Constraint Programming

Lab 2. Scheduling with OPL

Ruslan Sadykov

INRIA Bordeaux—Sud-Ouest

17 February 2022

Typical scheduling problem

Time intervals — activities, operations, or jobs to do, optional or obligatory

Temporal constraints — possible relations between the starting and completion times of activities

Specialised constraints — complex relations between activities due to the state and capacities of resources

Cost functions

- Necessary time to complete all activities (**makespan**)
- Cost for non-execution of optional activities
- Penalties for violating due dates of certain activities

Intervals

Syntax

```
dvar interval <taskName> <switches>
```

- Time window

  ```
  dvar interval masonry in 0..20;
  ```

- Job size (processing time)

  ```
  dvar interval windows size 5 in 0..7;
  ```

- Optional job

  ```
  dvar interval garden optional;
  ```

Intervals : linked variables

`endOf` — end of interval (completion time of job)

`startOf` — start of interval (starting time of job)

`lengthOf` — interval duration (can be different from the size if preemptions are allowed)

`sizeOf` — size of interval

`presenceOf` — 1, if interval is present, 0 otherwise (for optional intervals)

Intensity : calendar functions

Syntax

```
dvar interval <taskName> intensity F;
```

Here `F` is a step (piecewise constant) function.

Example

A job should be done during a week by a worker who works full-time during first five days and half-time on Saturday

```
stepFunction F = stepwise(100->5 ; 50->6 ; 0->7) ;
dvar interval decoration size 5..5 in 1..7 intensity F;
```

Precedence constraints

Syntax

```
endBeforeStart(a,b[,z])
```

Example

The ceiling should be dried during 2 days before being painted :

```
endBeforeStart(ceiling, painting, 2)
```

Other constraints

endBeforeStart

endAtStart

endAtEnd

startAtStart

startAtEnd

Cumulative constraints

Syntax

```
cumulFunction <functionName> = <function_expression>;
```

where expression can use `step`, `pulse`, `stepAtStart`, `stepAtEnd`

Cumulative function can be constrained :

```
cumulFunction workersUsage = ...;
...
workersUsage <= NbWorkers;
```

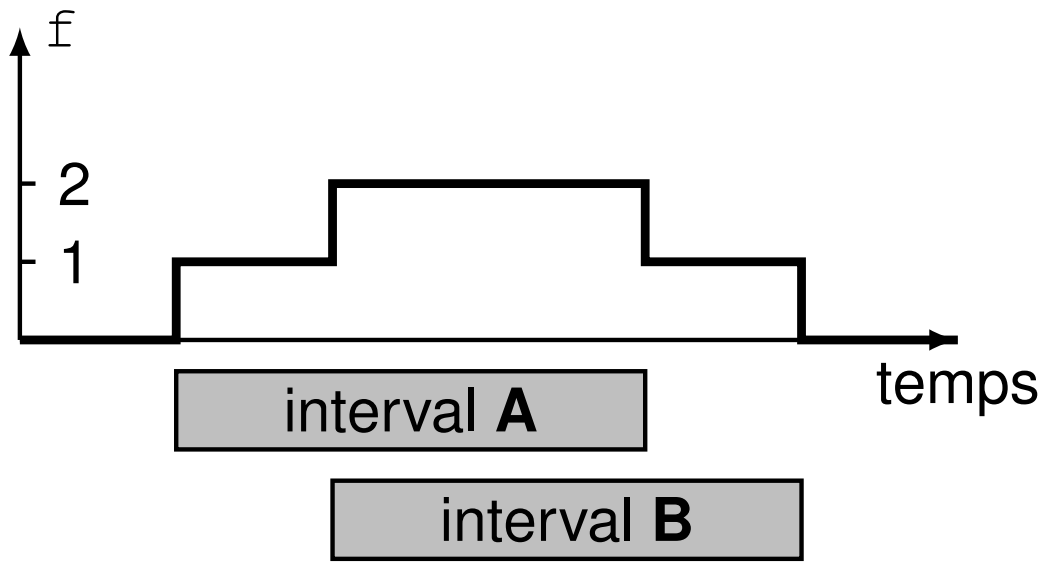
Function `pulse`

Syntax

```
cumulFunction f = pulse(u, v, h) ;
cumulFunction f = pulse(a, h) ;
cumulFunction f = pulse(a, hmin, hmax) ;
```

Example

```
cumulFunction f =
  pulse(A, 1)
  + pulse(B, 1) ;
```



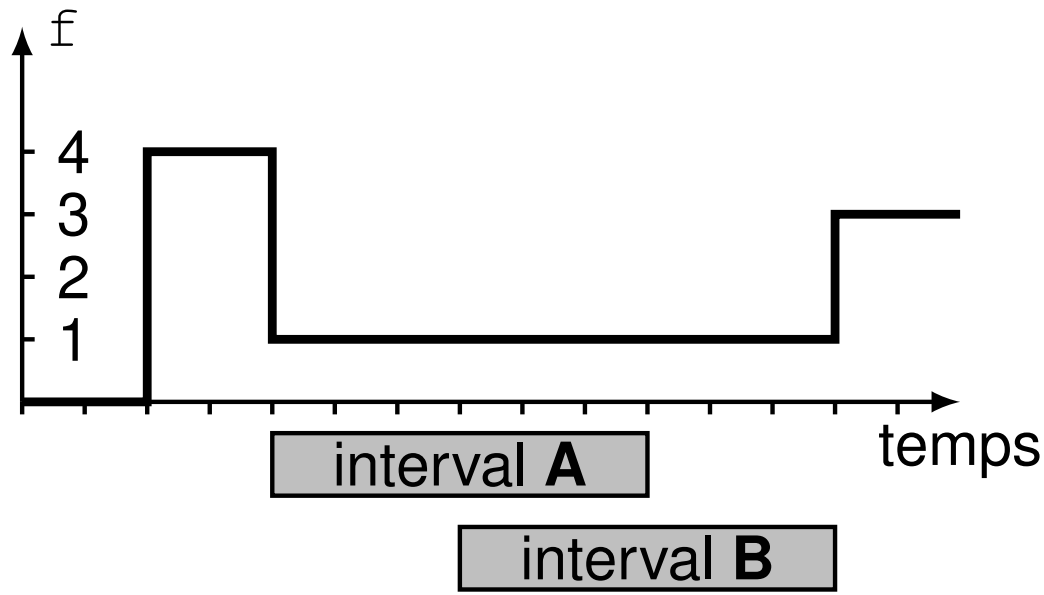
Functions `step`

Syntax

```
cumulFunction f = step(u, h) ;
cumulFunction f = stepAtStart(a, h) ;
cumulFunction f = stepAtEnd(a, hmin, hmax) ;
```

Example

```
cumulFunction f =
  step(2, 4)
  + stepAtStart(A, -3)
  + stepAtEnd(B, 2) ;
```



Sequencing

Sequencing variable represents a total order of a set of intervals.

Syntax

```
dvar sequence <seqName> in <intervalName> [types T] ;
```

Attention

Order of intervals does not necessarily establish the relative position of intervals in time.

Example

```
dvar sequence workers[w in WorkerNames] in
all(h in Houses,t in TskNames : Worker[t]==w) itvs[h][t]
types all(h in Houses,t in TskNames : Worker[t]==w) h;
```

Disjunctive global constraint

Syntax

```
noOverlap (<sequenceName> [,M]) ;
```

Example

- The set of activities should be scheduling a single machine.
- There is setup time necessary to pass from one activity to another, this setup time depends on the type of activities.
- No overlapping of activities in time.

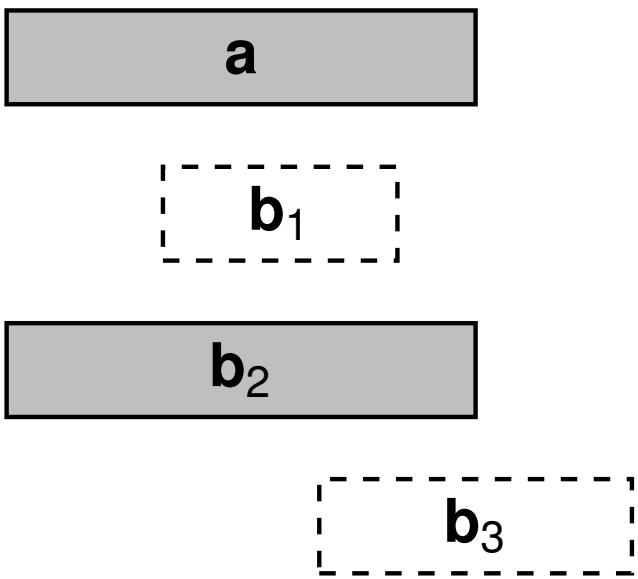
```
tuple triplet { int id1; int id2; int value; };
{triplet} M = { <i,j,ftoi(abs(i-j))> |
                i in Types, j in Types };
dvar interval A[i in 1..n] size d[i];
dvar sequence p in A types T;
subject to {
   noOverlap(p, M);
};
```

Alternative activities

Interval `a` is executed if and only if one of intervals in `B` is executed. In this case, they are synchronized.

Syntax

```
alternative(a,B) ;
```



Example

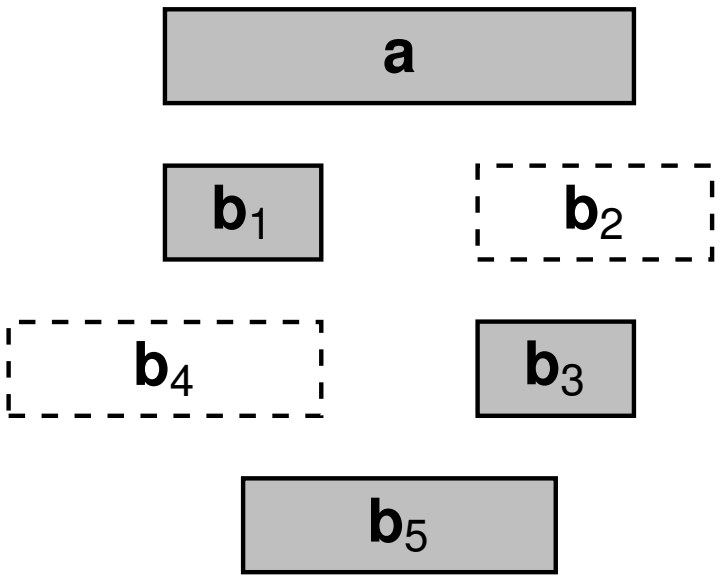
```
alternative(tasks[h] [t],
  all(s in Skills : s.task==t)
  wtasks[h] [s]) ;
```

Spanning activities

Interval `a` « spans » all intervals executed in `B` : `a` starts in the beginning of the first interval in `B` and completed at the end of the last one.

Syntax

```
span(a,B) ;
```



Example

```
span(house[i],
  all(t in tasks  : t.house == i)
  tasks[t]) ;
```

Synchronized activities

All intervals executed in `B` start and complete at the same time as interval `a`.

Syntax

```
synchronize(a,B);
```

Example

```
synchronize(task[i], all(o in opers  : o.task == i)
            tiopers[o]);
```