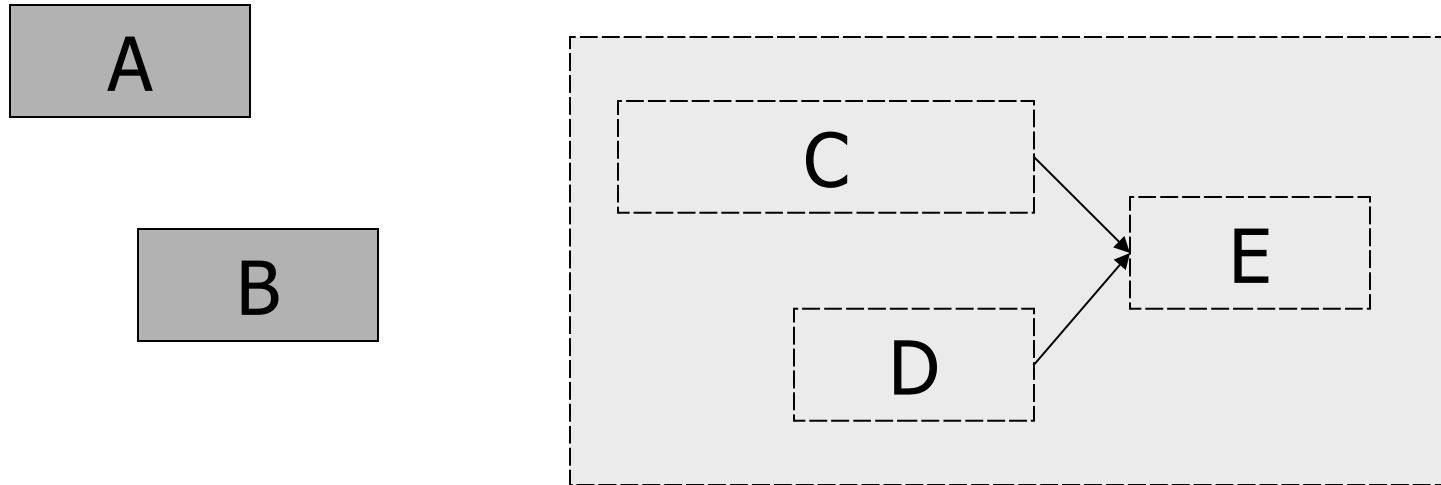
- Optional time-intervals in scheduling:
 - Some activities can be left unperformed



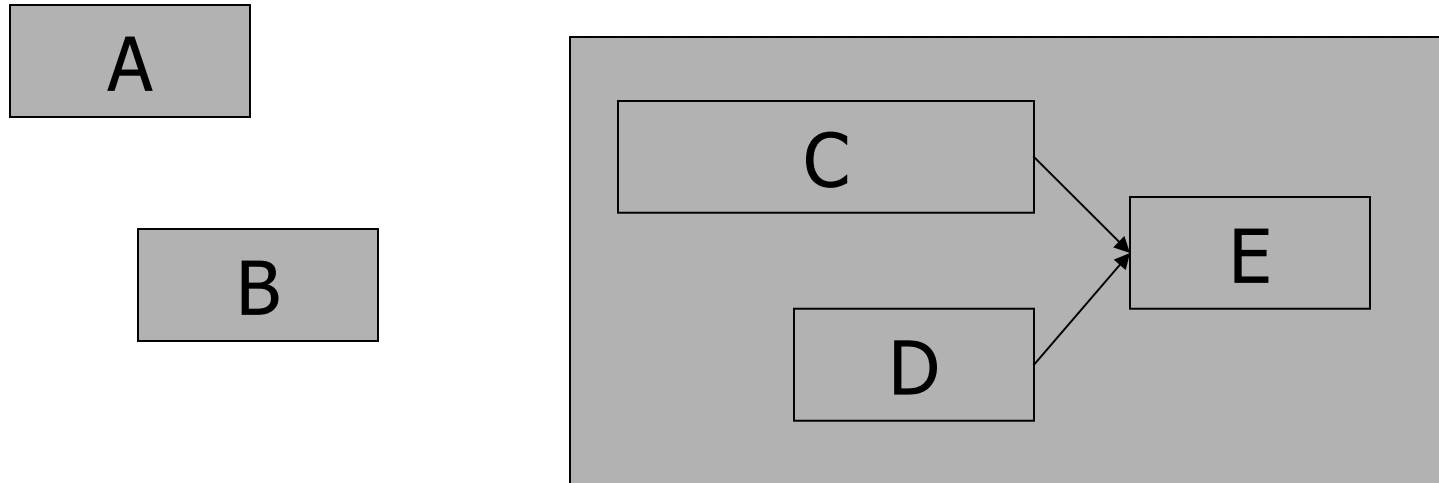
- Optional time-intervals in scheduling:
 - Some activities can be left unperformed



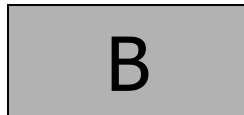
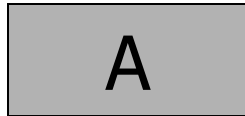
- Optional time-intervals in scheduling:
 - Some parts of the schedule can be left unperformed



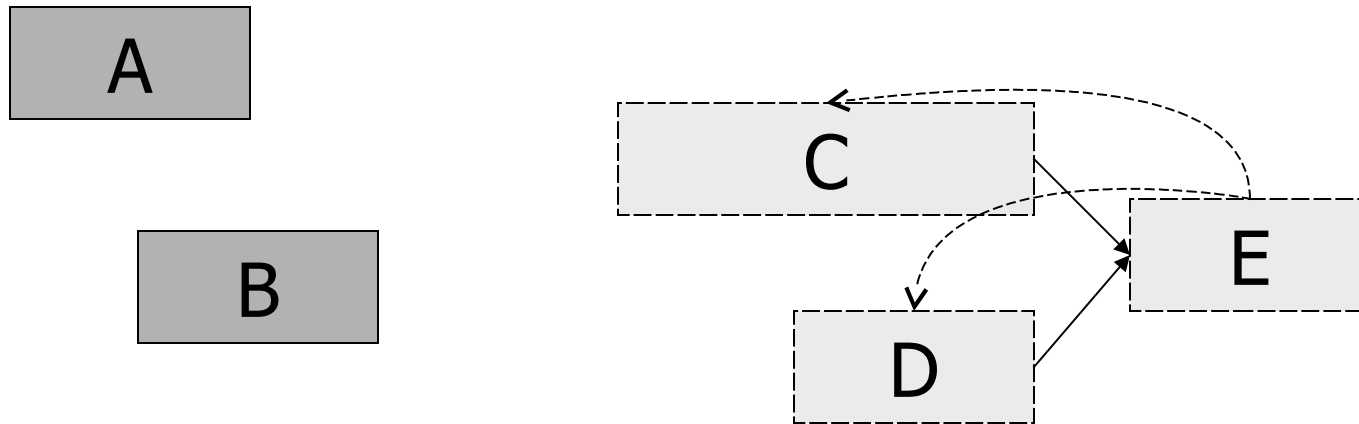
- Optional time-intervals in scheduling:
 - Some parts of the schedule can be left unperformed



- Optional time-intervals in scheduling:
 - Some parts of the schedule can be left unperformed

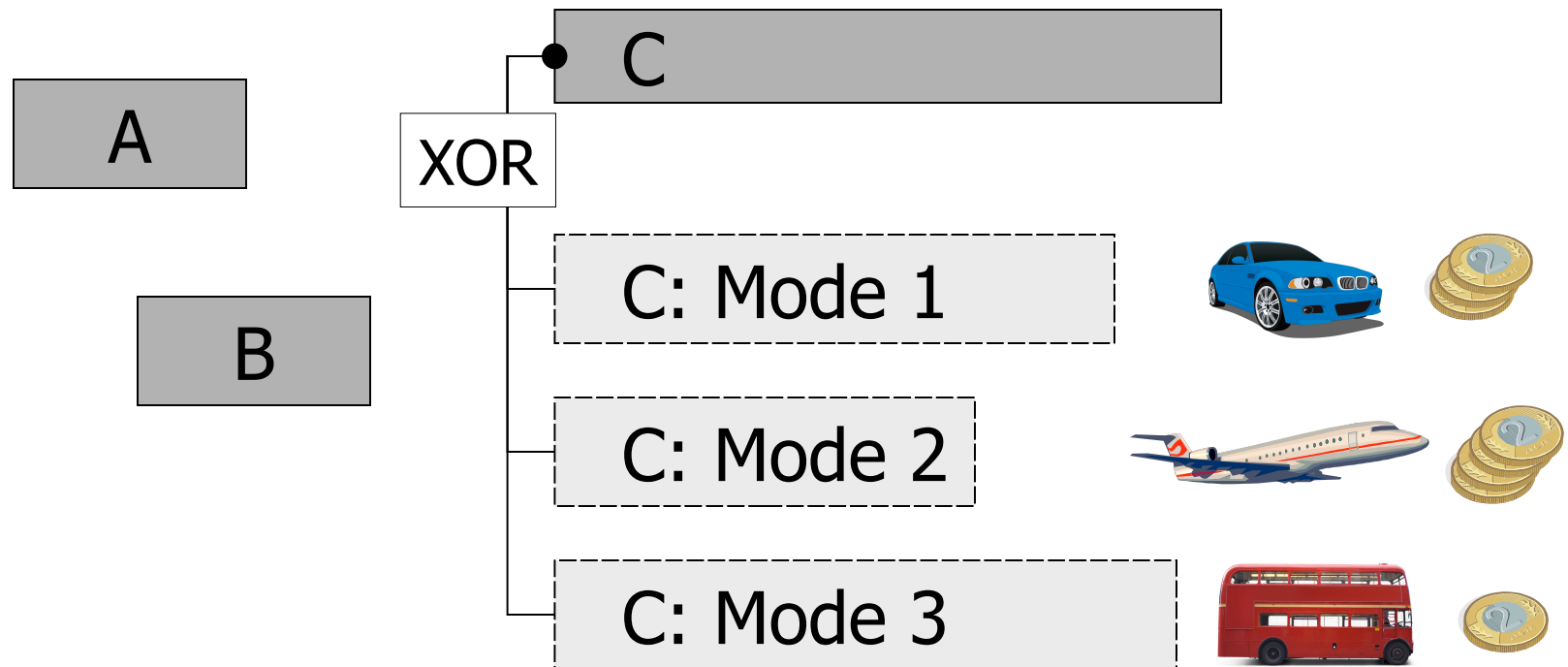


- Optional time-intervals in scheduling:
 - Some parts of the schedule can be left unperformed

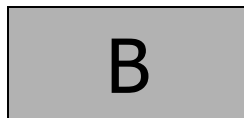
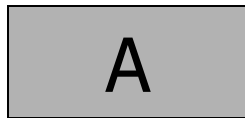


- Dependencies: $\text{exec}(E) \Rightarrow \text{exec}(C) \wedge \text{exec}(D)$

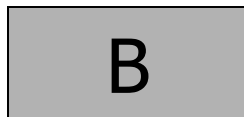
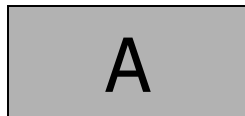
- Optional time-intervals in scheduling:
 - Alternative combination of resources



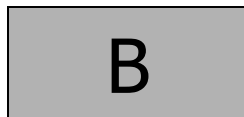
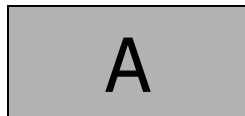
- Optional time-intervals in scheduling:
 - Alternative combination of resources



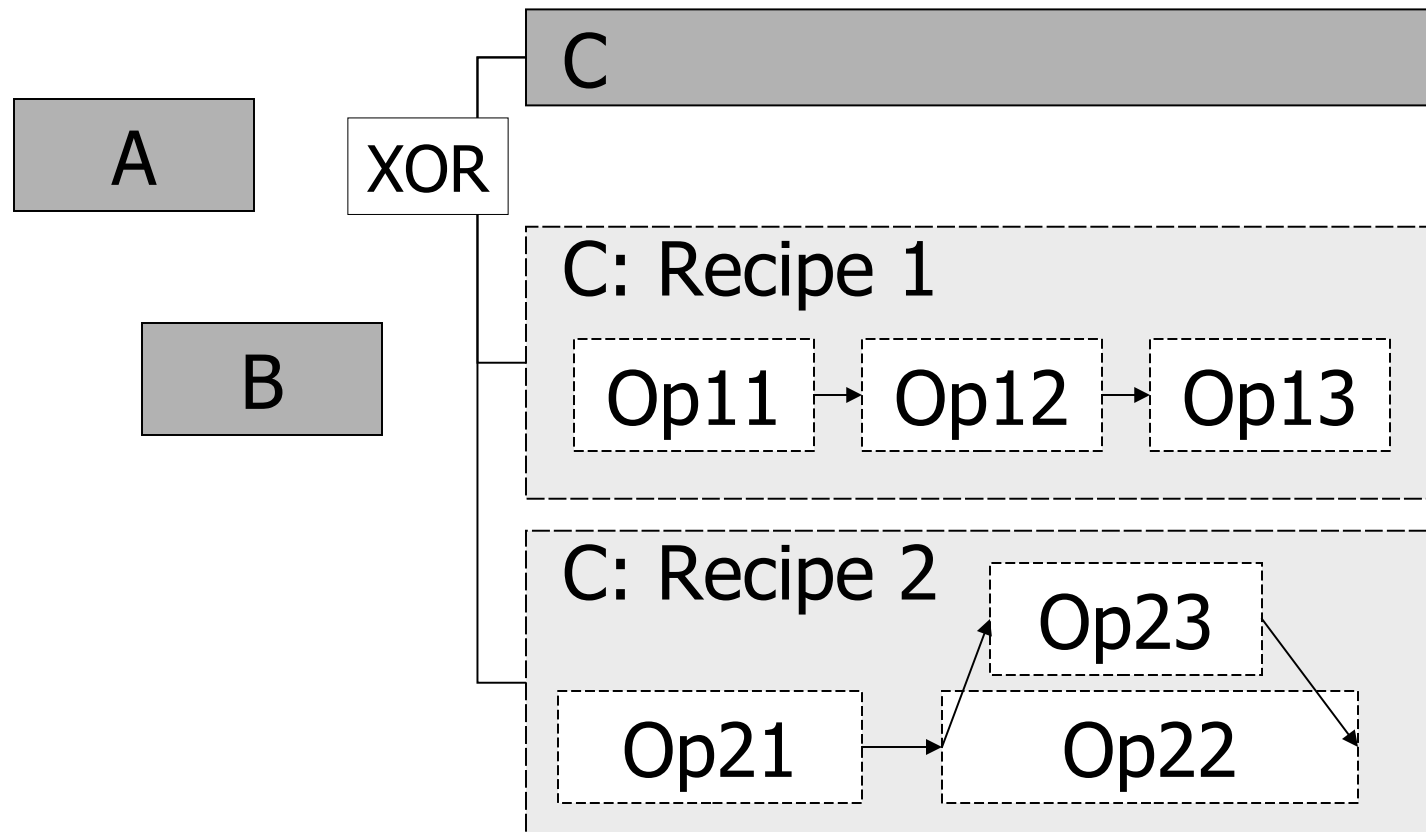
- Optional time-intervals in scheduling:
 - Alternative combination of resources



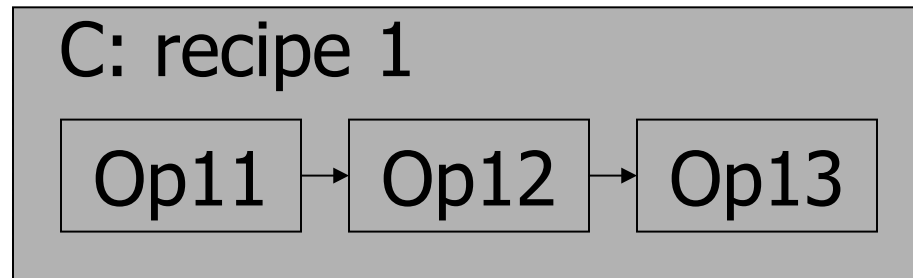
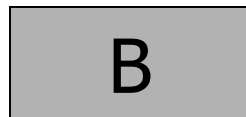
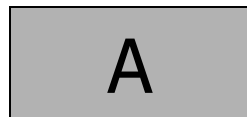
- Optional time-intervals in scheduling:
 - Alternative combination of resources



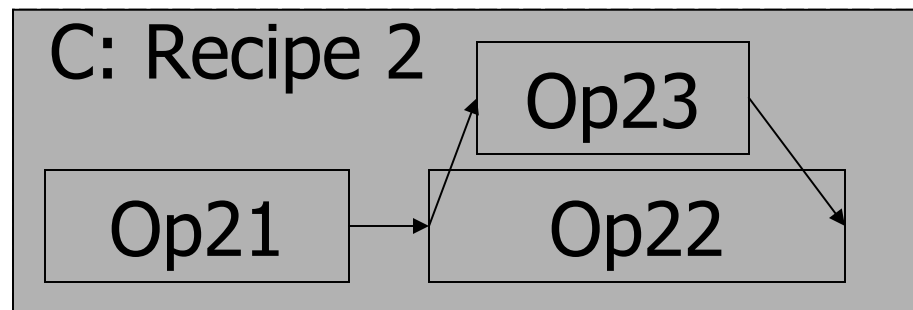
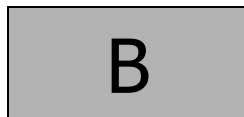
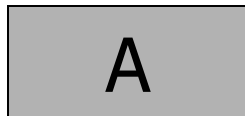
- Optional time-intervals in scheduling:
 - Alternative recipes (common in manufacturing)



- Optional time-intervals in scheduling:
 - Alternative recipes (common in manufacturing)



- Optional time-intervals in scheduling:
 - Alternative recipes (common in manufacturing)



- Objective: Make it easier to **model** and **solve** scheduling problems involving **optional** activities/tasks/processes/recipes/...
- Constraint Optimization framework
- Basic ingredients:
 - Optional time-intervals **variables** (a, b, c, \dots)
 - Logical **constraints** ($\text{exec}(a) \Rightarrow \text{exec}(b)$)
 - Precedence **constraints** ($\text{endBeforeStart}(a, b)$)
 - Decomposition **constraints** ($\text{span}(a, \{b_1, \dots, b_n\})$)
 - Alternative **constraints** ($\text{alternative}(a, \{b_1, \dots, b_n\})$)

MODEL

PROBLEM SOLVING

Basic constraints:
logical, precedence,
span, alternative

**Time-interval
variables**

**Basic constraint
propagation**

What we won't talk about here



Changing the rules of business

MODEL

Resource-related constraints:
sequencing, cumul,
states, calendars

Basic constraints:
logical, precedence,
span, alternative

**Time-interval
variables**

PROBLEM SOLVING

**Resource-related
constraint
propagation**

**Basic constraint
propagation**

**Search &
optimization
techniques**

- Extension of classical CSPs
- **A new type of first class citizen decision variable: Time-interval variable**
- Domain of values for a time-interval variable a :

$$\text{Dom}(a) \subseteq \{\perp\} \cup \{ [s,e) \mid s,e \in \mathbb{Z}, s \leq e \}$$

Non executed

Interval of integers

- Domain of values for a time-interval variable a :

$$\text{Dom}(a) \subseteq \{\perp\} \cup \{ [s,e) \mid s,e \in \mathbb{Z}, s \leq e \}$$

- Notations: Let a be a **fixed** time-interval variable

- If $a = \{[s,e)\}$ (a is executed), we denote:

$$\mathbf{x(a)=1, s(a)=s, e(a)=e, d(a)=e-s}$$

- If $a = \{\perp\}$ (a is non-executed), we denote:

$$\mathbf{x(a)=0}$$
 (in this case, $s(a), e(a), d(a)$ are meaningless)

- Execution unary constraint $\text{exec}(a)$ means that a is executed ($x(a)=1$)

- 2-SAT clauses over execution constraints:

$$[\neg]\text{exec}(a) \vee [\neg]\text{exec}(b)$$

- Expressivity:

- Same execution status:

$$\neg\text{exec}(a) \vee \text{exec}(b), \text{exec}(a) \vee \neg\text{exec}(b)$$

- Incompatibility: $\neg\text{exec}(a) \vee \neg\text{exec}(b)$

- Implication: $\neg\text{exec}(a) \vee \text{exec}(b)$

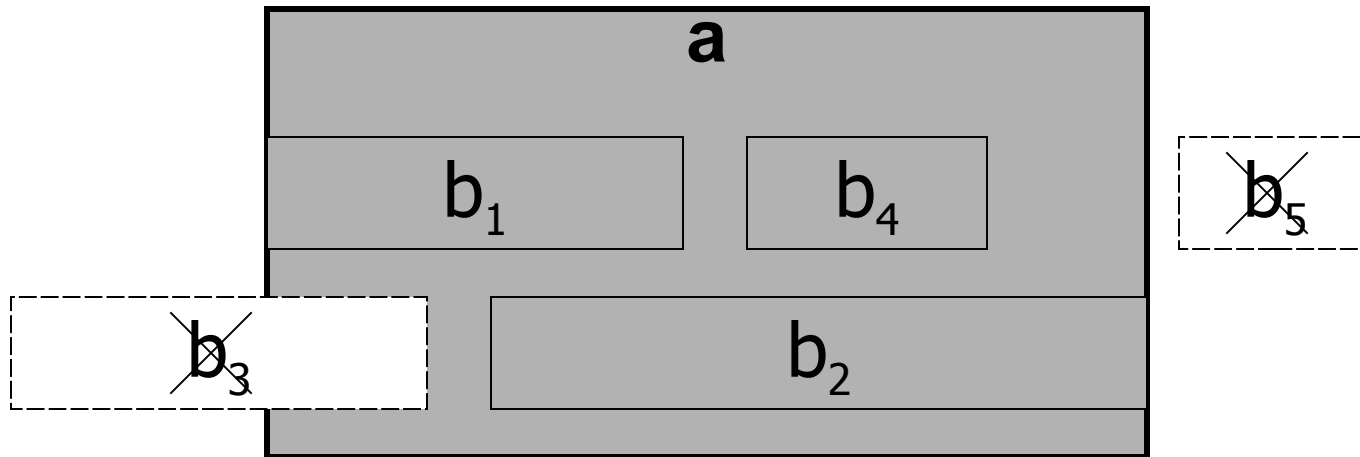
- Simple Precedence Constraints $t_i + z \leq t_j$ reified by execution statuses

- Example: `endBeforeStart(a,b,z)` means

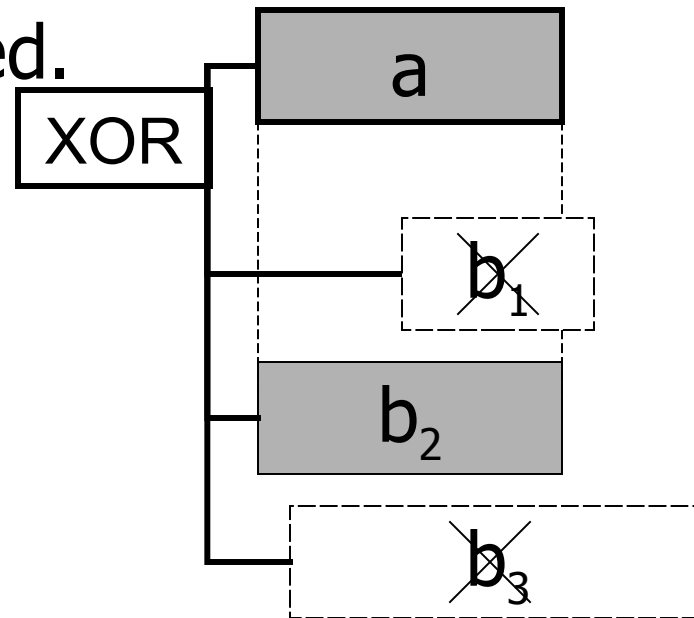
$$x(a) \wedge x(b) \Rightarrow e(a) + z \leq s(b)$$

- | | |
|--------------------------------|-----------------------------|
| <code>startBeforeStart,</code> | <code>startBeforeEnd</code> |
| <code>endBeforeStart,</code> | <code>endBeforeEnd</code> |
| <code>startAtStart,</code> | <code>startAtEnd</code> |
| <code>endAtStart,</code> | <code>endAtEnd</code> |

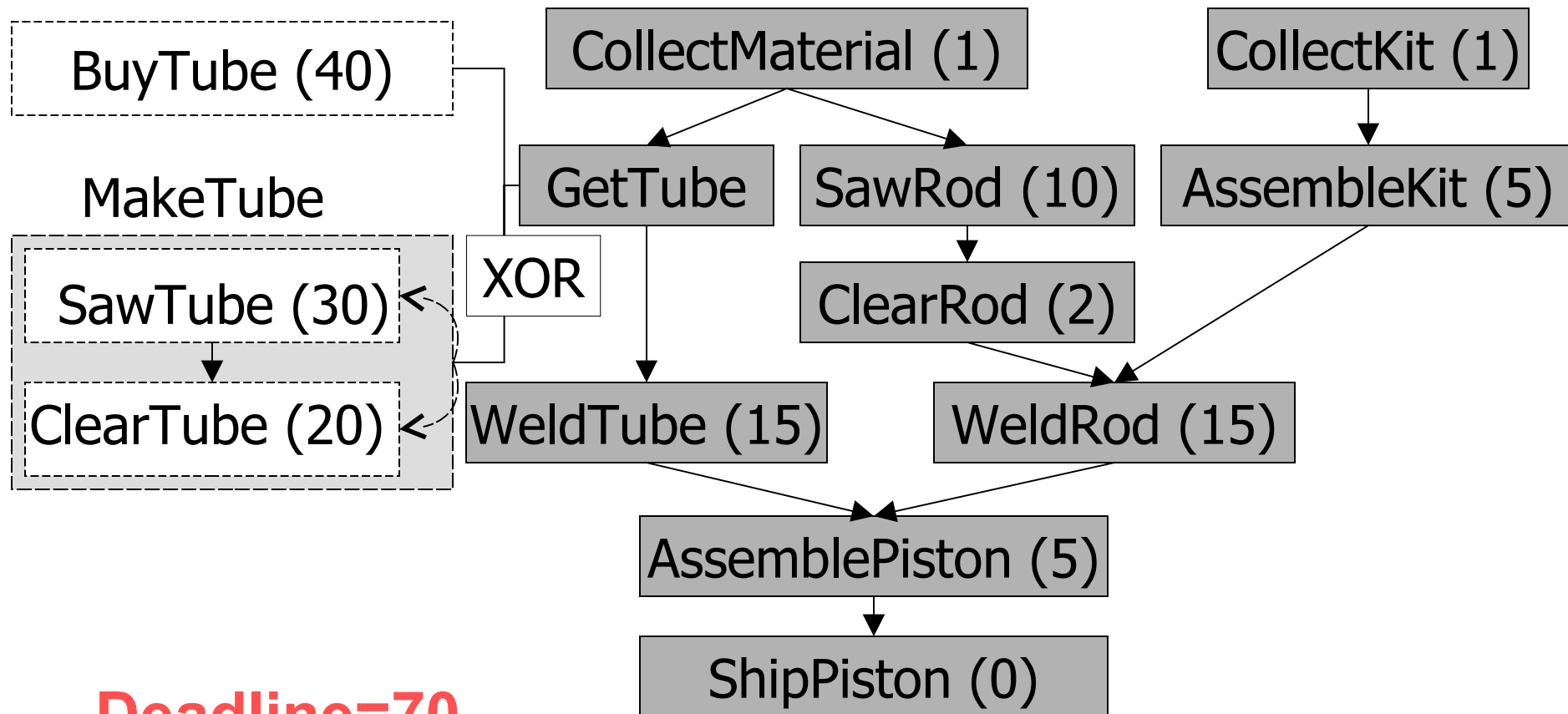
- Span constraint $\text{span}(a, \{b_1, \dots, b_n\})$ means that if a is executed, it spans all executed intervals from $\{b_1, \dots, b_n\}$. Interval a is not executed iff none of intervals $\{b_1, \dots, b_n\}$ is executed.



- Alternative constraint $\text{alternative}(a, \{b_1, \dots, b_n\})$ means that if a is executed, then exactly one of the $\{b_1, \dots, b_n\}$ is executed and synchronized with a . Interval a is not executed iff none of intervals $\{b_1, \dots, b_n\}$ is executed.



- Inspired from [Barták&Čepek 2007]



Deadline=70

Model: Simple example (ILOG OPL Studio)



Changing the rules of business

ILOG OPL Development Studio IDE

File Edit Navigate Run Window Help

OPL.mod

```
using CP;

dvar interval CollectMaterial size 1;
dvar interval CollectKit      size 1;
dvar interval GetTube;
dvar interval WeldTube        size 15;
dvar interval SawRod          size 10;
dvar interval ClearRod        size 2;
dvar interval AssembleKit     size 5;
dvar interval WeldRod         size 15;
dvar interval AssemblePiston size 5;
dvar interval ShipPiston      in 0..70 size 0;
dvar interval BuyTube         optional size 40;
dvar interval SawTube         optional size 30;
dvar interval ClearTube       optional size 20;
dvar interval MakeTube        optional;

constraints {
    span(MakeTube, [SawTube, ClearTube]);
    presenceOf(MakeTube) => (presenceOf(SawTube) && presenceOf(ClearTube));
    alternative(GetTube, [BuyTube, MakeTube]);
    endBeforeStart(CollectMaterial, GetTube);
    endBeforeStart(CollectMaterial, SawRod);
    endBeforeStart(CollectKit, AssembleKit);
    endBeforeStart(SawTube, ClearTube);
    endBeforeStart(GetTube, WeldTube);
    endBeforeStart(SawRod, ClearRod);
    endBeforeStart(ClearRod, WeldRod);
    endBeforeStart(AssembleKit, WeldRod);
    endBeforeStart(WeldTube, AssemblePiston);
    endBeforeStart(WeldRod, AssemblePiston);
    endBeforeStart(AssemblePiston, ShipPiston);
}
```

Solution

Name	Value
Data	
Decision variables (14)	
AssembleKit	<1 1 6 5>
AssemblePiston	<1 56 61 5>
BuyTube	<1 1 41 40>
ClearRod	<1 11 13 2>
ClearTube	<0 0 0 0>
CollectKit	<1 0 1 1>
CollectMaterial	<1 0 1 1>
GetTube	<1 1 41 40>
MakeTube	<0 0 0 0>
SawRod	<1 1 11 10>
SawTube	<0 0 0 0>
ShipPiston	<1 61 61 0>
WeldRod	<1 13 28 15>
WeldTube	<1 41 56 15>
Decision expressions	
Constraints	
Post-processing data	

Property	Value
----------	-------

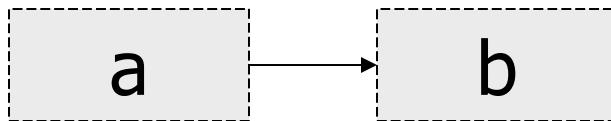
- Time interval variable domain representation:
tuple of ranges:
 - $[x_{\min}, x_{\max}] \subseteq [0, 1]$: current execution status
 - $[s_{\min}, s_{\max}] \subseteq \mathbb{Z}$: **conditional** domain of start time **would the time-interval be executed**
 - $[e_{\min}, e_{\max}] \subseteq \mathbb{Z}$: **conditional** domain of end time **would the time-interval be executed**
 - $[d_{\min}, d_{\max}] \subseteq \mathbb{Z}^+$: **conditional** domain of duration **would the time-interval be executed**

- Logical constraints are aggregated in an implication graph: all 2-SAT logical constraints $[\neg]\text{exec}(a) \vee [\neg]\text{exec}(b)$ are translated as implications $(\neg[\neg]\text{exec}(a) \Rightarrow [\neg]\text{exec}(b))$
- **Incremental transitive closure** of the implication graph allows detecting infeasibilities and querying in $O(1)$ whether $\text{exec}(a) \Rightarrow \text{exec}(b)$ for any (a,b)

- Precedence constraints are aggregated in a temporal network

- Precedence constraints are aggregated in a temporal network

- **Conditional reasoning:**

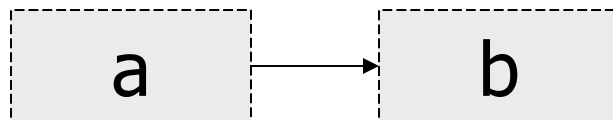


From logical network

$\text{exec}(a) \Rightarrow \text{exec}(b)$
 $\text{endBeforeStart}(a, b)$

- Precedence constraints are aggregated in a temporal network

- **Conditional reasoning:**



From logical network

$\text{exec}(a) \Rightarrow \text{exec}(b)$
 $\text{endBeforeStart}(a, b)$

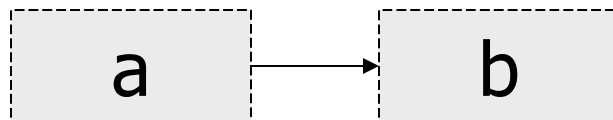
- Propagation on the conditional bounds of a (would a be executed) can assume that b will be executed too, thus:

$$e_{\max}(a) \leftarrow \min(e_{\max}(a), s_{\max}(b))$$

- Precedence constraints are aggregated in a temporal network

- Conditional reasoning:**

From logical network



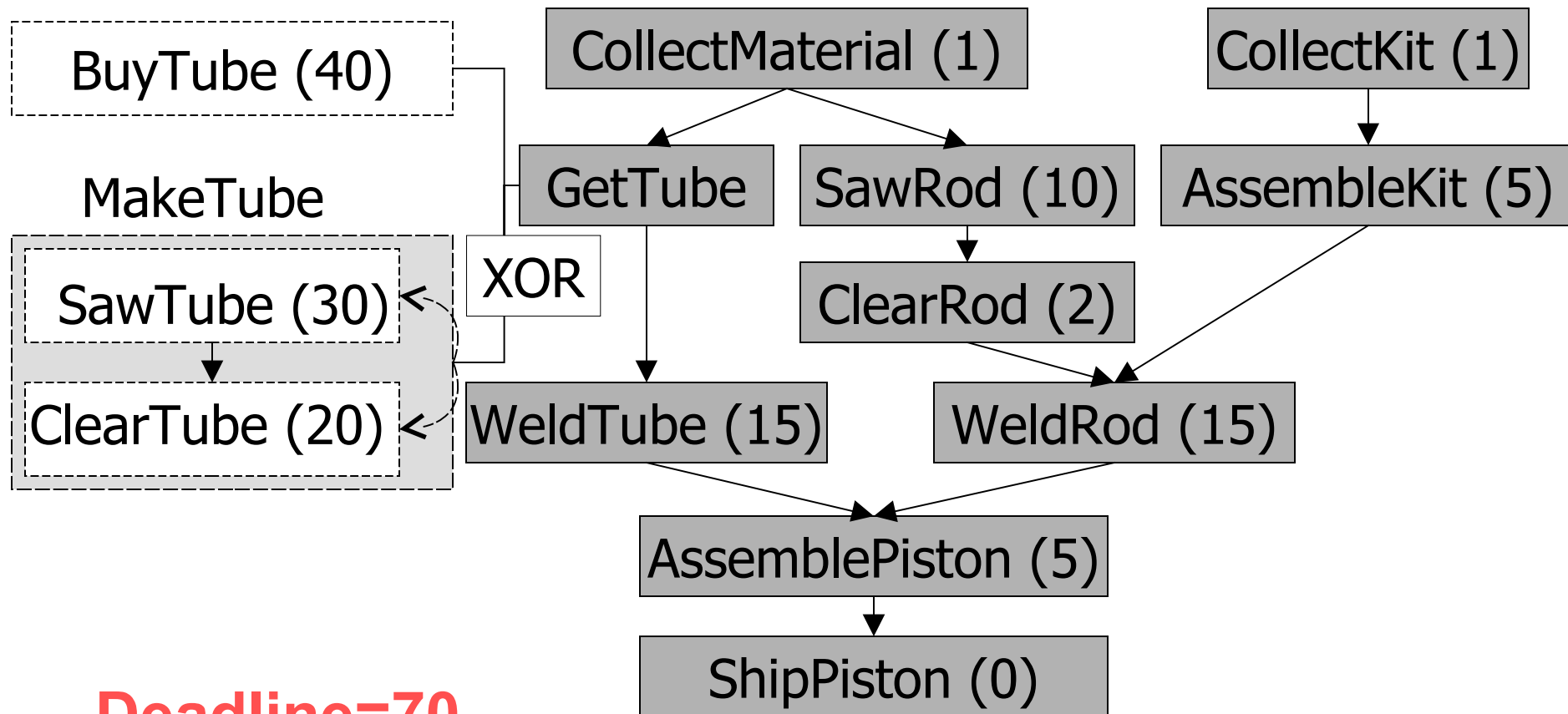
$\text{exec}(a) \Rightarrow \text{exec}(b)$
 $\text{endBeforeStart}(a, b)$

- Propagation on the conditional bounds of a (would a be executed) can assume that b will be executed too, thus:

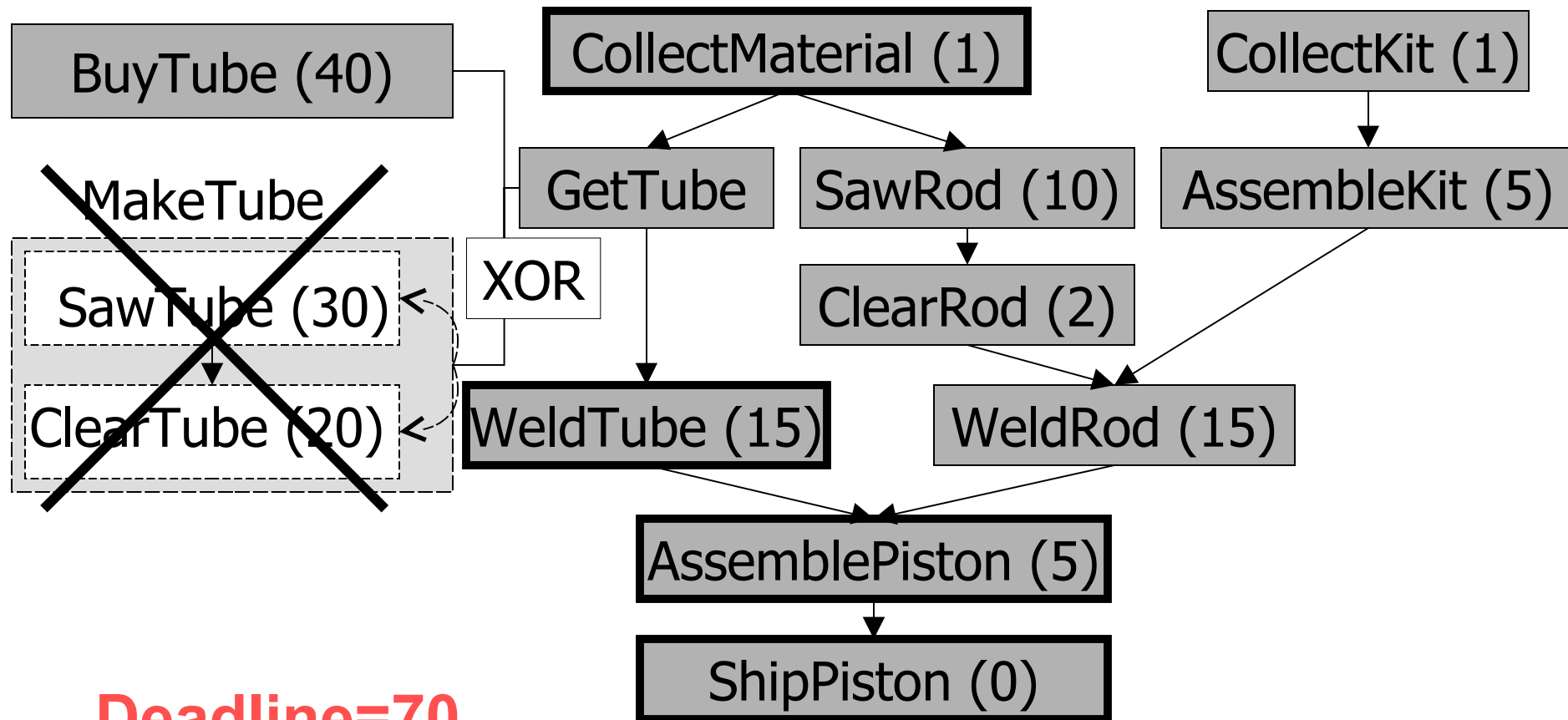
$$e_{\max}(a) \leftarrow \min(e_{\max}(a), s_{\max}(b))$$

- Bounds are propagated even on time intervals with still undecided execution status !**

- Inspired from [Barták&Čepek 2007]



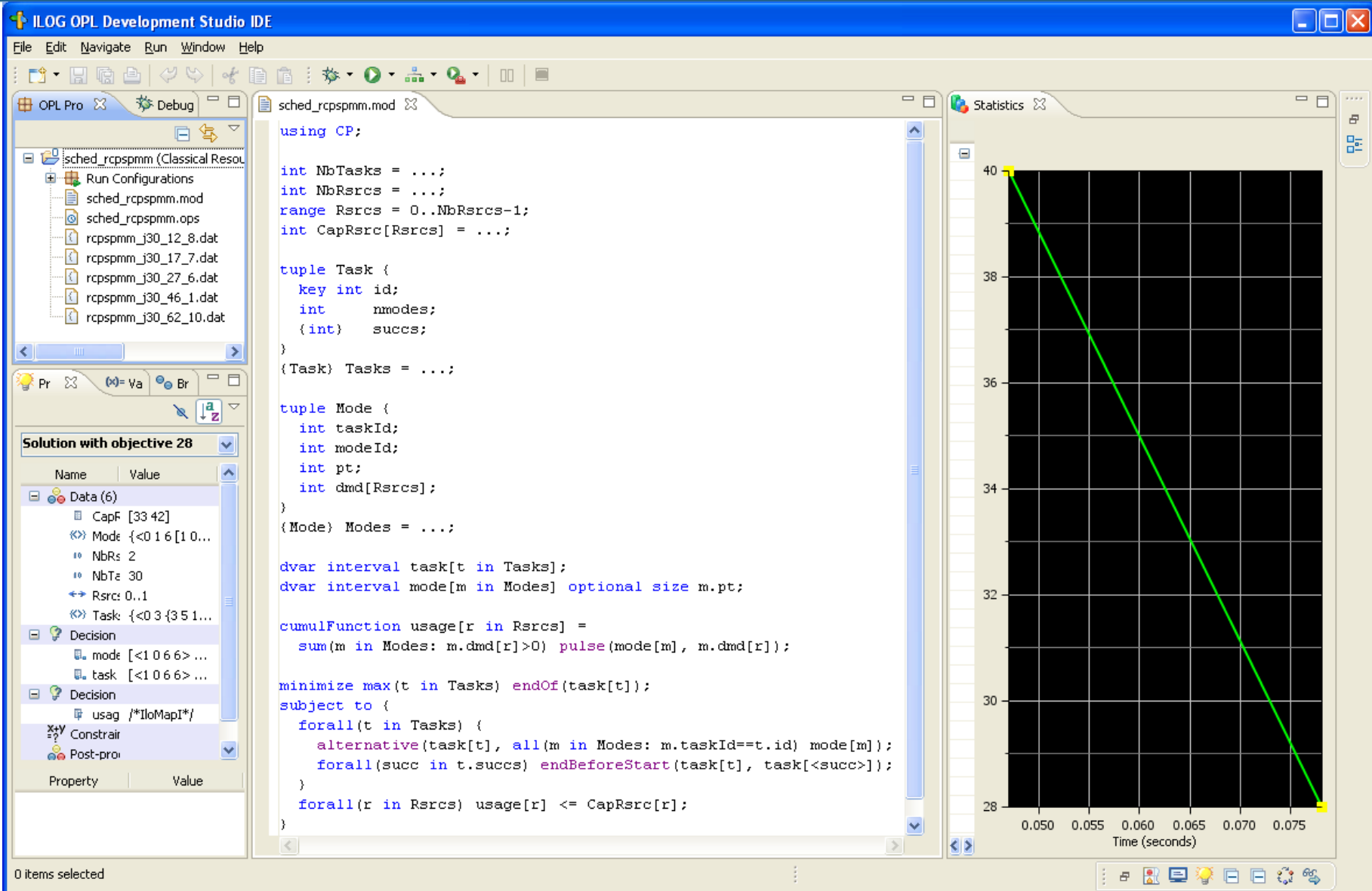
- Inspired from [Barták&Čepek 2007]



- Conditional time-intervals are the foundation of the new version of ILOG CP Optimizer (2.0) for **modeling** and solving detailed scheduling problems
- Model extensions:
 - Resources (sequencing, cumulative, state reasoning)
 - Calendars (resource efficiency curves, days off, etc.)
 - Expressions to use interval bounds (start, end, etc.) in classical CSP constraints on integer variables

- Conditional time-intervals are the foundation of the new version of ILOG CP Optimizer (2.0) for modeling and **solving** detailed scheduling problems
- Search:
 - Extension of SA-LNS [Laborie&Godard 2007] to handle optional time-interval variables

Example: Multi-Mode RCPSP





The screenshot displays the ILOG OPL Studio IDE interface. The main window shows the project 'sched_rcpspm' with a file tree on the left containing 'Run Configurations', 'sched_rcpspm.mod', 'sched_rcpspm.ops', and several data files. The central editor displays the 'Data' window with the following content:

```

using CP;

int NbTasks = ...;
int NbRsrcs = ...;
range Rsrcs = 0..NbRsrcs-1;
int CapRsrc[Rsrcs] = ...;

tuple Task {
    key int id;
    int nmodes;
    {int} succs;
}
{Task} Tasks = ...;

tuple Mode {
    int taskId;
    int modeId;
    int pt;
    int dmd[Rsrcs];
}
{Mode} Modes = ...;

dvar interval task[t in Tasks];
dvar interval mode[m in Modes] optional size m.pt;

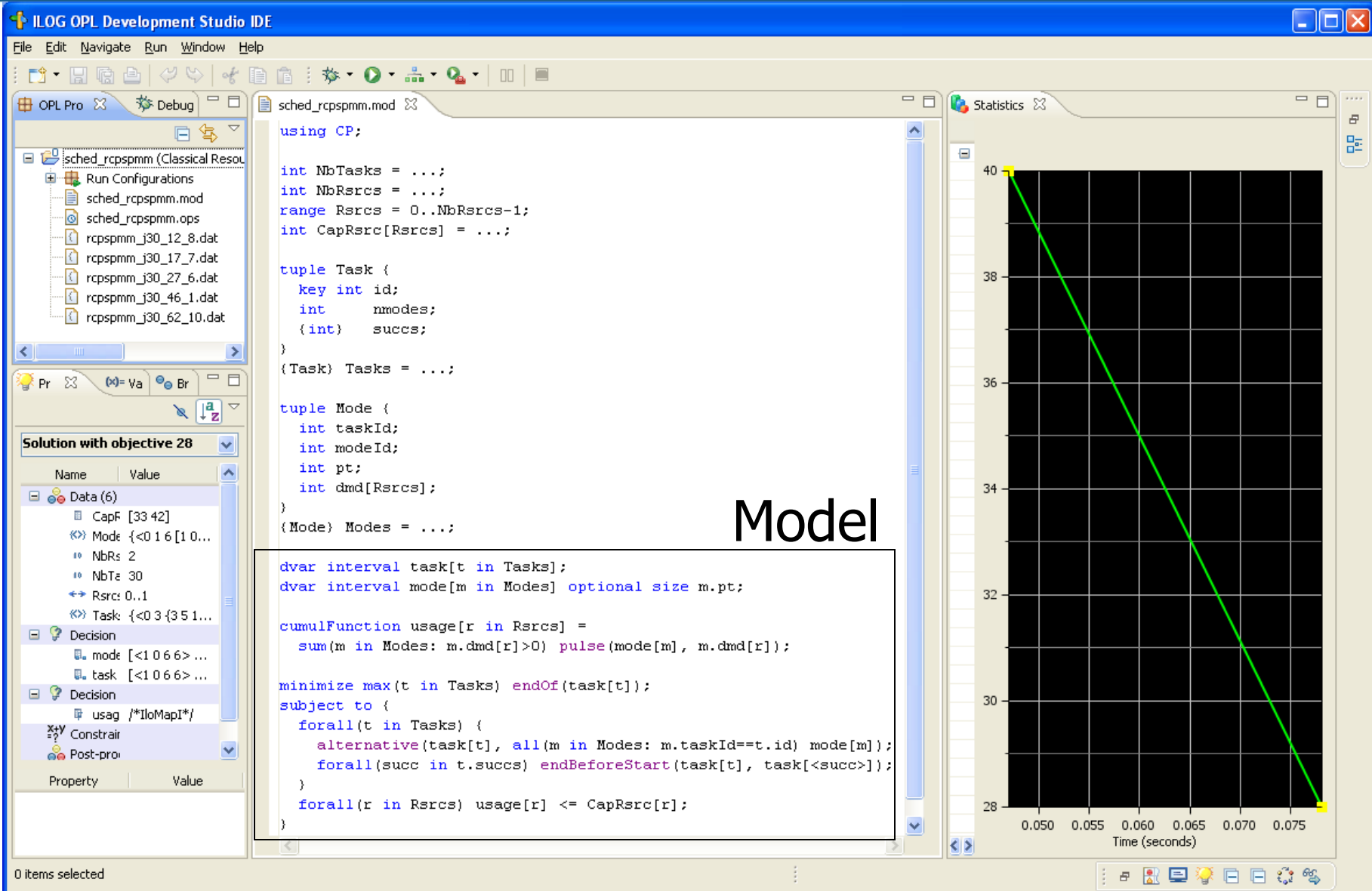
cumulFunction usage[r in Rsrcs] =
    sum(m in Modes: m.dmd[r]>0) pulse(mode[m], m.dmd[r]);

minimize max(t in Tasks) endOf(task[t]);
subject to {
    forall(t in Tasks) {
        alternative(task[t], all(m in Modes: m.taskId==t.id) mode[m]);
        forall(succ in t.succs) endBeforeStart(task[t], task[<succ>]);
    }
    forall(r in Rsrcs) usage[r] <= CapRsrc[r];
}

```

On the right, the 'Statistics' window shows a line graph with a green line representing a cumulative function over time. The x-axis is labeled 'Time (seconds)' and ranges from 0.050 to 0.075. The y-axis ranges from 28 to 40. The line starts at (0.050, 40) and ends at (0.075, 28).

Example: Multi-Mode RCPSP



Example: Multi-Mode RCPSP



Changing the rules of business

Trial Version:
<http://ilog.com/products/oplstudio/trial.cfm>

The screenshot displays the ILOG OPL Development Studio IDE interface. The main window shows the OPL model file `sched_rcpsmm.mod` with the following code:

```
using CP;

int NbTasks = ...;
int NbRsrcs = ...;
range Rsrcs = 0..NbRsrcs-1;
tuple Task {
    int id;
    int succs[NbRsrcs];
}
{Task} Tasks = ...;

tuple Mode {
    int taskId;
    int modeId;
    int pt;
    int dmd[Rsrcs];
}
{Mode} Modes = ...;

dvar interval task[t in Tasks];
dvar interval mode[m in Modes] optional size m.pt;

cumulFunction usage[r in Rsrcs] =
    sum(m in Modes: m.dmd[r]>0) pulse(mode[m], m.dmd[r]);

minimize max(t in Tasks) endOf(task[t]);
subject to {
    forall(t in Tasks) {
        alternative(task[t], all(m in Modes: m.taskId==t.id) mode[m]);
        forall(succ in t.succs) endBeforeStart(task[t], task[<succ>]);
    }
    forall(r in Rsrcs) usage[r] <= CapRsrc[r];
}
```

The left sidebar shows the project structure and a table of solution values:

Name	Value
Data (6)	
CapF	[33 42]
Mode	{<0 1 6 [1 0 ...
NbRs	2
NbTs	30
Rsrc	0..1
Task	{<0 3 {3 5 1 ...
Decision	
mode	[<1 0 6 6> ...
task	[<1 0 6 6> ...
Decision	
usag	/*IloMapI*/
Constr	
Post-pro	

The right sidebar shows a graph titled "Statistics" with a green line representing a cumulative function over time. The x-axis is labeled "Time (seconds)" and ranges from 0.050 to 0.075. The y-axis ranges from 28 to 40. The green line starts at approximately (0.050, 38) and ends at approximately (0.075, 28).