The Deadline Misses constraints in ILOG SOLVER [1] v2

Stefano Di Alesio[1,2], Shiva Nejati[2], Lionel Briand[2], and Arnaud Gotlieb[1]

[1]Certus Centre, Simula Research Laboratory, Norway
{stefano, arnaud}@simula.no
[2]SnT Software V&V Lab, University of Luxembourg, Luxembourg
{shiva.nejati, lionel.briand}@uni.lu

I. INTRODUCTION

In this report, we discuss the structure of the OPL constraint model we used to define the schedulability analysis problem of our work [1]. This document details constants, variables, constraints and objective function of the OPL constraint model.

II. OPL PROGRAM

```
/*********************************************
 * OPL 12.5 Model
 * Author: Stefano
 * Creation Date: 20 Jan 2013 at 15:13:43
 *********************************************/
using CP;

// T: Observation interval (range of time quanta)
int tq = ...;
range T = 0..tq-1;

// c: Number of Processor Cores
int c = ...;

// n: Number of tasks
int n = ...;
range J = 0..n-1;

tuple TaskExecution {
    int task;
    int execution;
}



/* Task Constants */

int priority[J] = ...;
int max_priority = max(j in J) priority[j];
int task_deadline[J] = ...;
int min_interarrival_time[J] = ...;
int max_interarrival_time[J] = ...;
int duration[J] = ...;
int triggers[J, J] = ...;
int dependent[J, J] = ...;



/* Task Execution Constants */
int task_executions[J] = ...;

{TaskExecution} A = {<j,k> | j in J, k in 0..task_executions[j]-1};
{TaskExecution} dependent_tasks[a in A] = {a1 | a1 in A : dependent[a.task, a1.task] == 1};
{TaskExecution} preceding_tasks[a in A] = {a1 | a1 in A : a1.task == a.task && a1.execution == a.execution-1};
{TaskExecution} triggered_tasks[a in A] = {a1 | a1 in A : triggers[a.task, a1.task] == 1};
{TaskExecution} triggering_tasks[a in A] = {a1 | a1 in A : triggers[a1.task, a.task] == 1};

int est[a in A] = 0;
int lst[a in A] = tq;
int eet[a in A] = 0;
int let[a in A] = tq;



/* Task Execution Variables */
```

```
56
57 dvar int arrival_time[a in A] in T;
58 dvar int start[a in A] in est[a]..lst[a];
59 dvar int end[a in A] in eet[a]..let[a];
60 dvar int active[a in A, t in T] in 0..1;
61
62 dexpr int eligible_for_execution[a in A] =
63     (prevc(A, a).task == a.task) ?
64         maxl(arrival_time[a], end[prevc(A, a)]) :
65         arrival_time[a];
66 dexpr int dependent_tasks_running[a in A, t in T] = sum(a1 in A : dependent[a.task, a1.task] == 1)(maxl(0, minl(1, end[a1] -
       t)) * maxl(0, minl(1, t - start[a1])));
67
68 dexpr int active_task[j in J, t in T] = sum(a in A : a.task == j) active[a, t];
69 dexpr int load[t in T] = count(all(a in A)active[a, t], 1);
70 dexpr int task_execution_deadline[a in A] = arrival_time[a] + task_deadline[a.task];
71 dexpr int deadline_miss[a in A] = end[a] - task_execution_deadline[a];
72
73
74
75
76 /* Indexed Constraints */
77 constraint to1[A, A];
78 constraint to2[A, A];
79 constraint to3[A, A];
80 constraint to4[A, A];
81
82
83
84
85 /* Constraint Program */
86
87 maximize
88     sum(a in A) 2^deadline_miss[a];
89
90 subject to {
91
92     forall(a in A) {
93
94         wf1: eligible_for_execution[a] <= start[a];
95         wf2: start[a] + duration[a.task] <= end[a];
96
97
98         wf3: sum(t in T) active[a, t] == duration[a.task];
99
100        wf4: active[a, start[a]] == 1;
101        wf5: active[a, end[a]-1] == 1;
102
103        wf6: sum(t in T) (t <= start[a]-1 && active[a, t] == 1) == 0;
104        wf7: sum(t in T) (t >= end[a] && active[a, t] == 1) == 0;
105
106    }
107
108    forall(a in A : min_interarrival_time[a.task] == max_interarrival_time[a.task] && card(triggering_tasks[a]) == 0)
109        wf8: arrival_time[a] == a.execution * min_interarrival_time[a.task];
110
111    forall(a in A : prevc(A, a).task == a.task && min_interarrival_time[a.task] != max_interarrival_time[a.task] && card(
         triggering_tasks[a]) == 0) {
112        wf9: arrival_time[prevc(A, a)] + min_interarrival_time[a.task] <= arrival_time[a];
113        wf10: arrival_time[a] <= arrival_time[prevc(A, a)] + max_interarrival_time[a.task];
114    }
115
116
117    /* II. Temporal Ordering constraints */
118    forall(a in A) {
119
120        forall(a1 in preceding_tasks[a])
121            to1[a, a1]: start[a] >= end[a1];
122
123        forall(a1 in triggered_tasks[a])
124            to2[a, a1]: end[a] == arrival_time[a1];
125
126        forall(a1 in dependent_tasks[a]) {
127            to3[a, a1]: start[a] != start[a1];
128            to4[a, a1]: start[a] <= start[a1]-1 => start[a1] >= end[a];
129        }
130    }
131
132
133
134    /* III. Multi-core Constraint */
135    forall(t in T)
136        mc: load[t] <= c;
137
```

```
138
139
140     /* IV. Preemptive Scheduling Constraint */
141     forall(t in T, a0 in A, a1 in A : priority[a0.task] < priority[a1.task])
142         ps: (active[a1, t] + c >= load[t] + active[a0, t] ||
143               eligible_for_execution[a1] >= t+1 ||
144               dependent_tasks_running[a1, t] >= 1 ||
145               end[a1] <= t);
146
147
148
149
150     /* V. Good CPU Usage Constraints */
151
152     forall(a in A, t in T)
153         gcu1: load[t] == c ||
154               active[a, t] == 1 ||
155               eligible_for_execution[a] >= t+1 ||
156               dependent_tasks_running[a, t] >= 1 ||
157               end[a] <= t;
158
159
160     forall(a0 in A, a1 in A : priority[a0.task] <= priority[a1.task], t in T : t < tq-1)
161         gcu2: active[a1, t] == 0 ||
162               active[a1, t+1] == 1 ||
163               end[a1] == t+1 ||
164               active[a0, t+1] == 0;
165
166     forall(a in A : priority[a.task] == max_priority)
167         gcu3: end[a] - start[a] == duration[a.task];
168
169 }
```

Listing 1: OPL Program for Testing Deadline Misses

REFERENCES

[1] Stefano Di Alesio, Shiva Nejati, Lionel Briand, Arnaud Gotlieb *"Stress Testing of Task Deadlines: A Constraint Programming Approach"*, 2013