The Regular Unwinding Framework

Benoît Dupont de Dinechin

STMicroelectronics AST Embedded Systems Research Laboratory
Via Cantonale 16E 6928 Manno Switzerland

Abstract

Modulo scheduling is a 1-periodic cyclic scheduling technique that originates from compiler instruction scheduling. To solve resource-constrained modulo scheduling problems, we propose regular unwinding, a new framework based on the unwinding of the cyclic scheduling problem and the acyclic scheduling under a regularity constraint. Given λ the modulo schedule period, a λ-regular unwinded schedule is such that two successive instances of any generic operation are scheduled at least λ cycles apart. We show the equivalence between modulo schedules of period λ and λ-regular unwinded schedules of pseudo-polynomial size. We apply the regular unwinding framework to solve in pseudo-polynomial time several modulo scheduling problems with UET operations on parallel processors and on typed task systems.

Key words: cyclic scheduling, modulo scheduling, instruction scheduling, monotone interval order, backward scheduling, modified deadlines

Introduction

Modulo scheduling [13][14] is the mainstream cyclic instruction scheduling technique used by optimizing compilers for the software pipelining of loops on instruction-level parallel processors. In modulo scheduling problems, a set of operations $\{O_i\}_{1 \leq i \leq n}$ is repeatedly executed with an integral period of λ cycles, traditionally known as the *initiation interval*, subjected to renewable resource constraints and uniform dependences with dependence delays. The objective of modulo scheduling is to build a schedule that minimizes λ.

The current techniques to solve modulo scheduling problems in compilers are domain-specific heuristics, some of them being satisfactory in practice [15].

Email address: Benoit.Dupont-de-Dinechin@st.com (Benoît Dupont de Dinechin).

However, no machine scheduling results have been applied to modulo scheduling and the only known relaxations are the obvious ones: either remove all resource constraints; or remove all dependence constraints. While this yields lower bounds on the period λ, no modulo scheduling relaxations are known that include dependence circuits and resource constraints.

In this work, we propose the *regular unwinding* framework to solve resource-constrained modulo scheduling problems. This framework combines two ideas:

p-Unwinding is the creation of an acyclic scheduling problem by instancing p iterations of the modulo scheduling problem operation set $\{O_i\}_{1 \leq i \leq n}$ and by instancing the dependences accordingly.

λ-Regularity is the constraint that any two operation instances O_i^k, O_i^{k+1} created by the unwinding of an operation O_i of the modulo scheduling problem are scheduled at least λ cycles apart.

Thanks to the λ-regularity condition, we show in Section 2 that: either the p-unwinded schedule becomes λ-stationary with all the O_i^k, O_i^{k+1} scheduled exactly λ cycles apart; or, the λ-regularized p-unwinded scheduling problem is infeasible for some p, implying there is no modulo schedule at period λ. From the λ-stationary part of the p-unwinded schedule, we extract a 1-periodic cyclic schedule with period λ, that is, a modulo schedule. The λ-regularity can be enforced by introducing dependence arcs with a positive latency λ between any two operations O_i^k, O_i^{k+1}.

Like the classic modulo scheduling framework [13][14], our regular unwinding framework requires that a period λ be assumed before trying to schedule, a difficulty addressed by performing a dichotomy search for a feasible value of λ. The novelty of regular unwinding however, is that for any assumed value of λ, acyclic machine scheduling techniques apply to the λ-regular p-unwinded scheduling problems. Whenever a polynomial-time solution is known for this particular class of acyclic machine scheduling problems, the corresponding modulo scheduling problem is solved in pseudo-polynomial time.

The presentation is as follows. In Section 1, we provide necessary scheduling background, including extensions of the $\alpha|\beta|\gamma$ scheduling problem denotation and a survey of the scheduling algorithm of Leung, Palem and Pnueli [10]. In Section 2, we formulate the resource-constrained modulo scheduling problem and we establish the regular unwinding framework in the setting of renewable resources. In Section 3, we apply this framework to modulo scheduling problems with Unit Execution Time (UET) operations on parallel machines, by scheduling the p-unwinded problems with the Leung-Palem-Pnueli Algorithm (LPPA). In Section 4, we extend these results to the typed task systems [9].

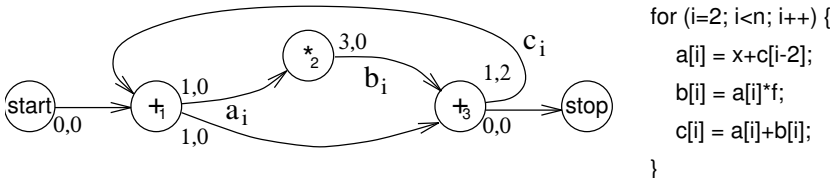

Fig. 1. Sample cyclic instruction scheduling problem.

1 Deterministic Scheduling Background

1.1 Resource-Constrained Cyclic Scheduling Problems

A *basic cyclic scheduling problem* [8] is defined by a set of generic operations $\{O_i\}_{1\leq i\leq n}$ to be executed repeatedly, thus defining a set of operation instances $\{O_i^k\}_{1\leq i\leq n}^{k>0}, k \in \mathbb{N}$. We call *iteration* k the set of operation instances $\{O_i^k\}_{1\leq i\leq n}$. For any $i \in [1, n]$ and $k > 0 \in \mathbb{N}$, let σ_i^k denote the schedule date of operation instance O_i^k. Basic cyclic scheduling problems are constrained by *uniform dependences* denoted $O_i \overset{\theta_i^j,\omega_i^j}{\prec} O_j$:

$$O_i \overset{\theta_i^j,\omega_i^j}{\prec} O_j \Longrightarrow \sigma_i^k + \theta_i^j \leq \sigma_j^{k+\omega_i^j} \quad \forall k > 0$$

The *latency* θ_i^j and the *distance* ω_i^j of uniform dependences are non-negative integers. The *carried* dependences are such that $\omega_i^j > 0$.

Let O_0^k and O_{n+1}^k be dummy operations defined by $\sigma_0^k \overset{\text{def}}{=} \min_{1\leq i\leq n} \sigma_i^k$ and $\sigma_{n+1}^k \overset{\text{def}}{=} \max_{1\leq i\leq n} \sigma_i^k$. Given a feasible cyclic schedule $\{\sigma_i^k\}_{0\leq i\leq n+1}^{k>0}$, its quality is measured by the asymptotic throughput $R_\infty \overset{\text{def}}{=} \lim_{k\to\infty} \frac{k}{\sigma_{n+1}^k}$.

In case of *cyclic scheduling on parallel processors* [8], basic cyclic scheduling problems are augmented with resource constraints. Each generic operation O_i has a processing time p_i and there are m identical processors available. Each O_i^k requires one of the processors to be available between dates $[\sigma_i^k, \sigma_i^k + p_i[$.

In the present work, we are interested in *resource-constrained cyclic scheduling problems*, where the resource constraints are adapted from the *resource-constrained project scheduling problems* (RCPSP) [3]. Precisely, we assume a set of *renewable resources*, also known as *cumulative resources*, whose availabilities are given by an integral vector $\vec{B}$. Each generic operation O_i is also associated with an integral vector $\vec{b}_i$ of resource requirements and this defines the resource requirements of all the operation instances $\{O_i^k\}^{k>0}$. For the cyclic scheduling problems we consider, the cumulative use of the resources by the operation instances executing at any given time must not exceed $\vec{B}$.

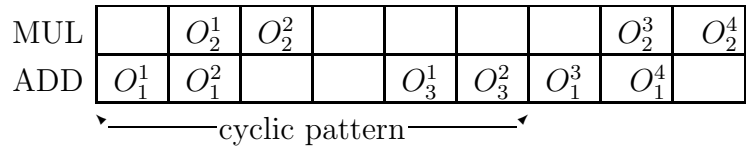

Fig. 2. Unwind and find pattern cyclic scheduling heuristic.

To illustrate the resource-constrained cyclic scheduling problems that arise from instruction scheduling of inner loops, consider the code and its dependence graph displayed in Figure 1. Here the dummy operations O_0 and O_{n+1} are labeled start and stop (to simplify the presentation, we did not include the memory access operations). Here, operation O_3 (c[i]=a[i]+b[i]) of iteration i must execute before operation O_1 (a[i]=x+c[i-2]) of iteration i+2 and this creates the uniform with distance 2 between O_3 and O_1 (arc c_i in Figure 1).

Assume this code is compiled for a microprocessor with an adder and a multiplier that operate in parallel. The adder and the multiplier may start a new operation every cycle. However, due to pipelined implementation, the multiplier result is only available after 3 cycles. This resource-constrained cyclic scheduling problem is defined by $p_1 = p_2 = p_3 = 1$, $\vec{B} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $\vec{b}_1 = \vec{b}_3 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\vec{b}_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$. The dependences are $O_1 \overset{1,0}{\prec} O_2$, $O_1 \overset{1,0}{\prec} O_3$, $O_2 \overset{3,0}{\prec} O_3$, $O_3 \overset{1,2}{\prec} O_1$.

The *K-periodic schedules* are solutions to cyclic scheduling problems that satisfy the property $\exists \lambda \in \mathbb{Q} : \sigma_i^k + K\lambda = \sigma_i^{k+K}$. Given a cyclic scheduling problem on parallel processors with uniform dependences, there always exists K such that the K-periodic schedules contain an optimal solution [8], but this K is problem instance dependent. The asymptotic throughput R_∞ of a periodic schedule equals $\frac{1}{\lambda}$, so the *periodic scheduling problem* is the construction of a K-periodic schedule of minimum λ.

An effective heuristic technique for K-periodic scheduling is to schedule the successive instances of the set of generic operations in acyclic context, until a repeating pattern of K successive iterations appears in the schedule [2][1]. To ensure convergence, these "unwind and find pattern" techniques require that a span-limiting uniform dependence $O_{n+1} \prec O_0$ be added to the original problem. In Figure 2, we apply the unwind and find pattern heuristic on our sample problem to construct a 2-periodic schedule of period $\lambda = 3$.

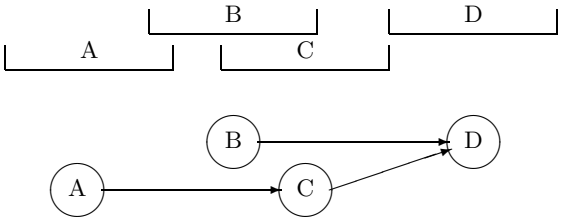

Fig. 3. Set of intervals and the corresponding interval-order graph.

1.2 Machine Scheduling Problem Denotation

In parallel machine scheduling problems, an operation set $\{O_i\}_{1\leq i\leq n}$ is processed on m identical processors. Each operation O_i requires the exclusive use of one processor for p_i time units, starting at its *schedule date* σ_i. Scheduling problems may involve *release dates* r_i and *due dates* d_i. This constrains the schedule date σ_i of operation O_i as $\sigma_i \geq r_i$ and there is a penalty whenever $C_i > d_i$, with C_i the *completion date* of O_i defined as $C_i \stackrel{\text{def}}{=} \sigma_i + p_i$. For problems where $C_i \leq d_i$ is mandatory, the d_i are called *deadlines*. A *dependence* $O_i \prec O_j$ between two operations constrains the schedule with $\sigma_i + p_i \leq \sigma_j$. The *dependence graph* has one arc (O_i, O_j) for each dependence $O_i \prec O_j$.

Machine scheduling problems are denoted by a triplet notation $\alpha|\beta|\gamma$, where α describes the processing environment, β specifies the operation properties and γ defines the optimality criterion. For the deterministic machine scheduling problems, the common values of α, β, γ are:

α : 1 for a single processor, P for parallel processors, Pm for the given m parallel processors. We introduce ΣP to denote *typed task systems* [9].
β : r_i for release dates, d_i for deadlines (if $\gamma = \bullet$) or due dates, $p_i = 1$ for Unit Execution Time (UET) operations.
γ : $\bullet$ for the feasibility, C_{max} or L_{max} for the minimization of these objectives.

The *makespan* is $C_{max} \stackrel{\text{def}}{=} \max_i C_i$ and the *maximum lateness* is $L_{max} \stackrel{\text{def}}{=} \max_i L_i : L_i \stackrel{\text{def}}{=} C_i - d_i$. The meaning of the additional β fields is:

$prec(l_i^j)$ Dependence delays l_i^j, where $O_i \prec O_j$ implies $\sigma_i + p_i + l_i^j \leq \sigma_j$.
$prec(l_i^j = l)$ All the dependence delays l_i^j have the same value l.
$inTree$ The dependence graph is an in-tree.
$intOrder(mono\ l_i^j)$ The dependence graph weighted by $w(O_i, O_j) \stackrel{\text{def}}{=} p_i + l_i^j$ is a monotone interval order (see below).
$chainOrder(mono\ l_i^j)$ Case of $intOrder(mono\ l_i^j)$, where the dependence graph is the transitive closure of a chain that connects all operations.
$subChainOrder(mono\ l_i^j)$ Case of $intOrder(mono\ l_i^j)$, where the dependence graph is the transitive closure of a chain that connects some operations. Other operations are independent.

As introduced by Papadimitriou & Yannakakis [12], an *interval-order* is defined by an incomparability graph that is chordal. An interval-order is also the transitive orientation of the complement of an interval graph [12] (see Figure 3). A *monotone interval-order* graph [11] is an interval-order graph (V, E) with a non-negative weight function w on the arcs such that, given any $(v_i, v_j), (v_i, v_k) \in E : \operatorname{pred} v_j \subseteq \operatorname{pred} v_k \Rightarrow w(v_i, v_j) \leq w(v_i, v_k)$. Here $\operatorname{pred} v_j$ and $\operatorname{pred} v_k$ respectively denote the predecessors of v_j and v_k.

We extend further the $\alpha|\beta|\gamma$ scheduling problem denotation to cyclic scheduling problems by introducing the following β fields:

$prec(\theta_i^j, \omega_i^j)$ Uniform dependences, implying cyclic scheduling.
$circuit(\theta_i^j, \omega_i^j)$ The dependence graph is a single circuit that includes all the operations.
$subCircuit(\theta_i^j, \omega_i^j)$ The dependence graph is a single circuit that includes some operations; other operations are independent.
$\sum \omega_i^j = k$ The sum of the ω_i^j of the dependence graph is $k \in \mathbb{N}$. Because the $\omega_i^j \in \mathbb{N}$, in case $\sum \omega_i^j = 1$ only one ω_i^j is non-zero.
$\pi_i = \lambda_i$ The processing period of operation O_i is λ_i.

The Graham List Scheduling Algorithm (GLSA) is a classic scheduling algorithm where the time steps are considered in sequential order. For each time step, if a processor is idle, the highest priority operation available at this time is scheduled. An operation is available if the current time step is not earlier than its release date and all its predecessors have completed their execution early enough to satisfy the entering dependences of this operation. The GLSA is optimal for $P|r_i; d_i; p_i = 1|\bullet$ and $P|r_i; p_i = 1|L_{max}$ when using the earliest deadlines (or due dates) d_i first as priority [4] (Jackson's rule).

Definition 1 *Given a scheduling problem, a* pseudo-schedule *is a set of dates, one per operation, that satisfies the dependence constraints, the release dates and the deadlines of the scheduling problem.*

1.3 The Scheduling Algorithm of Leung, Palem and Pnueli

The Leung-Palem-Pnueli Algorithm (LPPA) [10] is a parallel machine scheduling algorithm based on deadline modification and the use of the *lower modified deadline first* priority in a GLSA. Generally speaking, the *modified deadlines* $\{d'_i\}$ of a scheduling problem are such that $\forall i \in [1, n] : \sigma_i + p_i \leq d'_i \leq d_i$ in any schedule. By using the *fixpoint modified deadlines* [1] computed as explained

[1] Leung, Palem and Pnueli call them "consistent and stable modified deadlines".

below, this GLSA solves the following problems in polynomial time:

- $1|prec(l_i^j \in \{0,1\}); r_i; d_i; p_i = 1|\bullet$
- $P2|prec(l_i^j \in \{-1,0\}); r_i; d_i; p_i = 1|\bullet$
- $P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$
- $P|inTree(l_i^j = l); d_i; p_i = 1|\bullet$

To find its modified deadlines $\{d'_i\}_{1 \le i \le n}$, the LPPA computes the *optimal backward schedule date* σ'_i of each operation O_i and updates its current modified deadline as $d'_i \leftarrow \sigma'_i + p_i$. This process of deadline modification is iterated over all operations until a fixpoint is reached.

A backward scheduling problem $B(O_i, S_i)$ comprises an operation O_i, an operation set $S_i \not\ni O_i$ that contains the *supporting set* of O_i, the dependence-consistent release dates $\{r'_i\}$ and the current modified deadlines $\{d'_i\}$. The supporting set of O_i contains the operations that affect its modified deadline r'_i. Defining S_i as the union of the successors of O_i and the operations independent from O_i ensures it contains the supporting set of O_i [10].

A *backward schedule* $\{\sigma'_j\}$ for $B(O_i, S_i)$ satisfies the transitive dependence delays l_i^{j+} from O_i to any $O_j \in S_i$, the resource constraints of m parallel processors (assuming UET operations), the dependence-consistent release dates and the current modified deadlines [10]:

- $\sigma'_i + p_i + l_i^{j+} \le \sigma'_j,\ \forall O_j \in S_i : O_i \prec O_j$
- $\forall t \in \mathbb{N} : |\{O_j \in \{O_i\} \cup S_i : \sigma'_j = t\}| \le m$
- $\forall O_j \in \{O_i\} \cup S_i : r'_j \le \sigma'_j < d'_j$

An *optimal backward schedule* for $B(O_i, S_i)$ maximizes σ'_i.

For any $B(O_i, S_i)$ problem, Leung, Palem and Pnueli [10] propose the following BackwardSchedule algorithm to compute its optimal backward schedule:

- Binary search for the latest date p of O_i such that the constrained backward scheduling problem $(r'_i = p) \wedge B(O_i, S_i)$ is feasible. If there is such p, define the modified deadline of O_i as $d'_i \stackrel{\text{def}}{=} p + p_i$. Else the original scheduling problem is infeasible.
- To find if a constrained problem $(r'_i = p) \wedge B(O_i, S_i)$ is feasible, convert the transitive dependences from O_i to all the other $O_j \in S_i$ into release dates; then, remove the dependences. This yields a relaxation, which is a scheduling problem $P|r_i; d_i; p_i = 1|\bullet$.
- Optimally solve this $P|r_i; d_i; p_i = 1|\bullet$ relaxation using a lower-level GLSA with the earliest d_i first priority (Jackson's rule). This gives the feasibility status of the relaxation.

Leung, Palem and Pnueli [10] also propose the following ModifiedDeadlines algorithm that enables to compute the fixpoint modified deadlines with only n applications of the BackwardSchedule algorithm:

(i) Propagate the constraints of the release dates and the deadlines along the transitive dependence delays. This yields the dependence-consistent modified release dates $\{r'_i\}_{1\leq i\leq n}$ and the initial modified deadlines $\{d'_i\}_{1\leq i\leq n}$.
(ii) Define and initialize sets P as $\emptyset$ and U as $\{O_i\}_{1\leq i\leq n}$.
(1) Select $O_i \in U$ such that r'_i is maximal and O_i has no successors in U.
(2) Compute d'_i by BackwardSchedule of $B(O_i, P)$ and update d'_j for all $O_j \in U - \{O_i\}$ along the transitive dependence delays l_j^{i+}.
(3) Remove O_i from U and add it to P. Go to (1) until U is empty.

If any backward scheduling problem $B(O_i, P)$ is infeasible, then the original scheduling problem is also infeasible. Theorem 7 of [10] proves that the fixpoint modified deadlines computed by the BackwardSchedule algorithm are actual deadlines of the original problem whenever this problem is feasible.

Let us now consider the set of dates $\{\sigma_i^* \stackrel{\text{def}}{=} d'_i - 1\}_{1\leq i\leq n}$ derived from the fixpoint modified deadlines computed by the ModifiedDeadlines algorithm. These dates are not a schedule, as some resource constraints may be violated.

Proposition 2 ([10]) *The set of dates $\{\sigma_i^* \stackrel{def}{=} d'_i - 1\}_{1\leq i\leq n}$ computed by the ModifiedDeadlines algorithm of Leung, Palem and Pnueli is a pseudo-schedule.*

2 The Regular Unwinding Framework

2.1 Resource-Constrained Modulo Scheduling Problems

A *modulo scheduling problem* is a cyclic scheduling problem where all operations have the same processing period $\lambda \in \mathbb{N}$, also called the *initiation interval*. Being 1-periodic instead of K-periodic, the modulo schedules might exclude all optimal solutions of a given cyclic scheduling problem instance. However, in applications to software compilation and high-level synthesis, 1-periodic schedules are preferred because of the code size increase of the K-periodic schedules. In cases where such code size increase can be afforded, pre-unrolling the loop and optimizing the common computations across the unrolled loop body before modulo scheduling yields better results.

Compared to cyclic scheduling problems, a main simplification of modulo scheduling problems is that they can be described, solved or relaxed by only

considering the set of generic operations $\{O_i\}_{1\leq i\leq n}$. In particular, by introducing the *modulo schedule dates* $\{\sigma_i\}_{1\leq i\leq n}$ such that $\forall i \in [1,n], \forall k > 0 : \sigma_i^k = \sigma_i + (k-1)\lambda$, the uniform dependence constraints become:

$$O_i \overset{\theta_i^j,\omega_i^j}{\prec} O_j \Longrightarrow \sigma_i^k + \theta_i^j \leq \sigma_j^{k+\omega_i^j} \Longrightarrow \sigma_i + \theta_i^j - \lambda\omega_i^j \leq \sigma_j$$

In our extended $\alpha|\beta|\gamma$ scheduling problem denotation, a modulo scheduling problem at period λ is specified by adding $\pi_i = \lambda$ in the β field.

Let $\{\sigma_i\}_{1\leq i\leq n}$ denote the modulo schedule dates of a set of generic operations $\{O_i\}_{1\leq i\leq n}$. A *resource-constrained modulo scheduling problem* is defined by [7]:

- Uniform dependence constraints: for each such dependence $O_i \overset{\theta_i^j,\omega_i^j}{\prec} O_j$, a valid modulo schedule satisfies $\sigma_i + \theta_i^j - \lambda\omega_i^j \leq \sigma_j$. The dependence graph without the carried dependences is a DAG.
- Cumulative modulo resource constraints: each operation O_i requires $\vec{b}_i \geq \vec{0}$ resources for all the time intervals $[\sigma_i + k\lambda, \sigma_i + k\lambda + p_i[, k \in \mathbb{N}$ and the total resource use at any time cannot exceed a given availability $\vec{B}$. The integer value p_i is the processing time of O_i.

The first difficulty of modulo scheduling problems is that the constraints are parametric with the period λ. To solve modulo scheduling problems, one has to select a value of λ, then try to build a schedule. If scheduling fails, a new attempt is made at a higher λ. In the classic modulo scheduling framework [14], the search of a feasible period λ starts from $\lambda_{min} \overset{\text{def}}{=} \max(\lambda_{rec}, \lambda_{res})$, where:

$$\lambda_{rec} \overset{\text{def}}{=} \max_C \left\lceil \frac{\sum_C \theta_i^j}{\sum_C \omega_i^j} \right\rceil : \quad C \text{ dependence circuit}$$
$$\lambda_{res} \overset{\text{def}}{=} \max_{1\leq r\leq R} \left\lceil \frac{\sum_{i=1}^n p_i b_i^r}{B^r} \right\rceil : \quad R = \dim(\vec{B})$$

That is, λ_{rec} is the minimum λ such that there are no positive latency circuits in the dependence graph and λ_{res} is the minimum λ such that the renewable resources $\vec{B}$ are not over-subscribed.

Once a particular λ is assumed, most modulo scheduling problems still cannot be solved by classic machine scheduling techniques. Indeed, the modulo resource constraints introduce operation resource requirements of infinite extent. Also, the uniform dependence graph may include circuits (directed cycles), unlike machine scheduling precedence graphs that are acyclic. Even without the circuits, some modulo dependence latencies $\theta_i^j - \lambda\omega_i^j$ may also be negative.

The classic modulo scheduling heuristic is the modulo list scheduling algorithm

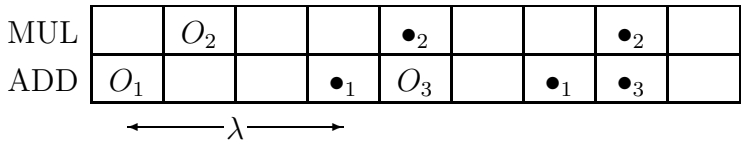

Fig. 4. Modulo list scheduling with $\lambda = 3$.

of Rau & Glaeser [13]. Given a value of λ, this heuristic applies a Graham list scheduling algorithm where each operation O_i requires $\vec{b_i}$ resources. Then any resource used by O_i at date t is also considered busy at dates $t + k\lambda, \forall k \in \mathbb{N}$ in order to satisfy the modulo resource constraints. Modulo list scheduling must be repeatedly applied with increasing values of λ until it succeeds.

Let us illustrate the classic modulo scheduling on the example of Figure 1. The dependence circuit $\{O_1 \overset{1,0}{\prec} O_2, O_2 \overset{3,0}{\prec} O_3, O_3 \overset{1,2}{\prec} O_1\}$ imply $\sigma_2 - \sigma_1 \geq 1 \wedge \sigma_3 - \sigma_2 \geq 3 \wedge \sigma_1 - \sigma_3 \geq 1 - 2\lambda$. By summing these inequalities we get:

$$0 \geq 1 + 3 + 1 - 2\lambda \Longrightarrow \lambda \geq \lambda_{rec} = \left\lceil \frac{1+3+1}{2} \right\rceil = 3$$

There is only one adder and there are two additions executed per iteration, so the resource constraints yield another lower bound $\lambda \geq \lambda_{res} = \left\lceil \frac{2}{1} \right\rceil$. Therefore, the minimum feasible λ of this example is $\lambda_{min} \overset{\text{def}}{=} \max(\lambda_{rec}, \lambda_{res}) = 3$. Applying modulo list scheduling assuming $\lambda = 3$ yields the modulo schedule illustrated in Figure 4, which spans 5 cycles.

2.2 Unwinding the Modulo Scheduling Problem

Given a modulo scheduling problem with the operation set $\{O_i\}_{1 \leq i \leq n}$, uniform dependences $\{O_i \overset{\theta_i^j, \omega_i^j}{\prec} O_j\}_{(i,j) \in E}$, release dates $\{r_i\}_{1 \leq i \leq n}$ and deadlines $\{d_i\}_{1 \leq i \leq n}$ its *p-unwinded scheduling problem* is defined by:

Operation Set $\{O_i^k\}_{1 \leq i \leq n}^{1 \leq k \leq p}$ with schedule dates $\{\sigma_i^k\}_{1 \leq i \leq n}^{1 \leq k \leq p}$
Dependence Constraints $\sigma_i^k + \theta_i^j \leq \sigma_j^{k + \omega_i^j}, \forall (i,j) \in E, \forall k \in [1, p - \omega_i^j]$
Resource Requirements $\vec{b}_i^k \overset{\text{def}}{=} \vec{b}_i \wedge p_i^k \overset{\text{def}}{=} p_i, \forall i \in [1,n], \forall k \in [1,p]$
Release Dates $r_i^k \overset{\text{def}}{=} r_i + (k-1)\lambda, \forall i \in [1,n], \forall k \in [1,p]$
Deadlines $d_i^k \overset{\text{def}}{=} d_i + (k-1)\lambda, \forall i \in [1,n], \forall k \in [1,p]$

We assume a modulo scheduling problem with release dates and deadlines and the objective is to find a modulo schedule that minimizes the period λ.

We denote $\bar{\omega} \overset{\text{def}}{=} \max_{(i,j) \in E} \omega_i^j$, $\hat{\omega} \overset{\text{def}}{=} \lceil \frac{\max_i d_i - \min_j r_j}{\lambda} \rceil - 1$ and $\Omega \overset{\text{def}}{=} \max(\hat{\omega}, \bar{\omega})$.

Proposition 3 *The dependence graph of a p-unwinded scheduling problem is acyclic with non-negative dependence latencies for any $p > 0$.*

PROOF. The modulo scheduling problem is defined such that the dependence graph without the carried dependences is acyclic. Unwinding the carried dependences yield dependences $O_i^k \prec O_j^l$ with $l = k + \omega_i^j > k$. Therefore the p-unwinded scheduling problem dependence graph has a topological sort. □

Definition 4 *A p-unwinded schedule is s-successive λ-stationary starting at iteration q if $\exists q \in [1, p-s], \forall i \in [1, n], \forall k \in [q, q+s-1] : \sigma_i^k + \lambda = \sigma_i^{k+1}$.*

By induction, a s-successive λ-stationary p-unwinded schedule starting at iteration q also satisfies: $\forall k \in [1, s], \forall i \in [1, n] : \sigma_i^q + k\lambda = \sigma_i^{q+k}$.

Proposition 5 *Given a modulo schedule $\{\sigma_i\}_{1 \leq i \leq n}$ at period λ, for any $p > 0$ the set of dates $\{\sigma_i^k \stackrel{def}{=} \sigma_i + (k-1)\lambda\}_{1 \leq i \leq n}^{1 \leq k \leq p}$ is a p-successive λ-stationary p-unwinded schedule.*

PROOF. From the definition of the p-unwinded scheduling problems, the set of dates $\{\sigma_i^k \stackrel{\text{def}}{=} \sigma_i + (k-1)\lambda\}_{1 \leq i \leq n}^{1 \leq k \leq p}$ satisfies the dependence and the resource constraints. This p-unwinded schedule is also p-successive λ-stationary. □

Definition 6 *An interesting schedule is a p-unwinded schedule such that $\exists q \in [1, p] : \{\sigma_i \stackrel{def}{=} \sigma_i^q\}_{1 \leq i \leq n}$ is a modulo schedule.*

Lemma 7 *Given a p-unwinded scheduling problem, if two operations O_i^k and O_j^l overlap then $|l-k| \leq \hat{\omega} \stackrel{def}{=} \lceil \frac{\max_i d_i - \min_j r_j}{\lambda} \rceil - 1$.*

PROOF. If O_i^k and O_j^l overlap then $r_i^k < d_j^l$ and $r_j^l < d_i^k$. This is equivalent to $(k-l)\lambda < d_j - r_i$ and $(l-k)\lambda < d_i - r_j$. Therefore $|l-k|\lambda < \max(d_j - r_i, d_i - r_j) \Rightarrow |l-k| < \frac{\max(d_j - r_i, d_i - r_j)}{\lambda} \leq \lceil \frac{\max_i d_i - \min_j r_j}{\lambda} \rceil \Rightarrow |l-k| < \hat{\omega} + 1$. □

Theorem 8 *Given a λ-feasible modulo scheduling problem, any s-successive λ-stationary p-unwinded schedule with $s \geq \Omega$ is an interesting schedule.*

PROOF. Assume a Ω-successive λ-stationary p-unwinded schedule starting at iteration q. The proof principle is to show that the dates $\{\sigma_i \stackrel{\text{def}}{=} \sigma_i^q\}_{1 \leq i \leq n}$ satisfy the constraints of the modulo scheduling problem at period λ.

For any uniform dependence $O_i \overset{\theta_i^j,\omega_i^j}{\prec} O_j$ of the modulo scheduling problem, the corresponding constraint is $\sigma_i + \theta_i^j - \lambda\omega_i^j \leq \sigma_j$, that is, $\sigma_i^q + \theta_i^j \leq \sigma_j^q + \lambda\omega_i^j$. The p-unwinded schedule satisfies $\sigma_i^q + \theta_i^j \leq \sigma_j^{q+\omega_i^j}$ and we have $\sigma_j^{q+\omega_i^j} = \sigma_j^q + \lambda\omega_i^j$ from Definition 4 since $\omega_i^j \leq \bar{\omega} \leq \Omega$, so $\sigma_i^q + \theta_i^j \leq \sigma_j^q + \lambda\omega_i^j$.

For the resources, assume the dates $\{\sigma_i \overset{\text{def}}{=} \sigma_i^q\}_{1\leq i\leq n}$ violate the cumulative modulo resource constraints. This implies that $\exists t, \exists J \subseteq [1,n], \exists k_{i_1}, k_{i_2}, \ldots \in \mathbb{N} : t \in \cap_{j\in J}[\sigma_{i_j}+k_{i_j}\lambda, \sigma_{i_j}+k_{i_j}\lambda+p_{i_j}[\wedge \sum_{j\in J} \vec{b}_{i_j} > \vec{B}$. Let $m \in J$ such that k_{i_m} is minimal. Then all the intervals $[\sigma_{i_j}+(k_{i_j}-k_{i_m})\lambda, \sigma_{i_j}+(k_{i_j}-k_{i_m})\lambda+p_{i_j}[$ overlap at date $t-k_{i_m}\lambda$. For any $j \in J$, the date $\sigma_{i_j}+(k_{i_j}-k_{i_m})\lambda$ satisfies the release date and deadline constraints of operation $O_{i_j}^{q+k_{i_j}-k_{i_m}}$. We apply Lemma 7 to the operations pairs $(O_{i_m}^q, O_j^l \in \{O_{i_j}^{q+k_{i_j}-k_{i_m}}\}_{j\in J})$ so $\forall j \in J : 0 \leq k_{i_j}-k_{i_m} \leq \hat{\omega}$. Thanks to $\hat{\omega}$-successive λ-stationarity starting at iteration q, we have $\forall j \in J$: $\sigma_{i_j} + (k_{i_j} - k_{i_m})\lambda = \sigma_{i_j}^{q+k_{i_j}-k_{i_m}}$ and this yields the contradiction that the p-unwinded schedule has resource conflicts. So the dates $\{\sigma_i \overset{\text{def}}{=} \sigma_i^q\}_{1\leq i\leq n}$ do not violate the cumulative modulo resource constraints. □

Definition 9 *Two scheduling problems are equivalent iff the solution of one yields a solution of the other in pseudo-polynomial time or both are infeasible.*

Corollary 10 *A modulo scheduling problem at period λ is equivalent to any p-unwinded scheduling problem that has a Ω-successive λ-stationary schedule.*

PROOF. First assume a feasible modulo scheduling problem at period λ. By Proposition 5, this yields a p-successive λ-stationary p-unwinded schedule, including $p \geq \Omega$. Conversely, assume a p-unwinded scheduling problem that has a Ω-successive λ-stationary schedule. By Theorem 8, it is an interesting schedule. This yields a modulo schedule in pseudo-polynomial time. □

2.3 Regular Unwinded Scheduling Problems

We proved in §2.2 that λ-stationarity for Ω iterations yields a modulo schedule. We show now that λ-regularity for enough iterations implies λ-stationarity. Intuitively, if a λ-regular p-unwinded schedule is not λ-stationary, then the distance between some schedule dates σ_i^k and their deadline d_i^k strictly decreases as p increases, so for large enough p the p-unwinded problem is infeasible.

Definition 11 *Given a p-unwinded scheduling problem at period λ with operation set $\{O_i^k\}_{1\leq i\leq n}^{1\leq k\leq p}$, a corresponding schedule $\{\sigma_i^k\}_{1\leq i\leq n}^{1\leq k\leq p}$ is* regular *if: $\forall i \in$*

$[1,n], \forall k \in [1,p-1] : \sigma_i^k + \lambda \leq \sigma_i^{k+1}$.

Lemma 12 *If a λ-regular p-unwinded schedule is not s-successive λ-stationary, then: $\forall q \in [1,p-s], \exists i \in [1,n] : \sigma_i^q + s\lambda < \sigma_i^{q+s}$.*

PROOF. For any p-unwinded schedule that is not s-successive λ-stationary, by Definition 4: $\forall q \in [1,p-s], \exists i \in [1,n], \exists k \in [q,q+s-1] : \sigma_i^k + \lambda \neq \sigma_i^{k+1}$. By λ-regularity, $\sigma_i^l + \lambda \leq \sigma_i^{l+1}, \forall l \in [1,p-1]$. Adding these inequalities for $l \in [q,q+s-1]$ yields the result, as $\exists k \in [q,q+s-1] : \sigma_i^k + \lambda < \sigma_i^{k+1}$. □

Theorem 13 *Given a λ-feasible modulo scheduling problem, $\forall s > 0, \exists r > 0, \forall p > r$: any λ-regular p-unwinded pseudo-schedule is s-successive λ-stationary.*

PROOF. By contradiction, assume $\exists s > 0, \forall r > 0, \exists p > r$: a λ-regular p-unwinded pseudo-schedule is not s-successive λ-stationary. From Lemma 12 this implies $\forall q \in [1,p-s], \exists i \in [1,n] : \sigma_i^q + s\lambda + 1 \leq \sigma_i^{q+s}$.

From the definition of the deadlines of the p-unwinded scheduling problem, we have $d_i^q + s\lambda = d_i^{q+s}$, so each inequality becomes $\forall q \in [1,p-s], \exists i \in [1,n] : \sigma_i^q - d_i^q + 1 \leq \sigma_i^{q+s} - d_i^{q+s}$.

Summing over $i \in [1,n]$ yields the inequality $\forall q \in [1,p-s] : \sum_{1 \leq i \leq n}(\sigma_i^q - d_i^q) + 1 \leq \sum_{1 \leq i \leq n}(\sigma_i^{q+s} - d_i^{q+s})$. Then for each $q \in (1, 1+s, \ldots 1+zs)$, with $z \stackrel{\text{def}}{=} \lfloor \frac{p-1}{s} \rfloor$, we sum this inequality and obtain $\sum_{1 \leq i \leq n}(\sigma_i^1 - d_i^1) + z \leq \sum_{1 \leq i \leq n}(\sigma_i^{1+zs} - d_i^{1+zs})$.

By increasing $z \stackrel{\text{def}}{=} \lfloor \frac{p-1}{s} \rfloor$, we get $\sum_{1 \leq i \leq n}(\sigma_i^1 - d_i^1) + z \geq 0$, so $0 \leq \sum_{1 \leq i \leq n}(\sigma_i^{1+zs} - d_i^{1+zs}) \Rightarrow \exists i : \sigma_i^{1+zs} \geq d_i^{1+zs}$ and the λ-regular p-unwinded pseudo-schedule is infeasible for any $z \geq \sum_{1 \leq i \leq n}(\mathrm{d}_i^1 - \sigma_i^1)$.

In particular, $\sum_{1 \leq i \leq n}(d_i - r_i) = \sum_{1 \leq i \leq n}(\mathrm{d}_i^1 - r_i^1) \geq \sum_{1 \leq i \leq n}(d_i^1 - \sigma_i^1)$, so $z \geq \sum_{1 \leq i \leq n}(d_i - r_i)$ ensures infeasibility. Finally, a choice of $r \geq s \sum_{1 \leq i \leq n}(d_i - r_i) + s$ ensures that the λ-regular p-unwinded pseudo-schedule is infeasible. This completes the contradiction. □

Corollary 14 *Given a λ-feasible modulo scheduling problem, $\forall s > 0$, if $p > s \sum_{1 \leq i \leq n}(d_i - r_i) + s$ then any λ-regular p-unwinded pseudo-schedule is s-successive λ-stationary.*

Corollary 15 *A modulo scheduling problem is equivalent to any of its λ-regular p-unwinded scheduling problems, with $p > \bar{\rho} \stackrel{def}{=} \Omega \sum_{1 \leq i \leq n}(d_i - r_i) + \Omega$.*

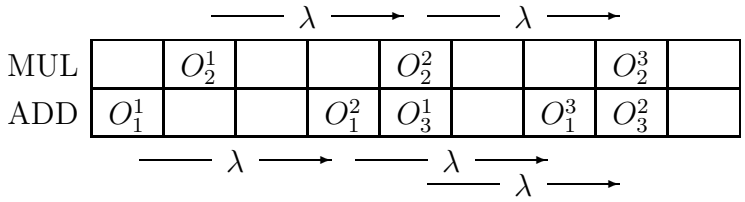

Fig. 5. Schedule for the 3-regular unwinded problem.

PROOF. Corollary 14 ensures that any λ-regular p-unwinded schedule with $p > \Omega \sum_{1 \leq i \leq n}(d_i - r_i) + \Omega$ is Ω-successive λ-stationary. By Corollary 10, this is equivalent to modulo scheduling. □

Corollary 15 provides the foundations of the regular unwinding framework: to solve a modulo scheduling problem at period λ, build a λ-regularized p-unwinded scheduling problem with $p > \bar{\rho}$. In practice, λ-regularizing a p-unwinded scheduling problem means adding enough constraints $\sigma_i^k + \lambda \leq \sigma_i^{k+1}$ (that is, dependence arcs $O_i^k \prec O_i^{k+1}$ of latency λ) to ensure that the p-unwinded schedules are λ-regular. Either this λ-regularized p-unwinded scheduling problem is feasible and we convert its solution to a modulo schedule; or, it is not feasible and there is no modulo schedule at period λ.

In Figure 5, we illustrate regular unwinding for $\lambda = 3$.

2.4 Semi-Regular Unwinded Scheduling Problems

Consider the cases where the p-unwinded schedule is only *partially λ-regular*, that is, λ-regularity only holds for the instances of a subset $\{O_i\}_{i \in R}$ of the generic operations, with $R \subset [1, n]$. Theorem 13 generalizes as follows:

Theorem 16 *Given a λ-feasible modulo scheduling problem, $\forall s > 0, \exists r > 0, \forall p > r$: any p-unwinded pseudo-schedule that is λ-regular on the index set $R \subset [1, n]$ is s-successive λ-stationary on $\{\sigma_i^k\}_{i \in R}^{1 \leq k \leq p}$.*

PROOF. Given the hypothesis, the theorem states that $\exists q \in [1, p - s], \forall i \in R, \forall k \in [q, q + s - 1] : \sigma_i^k + \lambda = \sigma_i^{k+1}$ (Definition 4). The proof is identical to the proof of Theorem 13, except that $i \in [1, n]$ is replaced by $i \in R$. □

Corollary 17 *Given a λ-feasible modulo scheduling problem and $R \subset [1, n]$, $\forall s > 0$, if $p > s \sum_{i \in R}(d_i - r_i) + s$ then any p-unwinded pseudo-schedule λ-regular on R is s-successive λ-stationary on $\{\sigma_i^k\}_{i \in R}^{1 \leq k \leq p}$.*

Among the partially λ-regular p-unwinded scheduling problems, we are especially interested in cases where R equals D, the index set of the generic operations that are not independent.

Definition 18 *A* λ-semi-regularized p-unwinded scheduling problem *is an unwinded scheduling problem where the* λ*-regularizing dependences are only added to the instances of the generic operations that are not independent.*

Definition 19 *The* dependent schedule *of a* p*-unwinded scheduling problem is the partial schedule* $\{\sigma_i^k\}_{i\in D}^{1\leq k\leq p}$*, with* D *the index set of the generic operations that are not independent in the modulo scheduling problem.*

Let us now focus on the properties of the dependence-consistent release dates of p-unwinded scheduling problems in case of λ-semi-regularization.

Lemma 20 *For any feasible* λ*-semi-regularized* p*-unwinded scheduling problem, the set of dependence-consistent release dates* $\{r'^k_i\}_{1\leq i\leq n}^{1\leq k\leq p}$ *defined as* $r'^k_i \stackrel{def}{=} \max(r_i^k, \max_{O_j^l\in \mathrm{pred}\, O_i^k}(r_j^l + p_j + l_j^{i+}))$ *is a* λ*-regular pseudo-schedule.*

PROOF. By definition, the set $\{r'^k_i\}_{1\leq i\leq n}^{1\leq k\leq p}$ satisfies the release dates and the dependence constraints of the p-unwinded scheduling problem. It also satisfies the deadlines else the p-unwinded scheduling problem is infeasible, so it is a pseudo-schedule. The λ-regularity follows from the λ-semi-regularization and from the λ-stationarity of the release dates of the independent operations. □

Definition 21 *A* q*-full* λ*-stationary schedule is a solution to a* p*-unwinded scheduling problem such that:* $\forall i \in [1,n], \forall k \in [q, p-1] : \sigma_i^k + \lambda = \sigma_i^{k+1}$

In other words, the q-full λ-stationary schedules are the s-successive λ-stationary p-unwinded schedules starting at iteration q with $q = p - s$.

Theorem 22 *For any feasible* λ*-semi-regularized* p*-unwinded scheduling problem, if the pseudo-schedule of the dependence-consistent release dates* $\{r'^k_i\}_{1\leq i\leq n}^{1\leq k\leq p}$ *is* $\bar{\omega}$*-successive* λ*-stationary starting at iteration* q*, then it is* q*-full* λ*-stationary.*

PROOF. The dependence-consistent release dates of the independent operations are λ-stationary, so we focus on the dependent operations. By hypothesis, $\exists q \in [1, p-\bar{\omega}], \forall i \in D, \forall k \in [q, q+\bar{\omega}-1] : r'^k_i + \lambda = r'^{k+1}_i$ (Definition 4). We need to prove $\forall i \in D, \forall k \in [q, p-1] : r'^k_i + \lambda = r'^{k+1}_i$ (Definition 21). This is done by induction once we show that $\forall q \in [1, p-\bar{\omega}]$, if $\forall i \in D, \forall k \in [q, q+\bar{\omega}-1] : r'^k_i + \lambda = r'^{k+1}_i$ then $\forall i \in D, r'^{q+\bar{\omega}}_i + \lambda = r'^{q+\bar{\omega}+1}_i$.

The set $\{r'^k_i\}_{i\in D}^{1\leq k\leq p}$ can be seen as the result of a longest path computation on the λ-semi-regularized p-unwinded problem dependence graph, augmented

with arcs of latency r_i between a root node and the dependent operations of the first iteration $\{O_i^1\}_{i\in D}$. The λ-regularizing arcs ensure $r'^k_i \geq r_i + k\lambda, \forall i \in D$.

By definition of a pseudo-schedule, the entering dependences of any dependent operation in iteration $q+\bar{\omega}$ are satisfied by the dependence-consistent release dates: $O_i^{q+\bar{\omega}-\omega_i^j} \overset{\theta_i^j}{\prec} O_j^{q+\bar{\omega}} \Rightarrow r'^{q+\bar{\omega}-\omega_i^j}_i + \theta_i^j \leq r'^{q+\bar{\omega}}_j$. First consider the carried dependences, that is, $\omega_i^j > 0$. As $\forall i \in D, \forall k \in [q, q+\bar{\omega}-1] : r'^k_i + \lambda = r'^{k+1}_i$, the dependence inequality becomes $r'^{q+\bar{\omega}-\omega_i^j+1}_i + \theta_i^j \leq r'^{q+\bar{\omega}}_j + \lambda$ after adding λ to both sides. Thus the date $r'^{q+\bar{\omega}}_j + \lambda$ satisfies the carried entering dependences of $O_j^{q+\bar{\omega}+1}, \forall j \in D$. Now consider the non-carried dependences, that is, $\omega_i^j = 0$. The dependence inequality is $r'^{q+\bar{\omega}}_i + \theta_i^j \leq r'^{q+\bar{\omega}}_j \Rightarrow r'^{q+\bar{\omega}}_i + \lambda + \theta_i^j \leq r'^{q+\bar{\omega}}_j + \lambda$, so the dates $\{r'^{q+\bar{\omega}}_j + \lambda\}_{j\in D}$ satisfy the non-carried dependences of iteration $q+\bar{\omega}+1$. Finally, no smaller pseudo-schedule dates are possible for $\{O_j^{q+\bar{\omega}+1}\}_{j\in D}$ by λ-semi-regularization. This implies $\forall j \in D, r'^{q+\bar{\omega}+1}_j = r'^{q+\bar{\omega}}_j + \lambda$. □

Corollary 23 *For any feasible λ-semi-regularized p-unwinded scheduling problem, if $p > \rho^D \overset{def}{=} \bar{\omega}\sum_{i\in D}(d_i - r_i) + \bar{\omega}$, then the pseudo-schedule of the dependence-consistent release dates $\{r'^k_i\}_{1\leq i\leq n}^{1\leq k\leq p}$ is $\rho^D - \bar{\omega}$-full λ-stationary.*

PROOF. From Corollary 17, if $p > \rho^D$ then the pseudo-schedule of the dependence-consistent release dates is $\bar{\omega}$-successive λ-stationary. This successive λ-stationarity appears at the latest at iteration $\rho^D - \bar{\omega}$. □

3 UET Modulo Scheduling on Parallel Machines

Although the regular unwinding framework applies to the resource-constrained modulo scheduling problems, we now focus on parallel machines with UET operations in order to obtain pseudo-polynomial time solutions.

3.1 Dependence-Free Modulo Scheduling

We start with a sufficient condition for the GLSA to build λ-regular schedules, even when the p-unwinded problem is not λ-regularized.

Theorem 24 *In case of UET operations and parallel machines, running the GLSA on a p-unwinded scheduling problem without dependences and with a priority function P such that $P(O_i^k) < P(O_j^l) \Rightarrow P(O_i^{k+1}) < P(O_j^{l+1})$ yields a λ-regular p-unwinded schedule.*

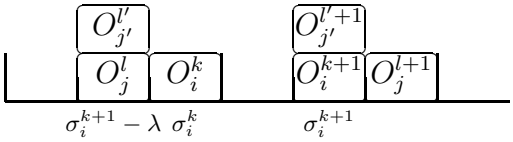

Fig. 6. Illustration of the proof of Theorem 24.

PROOF. [Figure 6] Assume that the p-unwinded schedule is not λ-regular. Let O_i^k be the earliest scheduled operation such that $\sigma_i^k + \lambda > \sigma_i^{k+1}$. At date $\sigma_i^{k+1} - \lambda$, operation O_i^k is available, as $r_i^k = r_i^{k+1} - \lambda \leq \sigma_i^{k+1} - \lambda$. Since operation O_i^k was not scheduled at date $\sigma_i^{k+1} - \lambda < \sigma_i^k$ even though it was available and from the UET hypothesis, there must exist operations $O_j^l, O_{j'}^{l'}, \ldots$ that are all scheduled at date $\sigma_i^{k+1} - \lambda$, meaning they have a higher priority than O_i^k. At date σ_i^{k+1}, the operations $O_j^{l+1}, O_{j'}^{l'+1}, \ldots$ are also available, because $\sigma_j^l = \sigma_i^{k+1} - \lambda \Rightarrow r_j^l \leq \sigma_i^{k+1} - \lambda \Rightarrow r_j^{l+1} \leq \sigma_i^{k+1}$. These operations cannot be scheduled earlier, that is, $\sigma_j^{l+1} < \sigma_i^{k+1}$, because $\sigma_j^{l+1} \geq \sigma_j^l + \lambda$ by λ-regularity and $\sigma_j^l = \sigma_i^{k+1} - \lambda$. Thus, operation O_i^{k+1} is scheduled before at least one of the operations $O_j^{l+1}, O_{j'}^{l'+1}, \ldots$, meaning it has a higher priority. This contradicts the hypothesis on the priority function P. □

Corollary 25 *The modulo scheduling problem $P|r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ can be solved in pseudo-polynomial time.*

PROOF. Build the p-unwinded scheduling problem with $p > \bar{\rho}$. This yields a problem $P|r_i; d_i; p_i = 1|\bullet$ that is optimally solved by the GLSA using the earliest d_i first priority. The priority $P(O_i^k) \stackrel{\text{def}}{=} d_i^k \stackrel{\text{def}}{=} d_i + (k-1)\lambda$ of a p-unwinded scheduling problem satisfies the condition of Theorem 24 [2], so the resulting schedule if feasible is λ-regular for $\bar{\rho}$ iterations. By Corollary 15, this is equivalent to modulo scheduling. □

3.2 Modulo Scheduling a Dependence Circuit

Consider modulo scheduling problems that include uniform dependences in addition to release dates and deadlines. When the p-unwinded problem is λ-regularized, this yields a dependence structure that is too complex in general for pseudo-polynomial time solutions. However we show now that regular unwinding of a dependence circuit yields a monotone chain-order graph, which is optimally scheduled by the LPPA.

[2] In case O_i^k and O_j^l have the same deadline, we order them by lower index i, j first.

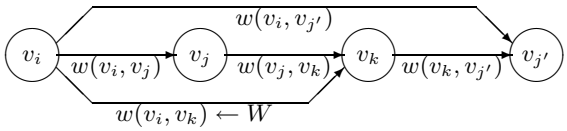

Fig. 7. Illustration of the proof of Theorem 27.

Lemma 26 *For any monotone chain-order graph, if increasing the weight of some arc violates the monotone property, then it can be restored without changing the transitive weights.*

PROOF. [Figure 7] Assume a chain order whose arc weights w satisfy the monotone property: $\forall (v_i, v_j), (v_i, v_k) \in E : \operatorname{pred} v_j \subseteq \operatorname{pred} v_k \Rightarrow w(v_i, v_j) \leq w(v_i, v_k)$. Select arbitrary v_i, v_k and increase $w(v_i, v_k)$ to some value W.

Consider the v_i successors v_j and $v_{j'}$. For v_j between v_i and v_k in the chain, $w(v_i, v_j) \leq w(v_i, v_k)$ still holds. For $v_{j'}$ after v_k in the chain, $w(v_i, v_k) \leq w(v_i, v_{j'})$ may no longer hold. Increasing $w(v_i, v_{j'})$ to $w(v_i, v_k) = W$ for all nodes $v_{j'}$ after v_k in the chain restores the monotone property. This increase does not change the transitive weights because $w(v_i, v_{j'}) = w(v_i, v_k) \leq w(v_i, v_k) + w(v_k, v_{j'})$, thanks to the non-negativity of the weight function. □

Theorem 27 *Unwinding a problem* $P|circuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ *and* λ*-regularizing it yields a* $P|chainOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$ *problem.*

PROOF. Unwinding a problem $P|circuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ yields a dependence graph that is a chain [3].

For any nodes v_i, v_j, v_k such that v_j is the direct successor of v_i in and v_k a transitive successor of v_i in the chain, create arc (v_i, v_k) and assign weight θ_i^j to all arcs leaving v_i. The result is a monotone chain order whose transitive latencies are the same as the transitive latencies of the original chain, so the scheduling problem constraints are unchanged.

Now add the λ-regularizing dependence arcs to the monotone chain order graph. Each such arc is parallel to an existing arc (v_i, v_k) and if not redundant, this has the effect of increasing $w(v_i, v_k)$ to λ. By Lemma 26, we restore the monotone property without changing the scheduling constraints. □

Corollary 28 *The modulo scheduling problem* $P|circuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ *can be solved in pseudo-polynomial time.*

[3] If $\sum \omega_i^j = k$ the p-unwinded dependence graph has k independent chains.

PROOF. Build the λ-regularized p-unwinded scheduling problem with $p > \bar{\rho}$. By Theorem 27, this yields a $P|chainOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$ problem. A monotone chain order is a particular case of monotone interval order, which is optimally solved by the LPPA. As any feasible schedule is λ-regular for $\bar{\rho}$ iterations, by Corollary 15 this is equivalent to modulo scheduling. □

Corollary 29 *Unwinding a problem* $P|subCircuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ *and* λ*-semi-regularizing it yields a* $P|subChainOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$ *problem.*

PROOF. First consider the operations in the dependence circuit. After unwinding, their instances are λ-regularized. Proof of Theorem 27 ensures that the resulting problem is $P|chainOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$. Now add the instances of the independent operations to the p-unwinded problem. Thanks to λ-semi-regularization, these operations instances remain independent. The end result is a $P|subChainOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$ problem. □

3.3 Properties of the Leung-Palem-Pnueli Pseudo-Schedule

Given a p-unwinded scheduling problem with UET operations, consider the pseudo-schedule $\{\sigma^{*k}_i \stackrel{\text{def}}{=} d'^k_i - 1\}_{1\le i\le n}^{1\le k\le p}$ computed by the ModifiedDeadlines algorithm of Leung, Palem and Plueli described in §1.3. These dates satisfy the dependence constraints of the p-unwinded problem, so if the p-unwinded problem is λ-regularized, its LPPA pseudo-schedule is λ-regular. A more difficult question we study in this section is whether the LPPA pseudo-schedule of a λ-semi-regularized p-unwinded scheduling problem is λ-regular.

Lemma 30 *Given a feasible* p*-unwinded scheduling problem,* $\{r'^k_i\}_{1\le i\le n}^{1\le k\le p}$ *its dependence-consistent release dates and* $O_i^k, i \in [1,n] - D$ *any independent operation, the supporting set of* O_i^k *is included in* $\{O_j^l\}_{1\le j\le n}^{\max(1,k-\hat{\omega})\le l\le p}$.

PROOF. Leung, Palem and Pnueli [10] characterize the supporting set at date t of operation O_i^k as $\mathsf{Support}(O_i^k, t) \stackrel{\text{def}}{=} \operatorname{succ} O_i^k \cup \{O_j^l \in \operatorname{indep} O_i^k : r'^l_j \ge t\}$. In case of the independent operations, $\operatorname{succ} O_i^k = \emptyset$ and taking $t = r'^k_i$ provides a safe approximation of the supporting set of O_i^k. Moreover, $r'^l_j \ge r'^k_i \Rightarrow d_j^l > r_i^k \Rightarrow (k-l)\lambda < d_j - r_i \Rightarrow k - l \le \hat{\omega} \Rightarrow l \ge k - \hat{\omega}$. □

Theorem 31 *For any* λ*-semi-regularized* p*-unwinded scheduling problem, if the pseudo-schedule of the dependence-consistent release dates* $\{r'^k_i\}_{1\le i\le n}^{1\le k\le p}$ *is*

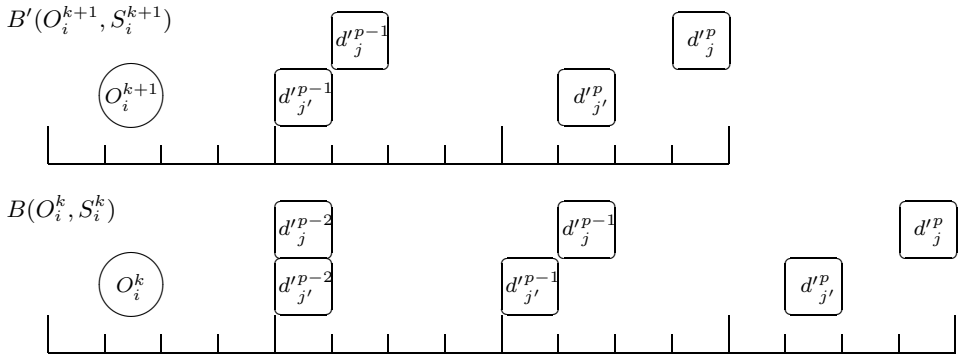

Fig. 8. Illustration of the proof of Theorem 31 with $\lambda = 4$.

q-full λ-stationary, the fixpoint modified deadlines of the algorithm of Leung, Palem and Pnueli are λ-regular starting at iteration $q + \hat{\omega}$.

PROOF. Follow the BackwardSchedule algorithm, which processes the operations in an order compatible with the highest dependence-consistent release dates r'^q_i first and starts from an initial set of deadlines that are the dependence-consistent deadlines. The dependence-consistent deadlines are λ-stationary in case of independent operations and are λ-regular in case of dependent operations thanks to λ-semi-regularization. So the dependence-consistent deadlines are λ-regular: $\forall i \in [1, n], \forall k \in [1, p-1] : d'^k_i + \lambda \leq d'^{k+1}_i$.

A first remark is that the fixpoint modified deadlines of the dependent operations $\{d^{*k}_i\}^{1 \leq l \leq p}_{i \in D}$ are guaranteed to be λ-regular. Indeed, in case O_i^k is a dependent operation, O_i^{k+1} is its direct successor by λ-semi-regularization, so the backward scheduling problem $B(O_i^k, S_i^k)$ is such that $O_i^{k+1} \in S_i^k$. The regularizing arc between O_i^k and O_i^{k+1} ensures $d^{*k}_i + \lambda \leq d^{*k+1}_i$. So there remains to prove that the fixpoint modified deadlines of the independent operations are λ-regular: $\forall i \in [1, n] - D, \forall k \in [q + \hat{\omega}, p-1] : d^{*k}_i + \lambda \leq d^{*k+1}_i$.

[Figure 8] Assume $i \in [1, n] - D$ and $k \in [q + \hat{\omega}, p]$. Because O_i^k is an independent operation, the fixpoint modified deadline d^{*k}_i is the result of optimal backward scheduling on problem $B(O_i^k, S_i^k)$, with $S_i^k \stackrel{\text{def}}{=} \{O_j^l\}^{\max(1,k-\hat{\omega}) \leq l \leq p}_{1 \leq j \leq n}$ by Lemma 30. Define $B'(O_i^{k+1}, S_i^{k+1})$ as problem $B(O_i^{k+1}, S_i^{k+1})$ with time translated by $-\lambda$. We claim that $B(O_i^k, S_i^k)$ is more constrained than $B'(O_i^{k+1}, S_i^{k+1})$, whose optimal backward scheduling resulted in $d^{*k+1}_i - \lambda$. First, the dependent-consistent release dates are λ-stationary for all the operations involved in S_i^k, because $k \geq q + \hat{\omega}$ and thanks to the q-full λ-stationarity of the modified release dates. Second, consider any deadline d'^l_j or d^{*l}_j that constrains $B(O_i^k, S_i^k)$. If $l = p$, there is no corresponding deadline in $B'(O_i^{k+1}, S_i^{k+1})$. Else, the corresponding deadline is $d'^{l+1}_j - \lambda \geq d'^l_j$ or $d^{*l+1}_j - \lambda \geq d^{*l}_j$ by the λ-regularity of the dependence-consistent deadlines and of the already computed fixpoint modified deadlines. In any case, $B(O_i^k, S_i^k)$ is more constrained than $B'(O_i^{k+1}, S_i^{k+1})$ and this implies $d^{*k}_i \leq d^{*k+1}_i - \lambda$. □

Corollary 32 *For any feasible λ-semi-regularized p-unwinded scheduling problem, if the dependence-consistent release dates are q-full λ-stationary, the fixpoint modified deadlines of Leung, Palem and Pnueli are λ-stationary from iteration $q+\hat{\omega}$ to iteration $p-\bar{\rho}+\Omega$ with $\bar{\rho} \stackrel{def}{=} \Omega \sum_{1 \leq i \leq n}(d_i - r_i) + \Omega$.*

PROOF. By Theorem 31, the LPPA pseudo-schedule is λ-regular starting at iteration $q+\hat{\omega}$. Corollary 14 applies to the pseudo-schedule $\{\sigma^{*k}_i\}^{q+\hat{\omega} \leq k \leq p}_{1 \leq i \leq n}$, so for $p > q+\hat{\omega}-1+\bar{\rho}$, this pseudo-schedule is Ω-successive λ-stationary. As the ModifiedDeadlines algorithm builds the pseudo-schedule in reverse topological sort order, Ω-successive λ-stationarity appears first in the higher iterations, then extends to the lower iterations because optimal backwards scheduling maximizes the pseudo-schedule dates. □

This result is useful by itself for heuristic modulo scheduling: when applying the LPPA to a λ-semi-regularized p-unwinded scheduling problem, there is no need to run the BackwardSchedule algorithm after the fixpoint modified deadlines become λ-regular for Ω iterations.

3.4 Modulo Scheduling a Dependence Sub-Circuit

While Corollary 28 provides a first pseudo-polynomial solution to modulo scheduling problems that involve both a dependence circuit with dependence delays and resource constraints, there remain strong motivations to include independent operations outside the dependence circuit in modulo scheduling problems. Indeed, given a complex modulo scheduling problem, the relaxation obtained by keeping only its most constraining dependence circuit and making the operations outside this circuit independent would be stronger than the existing practice of computing λ_{res} and λ_{rec} independently (§2.1).

Proposition 33 *Given a feasible problem $P|subCircuit(\theta^j_i, \omega^j_i); \sum \omega^j_i = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ at period λ, the pseudo-schedule of the dependence-consistent release dates $\{r'^k_i\}^{1 \leq k \leq p}_{1 \leq i \leq n}$ of its λ-semi-regularized p-unwinded problem is 2-full λ-stationary.*

PROOF. Consider the longest path problem introduced in the proof of Theorem 22 to compute the dependence-consistent release dates of the dependent operations. Assume $\exists i \in D : r'^2_i + \lambda < r'^3_i$. This implies the dependence circuit has a cumulative latency greater that λ, so the modulo scheduling problem is not feasible at period λ. Thus, whenever the modulo scheduling problem is feasible at period λ, $\forall i \in D : r'^2_i + \lambda = r'^3_i$, then Theorem 22 applies. □

Theorem 34 *Assume a GLSA schedule of a λ-semi-regularized p-unwinded scheduling problem such that the dependent schedule $\{\sigma_i^k\}_{i\in D}^{1\leq k\leq p}$ is s-successive λ-stationary starting at iteration q. If the GLSA priority function P satisfies $\forall k,l \in [q, q+s-1] : P(O_i^k) < P(O_j^l) \Rightarrow P(O_i^{k+1}) < P(O_j^{l+1})$, then the partial schedule $\{\sigma_i^k\}_{1\leq i\leq n}^{q+\hat{\omega}\leq k\leq q+s-\hat{\omega}}$ is λ-regular.*

PROOF. The schedule dates of the dependent operations are λ-stationary hence λ-regular for all iterations in $[q, q+s]$, so we only need to show that the schedule dates of the independent operations are regular for all iterations $k \in [q+\hat{\omega}, q+s-\hat{\omega}]$.

By Lemma 7, any two operations O_i^k and O_j^l can only overlap if $|l-k| \leq \hat{\omega}$, so the independent operations of iterations $k \in [q+\hat{\omega}, q+s-\hat{\omega}]$ can only overlap with the dependent operations whose iterations $l \in [q, q+s]$.

[Figure 6] Follow the proof principles of Theorem 24, a contradiction based on the definition of O_i^k as the earliest scheduled operation such that $\sigma_i^k + \lambda > \sigma_i^{k+1}$. In the current proof, operation O_i^k can only be independent by hypothesis. The proof of Theorem 24 considers operations $O_j^l, O_{j'}^{l'}, \ldots$ scheduled at date $\sigma_i^{k+1} - \lambda$ and needs that operations $O_j^{l+1}, O_{j'}^{l'+1}, \ldots$ be available at date σ_i^{k+1}. In the current proof, some O_j^l may be dependent. However the λ-stationarity of the dependent operations ensures that O_j^{l+1} is available at date σ_i^{k+1}, so the remainder of the proof of Theorem 24 carries unchanged. □

Corollary 35 *The modulo scheduling problem $P|subCircuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ can be solved in pseudo-polynomial time.*

PROOF. Build the λ-semi-regularized p-unwinded scheduling problem. By Corollary 29, this yields $P|subChainOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$. This problem is a particular case of $P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$, which is optimally solved by the LPPA.

Let p_1 be some unwinding amount. By Proposition 33, for the p_1-unwinded problems we consider, the pseudo-schedule of the dependence-consistent release dates $\{r'^k_i\}_{1\leq i\leq n}^{1\leq k\leq p_1}$ is 2-full λ-stationary. By Corollary 32, the LPPA pseudo-schedule $\{\sigma^{*k}_i \stackrel{\text{def}}{=} d'^k_i - 1\}_{1\leq i\leq n}^{1\leq k\leq p_1}$ is λ-stationary from iteration $2+\hat{\omega}$ to iteration $p_1 - \bar{\rho} + \Omega$. So for the LPPA GLSA that uses the fixpoint modified deadlines $P(O_i^k) \stackrel{\text{def}}{=} d'^k_i$ as priorities, we have $\forall k,l \in [2+\hat{\omega}, p_1 - \bar{\rho} + \Omega - 1], \forall i,j \in [1,n] : P(O_i^k) < P(O_j^l) \Rightarrow P(O_i^{k+1}) < P(O_j^{l+1})$.

Select p_2 such that the dependent schedule constructed by this GLSA is s_2-successive λ-stationary starting at iteration q_2, under the requirements $q_2 \geq$

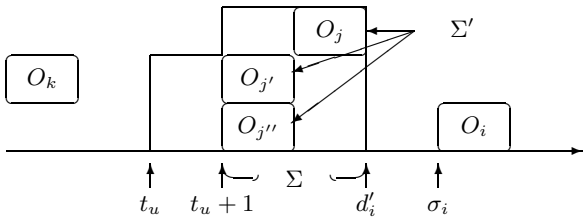

Fig. 9. The core proof of the Leung-Palem-Pnueli algorithm.

$2 + \hat{\omega}$ and $s_2 \geq \bar{\rho} + 2\hat{\omega}$. This is achieved by any dependent schedule that is s_3-successive λ-stationary with $s_3 \geq s_2 + 2 + \hat{\omega}$. By Corollary 17, this is ensured for any $p_2 > s_3 \sum_{i \in D}(d_i - r_i) + s_3$. Finally, by taking $p \stackrel{\text{def}}{=} \max(p_1 + \bar{\rho} - \Omega, p_2)$, Theorem 34 ensures the p-unwinded schedule is λ-regular for $\bar{\rho}$ successive iterations. As any feasible p-unwinded schedule is λ-regular for $\bar{\rho}$ iterations, by Corollary 15 this is equivalent to modulo scheduling. □

4 UET Modulo Scheduling on Typed Task Systems

In order to generalize the parallel machine modulo scheduling problems of Section 3 to typed task systems, we need to extend the algorithm of Leung, Palem and Plueli [10] to solve $\Sigma P|subChainOrder(monol_i^j); r_i; d_i; p_i = 1|\bullet$.

4.1 The Leung-Palem-Pnueli Algorithm Correctness Arguments

The correctness of the Leung-Palem-Pnueli algorithm (LPPA) is based on two arguments. The first proves that the fixpoint modified deadlines computed by the ModifiedDeadlines and the BackwardSchedule algorithms are deadlines of the original problem, so a schedule that misses a modified deadline on a feasible problem is not optimal. Moreover, in [10] Leung, Palem and Pnueli prove the correctness of the ModifiedDeadlines and the BackwardSchedule algorithms without relying on the dependence structure of the scheduling problem.

The second correctness argument of the LPPA is that the GLSA with the lowest fixpoint modified deadlines first as priorities does not miss any fixpoint modified deadlines. Let us call *core* this GLSA. Let O_i be the earliest operation that misses its fixpoint modified deadline d'_i in the core GLSA schedule. Leung, Palem and Pnueli [10] show that an earlier operation O_k necessarily misses its fixpoint modified deadline d'_k in the same schedule for each particular class of problems considered. This contradiction on the choice of O_i ensures that the core GLSA schedule does not miss any fixpoint modified deadline.

The details of the proof rely on a few definitions and observations illustrated

in Figure 9. An operation O_j is said *saturated* if $d'_j \leq d'_i$. Define $t_u < d'_i$ as the latest time step that is not filled with saturated operations. If $t_u < 0$ the problem is infeasible. Else, some scheduling slots at t_u are either empty or filled with unsaturated operations. Define the operation set $\Sigma \stackrel{\text{def}}{=} \{O_j \text{ saturated} : t_u < \sigma_j < d'_i\} \cup \{O_i\}$. Define the operation subset $\Sigma' \stackrel{\text{def}}{=} \{O_j \in \Sigma : r'_j \leq t_u\}$. For any operation $O_j \in \Sigma'$, there must exist a constraining operation O_k in the core GLSA schedule that prevents the scheduling of O_j at date t_u or earlier. This constraining operation O_k is a direct predecessor of operation O_j and the precedence constraint $\sigma_k + p_k + l_k^j = \sigma_j$ implies $\sigma_k + 1 + l_k^j > t_u$.

Consider problem $P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$. In an interval order, given two operations O_i and O_j, either $\operatorname{pred} O_i \subseteq \operatorname{pred} O_j$ or $\operatorname{pred} O_j \subseteq \operatorname{pred} O_i$. This is easily understood by referring to the underlying intervals that define the interval order. Also, any transitive predecessor (successor) of some operation in an interval order is also a direct predecessor (successor). Select some $O_{j'}$ among $O_j \in \Sigma'$ such that $|\operatorname{pred} O_j|$ is minimal. Each operation in $\operatorname{pred} O_{j'}$ is a predecessor of all operations in Σ' and no predecessor of $O_{j'}$ is in Σ'. Thus, the constraining operation O_k of $O_{j'}$ is a predecessor of all operations in Σ'.

A Σ-*stable backward schedule* is any optimal backward schedule $\{\sigma'_i\}_{i:O_i \in \{O_k\} \cup S_k}$ for some $B(O_k, S_k)$ with $\Sigma \subseteq S_k$ and the same dependence-consistent release dates and fixpoint modified deadlines as seen by the core GLSA. Thanks to the stability of the fixpoint modified deadlines, we may assume that a Σ-stable backward schedule exists for any Σ and O_k. A key observation is that any Σ-stable backward schedule must slot the $m(d'_i - t_u - 1) + 1$ operations of Σ between $t_u + 1$ and d'_i on m processors. This implies that at least one operation $O_j \in \Sigma'$ has a backward schedule date $\sigma'_j \leq t_u$.

Since $\exists O_j \in \Sigma' : \sigma'_j \leq t_u$ in any Σ-stable backward schedule, we have $\sigma'_k + 1 + l_k^j \leq t_u$ and the fixpoint modified deadline d'_k of O_k is such that $d'_k + l_k^j \leq t_u$. By the monotone property, $\operatorname{pred} O_{j'} \subseteq \operatorname{pred} O_i \Rightarrow l_k^{j'} \leq l_k^j$ for $O_{j'}$ selected above and $\forall O_j \in \Sigma'$, so $d'_k + l_k^{j'} \leq t_u$. However in the core GLSA schedule $\sigma_k + 1 + l_k^{j'} > t_u$, as operation O_k is constraining $O_{j'}$. Thus O_k misses its fixpoint modified deadline d'_k, contradicting the choice of O_i.

4.2 Extension of the Leung-Palem-Pnueli Proof to Typed Task Systems

Scheduling problems on *typed task systems* generalize the parallel machine scheduling problems by assigning a type τ_i to each operation O_i and by providing a bounded number of processors of each type. Jansen [9] gives a polynomial algorithm for the problem $\Sigma P|intOrder; p_i = 1|C_{max}$.

Definition 36 *The* type distribution *of a problem* $\Sigma P|\beta|\bullet$ *is creation of as*

many parallel parallel machine scheduling problems as there are types, where:

- *Each parallel machine scheduling problem only contains the processors and the operations of a particular type.*
- *The dependence graph of each parallel machine scheduling problem is obtained from the transitive closure of the original dependence graph weighted with the transitive latencies, induced by the operations of the particular type.*
- *The release dates and deadlines of each parallel machine scheduling problem are obtained by making them dependence-consistent with the transitive closure of the original dependence graph weighted with the transitive latencies*

Solving all the parallel machine scheduling problems resulting from type distribution is only a relaxation of the original problem on typed task systems. In simple cases, solving this relaxation also solves the original problem.

Lemma 37 *The GLSA with Jackson's rule optimally solves* $\Sigma P|r_i; d_i; p_i = 1|\bullet$

PROOF. Apply type distribution of the problem $\Sigma P|r_i; d_i; p_i = 1|\bullet$. This yields independent problems $P|r_i; d_i; p_i = 1|\bullet$, each of which is optimally solved by the GLSA with Jackson's rule. □

Theorem 38 *The algorithm of Leung, Palem and Pnueli extended to typed task systems solves any feasible* $\Sigma P|intOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$.

PROOF. In order to apply the LPPA to solve typed task systems, we need to show its correctness on the two arguments discussed §4.1. For the ModifiedDeadlines and the BackwardSchedule algorithms, we keep them unchanged except that the BackwardSchedule algorithm solves a series of $\Sigma P|r_i; d_i; p_i = 1|\bullet$ to find the optimal backward schedule dates. By Lemma 37, these subproblems are optimally solved by the GLSA with Jackson's rule. Thus the first correctness argument of the LPPA, that is, the fixpoint modified deadlines are deadlines of the original problem, still holds.

The second correctness argument of the LPPA is essentially unchanged. Let τ_i be the type of the earliest operation O_i that misses its fixpoint modified deadline d_i'. The contradiction proceeds by defining t_u, Σ, Σ' on the processors and the tasks whose type equals τ_i. As in the parallel machine case, monotonicity allows to show by contradiction that the core GLSA algorithm does not miss any modified deadlines, thus proving the second correctness argument of the LPPA on typed task systems and monotone interval orders. □

By using similar arguments as in §3.4, we now show that the modulo scheduling problem $\Sigma P|subCircuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ can be solved in pseudo-polynomial time.

Corollary 39 *Unwinding* $\Sigma P|subCircuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ *and* λ*-semi-regularizing it yields a* $\Sigma P|subChainOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$ *problem.*

PROOF. Same as Corollary 29, as resource constraints are not involved. □

Theorem 40 *The modulo scheduling problem* $\Sigma P|subCircuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$ *can be solved in pseudo-polynomial time.*

PROOF. Build the λ-semi-regularized p-unwinded scheduling problem. By Corollary 39, this yields $\Sigma P|subChainOrder(mono\ l_i^j); r_i; d_i; p_i = 1|\bullet$. By Theorem 38, this problem is optimally solved by the LPPA extended to typed task systems. Then follow the proof of Corollary 35 in order to show that for any $p \geq \max(p_1 + \bar{\rho} - \Omega, p_2)$, the p-unwinded schedule is λ-regular for $\bar{\rho}$ iterations. By Corollary 15, solving the p-unwinded problem for such p is equivalent to modulo scheduling.

The proof of Corollary 35 relies on Corollary 17, Corollary 32 and Theorem 34. Corollary 17 is independent of the resource constraints. Corollary 32 and Theorem 34 ultimately rely on the optimality of Jackson's rule for scheduling $P|r_i; d_i; p_i = 1|\bullet$ with the GLSA. By Lemma 37, the GLSA with Jackson's rule optimally solves $\Sigma P|r_i; d_i; p_i = 1|\bullet$ so Corollary 32 and Theorem 34 extend to typed task systems. □

Summary and Conclusions

Modulo scheduling refers to cyclic scheduling with uniform dependences and dependence delays, where the cyclic schedules must be 1-periodic with integral period. We propose a new framework for resource-constrained modulo scheduling, whose principle is to unwind the modulo scheduling problem and to apply acyclic scheduling while ensuring that the schedule is λ-regular. Under the general case of the RCPSP resource constraints [3], we prove the equivalence between modulo scheduling at period λ and solving λ-regularized p-unwinded

problems, for $p > \bar{\rho}$ of pseudo-polynomial size: either the λ-regularized p-unwinded schedule is λ-stationary for enough iterations to yield a modulo schedule of period λ; or, there is no modulo schedule of period λ.

Solving cyclic instruction scheduling problems by combining unwinding and acyclic scheduling has been proposed earlier [2][1][6]. These approaches build K-periodic cyclic schedules [7] where K is only known after the unwinded schedule is built. Work in [2][1] focuses on heuristics for instruction scheduling. Work by Chrétienne [6] studies the performance guarantees of Graham list scheduling on 1-regular unwinded problems, but is restricted to parallel processors and dependences without delays. These techniques require that the dependence graph be a single strongly connected component.

In the area of 1-periodic cyclic scheduling techniques, state-of-the-art modulo instruction scheduling [14][15] use job-based list scheduling extended to manage the modulo resource constraints. With job-based list scheduling, each operation is scheduled in priority order, unlike Graham list scheduling where the operations are scheduled in non-decreasing time order. In [14][15], the job-based modulo list scheduling is augmented with bounded backtracking and is reported with satisfactory results. In recent work, Brucker et al. [5] formulates neighborhood functions for the 1-periodic cyclic job-shop problems and apply them to Tabu search heuristics.

In contrast to those heuristic approaches and restricting assumptions, the regular unwinding framework enables resource-constrained modulo scheduling to benefit from any machine scheduling technique that applies to the regular unwinded scheduling problems. As first applications, we solve the following modulo scheduling problems in pseudo-polynomial time:

$P|r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$, based on the properties of the Graham list scheduling algorithm with Jackson's rule on the unwinded scheduling problems.

$P|circuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$, by applying the algorithm of Leung, Palem and Pnueli [10] to the unwinded scheduling problems.

$P|subCircuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$, by combining the previous results with a proof that the fixpoint modified deadlines computed by the algorithm of Leung, Palem and Pnueli [10] are almost λ-regular for unwinded problem whose independent operations are not λ-regularized.

$\Sigma P|subCircuit(\theta_i^j, \omega_i^j); \sum \omega_i^j = 1; r_i; d_i; p_i = 1; \pi_i = \lambda|\bullet$, by combining the previous result with a proof that the algorithm of Leung, Palem and Pnueli [10] applies to typed task systems [9] in case of monotone sub-chain orders.

These results are directly relevant to the practice of cyclic instruction scheduling that motivates our research. Given a resource-constrained and dependence-constrained modulo scheduling problem, summarizing all dependences into release dates and deadlines while keeping those on the most constraining recur-

rence circuit with $\sum \omega_i^j = 1$ yields a relaxation we can solve. This relaxation is stronger than removing either all resource constraints or all dependence constraints from the original modulo scheduling problem.

References

[1] A. Aiken, A. Nicolau, S. Novack. Resource-Constrained Software Pipelining. *IEEE Transactions on Parallel and Distributed Systems, 6, 12*, pages 1248–1270, 1995.

[2] F. Bodin, F. Charot. Loop Optimization for Horizontal Microcoded Machines. *Proceedings of the 1990 International Conference on Supercomputing*, pages 164–176, 1990.

[3] P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research 112*, 1999.

[4] P. Brucker. Scheduling Algorithms -4th ed.. *Springer Verlag*, 2004.

[5] P. Brucker, T. Kampmeyer. Tabu Search Algorithms for Cyclic Machine Scheduling Problems. *Journal of Scheduling, Vol. 8, No. 4*, July 2005.

[6] Ph. Chrétienne. On Graham's Bound for Cyclic Scheduling. *Parallel Computing 26(9)*, pages 1163–1174, 2000.

[7] B. Dupont de Dinechin. From Machine Scheduling to VLIW Instruction Scheduling. *ST Journal of Research vol. 1, no. 2*, 2004. `http://www.st.com/stonline/press/magazine/stjournal/vol0102/`

[8] C. Hanen, A. Munier. A Study of the Cyclic Scheduling Problem on Parallel Processors. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science, 57*, 1995.

[9] K. Jansen. Analysis of scheduling problems with typed task systems. *Discrete Applied Mathematics 52*, pages 223–232, 1994.

[10] A. Leung, K. V. Palem, A. Pnueli. Scheduling Time-Constrained Instructions on Pipelined Processors. *ACM TOPLAS 23, 1*, pages 73–103, Jan. 2001.

[11] K. V. Palem, B. Simons. Scheduling Time-Critical Instructions on RISC Machines. *ACM Transactions on Programming Languages and Systems (TOPLAS) 15, 4*, pages 632–658, Sept. 1993.

[12] C. Papadimitriou, M. Yannakakis. Scheduling Interval-Ordered Tasks. *SIAM J. of Computing, 8, 3*, pages 405–409, 1979.

[13] B. R. Rau, C. D. Glaeser. Some Scheduling Techniques and an Easily Schedulable Horizontal Architecture for High Performance Scientific Computing. *14th Annual Microprogramming Workshop on Microprogramming – MICRO-14*, pages 183–198, Dec. 1981.

[14] B. R. Rau. Iterative Modulo Scheduling. *The International Journal of Parallel Processing, 24, 1*, pages 3–64, Feb. 1996.

[15] J. Ruttenberg, G. R. Gao, A. Stoutchinin, W. Lichtenstein. Software Pipelining Showdown: Optimal vs. Heuristic Methods in a Production Compiler. *SIGPLAN Conference on Programming Language Design and Implementation – PLDI'96*, May 1996.