Coping with incomplete information in Real Time Scheduling Verification

Maurizio Talamo, Andrea Callia D'Iddio, Christian H. Schunck and Franco Arcieri

***Abstract*—**The scheduling of processes is of strategic importance in many areas of software engineering. Verifying the correctness of scheduling is critical to ensure data integrity, process efficiency and system security. Therefore a scheduling must often be verified quickly or even in "real time". In this case it may be inefficient or too resource consuming to gather all the data about individual jobs. However, missing information may prevent the unique identification of each job in the scheduling. In this paper we define a framework in which we model this information loss and we define methods and algorithms to check the correctness of a scheduling with regard to the order in which the jobs are executed in the presence of such ambiguities. Given a set of jobs and a specification, which is defined as a set of dependencies between the jobs, the task is to check if the order in which the jobs are scheduled satisfies the dependencies. We use partial order structures to mathematically model dependencies in the specification and in the order of execution of the scheduled jobs. If some jobs become indistinguishable due to information loss it is not possible to determine all the precedences between jobs and the notion of dominance is lost.

Here we use abstraction mechanisms at the combinatorial level to obtain new notions of dominance that enable the comparison of "approximated" partial orders. Based on these results, methods and algorithms are developed which can be used to test actual schedulings with regard to their specifications.

***Keywords*—**security, scheduling, smart cards, model checking.

I. Introduction

SCHEDULERS are software applications of strategic importance. Bugs in schedulers may therefore cause important efficiency and security problems. The security and stability of IT systems is in particular sensitive to such bugs if schedulers are managing new pieces of software that are integrated in open or distributed environments, or used in embedded systems: in such situations the environment in which the scheduler operates is not fully protected and any bug in the scheduler becomes a potential security threat.

Verifying the correct operation of a scheduler in such environments poses a number of important problems. In many security applications the scheduling must be analyzed in "real time" as it is not possible to test a scheduling a priori. A check a posteriori is too late to allow for intervention or prevention of damages for example in case denial of service attacks. Testing a scheduling in real time requires optimal use of limited resources like computational resources, time, and memory. In particular one must be able to check a scheduling even in the event that not all information about individual jobs is available which may prevent the unique identification of a job. This can be a direct consequence of how the environment in which the scheduler is operating has been setup (e.g. for thread pools), but may also be the result of a strategic choice to deal efficiently with the problem given limited resources. In the latter case the decision is made to analyze only a part of the information that is in principle available right away.

Here we model this information loss and develop an efficient verification application with the goal to check *the order in which the jobs of one or more schedulings are to be executed* against a *specification* representing the correct execution of the jobs. For this purpose we develop a new approach *abstracting* the scheduling process at the *combinatorial level*. This approach complements techniques used in model checking [1, 2, 3] and in particular *abstraction/refinement* methods used in model checking [4] and it can be applied in *process mining* [5, 6, 7]. In particular this approach can address issues related to applying *semantic model checking* [8] in such a context: it is not possible to apply the primitives used in these approaches to jobs about which limited information is available and which may be "legal" individually. Furthermore the approach avoids inefficient checking routines from analyzing all possibilities that could occur after a particular job has been executed in order to reach a decision whether to allow its execution. This becomes particularly important when high numbers of verifications need to be performed: in this case it is possible to *filter* correct and incorrect schedulings already at high levels of abstraction and to determine for which schedulings more information must be obtained to verify their correctness.

We represent a scheduling using a *partial order* of jobs, where the partial order represents the *dependencies* between them: if dependencies are satisfied in the scheduling, we assume that the outcome will be correct. Assuming a partial order to represent such dependencies between jobs is justified: while some job *B* might depend on the execution of a job *A*, at the same time *B* might be executed completely independently

M. Talamo is with the Nestor Laboratory and the Department of Mathematics, Tor Vergata University of Rome, Italy (e-mail: talamo@ nestor.uniroma2.it).

A. Callia D'Iddio is with the Department of Mathematics, Tor Vergata University of Rome, Italy (e-mail: calliadiddio@ nestor.uniroma2.it).

C. H. Schunck is with the Nestor Laboratory, Tor Vergata University of Rome, Italy (e-mail: schunck@ nestor.uniroma2.it).

F. Arcieri is with the Nestor Laboratory, Tor Vergata University of Rome, Italy (e-mail: arcieri@ nestor.uniroma2.it).

from another job *C* in the same process (i.e. *B* can be executed before, after, or concurrently with job *C* and vice versa). Moreover, cyclic dependencies must not exist as this would result in deadlocks, so dependencies cannot be symmetric.

We regard jobs as nodes in the partial order, and dependencies between jobs correspond to precedences between nodes. Due to the limited information available certain jobs that would be distinguishable in principle (given their full specifications) become indistinguishable as the available information does not allow a differentiation between these jobs. A verification based on the abstracted partial order may reveal the *correctness* or the *incorrectness* of a scheduling. If indistinguishable nodes do not allow to determine correctness or incorrectness, one can zoom in to a lower level of abstraction to solve ambiguities and recheck for errors, for example by using an *iterative* checking mechanism. In particular, one can use this approach to "zoom in" only on especially critical jobs instead of all the jobs. In the following sections, we will develop the model to represent the abstracted partial order of jobs and develop an algorithm to test a scheduling with regard to its specification.

II. Mathematical model

In this section we present the mathematical model that allows us to check a scheduling at different levels of abstraction. To do this a partial order model of a scheduling is defined in which several nodes may have the same value. An idea is to use a structure – first introduced by Gischer [9] and Pratt [10] to model concurrent processes – called *labeled poset*. A labeled poset is defined by a poset plus a node-labeling function. This function may be *non-injective*, and so it allows for nodes to have the same value. Labeled posets, by definition, contain all the information, because they contain not only the values but also the nodes labeled with these values, and these nodes represent the full specifications of jobs. Since we want to model the situation in which these specifications are not available, we will introduce *multipair structures*, which accurately reflect the available information. We then show how the correctness of a scheduling can be tested based on these multipair structures.

A. Labeled posets

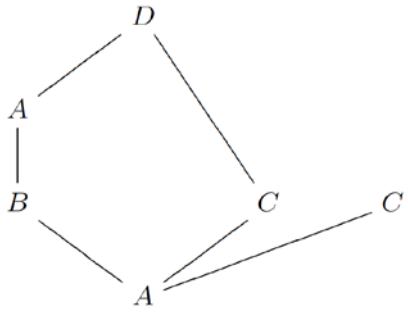
First we define labeled posets. Note that in the following we restrict ourselves to *strict* partial orders as the reflexive property of a non-strict partial order is not relevant for our purposes.

Definition 1 A **labeled poset** is defined as a 4-tuple $\mathcal{M} = (E, X, P, val)$ where the following holds.

- E is a set of **elements**, also called **nodes**.
- X is a set of **values**.
- $P \subseteq E \times E$ is a partial order over E.
- $val: E \to X$ is a surjective **labeling function** which assigns a value to each element.

Like simple posets, a labeled poset can be represented by a *labeled Hasse diagram*, i.e. a Hasse diagram in which elements are replaced by their corresponding values given by the labeling function.

Example 2 The following labeled Hasse diagram



represents the labeled poset $\mathcal{M} = (E, X, P, val)$ defined below.

- $E = \{1, 2, 3, 4, 5, 6\}$
- $X = \{A, B, C, D\}$
- $P = \{(1, 2), (2, 5), (1, 5), (5, 6), (2, 6), (1, 6), (1, 3), (3, 6), (1, 4)\}$
- $val(1) = A$, $val(2) = B$, $val(3) = C$, $val(4) = C$, $val(5) = A$, $val(6) = D$

B. Strong and weak dominance

Consider the labeled poset of example 2. Here value *B must dominate* value *D* because the unique node *B* precedes the unique node *D*. Also *A* must dominate *D*, because both nodes with value *A* precede the unique node *D*. However, *A* and *B* cannot be compared in terms of "must"-dominance, because not all nodes *A* precede the node with value *B* and vice versa. For the same reason, we cannot compare values *C* and *D* or values *A* and *C*. However, one can state that *A may* dominate *C*, because there exists a node *A* which dominates a node *C*, and for the same reason *C* may dominate *D*. Similarly, one can state that *A* may dominate *B* and *B* may dominate *A*. Below we formalize these notions of dominance, respectively called *strong* and *weak* dominance.

Definition 3 Let $\mathcal{M} = (E, X, P, val)$ be a labeled poset. We say that a value a **strongly dominates** a value $b \neq a$, in symbols $a \prec^{\mathcal{M}} b$, if a and b are in X and for each $i, j \in E$ such that $val(i) = a$ and $val(j) = b$ it holds that $(i, j) \in P$.

We write $a \prec b$ instead of $a \prec^{\mathcal{M}} b$ when $\mathcal{M}$ is clear from the context.

Example 4 In the labeled poset $\mathcal{M}$ of example 2, as said before, it holds that $B \prec D$ and $A \prec D$.

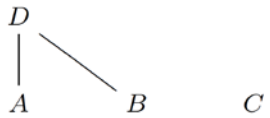
We now verify that the strong dominance relation is itself a partial order.

Proposition 5 The strong dominance relation $\prec$ is a partial order.

Proof: The irreflexive property is true by definition. We have to prove the transitivity property. Consider generic $a, b, c \in X$ such that $a \prec b$ and $b \prec c$. We have to prove that $a \prec c$. Consider generic $i, j \in E$ such that $val(i) = a$ and $val(j) = c$. Now we choose a generic k such that $val(k) = b$. Since $a \prec b$ then it must be $(i, k) \in P$. Moreover, since $b \prec c$ then it must be $(k, j) \in P$. Since P is a partial order, by the transitivity of P, it holds that $(i, j) \in P$. Since we have chosen generic i, j, then the property holds for every i, j, and so $a \prec c$. ⊔

From the point of view of "must"-dominance, the strong dominance relation contains all information and is a partial order. No approximations (like removal of pairs) are required to maintain the partial ordering property.

Example 6 Consider the labeled poset $\mathcal{M}$ of example 2. The partial order corresponding to its strong dominance relation can be depicted by the following Hasse diagram.



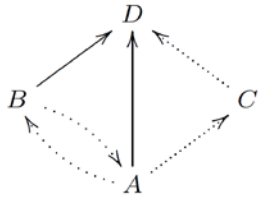
After strong dominance, we can similarly define the notion of *weak dominance*.

Definition 7 Let $\mathcal{M} = (E, X, P, val)$ be a labeled poset. We say that a value a **weakly dominates** a value b, in symbols $a \prec_w^{\mathcal{M}} b$, if a and b are in X and there exist $i, j \in E$ such that $val(i) = a$, $val(j) = b$ and $(i, j) \in P$.

Again, we write $a \prec_w b$ instead of $a \prec_w^{\mathcal{M}} b$ when $\mathcal{M}$ is clear from the context. It can be easily verified that, unlike strong dominance, the weak dominance relation may not be a partial order.

Example 8 Consider the labeled poset $\mathcal{M}$ of example 2. It holds that $A \prec_w B$ and $B \prec_w A$, so $\prec_w$ is not antisymmetric.

Example 9 In the following diagram strong dominance (plain arrows) and weak dominance (dotted arrows) are shown for labeled poset of example 2.



For readability, we will also write $SD(\mathcal{M})$ instead of $\prec^{\mathcal{M}}$ and $WD(\mathcal{M})$ instead of $\prec_w^{\mathcal{M}}$ to represent strong and weak dominance relations. Thus, formally, if $\mathcal{M} = (E, X, P, val)$ then we define $SD(\mathcal{M}) = \{(a, b) \in X \times X$ such that $a \prec^{\mathcal{M}} b\}$ and $WD(\mathcal{M}) = \{(a, b) \in X \times X$ such that $a \prec_w^{\mathcal{M}} b\}$.

C. Relating completeness and incompleteness

Labeled posets are a good model to represent, **in the same model**:

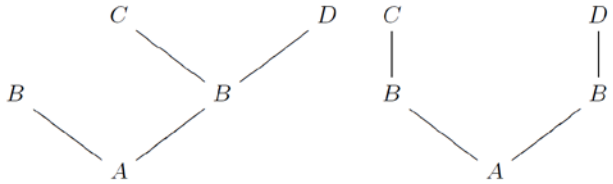
- the complete information (by nodes and pairs of nodes);
- the incomplete information (by values);
- and how these are related (by the non-injective labeling function).

However labeled posets contain information that cannot be obtained when a real scheduling is observed. Therefore we create a new model (multipair structures) that adequately represents the available information. In the following we will study both labeled posets and multipair structures. Labeled posets allows us to define notions, like completeness and correctness, which cannot be defined using multipair structures. Moreover we use labeled posets to formally prove that in many cases one can deduce completeness or correctness of a scheduling by using multipair structures.

D. Multipair structures and strong dominance

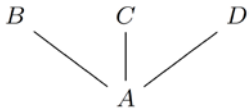
Let $\mathcal{M} = (E, X, P, val)$ be a labeled poset. As stated previously, in our context the observed jobs may not be necessarily "distinguishable" as the information that can be obtained about individual jobs may be incomplete. To model this fact we define a particular poset in which the set of the elements is X and a set of pairs is obtained by taking P and replacing nodes with corresponding values. If the labeling function *val* is injective (and then *val* is bijective) this representation is equivalent to the original labeled poset. We now model the information loss by considering the general case in which the labeling function *val* is *not* injective. In this case expressivity is invariably lost as shown in the following example.

Example 10 Consider the following tree-like posets.



These different labeled posets yield the same set of pairs, even if we adopt multisets to consider multiple instances of values and pairs. In fact both of them produce the multiset of nodes [*A*, *B*, *B*, *C*, *D*] and the multiset of pairs [(*A*,*B*), (*A*,*B*), (*A*,*C*), (*A*,*D*), (*B*,*C*), (*B*,*D*)]. Based on the structural representations one finds, for example, that in the first poset a node labeled by *B* precedes both *C* and *D* which is not true in the second poset.

A structural representation reflecting the "reduced" information and in particular the "incomparability" between the values *B*, *C*, *D* is given below.



It turns out that for our purposes the information about strong and weak dominance, together with information about the number of indistinguishable jobs and pairs, is sufficient to understand (at a certain level of abstraction) whether dependencies may be violated by a scheduling. This information can be obtained querying only over the set of pairs.

Definition 11 A **multipair structure** is a 4-tuple (X, ϕ, Q, ψ) defined as below.

- X is a set of values.
- $\phi : X \to N$ is a **multiplicity function** for X.
- $Q \in X \times X$ is a set of pairs of values in X.
- $\psi : Q \to N$ is a **multiplicity function** for Q.

A labeled poset $\mathcal{M} = (E, X, P, val)$ can be transformed into a multipair structure (X, ϕ, Q, ψ) by reading the values of pairs in P, and then counting values and pairs by the multiplicity functions. Formally, the transformation is explained below.

- X is already defined.
- $\phi(a) = n$ if there are n elements $i \in E$ s.t. $val(i) = a$.
- Q is the weak dominance relation of $\mathcal{M}$, that is $Q = \{(a, b)$ such that $val(i) = a$, $val(j) = b$ and $(i, j) \in P$ for some $i, j\}$.
- If there are exactly m pairs $(i, j) \in P$ such that $val(i) = a$ and $val(j) = b$ then $\psi(a, b) = m$.

Function ϕ counts how many times a value appears in the labeled poset, and ψ counts how many times a value precedes another value. Practically, a multipair structure is easy to implement, for example by using an array for X and ϕ and a matrix for Q and ψ.

Example 12 Consider again the labeled poset $\mathcal{M}$ of example 2. Following the previous criterion, the corresponding multipair structure (X, ϕ, Q, ψ) is constructed as below.

- $X = \{A, B, C, D\}$
- $\phi(A) = 2$, $\phi(B) = 1$, $\phi(C) = 2$, $\phi(D) = 1$
- $Q = \{(A,B), (B,A), (A,A), (A,D), (B,D), (A,C), (C,D)\}$
- $\psi(A, B) = 1$, $\psi(B, A) = 1$, $\psi(A, A) = 1$, $\psi(A, D) = 2$, $\psi(B, D) = 1$, $\psi(A, C) = 2$, $\psi(C, D) = 1$

Now we give a new characterization for strong dominance.

Theorem 13 Let $\mathcal{M} = (E, X, P, val)$ be a labeled poset, let (X, ϕ, Q, ψ) be the corresponding multipair structure and let $a, b \in X$ with $a \neq b$. Then $a \prec b$ if and only if $\psi(a, b) = \phi(a) \cdot \phi(b)$.

Proof:

- **"Only if" part (⇒)**. Suppose that $a \prec b$. We calculate $\psi(a, b)$. First, we choose an element i such that $val(i) = a$. There are $\phi(a)$ choices. Now we choose an element j such that $val(j) = b$ and $(i, j) \in P$. Since $a \prec b$, $(i, j) \in P$ for all j with $val(j) = b$. So there are $\phi(b)$ choices and therefore $\psi(a, b) = \phi(a) \cdot \phi(b)$.
- **"If" part (⇐).** Consider two elements i', j' such that $val(i') = a$ and $val(j') = b$. Suppose that $(i', j') \notin P$. We show that in this case $\psi(a, b) < \phi(a) \cdot \phi(b)$. First we choose an element i such that $val(i) = a$. There are two disjoint alternatives.
  - $i \neq i'$, i.e. there are $\phi(a) - 1$ choices for i. Now we choose an element j such that $val(j) = b$ and $(i, j) \in P$. Trivially at most $\phi(b)$ j's have this property. So, if w_1 is the total number of choices for (i, j) and $i \neq i'$, we have $w_1 \leq (\phi(a) - 1) \cdot \phi(b)$.
  - $i = i'$. Now we choose j such that $val(j) = b$ and $(i, j) \in P$. In this case there are strictly less than $\phi(b)$ choices for j since we have to exclude $j = j'$ (which has the value b) since $(i', j') \notin P$. So, if w_2 is the number of choices for (i, j) and $i = i'$ we have $w_2 < \phi(b)$.

Since the two alternatives count disjoint choices for (i, j), the total number of choices is given by

$$\psi(a, b) = w_1 + w_2 < w_1 + \phi(b)$$

$$\leq (\phi(a) - 1) \cdot \phi(b) + \phi(b) = \phi(a) \cdot \phi(b)$$

and then

$$\psi(a, b) < \phi(a) \cdot \phi(b)$$

⊔

For our purposes it is therefore sufficient to work on multipair structures instead of labeled posets. This is important as the multipair structures reflect the information about jobs that is indeed available at a given level of abstraction.

III. Applications in scheduling verification

We can now proceed to discuss how a scheduling can be checked for correctness. We will assume that the scheduling is observed as a multipair structure and therefore the verification algorithm will be based on these structures. To define the notion of correctness and to motivate the verification algorithm on multipair structures we use labeled posets as these eliminate ambiguities. The formal definitions of specification and scheduling are given below.

Definition 14 Given a labeled poset $\mathcal{M} = (E, X, P, val)$, called a **specification**, a **scheduling** of the specification $\mathcal{M}$ is a labeled poset $\mathcal{S} = (E', X', P', val')$ such that $E' \subseteq E$, and $val' = val|_{E'}$.

A. Complete schedulings

In the following we distinguish between *complete* and *partial* schedulings. A *complete* scheduling is fully specified in the sense that it contains all jobs to be executed and all their dependencies. In a *partial* scheduling some jobs or dependencies may not yet be included. The formal definition of complete scheduling is given below.

Definition 15 Let $\mathcal{M} = (E, X, P, val)$ be a specification and let $\mathcal{S} = (E', X', P', val')$ be a scheduling of $\mathcal{M}$. Then $\mathcal{S}$ is **complete** if and only if $E' = E$ (and then $X' = X \wedge val' = val$).

We now define a function to count the nodes labeled with the same value.

Definition 16 Let $\mathcal{M} = (E, X, P, val)$ be a labeled poset. Given an element $a \in X$ then $\Phi_{\mathcal{M}}(a) \equiv \{i \in E \mid val(i) = a\}$ and $\phi_{\mathcal{M}}(a) \equiv |\Phi_{\mathcal{M}}(a)|$.

We can exploit ϕ to verify if a scheduling is complete, as shown in the following theorem.

Theorem 17 Let $\mathcal{M} = (E, X, P, val)$ be a specification and let $\mathcal{S} = (E', X', P', val')$ a scheduling of $\mathcal{M}$. Then $\mathcal{S}$ is complete w.r.t. $\mathcal{M}$ if and only if for each $a \in X$ it holds that $\phi_{\mathcal{M}}(a) = \phi_{\mathcal{S}}(a)$.

Proof: Before proving the "if and only if" we observe that it is easy to verify that for each a ∈ X the following property holds.[1]

$$\Phi_{\mathcal{S}}(a) \subseteq \Phi_{\mathcal{M}}(a) \qquad (1)$$

[1] In fact, consider an element $a \in X$. Since $\mathcal{S}$ is a scheduling of $\mathcal{M}$ then $E' \subseteq E$ and $val' = val|_{E'}$. So, for each $j \in \Phi_{\mathcal{S}}(a)$ since $j \in E'$ then $j \in E$ and also $val(j) = val'(j) = a$ thus $j \in \Phi_{\mathcal{M}}(a)$, i.e. $\Phi_{\mathcal{S}}(a) \subseteq \Phi_{\mathcal{M}}(a)$.

- **"Only if" part (⇒).**
  We prove equivalently that if there exists $a \in X$ such that $\phi_{\mathcal{M}}(a) \neq \phi_{\mathcal{S}}(a)$ then $\mathcal{S}$ cannot be complete w.r.t. $\mathcal{M}$. Suppose that such a exists. This means that $\Phi_{\mathcal{S}}(a) \neq \Phi_{\mathcal{M}}(a)$. More precisely, since property (1) holds, then it can only be $\Phi_{\mathcal{S}}(a) \subset \Phi_{\mathcal{M}}(a)$. This means that there exists i such that $i \in \Phi_{\mathcal{M}}(a)$ but $i \notin \Phi_{\mathcal{S}}(a)$. Since $i \in \Phi_{\mathcal{M}}(a)$ then $i \in E$ and $val(i) = a$. Moreover, since $i \notin \Phi_{\mathcal{S}}(a)$ and $val(i) = a$ then it must be $i \notin E'$. So $E \neq E'$ and the scheduling $\mathcal{S}$ is not complete.
- **"If" part (⇐).** For each $a \in X$, since property (1) holds, then $\phi_{\mathcal{M}}(a) = \phi_{\mathcal{S}}(a)$ implies $\Phi_{\mathcal{S}}(a) = \Phi_{\mathcal{M}}(a)$. We already know that $E' \subseteq E$, so we only need to prove that $E \subseteq E'$. Consider a generic element $i \in E$. Let $a = val(i)$. So $i \in \Phi_{\mathcal{M}}(a)$ and then $i \in \Phi_{\mathcal{S}}(a)$, and this means, by definition of Φ, that $i \in E'$, so $E \subseteq E'$ and then $E = E'$, i.e. $\mathcal{S}$ is complete.

⊔

B. Correct schedulings

Assume we have a specification, represented by a partial order of jobs, and a complete scheduling, represented in the same way. Intuitively this scheduling is *correct* if it satisfies all dependencies in the specification, i.e. if the dependencies of the specification are also dependencies in the scheduling. This concept is formalized below.

Definition 18 Let $\mathcal{M} = (E, X, P, val)$ be a specification and let $\mathcal{S} = (E, X, P', val)$ be a complete scheduling of $\mathcal{M}$. Then $\mathcal{S}$ is correct with regard to $\mathcal{M}$ if and only if $P' \supseteq P$.

We also define a function to count the number of indistinguishable pairs.

Definition 19 Let $\mathcal{M} = (E, X, P, val)$ be a labeled poset. Given two elements a, b ∈ X then $\Psi_{\mathcal{M}}(a, b) \equiv \{(i, j) \mid (i, j) \in P \wedge val(i) = a \wedge val(j) = b\}$ and $\psi_{\mathcal{M}}(a, b) \equiv |\Psi_{\mathcal{M}}(a, b)|$.

The following theorem gives the criteria for checking whether a complete scheduling is correct with regard to a specification. The above stated definition of correctness is not applied directly, instead the criteria for strong and weak dominance and the ψ function are used.

Theorem 20 Let $\mathcal{M} = (E, X, P, val)$ be a specification and let $\mathcal{S} = (E, X, P', val)$ be a complete scheduling of $\mathcal{M}$. Then the following properties hold.

1. If $SD(\mathcal{S}) \supseteq WD(\mathcal{M})$ then the scheduling $\mathcal{S}$ is correct.
2. If there exists (a, b) such that $\psi_{\mathcal{S}}(a, b) < \psi_{\mathcal{M}}(a, b)$ then

the scheduling $\mathcal{S}$ is not correct. In particular if $WD(\mathcal{S}) \not\supseteq WD(\mathcal{M})$ then $\mathcal{S}$ is not correct.

3. If $SD(\mathcal{S}) \not\supseteq SD(\mathcal{M})$ then the scheduling $\mathcal{S}$ is not correct.
4. Otherwise (i.e. if $SD(\mathcal{M}) \subseteq SD(\mathcal{S}) \not\supseteq WD(\mathcal{M})$) it is not possible to establish if $\mathcal{S}$ is correct at the current level of abstraction.

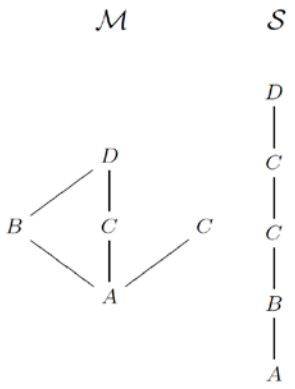
Proof:

1. Consider a generic pair $(i, j) \in P$. This means by definition of weak dominance that $(val(i), val(j)) \in WD(\mathcal{M})$ and therefore by hypothesis $(val(i), val(j)) \in SD(\mathcal{S})$. By definition of strong dominance, for each $l, m \in E$ such that $val(l) = val(i)$ and $val(m) = val(j)$ it holds that $(l, m) \in P'$, and therefore also $(i, j) \in P'$. Since $(i, j) \in P$ is generic, $P' \supseteq P$, i.e. $\mathcal{S}$ is correct.
2. If $\psi_{\mathcal{S}}(a, b) < \psi_{\mathcal{M}}(a, b)$ then $\Psi_{\mathcal{S}}(a, b) \subset \Psi_{\mathcal{M}}(a, b)$ so there exist $i, j \in E$ such that $(i, j) \in \Psi_{\mathcal{M}}(a, b)$ but $(i, j) \notin \Psi_{\mathcal{S}}(a, b)$. Since $(i, j) \in \Psi_{\mathcal{M}}(a, b)$ then $(i, j) \in P$ and $val(i) = a \wedge val(j) = b$. Moreover, since $(i, j) \notin \Psi_{\mathcal{S}}(a, b)$ and $val(i) = a \wedge val(j) = b$ then $(i, j) \notin P'$. So $P' \not\supseteq P$ and the scheduling $\mathcal{S}$ is not correct.
3. Suppose that $SD(\mathcal{S}) \not\supseteq SD(\mathcal{M})$. So there exists $(a, b) \in SD(\mathcal{M})$ such that $(a, b) \notin SD(\mathcal{S})$. Since $(a, b) \notin SD(\mathcal{S})$ then there exist $i, j \in E$ such that $val(i) = a \wedge val(j) = b$ but $(i, j) \notin P'$. But, since $val(i) = a \wedge val(j) = b$, the fact that $(a, b) \in SD(\mathcal{M})$ implies that $(i, j) \in P$. Since $(i, j) \notin P'$ but $(i, j) \in P$, then $P' \not\supseteq P$ i.e. $\mathcal{S}$ is not correct.
4. Since $SD(\mathcal{S}) \not\supseteq WD(\mathcal{M})$ there exists $(a, b) \in WD(\mathcal{M})$ such that $(a, b) \notin SD(\mathcal{S})$. Since $(a, b) \notin SD(\mathcal{S})$ there exist $i, j \in E$ such that $val(i) = a \wedge val(j) = b$ but $(i, j) \notin P'$. Now we have to verify if $(i, j) \in P$ to know if $\mathcal{S}$ can be correct. Since $SD(\mathcal{M}) \subseteq SD(\mathcal{S})$ and $(a, b) \notin SD(\mathcal{S})$ then $(a, b) \notin SD(\mathcal{M})$ so there exist $i', j' \in E$ such that $val(i') = a \wedge val(j') = b$ but $(i', j') \notin P$, and then the set $A = \{l, m \mid val(l) = a \wedge val(m) = b$ but $(l, m) \notin P\}$ is not empty. But, since $(a, b) \in WD(\mathcal{M})$ there also exist $i'', j'' \in E$ such that $val(i'') = a \wedge val(j'') = b$ and $(i'', j'') \in P$, so the set $B = \{l, m \mid val(l) = a \wedge val(m) = b$ and $(l, m) \in P\}$ is non-empty as well. Since $val(i) = a$ and $val(j) = b$ then (i, j) must be either in A or in B. In the first case $\mathcal{S}$ is not correct, and in the second case $\mathcal{S}$ may be correct. The only way to find out it is to refine $\mathcal{M}$ and $\mathcal{S}$.

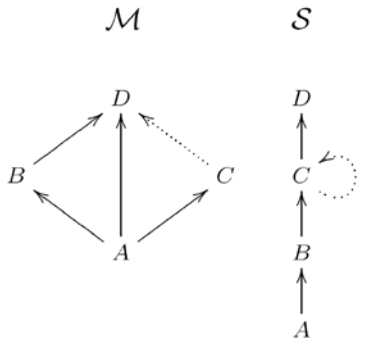
⊔

In the following examples we discuss some cases in which the criteria described in theorem 20 can be applied.

Example 21 Consider the specification $\mathcal{M}$ and the scheduling $\mathcal{S}$ depicted in the following labeled Hasse diagrams.

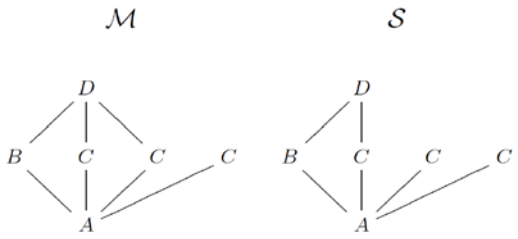


The diagrams representing strong and weak dominance of $\mathcal{M}$ and $\mathcal{S}$ are below.

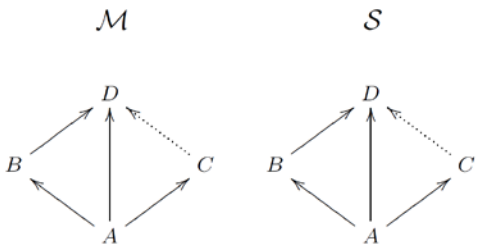


It is clear that strong dominance of $\mathcal{S}$ captures both strong and weak dominance of $\mathcal{M}$, so each dependency of $\mathcal{M}$ must be satisfied by $\mathcal{S}$. Here criterion (1) of theorem 20 is fulfilled, so, without making any refinement, we can state that $\mathcal{S}$ is correct.

Example 22 Consider the specification $\mathcal{M}$ and the scheduling $\mathcal{S}$ represented below.

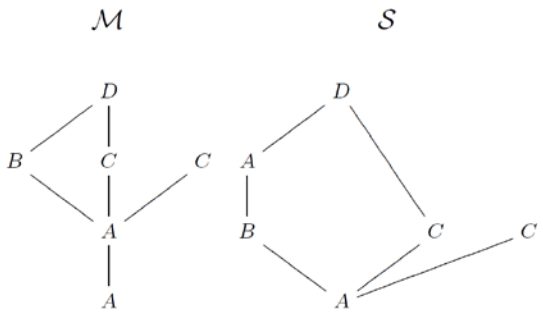


The diagrams for strong and weak dominance of $\mathcal{M}$ and $\mathcal{S}$ are the following.

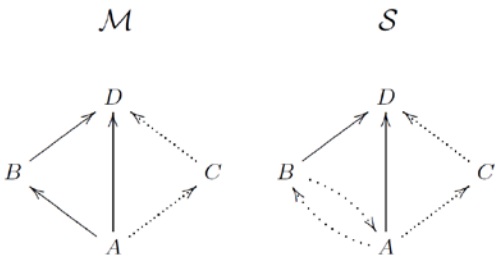


By looking at the diagrams for strong and weak dominance one might get the impression that the scheduling is substantially correct. Here, the pair counting helps. In fact, one finds that $\psi_{\mathcal{M}}(C,D) = 2$ and $\psi_{\mathcal{S}}(C,D) = 1$. So criterion (2) of theorem 20 is fulfilled and this means that $\mathcal{S}$ is not correct. In specification $\mathcal{M}$ the job C must precede the job D in two cases, and so $\mathcal{S}$ cannot satisfy the corresponding two dependencies because it has only one (C, D) pair.

Example 23 Consider the following labeled posets.

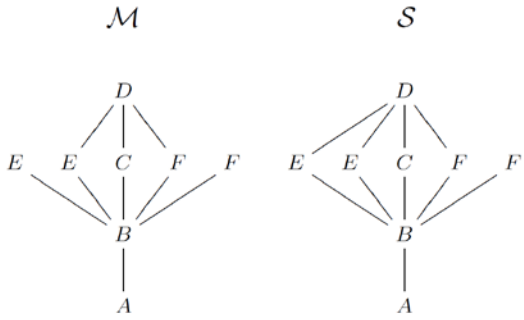


The strong/weak dominance diagrams are below.

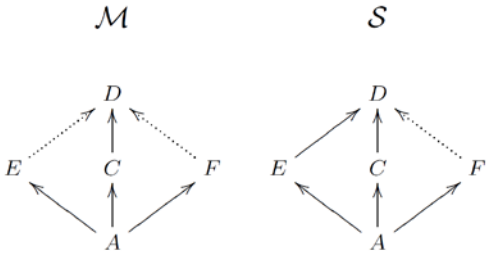


From these diagrams we can observe that $(A, B) \in SD(\mathcal{M})$ but $(A, B) \notin SD(\mathcal{S})$. So criterion (3) of theorem 20 is fulfilled and $\mathcal{S}$ is not correct. In fact, in $\mathcal{M}$ the job B must be preceded by A in all cases, and this is not satisfied by the scheduling $\mathcal{S}$.

Example 24 Consider the following labeled posets.



The diagrams for strong and weak dominance are depicted below.



In this case, criteria (1), (2) and (3) of theorem are not fulfilled, and one can erroneously think that $\mathcal{S}$ is correct. Also, in the scheduling $\mathcal{S}$ the value E strongly dominates D, and this dominance is only weak in the specification. But consider values F and D. The value F weakly dominates D in both models and $\psi_{\mathcal{S}}(F,D) = \psi_{\mathcal{M}}(F,D) = 1$, so the dependency (F,D) in $\mathcal{M}$ could be satisfied by the dependency (F,D) of $\mathcal{S}$. Since there are two jobs labeled with value F, it is unclear whether the F that dominates D in $\mathcal{S}$ is the same F which dominates D in $\mathcal{M}$. To know it we need to make a refinement, in order to know who really F is in those dependencies. In fact, criterion (4) is fulfilled.

We can also check the correctness of a *partial* scheduling by applying *locally* criterion (1) of theorem 20. Below we define *safe pairs*.

Definition 25 Let $\mathcal{M} = (E, X, P, val)$ be a specification and let $\mathcal{S} = (E', X', P', val')$ be a scheduling of $\mathcal{M}$. A pair $(a, b) \in X \times X$ is **safe** in $\mathcal{S}$ with regard to $\mathcal{M}$ if and only if $\Psi_{\mathcal{M}}(a, b) \subseteq \Psi_{\mathcal{S}}(a, b)$.

So a pair (a, b) is safe in a scheduling $\mathcal{S}$ of a specification $\mathcal{M}$ if each dependency (i, j) of $\mathcal{M}$ in which i and j are labeled respectively with a and b is also a dependency of $\mathcal{S}$. This can be used to check at runtime if a request to execute a job b of $\mathcal{S}$ can be satisfied even if the scheduling $\mathcal{S}$ is not fully specified. In fact, it suffices to verify if for each job a of $\mathcal{S}$ the pair (a, b) is safe w.r.t. $\mathcal{M}$. If so, all dependencies are satisfied and b can

be executed safely. If not, a *lazy* approach can be used, i.e. the execution of b is suspended until we have more information about the dependencies in which b is involved and in the mean time we can execute the other safe jobs.

This approach can also be used to *partially check* a complete scheduling. For example, if a complete scheduling is incorrect we can obtain its safe pairs to understand which jobs can be still safely executed, and adopting a "lazy" approach to manage unsafe jobs.

As said above, by applying locally criterion (1) of theorem 20 we can check if a pair is safe using only weak and strong dominance plus the function ϕ.

Theorem 26 Let $\mathcal{M} = (E, X, P, val)$ be a specification and let $\mathcal{S} = (E', X', P', val')$ be a scheduling of $\mathcal{M}$. A pair $(a, b) \in X \times X$ is safe if one of the following properties holds.

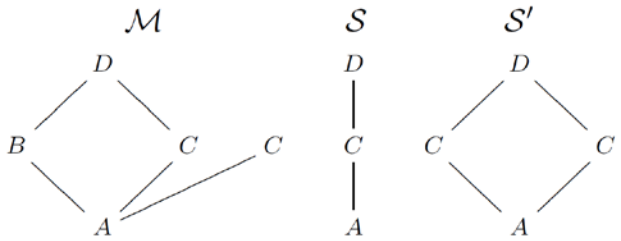
1. $(a, b) \notin WD(\mathcal{M})$
2. $\phi_{\mathcal{M}}(a) = \phi_{\mathcal{S}}(a) \wedge \phi_{\mathcal{M}}(b) = \phi_{\mathcal{S}}(b) \wedge (a, b) \in SD(\mathcal{S})$

Proof:

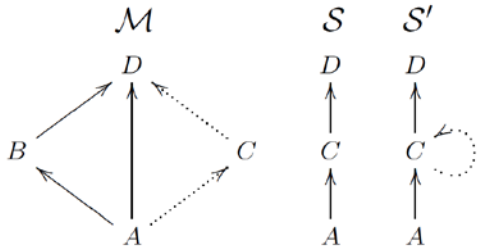
1. If $(a, b) \notin WD(\mathcal{M})$ then there are no pairs $(i, j) \in P$ such that $val(i) = a \wedge val(j) = b$. Thus $\Psi_{\mathcal{M}}(a, b) = \emptyset \subseteq \Psi_{\mathcal{S}}(a, b)$ and (a, b) is safe.
2. First we remember the property (1) cited in the proof of theorem 17. So $\phi_{\mathcal{M}}(a) = \phi_{\mathcal{S}}(a) \wedge \phi_{\mathcal{M}}(b) = \phi_{\mathcal{S}}(b)$ implies that $\Phi_{\mathcal{M}}(a) = \Phi_{\mathcal{S}}(a) \wedge \Phi_{\mathcal{M}}(b) = \Phi_{\mathcal{S}}(b)$. Consider now a generic pair $(i, j) \in \Psi_{\mathcal{M}}(a, b)$. We have to prove that (i, j) is also in $\Psi_{\mathcal{S}}(a, b)$. Since $(i, j) \in \Psi_{\mathcal{M}}(a, b)$ then $val(i) = a \wedge val(j) = b$, i.e. $i \in \Phi_{\mathcal{M}}(a)$ and $j \in \Phi_{\mathcal{M}}(b)$. Then it also holds that $i \in \Phi_{\mathcal{S}}(a)$ and $j \in \Phi_{\mathcal{S}}(b)$. This means that $i, j \in E'$ and $val'(i) = a \wedge val'(j) = b$ so, since $(a, b) \in SD(\mathcal{S})$ then it must be $(i, j) \in P'$. This means that $(i, j) \in \Psi_{\mathcal{S}}(a, b)$ and, since we considered generic i, j it holds that $\Psi_{\mathcal{M}}(a, b) \subseteq \Psi_{\mathcal{S}}(a, b)$, so (a, b) is safe.

⊔

Example 27 Consider the three labeled posets below.



Diagrams for strong and weak dominance are below.



The pair (C, D) is unsafe in the scheduling $\mathcal{S}$. In fact, even if C strongly dominates D in $\mathcal{S}$, the scheduling is incomplete and there is only one C job. This means that $\mathcal{S}$ could execute the wrong C job (the one which does not precede D in $\mathcal{M}$) and so it could not satisfy the (C, D) dependency of the specification. Instead, in $\mathcal{S}'$ the pair (C, D) is safe. This is because in $\mathcal{S}'$ there are two C jobs, like in the specification, and since $(C, D) \in SD(\mathcal{S}')$ then both C jobs dominate B, so the dependency (C, D) of $\mathcal{M}$ is surely satisfied.

C. Verification algorithms for multipair structures

Theorem 20 can be employed to build a verification algorithm for complete schedulings (see fig. 1), while the algorithm in figure 2 uses theorem 26 to calculate the set of unsafe pairs. Both algorithms have linear complexity w.r.t. the number of pairs of the weak dominance. Be $\mathcal{M}$ a labeled poset and (X, ϕ, Q, ψ) the corresponding multipair structure. $SD(\mathcal{M})$ can be deduced by applying theorem 13 and $Q = WD(\mathcal{M})$ (see the construction of multipair structure on page 4). By construction, $\phi(a) = \phi_{\mathcal{M}}(a)$ for each a, and $\psi(a, b) = \psi_{\mathcal{M}}(a, b)$ for each a, b. This implies that multipair structures are sufficient to apply theorems 17, 20 and 26, so they can be used instead of labeled posets in the corresponding algorithms.

INPUT: a scheduling $\mathcal{S}$ and a specification $\mathcal{M}$
$incorrect \leftarrow FALSE$
$toRefine \leftarrow \emptyset$
FOR EACH pair (a,b) **IN** $WD(\mathcal{M})$ do
  IF $(a,b) \in SD(\mathcal{S})$ **THEN** do nothing // **(Verifying criterion (1))**
  ELSE IF $\psi_{\mathcal{S}}(a,b) < \psi_{\mathcal{M}}(a,b)$ **THEN** $incorrect \leftarrow TRUE$ // **(criterion (2))**
  ELSE IF $(a,b) \in SD(\mathcal{M})$ **THEN** $incorrect \leftarrow TRUE$ // **(criterion (3))**
  ELSE $toRefine \leftarrow toRefine \cup \{a,b\}$ // **(criterion (4))**
  ENDFOR
IF $incorrect = TRUE$ **THEN OUTPUT** "Scheduling is incorrect!"
ELSE IF $toRefine = \emptyset$ **THEN OUTPUT** "Scheduling is correct!"
ELSE OUTPUT "You have to refine the following values: " + $toRefine$

Figure 1: Algorithm for correctness of complete schedulings

IV. A SPECIAL CASE: LABELED CHAINS

As for normal chains, a labeled chain is a labeled poset (*E*, *X*, *P*, *val*) in which all elements are comparable, that is for each $i, j \in E$ it holds that $i = j$ or $(i, j) \in P$ or $(j, i) \in P$. They can be used to represent linear order models, linear schedulings and similar execution models (like straight line programs, as we will see in the next section). While a normal chain can be represented by a sequence, a labeled chain can be represented in the same way but replacing elements with corresponding labels.

Example 28 The following sequence

$$(A, B, C, B, D)$$

represents the labeled chain $\mathcal{M} = (E, X, P, val)$ defined below.

- $E = \{1, 2, 3, 4, 5\}$
- $X = \{A, B, C, D\}$
- $P = \{(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5)\}$
- $val(1) = A$, $val(2) = B$, $val(3) = C$, $val(4) = B$, $val(5) = D$

Even though labeled chains are less expressive than generic labeled partial orders, information is lost when transforming them into multipair structures, as shown in the following example.

Example 29 Consider the following labeled chains.

$$(A, B, B, A)$$
$$(B, A, A, B)$$

These two different labeled chains produce the same multipair structure. In fact both of them yield the multiset of values [*A*,*A*,*B*,*B*] and the multiset of pairs [(*A*,*A*), (*A*,*B*), (*A*,*B*), (*B*,*A*), (*B*,*A*), (*B*,*B*)].

The structural representation (as labeled sequences) contains more information, for example that in the first chain there exists a node (labeled with) *A* which precedes all nodes *B*.

However, if we restrict to labeled chains, a simpler criterion for strong dominance can be given.

Theorem 30 Let $\mathcal{M} = (E, X, P, val)$ be a labeled chain and let (X, ϕ, Q, ψ) be the corresponding multipair structure. Let $a, b \in X$ with $a \neq b$. Then $a \prec b$ if and only if $(a, b) \in Q$ and $(b, a) \notin Q$.

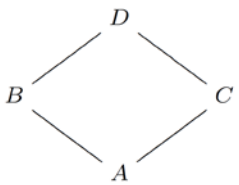
Proof:

- **"Only if" part (⇒).** Suppose that $a \prec b$. Consider generic i, j such that $val(i) = a$ and $val(j) = b$. Since $a \prec b$ then $(i, j) \in P$ and therefore $(a, b) \in Q$. Since P is antisymmetric then $(j, i) \notin P$ for all i, j with $val(i) = a$ and $val(j) = b$. Therefore $(b, a) \notin Q$.
- **"If" part (⇐).** Suppose that $(a, b) \in Q$ and $(b, a) \notin Q$. Consider generic i, j such that $val(i) = a$ and $val(j) = b$. Then $i \neq j$ because $val(i) = a \neq b = val(j)$. Moreover $(j, i) \notin P$ because otherwise $(b, a) \in Q$. Since the order is total then it must be $(i, j) \in P$. As i, j are generic we have $a \prec b$.

⊔

This theorem allows us to determine strong dominance using only the weak dominance relation Q without referring to ϕ and ψ.

Example 31 Consider the labeled chain $\mathcal{M}$ of example 28. The partial order corresponding to its strong dominance relation can be depicted by the following Hasse diagram.



INPUT: a scheduling $\mathcal{S}$ and a specification $\mathcal{M}$
$unsafe \leftarrow \emptyset$
FOR EACH pair (a,b) **IN** $WD(\mathcal{M})$ do
  IF $\phi_{\mathcal{M}}(a) \neq \phi_{\mathcal{S}}(a)$ **OR** $\phi_{\mathcal{M}}(b) \neq \phi_{\mathcal{S}}(b)$ **OR** $(a,b) \notin SD(\mathcal{S})$
    THEN $unsafe \leftarrow unsafe \cup \{(a,b)\}$
OUTPUT $unsafe$

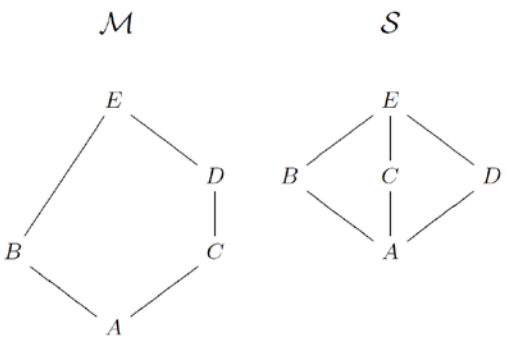
Figure 2: Algorithm to find unsafe pairs

In the examples below correctness and safeness criteria are applied to verify if a scheduling is correct with respect to a labeled chain specification.

Example 32 Let $\mathcal{M}$ be a labeled chain specification and let $\mathcal{S}$ be a complete scheduling of $\mathcal{M}$. Consider the criterion (1) of theorem 20 to detect the correctness of $\mathcal{S}$. If $SD(\mathcal{S}) \supseteq WD(\mathcal{M})$ then $WD(\mathcal{M}) = SD(\mathcal{M})$ and this implies that $\mathcal{M}$ has no ambiguities. An easy corollary of this proposition is that if a labeled chain specification $\mathcal{M}$ has ambiguous values then it is impossible to state that a scheduling of $\mathcal{M}$ is correct without refining these values. Then if the specification is ambiguous we can only detect the incorrectness, otherwise we have to refine it.

Example 33 Let $\mathcal{M}$ be the specification represented by the sequence (*A*,*B*,*A*,*B*,*B*) and let $\mathcal{S}$ be a complete scheduling represented by the sequence (*A*,*B*,*B*,*A*,*B*). In this case it holds that $\psi_{\mathcal{S}}(A,B) = 4$ and $\psi_{\mathcal{M}}(A,B) = 5$. Then criterion (2) of theorem 20 is fulfilled and we can conclude that $\mathcal{S}$ is not correct. In fact, in specification $\mathcal{M}$ the job *A* must precede the job *B* in five cases, and so $\mathcal{S}$ cannot satisfy all the corresponding five dependencies.

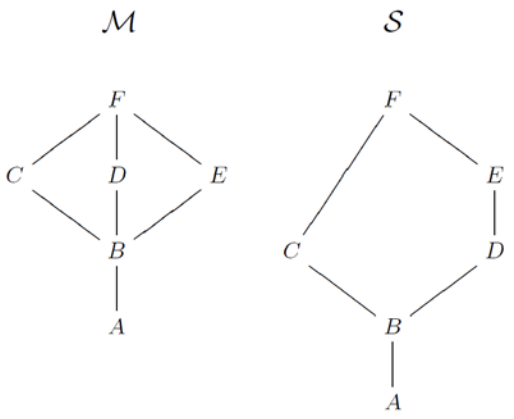
Example 34 Let $\mathcal{M}$ be the specification represented by the sequence (*A*,*B*,*C*,*C*,*D*,*B*,*E*) and let $\mathcal{S}$ be a complete scheduling represented by the sequence (*A*,*B*,*C*,*D*,*C*,*B*,*E*). Their strong dominance relations are represented by the following Hasse diagrams.



By observing the diagrams we notice that $(C,D) \in SD(\mathcal{M})$ but $(C,D) \notin SD(\mathcal{S})$, then criterion (3) of theorem 20 is fulfilled and $\mathcal{S}$ is not correct. In fact, in $\mathcal{M}$ the job *C* must precede the job *D* in all cases, and in $\mathcal{S}$ this is not true since *D* may also be executed before *C*.

Example 35 Consider the labeled chains $\mathcal{M}$ and $\mathcal{S}$ of previous example. We can reverse that example by considering now a specification $\mathcal{M}'$ = (*A*,*B*,*C*,*D*,*C*,*B*,*E*) and a complete scheduling $\mathcal{S}'$ of $\mathcal{M}'$ such that $\mathcal{S}'$ = (*A*,*B*,*C*,*C*,*D*,*B*,*E*). Now only criterion (4) of theorem 20 is fulfilled, so one needs to refine.

Example 36 Let $\mathcal{M}$ be a specification, let $\mathcal{S}$ be a partial scheduling of $\mathcal{M}$ and let (*a*, *b*) be a pair of values. Since theorem 26 applies, then we can deduce, similarly to what has been said in example 32, that if $(a, b) \notin WD(\mathcal{M})$ or $(a, b) \in SD(\mathcal{S})$ then values *a* and *b* cannot be ambiguous. But in this case we can still deduce safe pairs without refining. In fact, since in this case we focus on individual pairs, the specification $\mathcal{M}$ can also have ambiguities. Suppose for example that $\mathcal{M}$ corresponds to the sequence (*A*,*B*,*C*,*D*,*E*,*D*,*C*,*F*) and $\mathcal{S}$ corresponds to the sequence (*A*,*B*,*C*,*D*,*D*,*E*,*C*,*F*). The Hasse diagrams corresponding to their strong dominance relations are depicted below.



Applying the theorem 26 we can state that pairs (*A*, *B*), (*B*, *F*) and (*A*, *F*) are safe pairs.

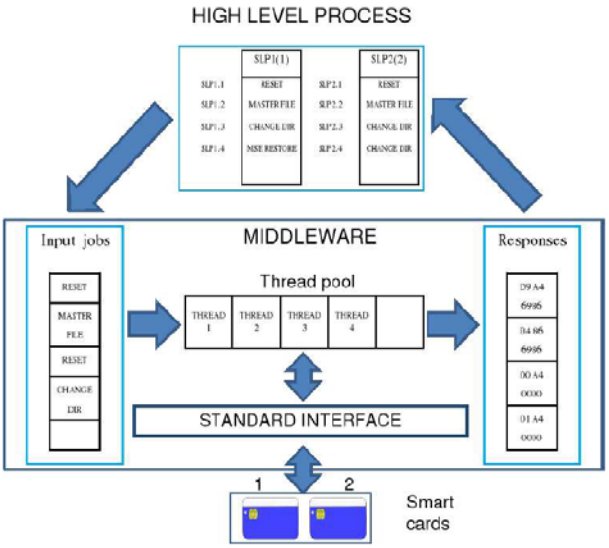


Figure 3: Smartcard interface system

V. A CASE STUDY: A SMART CARD INTERFACE SYSTEM

The Smartcards (SCs) are increasingly used for security applications like authentication and digital signature [11]. One of the significant roadblocks for SC use is SC interoperability: currently users are not able to easily connect SCs with SC applications independently of the type and manufacturer of the SC and application software. One solution would be a middleware that is able to handle a large variety of SCs and to connect them with various applications. Here we show how some of the methods developed in this work can be applied in such an environment.

Suppose we have one or more SCs inserted into SC readers and connected to several applications via a middleware (see figure 3) [12, 13, 14, 15]. We assume that on each SC one or more straight line programs (SLPs) can be executed for example to digitally sign a document. For the SLPs we use the notation `SLPn(m)` where the number `n` identifies the program and the number `m` identifies the smart card in which the program has to be executed. Such system usually uses a thread pool approach, placing the instructions of straight line programs in a queue, and a pool of threads executes them. The threads use a special middleware to get a standard interface with the smart cards. The structure of the system is depicted in figure 3.

Consider a generic set of SLPs (see figure 4). At the high-level process layer one can univocally identify an instruction as the sending application, the receiving SC and the order of commands are specified. Suppose now that we can observe the actual scheduling only at the middleware level i.e. in the thread pool. Here the association between a command in the thread and the sending application is broken, only the receiving smartcard is specified. Further the threads are essentially "anonymous" and only by evaluating the command itself it is possible to obtain some information about it. However doing this is costly and therefore we assume for the sake of this example that the limited information characterizing a command that can be efficiently obtained is limited to an element in

a)	SLP1(1)	b)	SLP2(2)
SLP1.1	A	SLP2.1	A
SLP1.2	B	SLP2.2	F
SLP1.3	H	SLP2.3	L
SLP1.4	C	SLP2.4	G
SLP1.5	B	SLP2.5	C
SLP1.6	D	SLP2.6	E

Figure 4: Two straight line programs with limited information about each job.

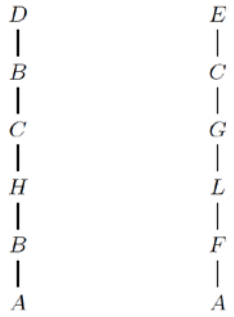


Figure 5: Labeled poset corresponding to the SLPs in figure 4

$\{A,B,C,D,E,F,G,H,L\}$. Figure 4 shows two SLPs to be executed on two different smartcards where the limited information available is listed instead of the command. Due to errors or during an attack threads may be relabeled with two effects: commands are executed in the wrong order and/or on the wrong SC. Therefore the only information one can rely on to verify the scheduling is the one letter characterization of the command. As a consequence certain commands may have become indistinguishable. The applications of our model to this situation is illustrated in the following examples.

Example 37 Consider the SLP of fig. 4(a) which, at process level, corresponds to the following chain:

```
(SLP1.1_A, SLP1.2_B, SLP1.3_H, SLP1.4_C,
        SLP1.5_B, SLP1.6_D)
```

This chain could also be regarded as the full specification of a correct scheduling which, if executed as described, ensures a correct outcome.

At middleware level, the high level information is lost, so the program number and the instruction number are not available anymore. We obtain the following *labeled chain*:

$$(A, B, H, C, B, D)$$

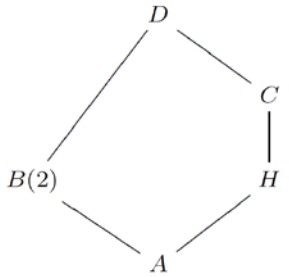
From this chain we can get the multipair structure (X, ϕ, Q, ψ) given by:

- $X = \{A,B,C,D,H\}$
- $\phi(x) = \begin{cases} 2 \text{ if } x = B \\ 1 \text{ if } x \in X \setminus \{B\} \end{cases}$
- $Q = \{(A,B), (A,H), (A,C), (A,D), (B,H), (B,C), (B,B), (B,D), (H,C), (H,B), (H,D), (C,B), (C,D)\}$
- $\psi(x, y) = \begin{cases} 2 \text{ if } (x, y) \in \{(A,B), (B,D)\} \\ 1 \text{ if } (x, y) \in Q \setminus \{(A,B), (B,D)\} \end{cases}$

Applying theorem 30 to Q we find that the strong dominance relation comprises the following pairs:

$$\prec = \{(A,B), (A,H), (A,C), (A,D), (B,D), (H,C), (H,D), (C,D)\}.$$

B and H as well as B and C are incomparable. The Hasse diagram with multiplicities (from ϕ) corresponding to the strong dominance relation is depicted below.



Example 38 Suppose that we have two SCs, and consider now both the SLPs shown in figure 4 which, at middleware level, correspond to the labeled poset in figure 5. The corresponding multipair structure (X, ϕ, Q, ψ) is given by:

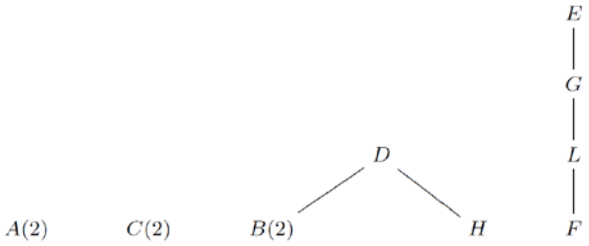
- $X = \{A,B,C,D,E,F,G,H,L\}$
- $\phi(x) = \begin{cases} 2 \text{ if } x \in \{A,B,C\} \\ 1 \text{ otherwise.} \end{cases}$
- Q and ψ are represented by the following matrix.

	A	B	C	D	E	F	G	H	L
A		2	2	1	1	1	1	1	1
B		1	1	2				1	
C		1		1	1				
D									
E									
F			1		1		1		1
G			1		1				
H		1	1	1					
L			1		1		1		

From this, applying theorem 13, we obtain the strong dominance relation:

$$\prec = \{(B,D), (H,D), (F,L), (F,G), (F,E), (L,G), (L,E), (G,E)\}$$

We can represent this relation by the following Hasse diagram with multiplicities.



In this examples we can apply the algorithm described in the previous section to verify the correctness of schedulings either for both smartcard together or for each individual smartcard. By observing and comparing a complete (partial) scheduling of jobs with this abstracted specification based on theorems 20 and 26 a scheduling can be checked against this specification.

VI. Conclusions

In this work we have developed methods and algorithms to describe and verify schedulings in the presence of undistinguishable jobs due to information loss. We discussed how dominance information in an approximated partial order of jobs can be obtained by counting indistinguishable nodes and pairs. This dominance information has been used to check the correctness of schedulings at a high level of abstraction. These methods can also be applied iteratively in order to verify the correctness of a scheduling without obtaining the full specification of each job. The efficiency of these algorithms increases with the ratio of the cardinalities of the strong dominance and weak dominance relations. Methods to increase this ratio will be developed for the specific applicative contexts (e.g. smartcards). In future work the verification problem discussed here w.r.t. to the order of jobs will be extended to take into account the potential replication of jobs or the introduction of jobs into the scheduling that do not exist in the specification. Further one can consider specific contexts where one can exploit the characteristics of particular partial order topologies (e.g. tree-like orders) to increase the efficiency of the verification algorithms.

REFERENCES

[1] E. Clarke, O. Grumberg, D. Peled , Model Checking, The MIT Press, Cambridge, MA, 2001.

[2] M. Popovic, I. Basicevic, An Approach to Formal Verification of Embedded Software, in *The 15th International Conference on COMPUTERS*, 2011, pp. 29-34. ISSN 1792-4251.

[3] Y. Aoki, S. Matsuura, A method for detecting unusual defects in enterprise system using model checking techniques, in *Proceedings of the 10th international conference on Software engineering, parallel and distributed systems*, p.165-171, February 20-22, 2011, Cambridge, UK.

[4] E. Clarke, O. Grumberg and D. Long, Model checking and abstraction, *ACM Trans. Program. Lang. Syst.* 16 (5), 1994, pp. 1512–1542.

[5] Wil van der Aalst et. al., Process Mining Manifesto, to be published in *Business Process Management Workshops 2011, Lecture Notes in Business Information Processing*, Vol. 99, Springer-Verlag, 2011.

[6] O. M. Hassan, M. S. Farag, M. M. MohieEl-Din, Reality Mining Via Process Mining, in *10th International Conference on Applied Informatics and Communications (AIC '10)*, ISSN: 1792-460X.

[7] M. S. Unluturk, K. Kurtel, A software method for managing event logs to improve quality and dependability of business processes, in *Proceedings of the 14th international conference on Computers*, p.89-94, July 23-25, 2010, Corfu Island, Greece.

[8] L. Boaro, E. Glorio, F. Pagliarecci and L. Spalazzi, Semantic Model Checking security requirements for web services, *Proc. of 2010 High Performance Computing & Simulation (HPCS'10)*, 2010.

[9] J. Gischer, Partial Orders and the Axiomatic Theory of Shuffle, Ph.D. Thesis, Computer Science Dept., Stanford University, 1984.

[10] V. Pratt, Modelling Concurrency with Partial Orders, *Internat. J. Parallel Programming* 15 (1), 1986, pp. 33–71.

[11] W. Rankl and W. Effing, *Smart Card Handbook*, 4th ed. West Sussex, UK: Wiley, 2010.

[12] M. Talamo, et al, Robustness and Interoperability Problems in Security Devices, in *Proceedings of 4th International Conferences on Information Security and Cryptology, INSCRYPT 2008*, 2008.

[13] M. Talamo, et al, Verifying extended criteria for the interoperability of security devices, in *Proceedings of 3rd International Symposium on Information Security*, IS08, ser. LNCS vol. 5332. Springer, 2008, pp. 1131-1139.

[14] M. Talamo, M. Galinium, C.H. Schunck, F. Arcieri, Interleaving command sequences: a threat to secure smartcard interoperability, in *The 10th International Conference on Information Security and Privacy (ISP 2011))*, Jakarta, Indonesia, 1-3 December 2011, ISBN: 978-1-61804-049-7.

[15] M. Talamo, M. Galinium, C.H. Schunck, F. Arcieri, Interleaving Commands: a Threat to the Interoperability of Smartcard Based Security Applications, in The International Journal of Computers and Communications, Issue 1, Volume 6, 2012, University Press, ISSN: 2074-1294 (http://www.universitypress.org.uk/journals/cc/17-840.pdf).