Real Time Scheduling Verification with Incomplete Information

Maurizio Talamo  
Nestor Laboratory and  
Department of Mathematics  
Tor Vergata University of Rome  
Italy  
talamo@nestor.uniroma2.it

Andrea Callia D’Iddio  
Department of Mathematics  
Tor Vergata University of Rome  
Italy  
calliadiddio@nestor.uniroma2.it

Christian H. Schunck, Franco Arcieri  
Nestor Laboratory  
Tor Vergata University of Rome  
Italy  
{schunck, arcieri}@nestor.uniroma2.it

Abstract: Schedulers play a critical role in ensuring the performance of IT systems and highly relevant for their security. In many security applications a scheduling must be analyzed rapidly and often this implies that not all information required to uniquely identify individual jobs can be obtained both for practical and efficiency reasons. Here we develop a framework for the automated, real time verification of a scheduling with limited information. Given a set of jobs and a specification of the dependencies between these jobs the task is to check whether the jobs has been scheduled in such an order that the dependencies are satisfied. Dependencies between jobs are specified using a partial order. Due to the loss of information some jobs may become indistinguishable and the partial order property may be lost. By introducing an approach based on abstraction at the combinatorial level we introduce a new framework to represent the approximated information and to define methods and algorithms to check the correctness of abstracted schedulings. Finally we present an application regarding scheduling process for smart cards. Further areas of application include model checking and process mining.

Key–Words: security, scheduling, smart cards, model checking

1 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 existing abstraction/refinement methods used in model checking [1] and it can be applied in process mining [2]. In particular this approach can address issues related to applying semantic model checking [3] 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.

ISBN: 978-1-61804-064-0

336
We represent a scheduling using a partial order of jobs, where the partial order represents the dependencies between the jobs. If the dependencies are satisfied in the scheduling and execution of the jobs, we assume that the outcome will be correct. 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.

2 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 [4] and Pratt [5] 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.

2.1 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 \( M = (E, X, P, \text{val}) \) 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 \( E \).
- \( \text{val} : E \rightarrow X \) 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

![Hasse diagram](image)

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

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

A labeled chain is a labeled poset \( (E, X, P, \text{val}) \) 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 \( (j, i) \in P \). 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 3** The following sequence

\((A, B, C, B, D)\)

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

- \( E = \{1, 2, 3, 4, 5\} \)
- \( X = \{A, B, C, D\} \)
- \( P = \{(1, 2), (1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (3, 5), (4, 5)\} \)
- \( \text{val}(1) = A, \text{val}(2) = B, \text{val}(3) = C, \text{val}(4) = B, \text{val}(5) = D \)
2.2 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 dominate 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 4 Let $\mathcal{M} = (E, X, P, \text{val})$ 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 $\text{val}(i) = a$ and $\text{val}(j) = b$ 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 5 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 6 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 $\text{val}(i) = a$ and $\text{val}(j) = c$. Now we choose a generic $k$ such that $\text{val}(k) = b$. 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 7 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 8 Let $\mathcal{M} = (E, X, P, \text{val})$ be a labeled poset. We say that a value $a$ weakly dominates a value $b$, in symbols $a \preceq_w b$, if $a$ and $b$ are in $X$ and there exist $i, j \in E$ such that $\text{val}(i) = a$, $\text{val}(j) = b$ and $(i, j) \in P$.

Again, we write $a \preceq b$ instead of $a \preceq_w b$ 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 9 Consider the labeled poset $\mathcal{M}$ of example 2. It holds that $A \preceq_w B$ and $B \preceq_w A$, so $\preceq_w$ is not asymmetric.

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

2.3 Multipair structures and strong dominance

Let $\mathcal{M} = (E, X, P, \text{val})$ 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 $\text{val}$ is injective (and then $\text{val}$ 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 $\text{val}$ is not injective. In this case expressivity is invariably lost as shown in the following examples.

Example 10 Consider the following labeled chains.

$(A, B, B, A)$

$(B, A, A, B)$
These different labeled chains yield the same set of pairs, even if we adopt multisets to consider multiple instances of values and pairs. In fact both labeled chains yield the values \([A,A,B,B]\) and the pairs \([\{A,B\}, \{A,B\}, \{A,C\}, \{A,D\}, \{B,C\}, \{B,D\}]\). The structural representation contains more information, for example that in the first chain there exists a node (labeled with) \(A\) which precedes all nodes \(B\).

If we generalize beyond chains, the loss of expressivity becomes even more evident. In tree-like orders, for example, branching emphasizes the loss of expressivity, as we can see in the following example.

**Example 11** Consider the following tree-like posets.

\[
\begin{array}{cccc}
C & D & C & D \\
A & B & B & A \\
& B & & \\
& & & B \\
B & & & \\
\end{array}
\]

Both posets 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.

\[
\begin{array}{c}
B \\
C \\
D \\
\end{array}
\]

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 12** A multipair structure is a 4-tuple \((X, \phi, Q, \psi)\) defined as below.

- \(X\) is a set of values.
- \(\phi : X \to \mathbb{N}\) is a multiplicity function for \(X\).
- \(Q \subseteq X \times X\) is a set of pairs of values in \(X\).
- \(\psi : Q \to \mathbb{N}\) is a multiplicity function for \(Q\).

A labeled poset \(M = (E, X, P, val)\) can be transformed into a multipair structure \((X, \phi, Q, \psi)\) 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 \(i \in E\) such that \(val(i) = a\).
- \(Q\) is the weak dominance relation of \(M\), that is \(Q = \{(a,b)\text{ such that } val(i) = a, val(j) = b\text{ and } (i,j) \in P \text{ for some } i,j\}\).
- 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 \(\phi\) counts how many times a value appears in the labeled poset, and \(\psi\) 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 \(\phi\) and a matrix for \(Q\) and \(\psi\).

**Example 13** Consider again the labeled poset \(M\) of example 2. Following the previous criterium, the corresponding multipair structure \((X, \phi, Q, \psi)\) 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 14** Let \(M = (E, X, P, val)\) be a labeled poset 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** (\(\Rightarrow\)). 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** (\(\Leftarrow\)). 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 \( \text{val}(j) = b \) 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 \( \text{val}(j) = b \) 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

\[
\psi(a, b) < \phi(a) \cdot \phi(b)
\]

\( \square \)

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.

### 2.3.1 Strong dominance and labeled chains

If we restrict the discussion to labeled chains, a simpler criterion for strong dominance can be given.

**Theorem 15** Let \( \mathcal{M} = (E, X, P, \text{val}) \) be a labeled chain and let \( (X, \phi, Q, \psi) \) 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:** Easy

This theorem allows us to determine strong dominance using only the weak dominance relation \( Q \) without referring to \( \phi \) and \( \psi \).

### 3 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 is given below.

**Definition 16** Given a labeled poset \( \mathcal{M} = (E, X, P, \text{val}) \), called a specification, a scheduling of the specification \( \mathcal{M} \) is a labeled poset \( \mathcal{S} = (E', X', P', \text{val}') \) such that \( E' \subseteq E \), and \( \text{val}' = \text{val}|_{E'} \).

#### 3.1 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 17** Given a specification \( \mathcal{M} = (E, X, P, \text{val}) \) and a scheduling \( \mathcal{S} = (E', X', P', \text{val}') \) of \( \mathcal{M} \), then \( \mathcal{S} \) is complete if and only if \( E' = E \) (and then \( X' = X \wedge \text{val}' = \text{val} \)).

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

**Definition 18** Given a labeled poset \( \mathcal{M} = (E, X, P, \text{val}) \) and given an element \( a \in X \) then \( \Phi_{\mathcal{M}}(a) \equiv \{ i \in E | \text{val}(i) = a \} \) and \( \phi_{\mathcal{M}}(a) \equiv |\Phi_{\mathcal{M}}(a)| \).

We can exploit \( \phi \) to verify if a scheduling is complete, as shown in the following theorem.

**Theorem 19** Let \( \mathcal{M} = (E, X, P, \text{val}) \) be a specification and let \( \mathcal{S} = (E', X', P', \text{val}') \) 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 \( a \in X \) the following property holds.

\[
\Phi_{\mathcal{S}}(a) \subseteq \Phi_{\mathcal{M}}(a)
\]

o “Only if” part \( (\Rightarrow) \).

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 \( \text{val}(i) = a \). Moreover, since \( i \notin \Phi_{\mathcal{S}}(a) \) and \( \text{val}(i) = a \) then it must be \( i \notin E' \). So \( E \neq E' \) and the scheduling \( \mathcal{S} \) is not complete.
"If" part (\(\iff\)). For each \(a \in X\), since property (1) holds, then \(\phi_S(a) = \phi_M(a)\) implies \(\Phi_S(a) = \Phi_M(a)\). 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 \(a = \text{val}(i)\). So \(i \in \Phi_M(a)\) and then \(i \in \Phi_S(a)\), and this means, by definition of \(\Phi\), that \(i \in E'\), so \(E \subseteq E'\) and then \(E = E'\), i.e. \(S\) is complete.

\(\square\)

3.2 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 20** Given a specification \(\mathcal{M} = (E, X, P, \text{val})\) and a complete scheduling \(S = (E, X, P', \text{val})\), then \(S\) 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 21** Given a labeled poset \(\mathcal{M} = (E, X, P, \text{val})\) and given two elements \(a, b \in X\) then \(\psi_M(a, b) \equiv \{(i, j) | (i, j) \in P \land \text{val}(i) = a \land \text{val}(j) = b\}\) and \(\psi_S(a, b) \equiv |\psi_M(a, b)|\).

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 \(\psi\) function are used.

**Theorem 22** Given a specification \(\mathcal{M} = (E, X, P, \text{val})\) and given a complete scheduling \(S = (E, X, P', \text{val})\) the following properties hold.

1. If \(SD(S) \supseteq WD(M)\) then the scheduling \(S\) is correct.

2. If there exists \((a, b)\) such that \(\psi_S(a, b) < \psi_M(a, b)\) then the scheduling \(S\) is not correct. In particular if \(WD(S) \nsubseteq WD(M)\) then \(S\) is not correct.

3. If \(SD(S) \nsubseteq SD(M)\) then the scheduling \(S\) is not correct.

4. Otherwise (i.e. if \(SD(M) \subseteq SD(S) \nsubseteq WD(M)\)) it is not possible to establish if \(S\) 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 \((\text{val}(i), \text{val}(j)) \in WD(M)\) and therefore by hypothesis \((\text{val}(i), \text{val}(j)) \in SD(S)\). By definition of strong dominance, for each \(l, m \in E\) such that \(\text{val}(l) = \text{val}(i)\) and \(\text{val}(m) = \text{val}(j)\) 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. \(S\) is correct.

2. If \(\psi_S(a, b) < \psi_M(a, b)\) then \(\psi_S(a, b) \subset \psi_M(a, b)\) so there exist \(i, j \in E\) such that \((i, j) \in \psi_M(a, b)\) but \((i, j) \notin \psi_S(a, b)\). Since \((i, j) \in \psi_M(a, b)\) then \((i, j) \in P\) and \(\text{val}(i) = a \land \text{val}(j) = b\). Moreover, since \((i, j) \notin \psi_S(a, b)\) and \(\text{val}(i) = a \land \text{val}(j) = b\) then \((i, j) \notin P'\). So \(P' \nsubseteq P\) and the scheduling \(S\) is not correct.

3. Suppose that \(SD(S) \nsubseteq SD(M)\). So there exists \((a, b) \in SD(M)\) such that \((a, b) \notin SD(S)\). Since \((a, b) \notin SD(S)\) then there exist \(i, j \in E\) such that \(\text{val}(i) = a \land \text{val}(j) = b\) but \((i, j) \notin P'\). But, since \(\text{val}(i) = a \land \text{val}(j) = b\), the fact that \((a, b) \in SD(M)\) implies that \((i, j) \in P\). Since \((i, j) \notin P'\) but \((i, j) \in P\), then \(P' \nsubseteq P\) i.e. \(S\) is not correct.

4. Since \(SD(S) \nsubseteq WD(M)\) there exists \((a, b) \in WD(M)\) such that \((a, b) \notin SD(S)\). Since \((a, b) \notin SD(S)\) there exist \(i, j \in E\) such that \(\text{val}(i) = a \land \text{val}(j) = b\) but \((i, j) \notin P'\). Now we have to verify if \((i, j) \in P\) to know if \(S\) can be correct. Since \(SD(S) \subseteq SD(M)\) and \((a, b) \notin SD(S)\) then \((a, b) \notin SD(M)\) so there exist \(i', j' \in E\) such that \(\text{val}(i') = a \land \text{val}(j') = b\) but \((i', j') \notin P\), and then the set \(A = \{l, m | \text{val}(l) = a \land \text{val}(m) = b\} \subset P\) is not empty. But, since \((a, b) \in WD(M)\) there also exist \(i'', j'' \in E\) such that \(\text{val}(i'') = a \land \text{val}(j'') = b\) and \((i'', j'') \in P\), so the set \(B = \{l, m | \text{val}(l) = a \land \text{val}(m) = b\} \subset P\) is non-empty also.

Since \(\text{val}(i) = a\) and \(\text{val}(j) = b\) then \((i, j)\) must be either in \(A\) or in \(B\). In the first case \(S\) is not correct, and in the second case \(S\) may be correct. The only way to find out it is to refine \(M\) and \(S\).
We can also check the correctness of a partial scheduling by applying locally criterion (1) of theorem 22. Below we define safe pairs.

**Definition 23** Let $\mathcal{M} = (E, X, P, \text{val})$ be a specification and let $S = (E', X', P', \text{val}')$ be a scheduling of $\mathcal{M}$. A pair $(a, b) \in X \times X$ is safe in $S$ with regard to $\mathcal{M}$ if and only if $\Psi_{\mathcal{M}}(a, b) \subseteq \Psi_S(a, b)$.

So a pair $(a, b)$ is safe in a scheduling $S$ 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 $S$. This can be used to check at runtime if a request to execute a job $b$ of $S$ can be satisfied even if the scheduling $S$ is not fully specified. In fact, it suffices to verify if for each job $a$ of $\mathcal{M}$ 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. Safe pairs and lazy approach can be used also in verification of complete scheduleings, because we can suspend the execution of an unsafe job until it gets refined, and in the mean time we can execute safe jobs.

As said above, by applying locally criterion (1) of theorem 22 we can check if a pair is safe using only weak and strong dominance plus the function $\phi$.

**Theorem 24** Let $\mathcal{M} = (E, X, P, \text{val})$ be a specification an let $S = (E', X', P', \text{val}')$ 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. $\phi_{\mathcal{M}}(a) = \phi_S(a) \land \phi_{\mathcal{M}}(b) = \phi_S(b) \land (a, b) \in SD(S)$

**Proof:**

Figure 1: Algorithm for correctness of complete scheduleings

We can also check the correctness of a partial scheduling by applying locally criterion (1) of theorem 22. Below we define safe pairs.

**Definition 23** Let $\mathcal{M} = (E, X, P, \text{val})$ be a specification and let $S = (E', X', P', \text{val}')$ be a scheduling of $\mathcal{M}$. A pair $(a, b) \in X \times X$ is safe in $S$ with regard to $\mathcal{M}$ if and only if $\Psi_{\mathcal{M}}(a, b) \subseteq \Psi_S(a, b)$.

So a pair $(a, b)$ is safe in a scheduling $S$ 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 $S$. This can be used to check at runtime if a request to execute a job $b$ of $S$ can be satisfied even if the scheduling $S$ is not fully specified. In fact, it suffices to verify if for each job $a$ of $\mathcal{M}$ 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. Safe pairs and lazy approach can be used also in verification of complete scheduleings, because we can suspend the execution of an unsafe job until it gets refined, and in the mean time we can execute safe jobs.

As said above, by applying locally criterion (1) of theorem 22 we can check if a pair is safe using only weak and strong dominance plus the function $\phi$.

**Theorem 24** Let $\mathcal{M} = (E, X, P, \text{val})$ be a specification an let $S = (E', X', P', \text{val}')$ 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. $\phi_{\mathcal{M}}(a) = \phi_S(a) \land \phi_{\mathcal{M}}(b) = \phi_S(b) \land (a, b) \in SD(S)$

**Proof:**

Figure 2: Algorithm to find unsafe pairs

1. If $(a, b) \notin WD(\mathcal{M})$ then there are no pairs $(i, j) \in P$ such that $\text{val}(i) = a \land \text{val}(j) = b$. Thus $\Psi_{\mathcal{M}}(a, b) = \emptyset \subseteq \Psi_S(a, b)$ so $(a, b)$ is safe.

2. First we remember the property (1) cited in the proof of theorem 19. So $\phi_{\mathcal{M}}(a) = \phi_S(a) \land \phi_{\mathcal{M}}(b) = \phi_S(b)$ implies that $\Phi_{\mathcal{M}}(a, b) = \Phi_S(a, b)$. Consider now a generic pair $(i, j) \in \Psi_{\mathcal{M}}(a, b)$. We have to prove that $(i, j)$ is also in $\Psi_S(a, b)$. Since $(i, j) \in \Psi_{\mathcal{M}}(a, b)$ then $\text{val}(i) = a \land \text{val}(j) = b$, i.e. $i \in \Phi_{\mathcal{M}}(a)$ and $j \in \Phi_{\mathcal{M}}(b)$. Then it also holds that $i \in \Phi_S(a)$ and $j \in \Phi_S(b)$. This means that $i, j \in E'$ and $\text{val}'(i) = a \land \text{val}'(j) = b$ so, since $(a, b) \in SD(S)$ then it must be $(i, j) \in P'$. This means that $(i, j) \in \Psi_S(a, b)$ and, since we considered generic $i, j$ it holds that $\Psi_{\mathcal{M}}(a, b) \subseteq \Psi_S(a, b)$, so $(a, b)$ is safe.

3.3 Verification algorithms for multipair structures

Theorem 22 can be employed to build a verification algorithm for complete scheduleings (see fig. 1), while the algorithm in figure 2 uses theorem 24 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 $(X, \phi, Q, \psi)$ the corresponding multipair structure. $SD(\mathcal{M})$ can be
deduced by applying theorem 14 and $Q = WD(M)$ (see the construction of multipair structure on page 4). By construction, $\phi(a) = \phi_M(a)$ for each $a$, and $\psi(a, b) = \psi_M(a, b)$ for each $a, b$. This implies that multipair structures are sufficient to apply theorems 19, 22 and 24, so they can be used instead of labeled posets in the corresponding algorithms.

4 A case study: a smart card interface system

Smartcards (SCs) are increasingly used for security applications like authentication and digital signature [6]. 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) [7, 8, 9]. 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 $SLP_n(m)$ 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 $\{A, B, C, D, E, F, G, H\}$. 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 example.

Example 25 Suppose that we have two SCs, and consider the SLPs shown in figure 4 which, at middleware level, correspond to the labeled poset in figure 5. The corresponding multipair structure $(X, \phi, Q, \psi)$ 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 $\psi$ are represented by the following matrix.
From this 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 example 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 22 and 24 a scheduling can be checked against this specification.

5 Conclusion

We have seen how to get dominance information in an approximated partial order of jobs by counting indistinguishable nodes and pair. As we have explained, this dominance information can be used to check correctness of schedulings at high level of abstraction. This checking can be put in an iterative refinement cycle which tries to determine the correctness of a scheduling without using the whole specification of each job. Given an approximated partial order, we can define its precision degree as the ratio between the cardinals of its strong dominance relation and its weak dominance relation. If this ratio is high the algorithm become more effective. Methods to increase this ratio will be developed for the specific applicative contexts.

References:


