The Asynchronous Committee Meeting Problem

Javier Esparza[1] and Bernhard von Stengel[2]

[1] Institut für Informatik, Universität Hildesheim, Samelsonplatz 1, 31141 Hildesheim, Germany. email: esparza@informatik.uni-hildesheim.de

[2] Informatik 5, Universität der Bundeswehr München, 85577 Neubiberg, Germany. email: i51bbvs@rz.unibw-muenchen.de

Abstract. The committee meeting problem consists in finding the earliest meeting time acceptable to every member of a committee. We consider an asynchronous version of the problem that does not presuppose the existence of a global clock, where meeting times are maximal antichains in a poset of 'local times', and propose an efficient algorithm to solve it. A generalization, the private meeting problem, where the earliest time for a meeting *without* some committee members is sought, turns out to be NP-complete. However, it can be solved in polynomial time if the poset is N-free, that is, representing a precedence of the arcs (and not the nodes) of an acyclic directed graph. This special case is relevant, because it allows to improve the key algorithm of the model checking technique developed in [4] for Petri nets.

Key words. Committee meeting problem, maximal antichains, N-free posets, Petri net unfolding

1 The asynchronous committee meeting problem

The committee meeting problem is an elementary, well-known problem of program design. It is for instance the first problem used by Chandy and Misra [3] to illustrate their UNITY programming methodology. The problem is to find the earliest meeting time acceptable to every member of a committee. Time instants are linearly ordered, and there exists a zero time without predecessor. Every committee member labels a subset of the time instants as 'free'. The desired output is the earliest time which is labeled as 'free' by all members.

A simple algorithm (with several variants, as shown in [3]) can be used to solve the problem: time zero is taken as the first candidate to a solution; if some person cannot meet at that time, she suggests her next free time as new candidate, and the procedure is iterated until nobody makes a new suggestion. It is not difficult to prove that, if some meeting time exists, then the algorithm yields the earliest one.

This problem presupposes the existence of a global clock, which can be viewed as a totally ordered set of times that are common to every committee member. We study an asynchronous version of the problem in which no global clock exists. In this version, every person has her own totally ordered set of local times, or, as we shall say, *states*. The states can also be seen as pauses between the execution of two tasks. The time taken by a task is not known in advance, and therefore it is not possible to ensure that, for instance, the first pause will simultaneously take place for all persons. However, in addition to the (fixed) sequential order of the tasks of one person, there may be dependencies between certain tasks of different persons, which impose additional precedences between the states. Thereby, the union of the (pairwise disjoint) sets of states is partially ordered.

The total order of the special synchronous case is thereby represented by introducing for each time instant separate states, one for each person, that are mutually incomparable and otherwise precede all states corresponding to later time instants. Furthermore, *any* partially ordered set (poset) may arise from the asynchronous meeting problem by partitioning the poset into chains (totally ordered subsets) which are assigned to committee members; here, only finite posets will be considered.

Figure 1.1 shows a case in which the committee consists of three people, A, B and C, whose sets of states contain 4, 5 and 3 elements, respectively. As in the synchronous case, every person labels a subset of her own states as 'free'. The free states are shown in boldface.

$$\begin{tikzcd}
A: & a_1 \arrow[rr] & & \mathbf{a_2} \arrow[rr] & & a_3 \arrow[rr] \arrow[dr] & & \mathbf{a_4} \\
B: & b_1 \arrow[r] \arrow[urr] & \mathbf{b_2} \arrow[rr] \arrow[drr] & & b_3 \arrow[rr] & & \mathbf{b_4} \arrow[r] & b_5 \\
C: & \mathbf{c_1} \arrow[rrr] \arrow[ur] & & & c_2 \arrow[rrr] \arrow[urrr] & & & \mathbf{c_3}
\end{tikzcd}$$

Fig. 1.1. An instance of the asynchronous version of the meeting problem.

Two states where one precedes the other are called *comparable*. An *antichain* is a set of pairwise incomparable states. Here, a maximal antichain with respect to set inclusion is called a *cut*. A cut can be interpreted as a snapshot of the committee, which tells which members have finished which tasks at a certain moment. For instance, $\{a_3, b_3, c_2\}$ is a cut of the partial order of Fig. 1.1, but $\{a_3, b_1, c_1\}$ is not, because a_3 and b_1 are comparable (and so A and B cannot simultaneously be in a_3 and b_1).

Cuts are the asynchronous counterpart of the global time instants in the asynchronous case. There is a natural notion of temporal precedence between cuts, which corresponds to the precedence between global time instants of the synchronous case: We say that a cut is *earlier* than another if every state of the first precedes (not necessarily strictly) a suitable state of the second. Independently of the relative speeds of the local clocks, if a cut Y is earlier than a cut Z, then Z can only be observed after Y. In Fig. 1.1, $\{a_4, b_3, c_2\}$ is earlier than $\{a_4, b_5, c_3\}$, but not earlier than $\{a_3, b_3, c_3\}$. It is easy to see that the 'earlier than' relation on cuts is a partial order.

The asynchronous committee meeting problem is to find an earliest cut containing one 'free' state for each member of the committee. In the example of Fig. 1.1, the reader can check that the unique earliest cut satisfying this condition is $\{a_4, b_4, c_3\}$.

2 Formalization and algorithm

Before proceeding to give an algorithm for this problem, we introduce further notations and conventions.

The set of states is denoted by X, which is a finite set to guarantee termination of the algorithm, possibly with a negative result if no common meeting time exists. The precedence relation between states is a partial order and denoted by $\leq$, so the poset under consideration is $(X, \leq)$. A state x is *smaller* than, or *precedes*, a state y if $x \leq y$. For a subset Y and an element x of X, say that x precedes Y if x precedes some element of Y. Given two cuts A, B, we denote by $A \sqsubseteq B$ that A is earlier than B, that is, each element of A precedes B. A *smallest* state is a $\leq$-minimal state (with no predecessor), but not necessarily smaller than all states. Similarly, an *earliest* cut is $\sqsubseteq$-minimal.

The problem is stated as follows. Each state is labeled with a label from a finite set $\mathcal{S}$, or unlabeled. Any two states that carry the same label (from $\mathcal{S}$) are comparable. Then, find an earliest cut containing one s-labeled element for every label s in $\mathcal{S}$.

The interpretation is that the set X is the disjoint union of the sets of states of the given persons. Each person has a separate label in $\mathcal{S}$ to mark certain of her states as 'free', while the others are unlabeled and thus marked as 'blocked' (these are not distinguished per person). The states of any one person are totally ordered.

We shall consider a slightly more general problem: Given a subset $\mathcal{G}$ of $\mathcal{S}$ of 'prescribed' labels, find an earliest cut containing one s-labeled element for every label s in $\mathcal{G}$. The asynchronous committee meeting problem corresponds to the case $\mathcal{G} = \mathcal{S}$.

A cut containing an (obviously at most one) s-labeled element for every label s in $\mathcal{G}$ shall be called a $\mathcal{G}$*-cut.* Notice that the set of $\mathcal{G}$-cuts may be empty, so that it has no $\sqsubseteq$-minimal element, and then the problem has no solution. However, if such a cut exists, then it is unique, so it is *the* earliest $\mathcal{G}$-cut:

Proposition 2.1. *Let the set of $\mathcal{G}$-cuts be nonempty. Then it has exactly one $\sqsubseteq$-minimal element.*

Proof. Let A, B be two $\mathcal{G}$-cuts, and C be the set of largest states preceding both A and B. It is easy to see that the elements of A preceding B and the elements of B preceding A belong to C, which comprise at least one of the s-labeled states in A or B for each label s in $\mathcal{G}$. A further consequence is that C is a cut and in fact the lattice-theoretic *meet*, the greatest lower bound with respect to $\sqsubseteq$, of A and B (it is well known [1] that the cuts of any poset form a lattice). So C is also a $\mathcal{G}$-cut. The meet of all $\mathcal{G}$-cuts is the unique earliest $\mathcal{G}$-cut. $\square$

To simplify the algorithm, we assume that the set of $\mathcal{G}$-cuts is nonempty, and denote by C its earliest element. This can be achieved for any poset $(X, \leq)$ by 'stacking' an extra $\mathcal{G}$-cut D on top of X, that is, D is a set of additional states (not in X), one for each label in $\mathcal{G}$, which are mutually incomparable and larger than any element of X. The algorithm is then applied to $X \cup D$ instead of X, and if it produces an earliest $\mathcal{G}$-cut containing elements of D, then X itself has no $\mathcal{G}$-cut, so the original problem has no solution.

The problem is then to design an algorithm that yields C as output. We use a variable A, which is initialized to the earliest cut of $(X, \leq)$, namely the set of smallest elements of X. A loop increases A until it becomes a $\mathcal{G}$-cut, while preserving $A \sqsubseteq C$. When the loop has ended, A is the output.

The algorithm uses the following notation: given an element x of X, the set $\downarrow x$ (read 'proper predecessors of x') consists of the states smaller than x and different from it. For a subset A of X, the set $\downarrow A$ is defined as the union of the sets $\downarrow x$ for every element x of A. The set of labels of A is denoted by $l(A)$.

```
$A :=$ set of smallest elements of $X$;
do  { invariant: $A$ is a cut and $A \sqsubseteq C$ }
    there exists $s$ in $\mathcal{G} \setminus l(A)$ $\longrightarrow$
        $x :=$ smallest $s$-labeled state not preceding $A$;
        $A :=$ set of smallest elements of $X \setminus \downarrow(A \cup \{x\})$
od
```

Let us prove the correctness of the algorithm. The loop guard simply states that A is not a $\mathcal{G}$-cut. The invariant then guarantees that, upon termination of the algorithm, A is a $\mathcal{G}$-cut and $A \sqsubseteq C$. By the $\sqsubseteq$-minimality of C, $A = C$. The invariant holds before the first execution of the loop because the set of minimal elements of X is the unique $\sqsubseteq$-minimal cut. The following lemma and proposition show that the invariant is preserved by the body of the loop.

Lemma 2.2. *Let Y be a subset of X, and let Z be set of smallest elements of $X \setminus \downarrow Y$. Then Z is a cut.*

Proof. Since the smallest elements of any set are pairwise incomparable, Z is an antichain. To prove that Z is a cut, let x be an arbitrary element of X. If $x \in X \setminus \downarrow Y$, then x is larger than some element of Z by the definition of Z. If $x \in \downarrow Y$, then x is smaller than some largest element y of Y. So $y \in Z$ by the definition of Z. In both cases, x is comparable with some element of Z, so Z is a cut. $\square$

Proposition 2.3. *Let A be a cut so that $A \sqsubseteq C$ and so that for some label s, no element of A is s-labeled. Let x be the smallest s-labeled state not preceding A. Let B be the set of smallest states of $X \setminus \downarrow(A \cup \{x\})$. Then B is a cut and $B \sqsubseteq C$.*

Proof. B is a cut by Lemma 2.2. It remains to show $B \sqsubseteq C$, i.e., every element of B is smaller than some element of C. By the definition of B, it suffices to prove that C is a subset of $X \setminus \downarrow(A \cup \{x\})$, or, equivalently, $C \cap (\downarrow A \cup \downarrow x) = \emptyset$.

$C \cap \downarrow A = \emptyset$ because $A \sqsubseteq C$ and C is a cut. To prove $C \cap \downarrow x = \emptyset$, observe that C contains one s-labeled element y, which is comparable with x and not smaller than an element of A, so $x \leq y$ by the definition of x; since C is an antichain, it contains no proper predecessor of x. $\square$

To prove termination, it suffices to show that the body of the loop strictly increases A with respect to the partial order $\sqsubseteq$. In other words, that A is earlier than, and different from, the set of smallest elements of $X \setminus \downarrow(A \cup \{x\})$. The first part is trivial. For the second part, observe that $x \notin A$ (because A contains no s-labeled element), but x is one of the smallest elements of $X \setminus \downarrow(A \cup \{x\})$.

To estimate the running time of the algorithm, its input has to be made precise. We choose as input the *diagram* $(X, \prec)$ of the poset, where $x \prec y$ if y is an *immediate successor* of x, that is, $x < y$ and there exists no state z with $x < z < y$. Since $(X, \leq)$ is finite, $\leq$ can be recovered as the reflexive and transitive closure of $\prec$.

The cut A can be increased with respect to $\sqsubseteq$ at most $|X|$ times. So the number of iterations of the loop is at most $|X|$. An iteration requires to compute the element x and the set of smallest elements of $X \setminus \downarrow(A \cup \{x\})$, which can be done in $O(|X|+|\prec|)$ time. So the algorithm requires at most $O\big(|X| \cdot (|X|+|\prec|)\big)$ time.

3 The asynchronous private meeting problem

A subset of the committee members wish to meet, for instance those belonging to a certain political party. However, they also want to make sure that, when they meet, the committee members of another party are busy, i.e., that none of them is in a 'free' state. To this end, a new set $\mathcal{H}$ of 'forbidden' labels is introduced besides the set $\mathcal{G}$. The asynchronous private meeting problem is to find an earliest cut C of $(X, \leq)$ so that:

(1) for every label $s \in \mathcal{G}$, some element of C is s-labeled, and

(2) for every label $s \in \mathcal{H}$, no element of C is s-labeled.

The set of cuts satisfying these two conditions may be empty. Furthermore, even if the set is nonempty it may happen that it has no unique $\sqsubseteq$-minimal element. Figure 3.1 shows such a poset (in the conventional way of drawing poset diagrams, with smaller elements further down). There, let the states a and b carry one label and a' and b' another, both labels belonging to $\mathcal{G}$, and the state x, indicated by a white dot, have a forbidden label in the set $\mathcal{H}$. Then both $\{a, a'\}$ and $\{b, b'\}$ are cuts satisfying (1) and (2), but their meet $\{a, x, b'\}$ is not.

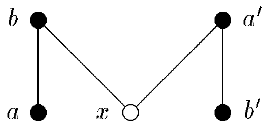

Fig. 3.1. A poset with no earliest cut fulfilling (1) and (2).

Thus, for a general poset, the asynchronous private meeting problem may not have a unique answer. Even finding one possible solution is much harder than in the previous case. This question turns out to be NP-complete already for $\mathcal{G} = \emptyset$. In this case, the condition on $\mathcal{G}$ is vacuously fulfilled, and the problem reduces to finding an earliest cut of $(X, \leq)$ containing no element with a label in $\mathcal{H}$. An even weaker question is whether there exists such a cut at all: if we color the elements with labels in $\mathcal{H}$ white and the rest black, the problem is to find a cut containing only black elements.

Proposition 3.1. *The following problem is NP-complete: Given a finite poset $(X, \leq)$ and each element of X being either black or white, find a cut in X containing only black elements.*

Proof. As before, call the elements of X states. A cut C in X is an antichain so that every state in X compares with an element of C. Since this is verified fast, the problem belongs to the complexity class NP.

The NP-complete problem 3SAT will be reduced to this problem, which will show that it is NP-complete [5]. An instance of 3SAT is a Boolean expression in conjunctive normal form, given by a conjunction of clauses, each of which is a disjunction of exactly three literals, which are either positive, given by a Boolean variable, or negative, given by a negated variable. The problem is to decide if this Boolean expression is satisfiable, that is, to find an assignment of truth values to the variables so that each clause evaluates to true.

Given an instance of 3SAT, construct the poset $(X, \leq)$ as follows. For each variable x of the expression, introduce two black states $p(x)$ and $n(x)$ and a white state $w(x)$ with $n(x) < w(x) < p(x)$; between these states, considered for all variables x, these are the only comparabilities, and the white states $w(x)$ will compare with no other states. Therefore, a cut C that is a solution to the problem contains for each variable x either $p(x)$ or $n(x)$ (which is interpreted as x assuming the value true or false, respectively), since each white node must compare with some element of C. Additional white states will be used to encode the clauses. If a clause c contains only positive literals x, y and z, say, introduce a white state $w(c)$ with $w(c) < p(x)$, $w(c) < p(y)$ and $w(c) < p(z)$ and no further comparabilities, so C is a solution only if it contains at least one of the states $p(x)$, $p(y)$ or $p(z)$, that is, if the clause c is true. Similarly, a clause c containing only negative literals $\overline{x}$ (x negated), $\overline{y}$ and $\overline{z}$ is represented by a white node $w(z)$ larger than $n(x)$, $n(y)$ and $n(z)$.

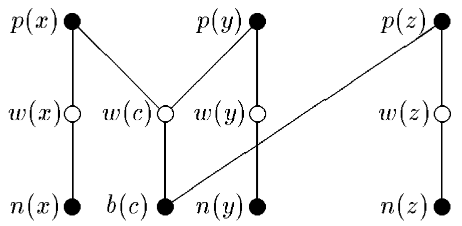

Fig. 3.2. Encoding of the clause c given by $x \vee y \vee \overline{z}$.

A 'mixed' clause c contains the literals x, y and $\overline{z}$ for variables x, y, z (or literals x, $\overline{y}$ and $\overline{z}$, in which case c will be encoded analogously). As shown in Fig. 3.2, a white node $w(c)$ is introduced with $w(c) < p(x)$ and $w(c) < p(y)$, and an additional black node $b(c)$ with $b(c) < w(c)$ and $b(c) < p(z)$ (note that $n(z) < w(c)$ would not work since then $n(z) < p(x)$). Then, a solution C has to contain at least $p(x)$, $p(y)$ or $b(c)$ to contain a state comparing with $w(c)$. The clause c is true if x or y is true, represented by $p(x) \in C$ respectively $p(y) \in C$, or if z is false, represented by $n(z) \in C$; only in this case, $b(c)$ can belong to C, since otherwise $p(z) \in C$ and $p(z)$ compares with $b(c)$. This encodes mixed clauses.

The poset is constructed in polynomial time from the 3SAT expression, which is satisfiable if and only if there is a cut containing only black nodes. □

4 The private meeting problem for N-free posets

In this section, we consider the special class of N-free posets, where the asynchronous private meeting problem always has a unique answer, and can be solved in polynomial time. These are the (always finite) posets $(X, \leq)$ satisfying the following condition:

(*) If x, y, z are states with $x \prec y$ and $x \prec z$, then $\downarrow y = \downarrow z$.

Notice that the poset of Fig. 3.1 does not satisfy this condition: b and a' are immediate successors of x, and $a \in \downarrow b$ but $a \notin \downarrow a'$.

The following equivalent conditions to (*) are better known in the literature: X contains no four distinct elements a, b, c, d with $a < b$, $c \prec b$, $c < d$ and otherwise being incomparable [6], or, more specially, with $a \prec b$, $c \prec b$, $c \prec d$, otherwise being incomparable [7]. In the latter case, the states a, b, c, d form an N-shaped part of the poset diagram, which is why such posets are called N-free. Another equivalent condition 'CAC' is chain-antichain-completeness: every maximal chain intersects every maximal antichain [6]. Finally, ordering the arcs of an acyclic directed graph (with $x \prec y$ if y is an arc starting at the endpoint of the arc x) yields an N-free poset, and conversely, for each N-free poset such a directed graph is easily constructed.

In the following, consider an N-free poset. First, we establish that the private meeting problem has a unique solution. That is, there is a unique earliest cut satisfying the conditions (1) and (2) in Sect. 3 above, where the labels of the elements of the cut include all labels from $\mathcal{G}$ and none from $\mathcal{H}$, if there is such a cut at all.

Proposition 4.1. *For an N-free poset, let the set of cuts satisfying conditions (1) and (2) be nonempty. Then it has exactly one $\sqsubseteq$-minimal element.*

Proof. Let A, B be two cuts satisfying (1) and (2). As in Proposition 2.1, the meet C of the cuts A and B consisting of the maximal states preceding both A and B is a $\mathcal{G}$-cut. We claim that C is also a subset of $A \cup B$ (in fact, it equals the set of smallest elements of $A \cup B$). Assume that some x in C belongs neither to A nor to B, and consider $a \in A$, $b \in B$ and states y, z such that $x \prec y \leq a$ and $x \prec z \leq b$. Since y precedes A but not B by the maximality of x, and B is a cut, $b' \in \downarrow y$ for some b' in B, but $b' \notin \downarrow z$, contradicting (*). Thus, since neither A nor B has elements with a label in $\mathcal{H}$, neither does C. So C satisfies (2) as well. $\square$

We assume that the set of cuts satisfying conditions (1) and (2) is nonempty, and denote by C its earliest element. The algorithm to compute C is an extension of that described in Sect. 2. We use a variable A, which is initialized to the set of smallest elements of X. A loop increases A with respect to $\sqsubseteq$ until it satisfies (1) and (2), while preserving $A \sqsubseteq C$. As before, $l(A)$ denotes the set of labels of A.

```
$A :=$ set of smallest elements of $X$;
do  { invariant: $A$ is a cut and $A \sqsubseteq C$ }
    there exists $s$ in $\mathcal{G} \setminus l(A) \longrightarrow$
        $x :=$ smallest $s$-labeled state not preceding $A$;
        $A :=$ set of smallest elements of $X \setminus \downarrow(A \cup \{x\})$
▯   there exists $s$ in $\mathcal{H} \cap l(A) \longrightarrow$
        $x :=$ an immediate successor of the unique $s$-labeled element of $A$;
        $A :=$ set of smallest elements of $X \setminus \downarrow(A \cup \{x\})$
od
```

The algorithm is proved correct along the lines of the first algorithm. The loop guards (both of which have to be false to terminate the loop) now state that (1) or (2) does not hold for A. So the invariant guarantees that, upon termination, A is a cut satisfying (1) and (2), and $A \sqsubseteq C$. By the $\sqsubseteq$-minimality of C, $A = C$.

The invariant holds before the first execution of the loop because the set of smallest elements of X is the unique earliest cut. The first alternative (i.e., the sequence of statements after the first guard) preserves the invariant by Proposition 2.3. The following proposition shows that the second alternative also preserves the invariant.

Proposition 4.2. *Let A be a cut with $A \sqsubseteq C$ that contains an element y carrying a label s from $\mathcal{H}$, and let x be an immediate successor of y. Let B be the set of smallest elements of $X \setminus {\downarrow}(A \cup \{x\})$. Then B is a cut and $B \sqsubseteq C$.*

Proof. B is a cut by Lemma 2.2. It remains to show $B \sqsubseteq C$. By the definition of B, it suffices to prove $C \subseteq X \setminus {\downarrow}(A \cup \{x\})$, that is, $C \cap ({\downarrow}A \cup {\downarrow}x) = \emptyset$.

$C \cap {\downarrow}A = \emptyset$ because $A \sqsubseteq C$. To prove $C \cap {\downarrow}x = \emptyset$, note that since C does not contain any s-labeled element and $A \sqsubseteq C$, some immediate successor z of y is smaller than some element of C. Since C is an antichain, $C \cap {\downarrow}z = \emptyset$. By assumption (*), ${\downarrow}x = {\downarrow}z$, so $C \cap {\downarrow}x = \emptyset$. □

To prove termination, it suffices to show that the body of the loop strictly increases A with respect to the partial order $\sqsubseteq$. This was already shown for the first alternative in Sect. 2. The proof for the second alternative is analogous.

The running time does not change compared to the first algorithm. Again, the number of iterations of the loop is at most $|X|$. The element x can be computed, for both alternatives, in $O(|X| + |{\prec}|)$ time, and so can the set of smallest elements of $X \setminus {\downarrow}(A \cup \{x\})$.

5 The private meeting problem for nonsequential processes

In this section we consider Petri nets, a well-known formal model of concurrent systems, thoroughly studied in the literature [8]. Petri nets can be given a partial order semantics in terms of certain labeled posets called nonsequential processes [2]. We shall observe that the private meeting problem for nonsequential processes corresponds to a problem stated in [4], and that nonsequential processes satisfy the condition of Sect. 4. As will be indicated, the new algorithm is more efficient than the algorithm described in [4].

A *net* is a triple $N = (S, T, F)$, where $S \cap T = \emptyset$ and $F \subseteq ((S \times T) \cup (T \times S))$. The elements of S are called *places*, the elements of T *transitions*, and the elements of F *arcs*. The *preset* (*postset*) of an element x of $S \cup T$ is the set of nodes y such that $(y, x) \in F$ ($(x, y) \in F$).

A *marking* of N is a mapping M from S to the nonnegative integers. A Petri net is a pair (N, M_0), where N is a net and M_0 an initial marking. Graphically, the places of a Petri net are represented by circles, and its transitions by boxes. If (x, y) is a arc, an arrow is drawn from x to y. Finally, if $M_0(s) = n$ for a place s, then n black dots or tokens are drawn inside the circle corresponding to s. Figure 5.1 shows a Petri net. A transition t is *enabled* at a marking M if $M(s) > 0$ for every place s in the preset of t. If a transition t is enabled at a marking M, then it can *occur*, yielding a marking obtained by removing one token from each place in the preset of t and adding one token to each place in the postset of t. Given a Petri net (N, M_0), a marking M of N is *reachable* if there exists a finite sequence of successively occurring transitions initially enabled at M_0 and eventually yielding M.

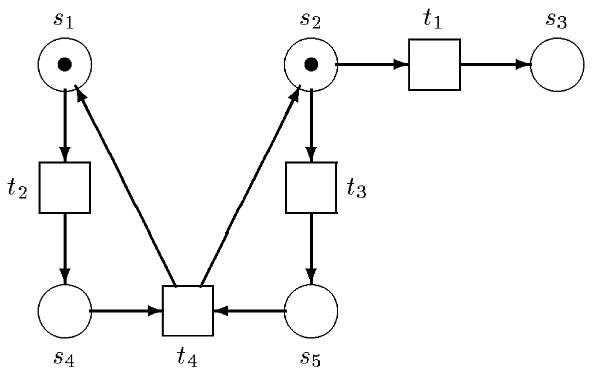

Fig. 5.1. A Petri net.

A *nonsequential process* (or just a process) of a Petri net (N, M_0) with $N = (S, T, F)$ is a pair (N', p), where N' is an acyclic net $N' = (B, E, F')$, and p is a labeling function $B \to S$ and $E \to T$. To avoid confusions, the elements of B are called *conditions*, instead of places, and the elements of E are called *events*, instead of transitions. The labeling p satisfies certain properties that allow to interpret N' as an 'unfolding' of the Petri net (N, M_0), corresponding to one of its possible concurrent runs. Most of these properties are not relevant here; the interested reader is referred to [2].

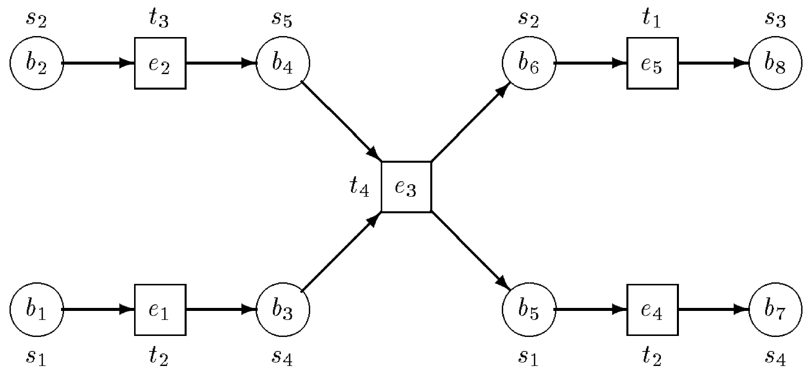

Fig. 5.2. A nonsequential process of the Petri net of Fig. 5.1.

Figure 5.2 shows a nonsequential process of the Petri net of Fig. 5.1. The names of the conditions and events are written inside the circles and boxes; the names of the places and transitions with which they are labeled are written close to them. If we put one token in the conditions b_1 and b_2, then we can simulate in the process a concurrent run of the Petri net; the occurrence of an event corresponds to the occurrence in the Petri net of the transition it is labeled with (for example, the occurrence of the event e_1 in the process corresponds to the occurrence of t_2 in the net); the set of conditions marked after the occurrence of the events corresponds via the labeling to a reachable marking of the Petri net. The process of Fig. 5.2 corresponds to a run in which the transitions t_2 and t_3 can occur independently, then transition t_4 can occur, and finally transitions t_1 and t_2 can occur independently.

Since the net N' of a process is acyclic, the reflexive and transitive closure of F' is a partial order, in the following denoted by $\leq$. This partial order will be of interest for the conditions of the process, considering only the poset $(B, \leq)$. This is a special poset, because N' has the property that the preset and postset of a condition contain at most one event. If we extend the process with an 'initial' event preceding all smallest conditions and with a 'terminal' event succeeding all largest conditions, then every condition has a unique event in its preset and its postset, so B can be seen as the set of *arcs* of a directed graph whose nodes are the events.

In this paper we restrict our attention to *safe* Petri nets, in which every reachable marking puts at most one token in a place (if this holds for the initial marking); then, a marking can be identified with the set of places with a token. The following proposition (see [2]) shows that the markings of the Petri net reached along the run correspond to the cuts of the poset $(B, \leq)$.

Proposition 5.1. *Let (N', p) be a nonsequential process of a safe Petri net (N, M_0), where $N' = (B, E, F')$. Then*

(a) two conditions carrying the same label are comparable with respect to $\leq$,

(b) if A is a cut of $(B, \leq)$, then $p(A)$ defines a reachable marking of (N, M_0).

In [4], a new model checking technique for safe Petri nets is presented, based on nonsequential process semantics. The paper shows that the verification of an arbitrary property expressible in a certain temporal logic can be reduced to the following problem: given a finite process of the safe Petri net (N, M_0), and two sets $\mathcal{G}$ and $\mathcal{H}$ of places of N, find an earliest cut such that its associated marking puts a token in each place of $\mathcal{G}$ and no token in the places of $\mathcal{H}$. (Actually, the problem discussed in [4] is that of finding a *latest* cut. It is easily reduced to the problem of finding an earliest cut.) For instance, if in the example of Fig. 5.2 we take $\mathcal{G} = \{s_1\}$ and $\mathcal{H} = \{s_2\}$, then the solution is the cut $\{b_1, b_4\}$, which corresponds to the reachable marking which puts tokens on s_1 and s_5.

As shown in [4], if this problem concerning processes can be solved in polynomial time, then some verification problems which required exponential time with the existing algorithms could be solved in polynomial time.

To reduce this problem to the asynchronous private meeting problem, construct the poset $(B, \leq)$. Choose as the set of labels $\mathcal{S}$ the set of places of the net N, and label a condition $b \in B$ with $p(b)$. Any two conditions carrying the same label are comparable by Proposition 5.1(a). The following proposition shows that the poset $(B, \leq)$ satisfies the condition (*) of Sect. 4, which allows to apply the polynomial algorithm given there. The proposition is also implicit in the above remarks on directed graphs.

Proposition 5.2. *Let $(B, \leq)$ be a poset obtained from a nonsequential process. Let $b, c, d \in B$ be conditions such that c and d are immediate successors of b. Then $\downarrow c = \downarrow d$.*

Proof. Let $N' = (B, E, F')$ be the net from which the poset $(B, \leq)$ is obtained. Every condition of B has a unique event in its poset. Let e be the unique event of the postset of b, which is the unique event in the preset of c and d. So both $\downarrow c$ and $\downarrow d$ are equal to the set of conditions smaller than e. $\square$

In [4], the problem of finding the earliest marking was solved by means of a reduction to Linear Programming. This reduction shows that the problem is solvable in polynomial time, although still with large exponents for the known polynomial algorithms for Linear Programming. The algorithm proposed here is more efficient, and does not require to construct a system of inequalities, because it works directly on the poset.

Acknowledgements

An early version of the algorithm was developed in discussions with Stefan Hougardy and Christoph Hundack, Bonn. Eike Best gave very helpful suggestions on exposition. A referee indicated the connection to N-free posets. Partial support was provided by the ESPRIT basic research working group CALIBAN and by the Volkswagen foundation.

References

[1] G. Behrendt: Maximal antichains in partially ordered sets. Ars Combinatoria 25 C (1988), 149–157.

[2] E. Best and C. Fernández: Nonsequential Processes – A Petri Net View. EATCS Monographs on Theoretical Computer Science Vol. 13, Springer Verlag (1988).

[3] K. M. Chandy and J. Misra: Parallel Program Design – A Foundation. Addison-Wesley (1988). (The meeting problem is considered on pages 14–18.)

[4] J. Esparza: Model Checking Using Net Unfoldings. TAPSOFT'93: Theory and Practice of Software Development, M. C. Gaudel and J. P. Jouannaud (eds.), Lecture Notes in Computer Science 668 (1993), 613–628.

[5] M. R. Garey and D. S. Johnson: Computers and Intractability – A Guide to the Theory of NP-Completeness. Freeman (1979).

[6] P. A. Grillet: Maximal chains and antichains. Fundamenta Mathematicae 65 (1969), 157–167.

[7] B. Leclerc and B. Monjardet: Ordres "C.A.C.". Fundamenta Mathematicae 79 (1973), 11–22.

[8] W. Reisig: Petri Nets – An Introduction. EATCS Monographs on Theoretical Computer Science Vol. 4, Springer Verlag (1985).