Eindhoven University of Technology
Department of Mathematics and Computing Science

Conservative Adaption of Workflow

by

M. Voorhoeve and W. van der Aalst

96/24

ISSN 0926-4515

Computing Science Reports 96/24
Eindhoven, December 1996

Conservative Adaptation of Workflow

Marc Voorhoeve
Wil van der Aalst
(email: {wsinmarc,wsinwa}@win.tue.nl)

Eindhoven University of Technology

Abstract

Business processes and, therefore, workflow models are often adapted. Adaptations can be caused either by procedural changes, affecting all future cases or on an ad-hoc basis for individual cases. Either way, cases with similar but different definitions will coexist in an organization. One can *conserve* the similarity of such cases by considering them as *extensions* or *reductions* of one and the same common ancestor. The extension/reduction concept is based on selectively blocking and skipping tasks, combined with *delay bisimilarity*. Views can be defined and used for monitoring the flow of cases on a high level. An example case study is given, using Petri Nets as the modeling framework.

Keywords: Workflow, Bisimulation, Petri Nets

1 Introduction

Efficiency has become the key aspect that organizations are competing in. In order to get the most work done with the least number of resources, computerized support is vital. Already most tasks have adequate support from applications, so the attention has shifted towards supporting the workflow (WF), i.e. the interplay of various tasks aimed at realizing an organization's goals.

WF management (WFM) systems can be used to define tasks with the resources involved therein and the applications supporting them (cf. [WFM94], [Kou95], [HaL91]). These tasks are combined to form *processes*. A process can be compared to a program, where the statements are tasks. Constructs from programming, like sequencing, choice and iteration all have their counterpart in WF process definitions.

A process can be monitored by observing its overall state. This state is composed of *cases* belonging to the process. Each case has its individual case state, which is determined by the possible tasks it can directly undergo and the case states that will result from executing these tasks. Each case must have a final "completed" state. The ability to monitor the overall state of a WF process and thus discover resource bottlenecks and slack is a major advantage of WFM systems. With this information, resources can be reallocated.

A process never terminates (in principle). New cases enter the process, undergo a series of tasks and leave completed. The "empty" overall state wherein no cases are being treated is unlikely to occur. This aspect makes it hard to maintain a WF process. An application can be recompiled while it is not being used. An operating system can be shut down, modified and rebooted. However, the ongoing work in the organization cannot be frozen for a process update. After a process update, cases that were initially started by an earlier version of the process will still be present and need to be handled properly in spite of the update. Similar problems are encountered e.g. in the telecommunications industry.

A radical solution is to define a new process while retaining the old one. New cases enter the new process. When the old process has reached the empty state it can be discarded. The drawback of such a solution is that the connection between the old and new process is lost, and that they must be monitored separately. Also, the old cases will not benefit from possible improvements in the new process. A similar problem occurs for cases derived from a standard process by minor ad-hoc modifications.

In this paper we indicate how similar processes can be treated and viewed as instances of one and the same common extension. Cases disregard "new" tasks, either by skipping (moving to the successor state immediately) or blocking (not executing them at all). The advantage of such an approach is that the overall state of a process and its modifications can be monitored together.

Monitoring the overall state of a complex process will still be cumbersome. An answer to this problem lies in the definition of *views*. By renaming and abstraction of tasks, the complexity of a process can be considerably reduced, concentrating on the essential features in its overall state. The underlying concept behind the extension, reduction and simplification of processes and their views is *delay bisimilarity* (cf. [Gla93], [Weij89]).

In this paper we restrict ourselves to the process part of WFM. The resource part and the scheduling problems involved there are at least of equal importance. However we believe that the resource part also benefits from a better understanding of the process part.

2 Process Model

For the modeling of a process we use the Petri net formalism [Rei85], more specifically free choice nets [DeE95]. Petri nets are a good formalism to base WFM tools upon (cf. [ElN93], [AHH94] and [Aal96]). On the one hand, Petri nets are graphical and easy to use. On the other hand, Petri nets do have a formal semantics and analysis methods are abundant. In the remainder of this paper, we assume a basic knowledge of Petri nets.

2.1 Notations and Conventions

We assume the usual facts and notations about sets and functions. The *product* $A \times B$ of two sets A, B is the set of ordered pairs $\{(a, b) \mid a \in A \wedge b \in B\}$. A *relation* between A and B is a subset of $A \times B$. This can be generalized to three or more sets, giving sets of ordered triples, ternary relations, and so on.

We introduce *bags* to formalize the states of Petri nets. The set $\mathbb{N}^A$ of *bags* over a given set A is the set of functions with domain A and range $\mathbb{N}$. If A is finite, say $A = \{a, b, c\}$ the elements of $\mathbb{N}^A$ are denoted by superscripting the domain element with the value, e.g. $a^1b^2c^0$. If this does not lead to confusion, 1 superscripts and domain elements with 0 superscripts may be omitted, so the above bag can also be denoted as ab^2. To each subset S of A corresponds a bag $\mathrm{bag}(S)$ in $\mathbb{N}^A$

satisfying $\text{bag}(S)(a) = 1$ if $a \in S$ and $\text{bag}(S)(a) = 0$ if $a \notin S$. Equality on bags is defined as follows: two bags are equal iff they have the same domain and the bag values are the same on that domain. When introducing Petri nets, a net's state will correspond to a bag which has a domain consisting of the *places* of the net.

Let $B \in \mathbb{N}^A$ and $S \subseteq A$. Then $B + S$ is the bag B' defined by $B'(a) = B(a)$ if $a \notin S$ and $B'(a) = B(a) + 1$ if $a \in S$. We write $S \leq B$ if there exists a $B' \in \mathbb{N}^A$ such that $B' + S = B$; if so, this (unique) B' is denoted by $B - S$.

2.2 Petri Nets

A *Petri net* N is a triple (S, T, F), where S and T are finite disjoint sets (of *places* and *transitions* respectively) and $F \subseteq (S \times T) \cup (T \times S)$ the *flow relation*. The elements of $S \cup T$ are called *nodes* of N. Given a node x of N, the *preset* ${}^\bullet x$ of x is the set $\{y \mid (y, x) \in F\}$ and the *postset* $x^\bullet$ of x is the set $\{y \mid (x, y) \in F\}$. A *path* between nodes x, y of N is a sequence of nodes $x = x_1, \ldots, x_n = y$ of N such that $x_i \in {}^\bullet x_{i+1}$ for all $1 \leq i < n$. A path between x and x may consist of the singleton sequence. The *length* of such a path is the number of elements in the sequence minus one.

The Petri net N is a *free choice* net (FC net) iff the presets of transitions are either disjoint or coincide. So for any $t, u \in T$, ${}^\bullet t = {}^\bullet u$ or ${}^\bullet t \cap {}^\bullet u = \emptyset$. An S-system is a net for which every transition has one input and one output place. S-systems are FC nets.

A *marking* of N is a bag $M \in \mathbb{N}^S$. A transition $t \in T$ is *enabled* by M iff ${}^\bullet t \leq M$. The enabling relation between markings and transitions is denoted by $M\,[t\rangle$. A transition t enabled by M can *fire*, leading to a *successor* marking M' given by $M' = (M - {}^\bullet t) + t^\bullet$. This ternary relation between markings and transitions is denoted by $M\,[t\rangle\,M'$. A marking M' is *reachable* from M (notation $M\,[*\rangle\,M'$) iff either $M' = M$ or there exist $t_1, \ldots, t_n \in T$ and $M_1, \ldots, M_n \in \mathbb{N}^S$ such that $M\,[t_1\rangle\,M_1\,[t_2\rangle\,\ldots\,[t_n\rangle\,M_n$ and $M_n = M'$.

M is called *live* iff for every M' with $M\,[*\rangle\,M'$ and every $t \in T$ there exists a marking M'' with $M'\,[*\rangle\,M''\,[t\rangle$. M is called *bounded* iff the set $\{M' \mid M\,[*\rangle\,M'\}$ is finite. M is called a *home marking* iff for every M' with $M\,[*\rangle\,M'$ one has $M'\,[*\rangle\,M$. There exist efficient algorithms for deciding these properties for FC nets (see [DeE95]).

We now introduce *labeled Petri nets* presupposing a set A of *labels*. Now a *labeled* Petri net is a pair (N, L), where $N = (S, T, F)$ is a Petri net and L is a function with domain T and range $A \cup \{\tau\}$, where $\tau \notin A$ is a special "silent" label. Let M, M' be states of a labeled net (N, L). If $M\,[t\rangle\,M'$ in N and $L(t) = a$, we write $M \xrightarrow{a} M'$. The ternary $\_ \xrightarrow{\_} \_$ relation between states and tasks is called the successor relation.

Nets are drawn as bipartite directed graphs, with box-like transitions and circular places as nodes. The flow relation is depicted by arrows. Labeled nets are depicted by inscribing the label inside the transitions. The label τ may be omitted.

2.3 Workflow Processes

WF processes can be easily modeled by Petri nets. Tasks are assigned to transitions and case states correspond to markings. The execution of a task, leading to a successor state, corresponds to the firing of a transition. An example WF process is given in Figure 1.

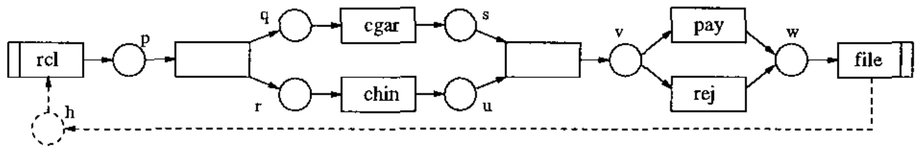


Figure 1: Claim handling net

The task labels are inscribed in the transitions. The transitions without label can fire without any task being executed. The remaining labels have no formal meaning and are added for clarity only. We describe the process associated to this net. Initially, a case is in the state h^1. The place h is drawn dashed for reasons indicated below, A new claim for car damage insurance can be started by the claim reception (*rcl*) task. After receiving the claim, the process forks and two tasks can be executed independently, contacting the garage (*cgar*) and checking the insurance policy (*chin*). After completion of both tasks, a join action takes place, after which a decision can be made either to pay the damage (*pay*) or reject the claim (*rej*). Either way, the claim is filed (*file*) and the case is closed.

Every WF process must have a "home" place h, corresponding to an initial or final case state. Instead of drawing it and its connected edges (dashed in the figure), the transitions with h in their preset or postset are barred at the left-hand, respectively right-hand side. These are the initial tasks (receiving a call, letter or order form) and final tasks (e.g. filing) respectively.

It is not the case that cases, once finished, reenter the net. In fact, the "home" place construction has been given for technical reasons. The essential feature to consider here is that there are initial and final tasks.

For the sake of clarity, large nets can be divided into subnets. Subnets are depicted with rounded corners and their labels have a different font (subnet labels do not refer to tasks). Figure 2 indicates a possible division of our earlier net into subnets.

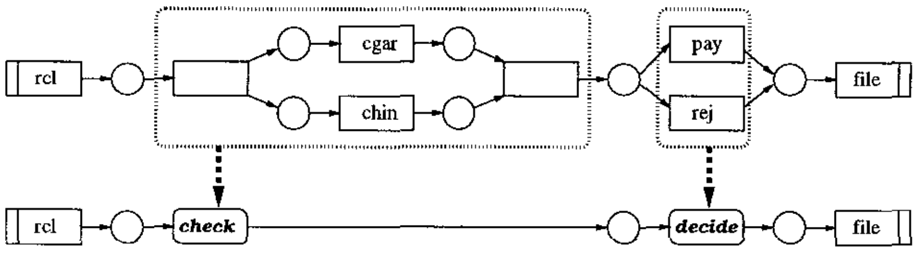


Figure 2: Claim handling net subdivision

We now define WF processes formally as a subclass of labeled Petri nets. The set A of labels represents the set of tasks to be executed. A WF process P is a labeled Petri net (N, L), where

$N = (S, T, F)$ is an FC net. The transitions t with $L(t) = \tau$ are transitions without task. The net N must have a special "home" place $h \in S$, and the marking h^1 must be a live and bounded home marking. The net component N from a WF process (N, L) is called a WF net. When no confusion is possible, "net" and "process" are used as synonyms. The markings reachable from h^1 are called *states* of P. By the conditions imposed on N, the states of P are *safe*, i.e. each place can contain at most one token. So the states of P can be described by sets instead of bags. Let M, M' be states of P. If $M = M'$ or $M\,[t_1\rangle\, M_1 \ldots [t_n\rangle\, M'$ for certain states $M_1, \ldots, M_{n-1}$ different from h^1, we write $M \xrightarrow{*} M'$. A transition t with $L(t) = \tau$ is called *silent*. These transitions should be used for synchronization purposes only. If $M \xrightarrow{a} M'$, the task a may be executed in the state M and its execution leads to the state M'. If $M \xrightarrow{\tau} M'$, the state M' may evolve autonomously (i.e. without any task being executed) from M.

The relation $M \xrightarrow{*} M'$ signifies that M can evolve into M' without passing through the state h^1. It is the reachability relation in a net that is derived from N by splitting h into an initial and terminal place.

A WF process can be modeled directly or indirectly by transforming a CCS [Mil89] or ACP [BaV95] term into a net. The transformation rules are indicated e.g. in [GlV87]. The ACP term corresponding to the net in Figure 1 is $rcl.\tau.(chin \| cgar).\tau.(pay + rej).file$. This approach may be preferred by modelers having a background in (concurrent) programming.

One must bear in mind that the above way of modeling WF processes abstracts from several important aspects. The first such aspect is the existence of *case variables*. A case usually possesses *variables* that are both set and accessed by the tasks being executed. These variables may influence the WF process itself, disabling certain tasks within the process being modeled. A second aspect is the attribution of tasks to resources. This aspect could prevent execution of certain tasks in a certain state, because they lack the necessary resources. Taking either aspect into account can easily lead to a net that lacks the FC property. However, it is our belief that one should refrain from modeling the workflow at a too detailed level initially. So these aspects are not taken into account, which makes it acceptable to require the WF nets to be FC nets.

3 Process Equivalence and Reduction Relations

Different labeled nets may represent one and the same flow of work. For example the nets P, Q and R in Figure 3 all three consist of an initial task a followed by a final task b. In order to make this observation tangible, an *equivalence relation* is established between WF processes.

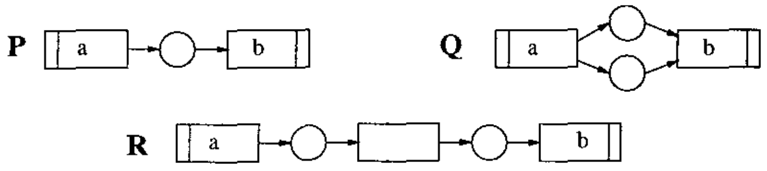


Figure 3: Different nets representing the same process

There exist many equivalence relations that can be considered for this purpose (c.f. [Gla93]).

Here, we have chosen *delay bisimilarity*, which has the advantage of a rather simple definition, whereas some essential distinctions between processes (like the difference between internal and external choice described below) are preserved.

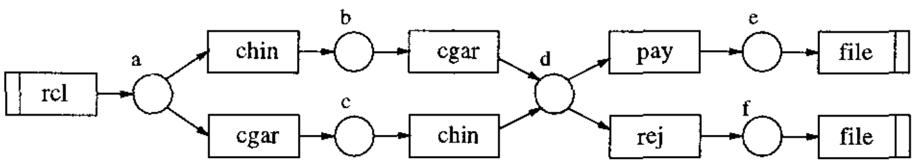


Figure 4: Equivalent claim handling net

The net in Figure 4 is delay bisimilar to our claim handling net. This is because to every case state of the one net corresponds a case state of the other allowing the same tasks, possibly after firing some silent transitions. Executing corresponding tasks in corresponding states of either net again leads to corresponding states. The states and the successor relation for the nets in Fig 1 and Fig 4 respectively are given in Figure 5. The correspondence between states is given by dotted lines. Note that the *h* states of both processes do correspond.

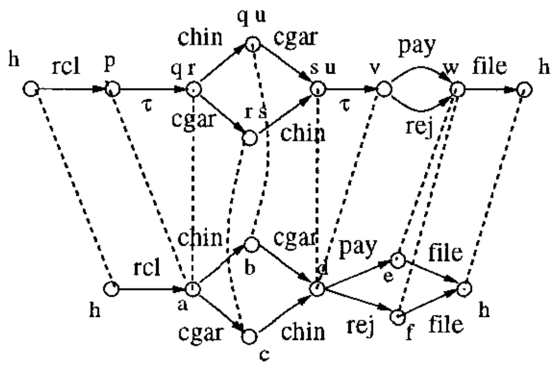


Figure 5: A delay bisimulation between processes

We call two WF processes *equivalent* iff such a relation can be constructed between their states. The relation is called a *delay bisimulation* (cf. [Gla93], [Weij89]). In the appendix the formal definition can be found. When considering the processes in Figure 1 and 4 equivalent, it is tacitly assumed that only one task can be executed at a time for a certain case.

An aspect that deserves attention in process equivalence is the treatment of silent transitions.

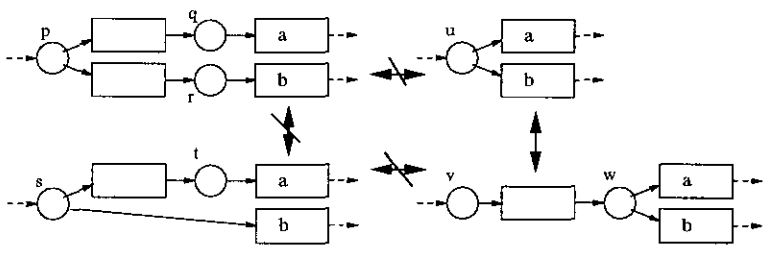


Figure 6: Choices with silent transitions

In Figure 6, four alternatives are given that involve a choice between tasks a and b. In all cases, the dashed arrows are connected in one and the same way to some environment net. Of these four alternatives, only the two in the right half are equivalent. The net in the top left quarter can evolve autonomously from the state p (where the a and b tasks are both still possible) to either the state q or r (where only a or only b is possible). A *nondeterministic* or *internal* choice has been made before any a or b task is executed. In the net in the bottom left quarter, the state s (allowing both tasks) can evolve autonomously into t (allowing only a).

In the nets in the right half, the choice between a and b is retained until either of them is started. This can be regarded as a *deterministic* or *external* choice. States that are related by a delay bisimulation essentially possess the same options for continuation.

WF processes are often adapted e.g. by adding extra tasks. We shall define a relation between processes that embodies a structured way of adaptation. Informally, a WF process can be *reduced* by retaining its net and adding variables and instructions to skip (or hide) certain transitions and block (forbid) others. Here "skipping a transition" means firing it without executing the associated task (thus autonomously moving to its successor state), while "blocking a transition" means not firing it (at all or unless a certain event has taken place). Typically, blocking will occur in a choice context and skipping in a sequential context.

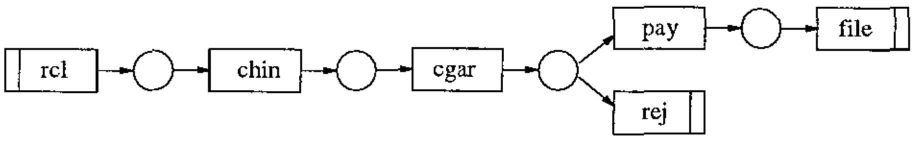


Figure 7: Reduction of claim handling

In Figure 7 a reduction of the WF process in Figure 1 is depicted. In the reduced net, the *cgar* task is blocked until *chin* has been executed. Furthermore, the *file* task is skipped in case of a rejection. This example indicates that the skipping and blocking of tasks may be conditional; it may depend on the *history* of a case whether certain tasks are skipped and/or blocked. *Extension* is the inverse of reduction; P is an extension of Q iff Q is a reduction of P.

We formally define the concept of reduction. Let P_1, P_2 be WF processes. Then P_1 is a reduction of P_2 iff a relation R can be constructed between the states of P_1 and P_2 with the following property. Let M_1, M_1' be states of P_1 and M_2 a state of P_2 such that $(M_1, M_2) \in R$. If $M_1 \xrightarrow{\tau} M_1'$ there must exist a state M_2' of P_2 such that $M_2 \xrightarrow{*} M_2'$ and $(M_1', M_2') \in R$. If $M_1 \xrightarrow{a} M_1'$, there must exist states M_2', M_2'' of P_2 such that $M_2 \xrightarrow{*} M_2'' \xrightarrow{a} M_2'$.

Figure 8 displays a reduction relation between the states of the processes in Figure 7 and 4. Reduction of processes is *transitive*: if A is a reduction of B and B a reduction of C then A is a reduction of C. The third relation can be obtained by composing the first two.

A delay bisimulation is a reduction relation in both directions. So if A and B are equivalent WF processes, then A is a reduction of B and vice versa. The converse is not true: WF processes can be reductions of one another without being equivalent.

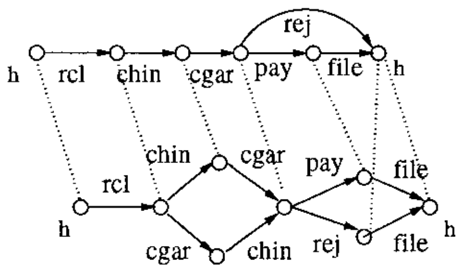


Figure 8: A reduction relation

4 Equivalence, Reduction and Extension Rules

Algorithms for establishing equivalence, reduction or extension of given WF processes are important for tools that support the definition and adaptation of workflow processes. Here we define rules for transforming a process into an equivalent, extended or reduced one. In this way a process designer equipped with a graphical editor can modify his processes interactively by selecting parts of his model and invoking the rules he has in mind.

We start by informally defining rules for process equivalence. A formal definition can be found in the appendix. Illustrative examples can be found in Figure 9. Each rule can be applied in two directions, forward and backward. The backward direction (from right to left) often simplifies a net. A description is given for one direction only. Some rules are applicable only to *equivalent places*. An equivalence relation called *place bisimulation* [AuS92] exists for the places of a Petri net. This equivalence can be efficiently decided for any Petri net. A formal definition of place bisimulation can be found in the appendix. Informally, tokens in equivalent places can be interchanged without affecting the behavior of a net.

interleave/factorize: By interleaving, a WF net can be transformed into an S-system. Every state corresponds to a place in the interleaved net and for each possible state transformation a transition is added. We call the inverse operation factorization.

unfold/fold: Equivalent places can be folded into a single place. The pre- and postset of the folded place are the union of the pre-, respectively postsets of the folded places. The unfold transformation is allowed iff the resulting unfolded places are equivalent (which is not always the case).

addrplace/delrplace: A place p is called *redundant* iff for any (reachable) state M and transition t with $M + \{p\} [t\rangle$ it holds that $M [t\rangle$. Redundant places can be deleted at will and added if the free choice property is conserved. This rule is a consequence of the interleave rule that can be verified locally.

addtau/deltau: A silent transition can be added between sets of places that are equivalent in the sense that a 1-1 correspondence can be given between the places in the preset and postset of the silent transition, such that corresponding places are equivalent or equal.

addrtrans/delrtrans: If places p, q are connected via zero or more silent transitions followed by some transition t, a duplicate of t can be added connecting p to q.

splittau/mertau: A place p can be split into places q, r, adding a silent transition between q and r. The incoming edges of p can be distributed between q and r and all outgoing edges of p must

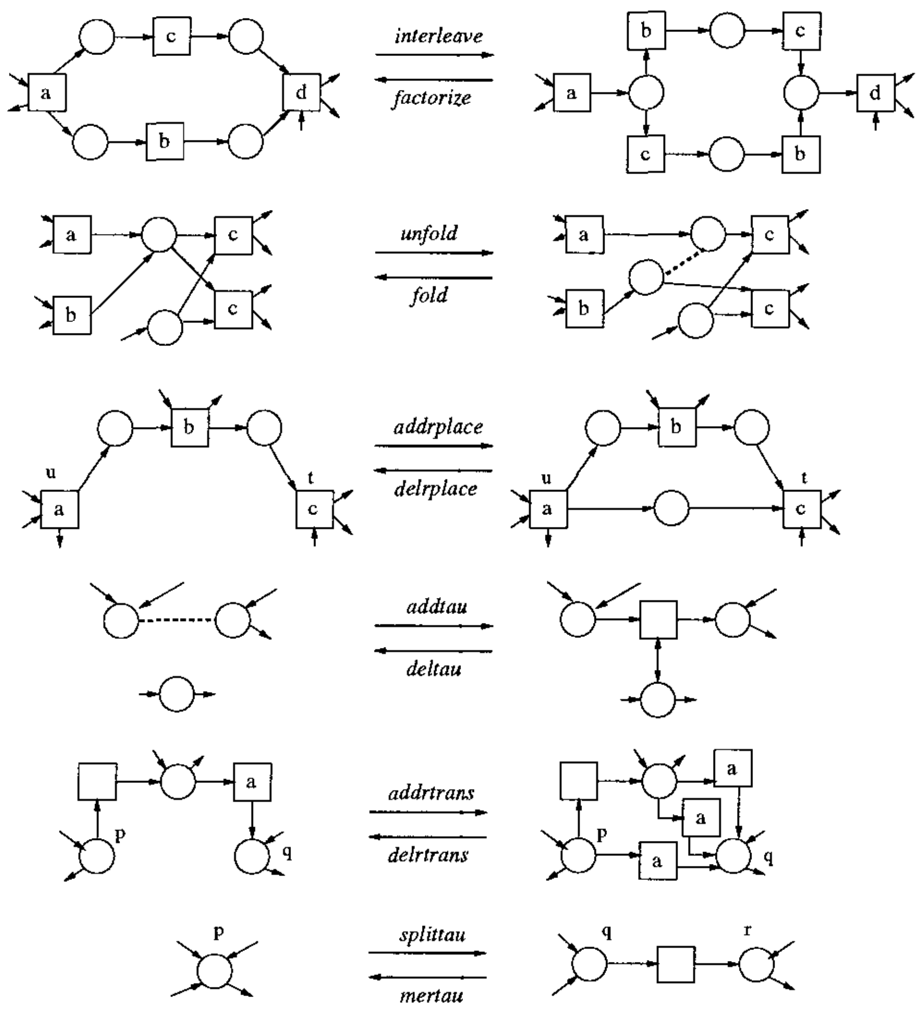


Figure 9: Equivalence rules

be attributed to r.

Next, we define extension and reduction rules, shown in Figure 10. In contrast to the equivalence rules above, the processes on the left-hand and right-hand side will be different. Note the phenomenon in the delplace/addplace rule that *deleting* a place results in *extending* the net. This follows from the fact that each place represents a *condition* that may impede the occurrence of an action.

unskip/skip: A WF process can be extended by attributing a task to a silent transition and reduced by making a task transition silent.

addsub/block: A WF process can be extended by inserting a subnet between two places and reduced by removing such a subnet.

delplace/addplace: A WF process can be extended by deleting a place and reduced by adding one. This is a consequence of the addsub and interleave rules.

The addsub/block and delplace/addplace rules are allowed only if the resulting process remains a

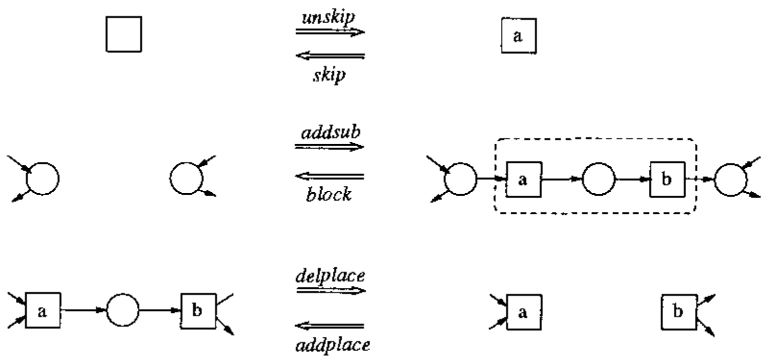
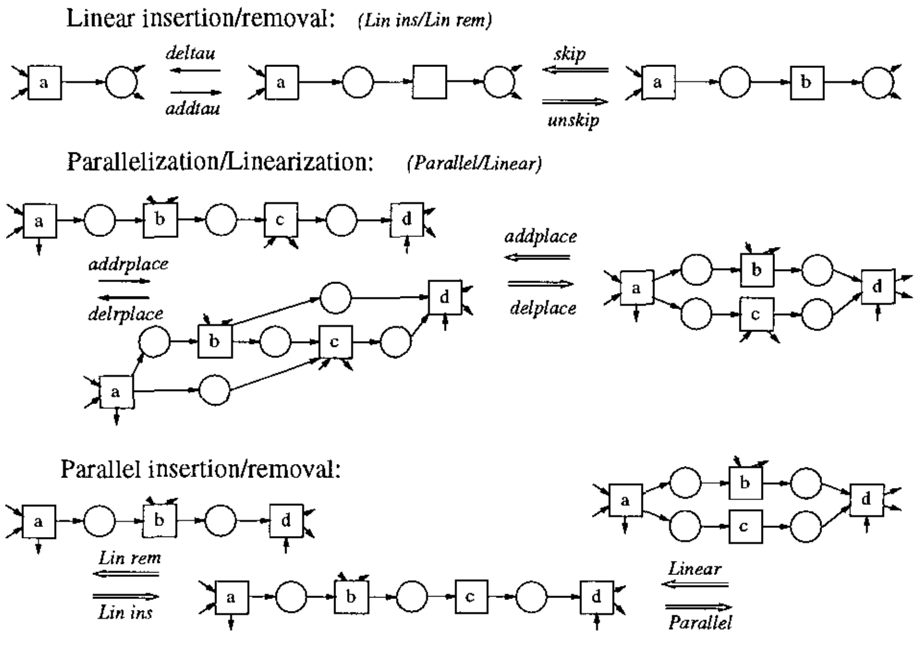


Figure 10: Extension (⇒) and reduction (⇐) rules

WF process. The delplace/addplace rules can be considered to be generalizations of the "redundant" place equivalence rules.



Linear insertion/removal: (Lin ins/Lin rem)
deltau
addtau
skip
unskip
Parallelization/Linearization: (Parallel/Linear)
addrplace
delrplace
addplace
delplace
Parallel insertion/removal:
Lin rem
Lin ins
Linear
Parallel

Figure 11: Extension derivations

Figure 11 shows some extension (and reduction) rules that can be derived from the above rules. They illustrate that the equivalence, extension and reduction rules allow for adaptations encountered in practice. The linear insertion rule indicates that a process can be extended by inserting a new task between two old ones. The parallelization rule shows how a sequential ordering of tasks can be converted into a parallel one. These two rules can be combined to yield the parallel insertion rule. The claim handling net in Figure 1 can be shown to be an extension of the one in Figure 7 by the addtau, parallelization, linear insertion and fold rules.

We prove in the appendix that WF processes are equivalent iff they can be transformed into one another by means of the process equivalence rules. They are an extension/reduction of one another iff they can be transformed into one another by means of the extension/reduction and equivalence rules.

5 Process Views

We will now treat the visualization of the combined state of a process, i.e. all cases currently being treated. In doing so, one must maintain a balance between accuracy and amount of information offered. A too accurate representation of the state (e.g. an overview of every case with its variables and history) cannot be visualized. Instead, such information should be stored in a database and queried.

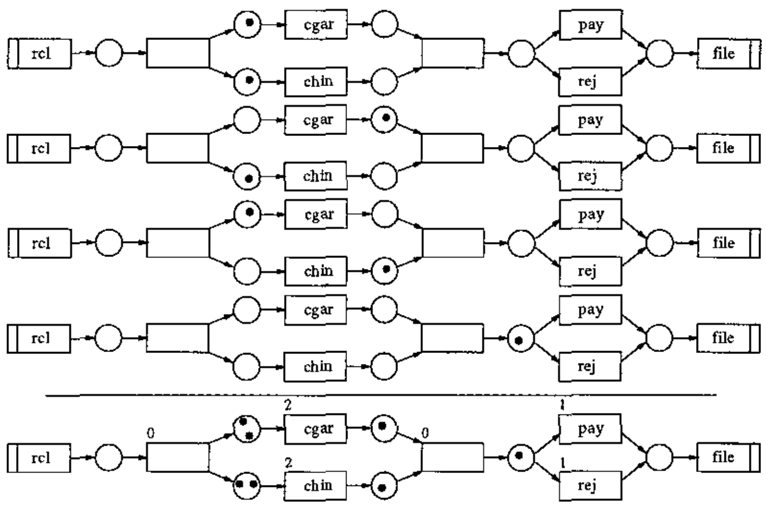


Figure 12: Combined state

As indicated earlier, a Petri Net based WF model allows one to visualize the state. This is done by depicting the state (marking) of each case as a distribution of *tokens* in the places and superposing them, like in Figure 12. Note that some information is lost in the process: it looks as if the join transition is enabled in the superposition, whereas the examination of the individual cases shows this not to be the case. By adding extra information (e.g. the number of times each transition is enabled), some of this lost information can be retrieved without overloading the picture.

We believe that a picture of the WF net where the number of tokens per place and the "enable count" per transition is added gives an adequate overview of the state, allowing an early discovery of bottlenecks and slack.

An advantage for defining extension and reduction relations for WF processes is that the cases of all processes that are reductions of a single extension can still be viewed in combination. In Figure 13 this is depicted for our running example. Note that information is lost again, since cases can be depicted with options for continuation they lack.

Still, monitoring a complex WF process can become a burden, especially when there exist many variants with ad-hoc tasks added to them. To reduce the complexity of such nets one can define *views*. A view corresponds to a function that renames certain tasks and abstracts from other tasks. A manager monitoring the car claim process might be interested in two groups of resources: the "check" and "decide" workers. By renaming the "check" tasks to *chk* and "decide" tasks to *dec* and removing the label of the other tasks, the top net in Figure 14 is constructed. This net can be

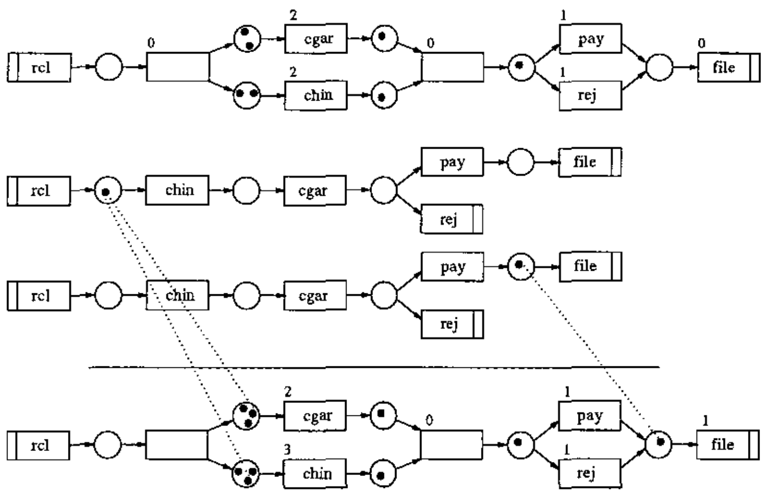


Figure 13: Combined state of process with reduction

simplified by the interleave, delrtrans and mertau rules to the bottom net, showing in one glance that the current workload consists of six "check" tasks (four of which are enabled) and five "decide" tasks (one of which is enabled).

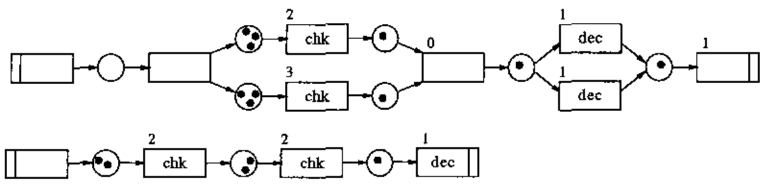


Figure 14: View of the claim process state

6 Example

The "WWWizz" agency offers support to companies for presenting themselves on the Internet. The agency gives courses, develops company-specific style guidelines and develops and maintains web sites. Figure 15 gives the WF process (with subnets) for clients that eventually want to maintain their own site.

Figure 15 contains four subnets. The acquisition subnet (*acq*) creates cases (*client*) that enter the construction subnet (*cons*). After the construction subnet a web site has been started and guidelines have been prepared. The site can be iteratively maintained for a while (*maint*), during which the guidelines can be updated (*gupd*). The client can decide to initiate a guidelines release subnet (*grel*), resulting in a company-specific guidelines manual. After releasing the guidelines, the site itself can be released (*srel*), meaning that the maintenance responsibility shifts from WWWizz to the client. The maintenance site then becomes a support site. The support action (*supp*) is possible in that state until the termination of the support phase (*file*). Of course, the normal flow in Figure 15 can be aborted in earlier stages, but this has not been included in the figure.

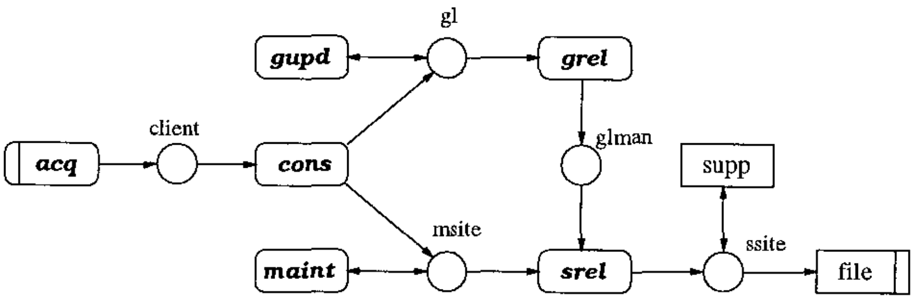


Figure 15: WWWizz net

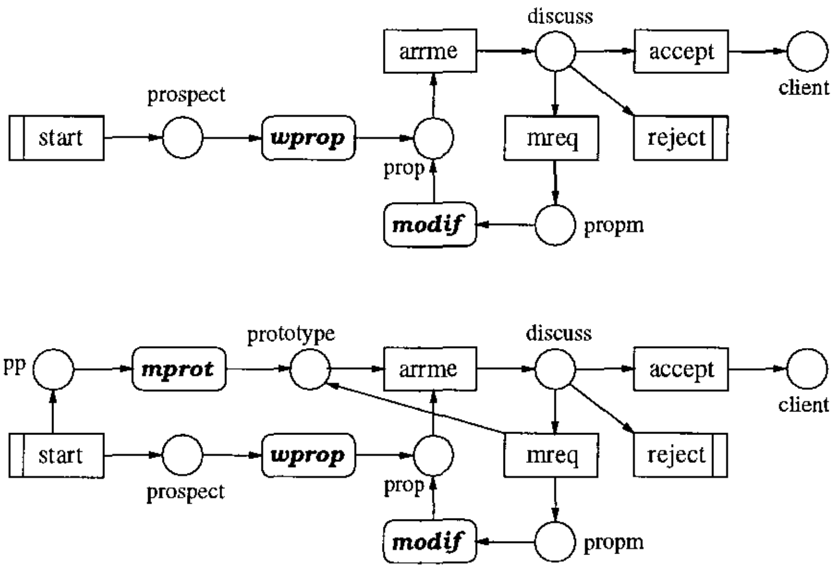


Figure 16: WWWizz acquisition subnets

Figure 16 gives the acquisition subnet in a standard form (at the top) and an extended form (bottom). The start activity (*start*) generates prospects that enter a proposal writing subnet (*wprop*). A meeting is then arranged (*arrme*), where the proposal is discussed with the prospective client. The outcome of this discussion can be either acceptance (*accept*), rejection (*reject*) or a request to modify the proposal (*mreq*). This modification then takes place in a subnet (*modif*).

In the extended net, the proposal is accompanied by a prototype, consisting of a few web pages containing material specific to the prospect, which is prepared in the *mprot* subnet. The WWWizz marketing people have found out that this approach gives a higher success rate, justifying the extra costs. Prototypes are not modified. The new net is an extension by parallel insertion.

In Figure 17, the proposal subnet is depicted at the top. A session with the user is held to determine his requirements, which are then documented (*ureq*). From this document, the infrastructure requirements (hardware and software) are extracted (*isreq*). In parallel, the available infrastructure at the client's site is assessed (*avis*). From these data, the costs of creating and maintaining the required web site are computed (*ccost*).

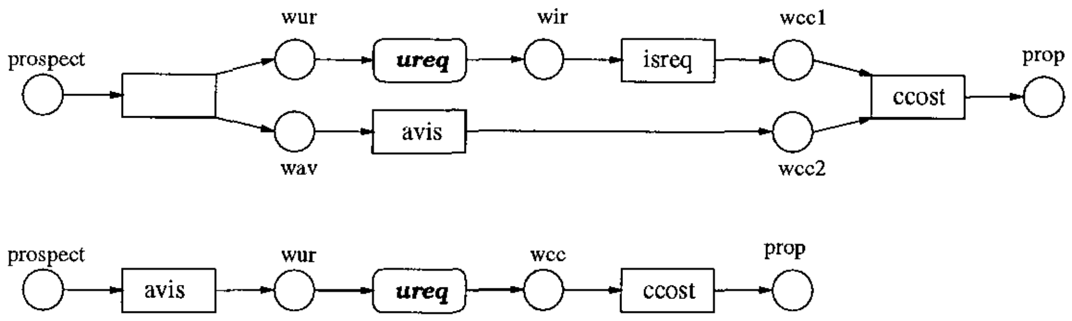


Figure 17: WWWizz proposal subnets

The bottom net is a modification that has been made for a prospect that has no budget for new infrastructure. So first his available infrastructure is determined and his requirements then have to fit to what is available. This is a reduction by linearization and linear removal.

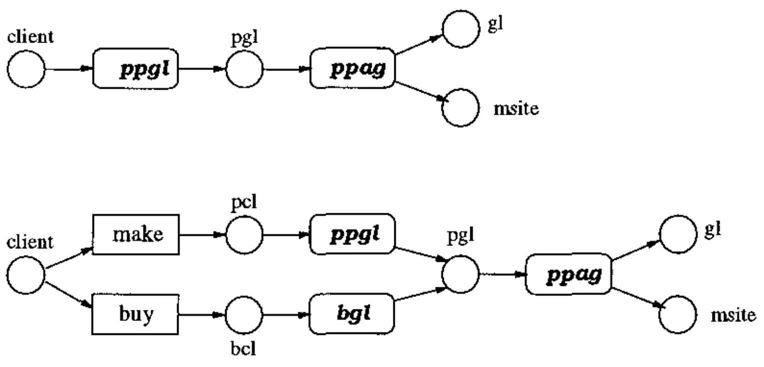


Figure 18: WWWizz construction subnets

In Figure 18, the construction subnet is depicted. At the top is the old situation. First, provisional guidelines are prepared (*ppgl*), after which web pages are produced and the guidelines updated if necessary (*ppag*). At the bottom, an extension of this net is depicted. The preparation of guidelines is a substantial amount of independent work. The WWWizz employees that are capable of doing this work have become a bottleneck resource. So the possibility has been added to outsource this activity to free-lancers or advertising agencies. First, for every client a make-or-buy decision is introduced. The "make" clients then follow the old path. For the "buy" clients, a new subnet (*bgl*) is entered. It can be verified that the bottom net is an extension of the top one by applying the addsub and linear insertion rules.

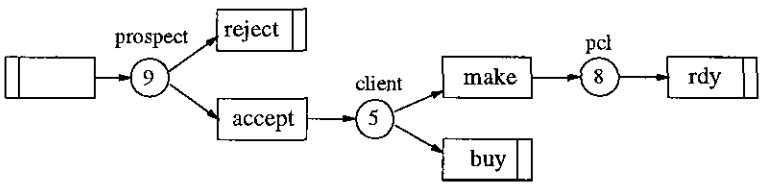


Figure 19: WWWizz provisional guidelines preparation view

In Figure 19, an overall state view is depicted for the manager of the provisional guidelines employees. He is largely responsible for the make-or-buy decision. He must be able to monitor the work that his people are busy with and to estimate the amount of work needed in the near future. His view abstracts from all tasks but the ones depicted in the figure. The reduction is then obtained by invoking the fold, delrtrans and mertau rules.

In the state depicted, his people are busy with eight guidelines. Five clients await the make-or-buy decision and nine prospects are negotiated with. Of course, it would be better to add more detail to the *ppgl* activity, to see how near these eight guidelines are to completion. Also the 'enable count' of each transition can be added.

7 Conclusions

The present paper gives several notions regarding workflow processes. Process equivalence can be used to modify processes (making them simpler or clearer) while retaining the behavior that is being modeled. Process extension (reduction) embodies the idea of adding (deleting) options. It is possible to define the "least common extension" and "greatest common reduction" of a set of processes derived from a single generic process.

To further elaborate the value of process extension and reduction in practice, we consider adaptive, ad-hoc changes and structural, permanent ones. For the adaptive changes, one may imagine the situation where an organization has several template processes that serve as a starting point for the derivation of many actual processes. Every time a new case enters the organization, a template process is selected and modified to fit the requirements of that specific case.

The notions of extension and reduction allow the organization to specify constraints upon the ways that processes are modified. The organization may e.g. fix a greatest common reduction that just possesses the necessary minimum actions, or, a least common extension that limits the actions that can be added. It can be checked that such constraints are met.

For the structural part, we stress the fact that a case of process P in any state "fits" into some state of an extension Q of P. This allows *structural* changes to be made with impunity to a process while it is running and lots of cases being treated. Finally, the paper shows how processes can be monitored and defines *views* of the current work in progress. These views can be used to monitor cases being executed according to many variants of a common WF process.

There is still a lot of work to do in order to allow the above concepts to have any impact in practice. One of the first efforts is incorporating the ideas sketched here into a workflow editing tool.

We conclude with a few technical remarks. The underlying equivalence notion is delay bisimilarity. One could also consider to adopt *branching* or *weak* bisimilarity (cf. [Gla93]), resulting in a slightly different set of "tau" rules. Weak bisimilarity can be obtained by adding an extra case to the "rtrans" rules. However branching bisimilarity requires a different approach; the rtrans rules must be weakened and the fold rules strengthened, requiring a new kind of place bisimulation.

Acknowledgement

The authors wish to thank Twan Basten for many fruitful discussions, during which the idea of

structured modification involving blocking and skipping of actions was born (cf. [BaA96]).

References

Aal96 W.M.P. van der Aalst: *Petri-net-based Workflow Management Software*, in: A. Sheth (ed.) *Proc. NFS Workshop on WF and Proc. Aut. Inf. Sys.*, pp. 114-118, 1996.

AHH94 W.M.P. van der Aalst, K.M. van Hee, G.J. Houben: *Modelling Workflow Management Systems with High-level Petri Nets*, in: G. De Michelis, C. Ellis, G. Memmi: *Proc. 2nd Workshop Comp.-Supp. Coop. Work*, pp. 31-50, 1994.

AuS92 C. Autant, Ph. Schnoebelen: *Place bisimulations in Petri nets*, in: K. Jensen (ed.) *Proc. Appl. Theor. Petri Nets*, pp. 45-61, LNCS 616, Springer 1992.

BaA96 T. Basten, W.M.P. van der Aalst: *A Process-Algebraic Approach to Life-Cycle Inheritance, Inheritance = Encapsulation + Abstraction*, Computing Science Report 96-05, Eindhoven Univ. Tech. 1996.

BaV95 J.C.M. Baeten, C. Verhoef: *Concrete Process Algebra*, in: Abramsky, Gabbay, Maibaum (eds.) *Handb. Logic Comp. Sci.*, Vol. 4, pp. 149-268, Oxford Univ. Press 1995.

DeE95 J. Desel, J. Esparza: *Free Choice Petri Nets*, Cambridge Univ. Press 1995.

ElN93 C.A. Ellis and G.J. Nutt: *Modelling and Enactment of Workflow Systems*, in: M. Ajmone Marsan (ed.) *Proc. Appl. Theor. Petri Nets*, pp. 1-16, LNCS 691, Springer 1993.

GlV87 R.J. van Glabbeek, F.W. Vaandrager: *Petri Net Models for Algebraic Theories of Concurrency*, in: de Bakker et al. (eds.) *Proc. PARLE, Vol. II: Parallel Languages*, pp. 224-242, LNCS 259, Springer 1987.

Gla93 R.J. van Glabbeek: *The Linear time - Branching time Spectrum II (extended abstract)* in: E. Best (ed.): *Proc. CONCUR '93*, pp. 66-81, LNCS 715, Springer 1993.

HaL91 K. Hayes and K. Lavery: *Workflow Management Software: the Business Opportunity*, Ovum 1991.

Kou95 T.M. Koulopoulos: *The Workflow Imperative*, Van Nostrand 1995.

Mil89 R. Milner: *Communication and Concurrency*, Prentice-Hall 1989.

Rei85 W. Reisig: *Petri Nets*, Springer 1985.

Weij89 W.P. Weijland: *Synchrony and Asynchrony in Process Algebra*, Ph. D. Thesis, Univ. Amsterdam 1989.

WFM94 Workflow Management Coalition: *Workflow Reference Model*, 1994.

Appendix

In this appendix, formal definitions and proofs are added. We suppose the basic notions about sets, functions and relations to be known with the standard notations for them. Throughout this section, we suppose that $P_1 = (N_1, L_1)$ and $P_2 = (N_2, L_2)$ are WF processes, where $N_1 = (S_1, T_1, F_1)$ is a WF net with home place h_1. and $N_2 = (S_2, T_2, F_2)$, with home place h_2. With $a, b, \ldots$ we denote tasks and α is either a task or τ.

The simplest equivalence notion between WF processes is isomorphism. P_1 and P_2 are isomorphic iff there exists a bijection f between their places and transitions such that $h_2 = f(h_1)$ and $F_2 = \{(f(x), f(y)) \mid (x, y) \in F_1\}$.

We start with the definition of simulation and bisimulation relations. First, an auxiliary notation is defined. Let P be a WF process and let M, M' be states of P. We write $M \overset{\alpha}{\Longrightarrow} M'$ iff either $\alpha = \tau \land M = M'$ or $M \xrightarrow{\alpha} M'$ or there exists a state M'' such that $M \overset{\tau}{\Longrightarrow} M'' \xrightarrow{\alpha} M'$.

Definition 1 *A* strong simulation *between P_1 and P_2 is a relation R between the states of P_1 and those of P_2 such that if $(M_1, M_2) \in R$ and $M_1 \xrightarrow{\alpha} M_1'$, there exists an M_2' such that $M_2 \xrightarrow{\alpha} M_2'$.*
A delay simulation *between P_1 and P_2 is a relation R between the states of P_1 and those of P_2 such that if $(M_1, M_2) \in R$ and $M_1 \overset{\alpha}{\Longrightarrow} M_1'$, there exists an M_2' such that $M_2 \overset{\alpha}{\Longrightarrow} M_2'$.*
A (delay/strong) bisimulation is a (delay/strong) simulation R such that its inverse R^{-1} is also a (delay)/strong simulation.
P_1 and P_2 are equivalent *iff there exists a delay bisimulation between P_1 and P_2 such that their home states are related.*

The relation composition of two (delay/strong) simulations is a (delay/strong) simulation, so equivalence is indeed an equivalence relation. Note that P_1 and P_2 may be one and the same process. We now formally define place bisimulation.

Definition 2 *A place bisimulation between the places of a WF process P is a symmetric and reflexive relation R between the places of P such that:*
If t is a transition of P with $^\bullet t = \{s_1, \ldots, s_n\}$, $t^\bullet = \{s_1', \ldots, s_m'\}$ and $r_1, \ldots, r_n$ are places of P such that $\{(s_1, r_1), \ldots, (s_n, r_n)\} \subseteq R$
then there exists a transition u of P with $L(u) = L(t)$, $^\bullet u = \{r_1, \ldots, r_n\}$, $u^\bullet = \{r_1', \ldots, r_m'\}$ and $\{(s_1', r_1'), \ldots, (s_n', r_n')\} \subseteq R$.
Places of a WF net that are related by a place bisimulation are called equivalent.

Note that a place bisimulation within an S-system is a strong bisimulation. We now define rules for equivalence of WF processes. Suppose P_1 given as before. The rules below tell us how to construct an equivalent WF process P_2. The $\bullet$ symbol used below always refers to P_1. The interleave construction is the the hardest one. Interleaving a subnet of a given net leads to the well-known state explosion and cannot be obtained polynomially (in the size of the subnet). Its inverse (factorization) is even harder: it is difficult to even identify subnets of a given net that can be factorized. To apply interleaving, one must identify a subnet P_3 of P_1 with certain properties. Define $T_i = \{t \in T_3 \mid {}^\bullet t \cap S_3 = \emptyset\}$, $T_o = \{t \in T_3 \mid t^\bullet \cap S_3 = \emptyset\}$, $T_b = (T_3 \setminus T_i) \setminus T_o$. Now P_3 must satisfy

$S_3 \subseteq S_1 \setminus \{h_1\}, T_3 \subseteq T_1, F_3 \subseteq F_1, L_3 = L_1 \cap T_3$,
$\forall s \in S_3 : {}^\bullet s \cup s^\bullet \subseteq T_3$,
$\forall t \in T_i \cup T_b : t^\bullet \subseteq S_3$,
$\forall t \in T_b \cup T_o : {}^\bullet t \subseteq S_3$.

This subnet must have the property that its *closure* is a WF process. The closure $\bar{P}_3$ of P_3 satisfies $\bar{S}_3 = S_3 \cup \{h_1\}, \bar{T}_3 = T_3, \bar{F}_3 = F_3 \cup \{(h_1, t) \mid t \in T_i\} \cup \{(t, h_1) \mid t \in T_o\}, \bar{L}_3 = L_3$ and $\bar{h}_3 = h_1$. The simplest way to satisfy these requirements is to set P_3 to P_1 with h_1 deleted. In this case, $\bar{P}_3 = P_1$. Now we formulated the construction's precondition, we can define the interleave construction. The other constructions are local and easy to implement. Their inverses have not been treated, but are as simple to formulate.

i) Let Σ be the set of states of $\bar{P}_3$. We write $M\,[t\rangle\, M'$ iff $M, M' \in \Sigma$ and $t \in T_3$ and the state transition takes place in $\bar{P}_3$. The interleave construction sets $S_2 = (S_1 \setminus S_3) \cup (\Sigma \setminus \{h_1\})$, $T_2 = (T_1 \setminus T_3) \cup \{(t, M') \mid h_1^1\,[t\rangle\, M'\}, \{(M, t, M') \mid t \in T_b \wedge M\,[t\rangle\, M'\}, \{(M, t) \mid M\,[t\rangle\, h_1^1\}$, $F_2 = \{((M, t, M'), M') \mid (M, t, M') \in T_2\} \cup \{(M, (M, t, M')) \mid (M, t, M') \in T_2\}, h_2 = h_1^1$ and $L_2(M, t, M') = L_1(t)$.

ii) Let $r, s \in S_1$ be equivalent. The fold construction sets $S_2 = S_1 \setminus \{r\}$, $T_2 = T_1$, $F_2 = F_1 \setminus (T \times \{r\} \cup \{r\} \times T) \cup \{(u, s) \mid u \in {}^\bullet r\} \cup \{(s, u) \mid u \in r^\bullet\}, h_2 = h_1$ and $L_2 = L_1$.

iii) Let $r_1, \ldots, r_n, s_1, \ldots, s_n \in S_1$ with r_i equivalent to s_i for $i = 1, \ldots, n$. Choose $t \notin T_1$. The addtau construction sets $S_2 = S_1$, $T_2 = T_1 \cup \{t\}$, $F_2 = F_1 \cup \{(r_1, t), \ldots, (r_n, t), (t, s_1), \ldots, (t, s_n)\}, h_2 = h_1$ and $L_2 = L_1 \cup \{(t, \tau)\}$.

iv) Let M, M' be subsets of S_1 such that there exist $t_1, \ldots, t_n$ in T such that $M = {}^\bullet t_1, t_1{}^\bullet = {}^\bullet t_2, \ldots, t_{n-1}{}^\bullet = {}^\bullet t_n, t_n{}^\bullet = M'$ and $L(t_1) = \ldots = L(t_{n-1}) = \tau$ and $L(t_n) = \alpha$. Choose $t \notin T_1$. The addrtrans construction sets $S_2 = S_1$, $T_2 = T_1 \cup \{t\}$, $F_2 = F_1 \cup \{(s, t) \mid s \in M\} \cup \{(t, s) \mid s \in M'\}, h_2 = h_1$ and $L_2 = L_1 \cup \{(t, \alpha)\}$.

v) Let $r \in S_1$ be a redundant place of N_1. The delrplace construction sets $S_2 = S_1 \setminus \{r\}$, $T_2 = T_1$, $F_2 = (F_1 \cap S_2 \times T_2) \cap T_2 \times S_2, h_2 = h_1$ and $L_2 = L_1$. For WF nets, redundant places can be found by computing place invariants. The place r is redundant in N_1 iff N_1 initialized with a single token in h satisfies an invariant of the form $|r| = \Sigma_{s \in S_2} g_s . |s|$ with $|p|$ denoting the number of tokens in a place p and the g_s being nonnegative rational weights. For the addrplace rule, one can concoct such an invariant and then connect the new place r to transitions in a way to satisfy this invariant. The weights in the concocted invariant must be chosen with care.

vi) Let $t \in T_1$ with $L_1(t) = \tau$ such that all places in ${}^\bullet t$ have no other transitions in their postsets. So formally, $\forall s \in {}^\bullet t : s^\bullet = \{t\}$. Moreover, if $\{h_1\} = {}^\bullet t$, then $t^\bullet$ must be a singleton, say $\{\eta\}$. Choose $s_{i,j} \notin S_1$ for $i \in {}^\bullet t, j \in t^\bullet$. The mertau construction sets $T_2 = T_1 \setminus \{t\}$, $S_2 = S_1 \setminus ({}^\bullet t \cup t^\bullet) \cup \{s_{i,j} \mid i \in {}^\bullet t, j \in t^\bullet\}$, $F_2 = F_1 \cap (S_2 \times T_2 \cup T_2 \times S_2) \cup \{(t, s_{i,j}) \mid (t, i) \in F_1\} \cup \{(s_{i,j}, u) \mid (j, u) \in F_1\}$, $h_2 = h_1$ if $h_1 \notin {}^\bullet t$, $h_2 = s_{h_1,\eta}$ otherwise and $L_2 = L_1$.

We now state and prove our first main theorem.

Theorem 1 *WF processes P, Q are equivalent iff they can be transformed (up to isomorphism) into one another by applying the above rules.*

Proof: The "if" part (soundness) is proved by constructing a delay bisimulation in all the cases above. For instance in the mertau case, the correspondence between the states of P and Q is given by replacing a place $\sigma \in {}^\bullet t$ of a state of P by the combination of $s_{\sigma,j}$'s and a place $\sigma \in t^\bullet$ by the combination of $s_{i,\sigma}$'s.

The "only if" part (completeness) is proved as follows. Suppose P and Q are equivalent WF processes. By applying the interleave rule, we may suppose that their nets N and M are S-systems. We shall modify P by the transformation rules to arrive at a process P' that is equivalent to Q in such a way that there exists a delay bisimulation R' between P' and Q that is injective, i.e. no two places of P are related to the same place of Q. By modifying Q in the same way, we obtain a bijective delay bisimulation. By invoking the addrtrans and deltau rules, we then arrive at systems with isomorphic nets.

Let R be the delay bisimulation between the markings (places) of N and M. We take the transitive closure $(R \circ R^{-1})^*$ of the composition of R with its inverse and write $r \sim s$ iff $(r, s) \in (R \circ R^{-1})^*$. Since R was a delay bisimulation, we may deduce that $\sim$ is a delay bisimulation too, so if $r \sim s$ and $r^1 \overset{\alpha}{\Longrightarrow} \rho^1$ there exists a σ such that $\rho \sim \sigma$ and $s^1 \overset{\alpha}{\Longrightarrow} \sigma^1$. By applying the addrtrans rule, we can modify N to the effect that $r^1 \xrightarrow{\alpha} \rho^1$ iff $r^1 \overset{\alpha}{\Longrightarrow} \rho^1$. This means that $\sim$ has become a strong bisimulation, and since N is an S-system, $\sim$ is a place bisimulation in the modified system.

Be repeatedly invoking the fold rule to N we arrive at the desired injective delay bisimulation. Thus we can arrive by successive modifications at isomorphic processes, concluding our completeness proof.

Note that the delrplace/addrplace and mertau/splittau constructions have not been used in the completeness proof. This indicates that these constructions are redundant within the chosen notion of equivalence.

To arrive at a formalization of the extension and reduction rules, yet another auxiliary notion is defined.

Definition 3 *The* flow completion *of a WF process is obtained by adding a silent transition with the same pre- and postset to every transition of the original process. Let P_1 and P_2 be WF processes as before. Then P_2 is the* flow completion *$\Phi(P_1)$ of P_1 iff $S_2 = S_1$, $T_2 = T_1 \times \{0, 1\}$, $F_2 = \{(s, (t, x)) \mid ((s, t), x) \in F_1 \times \{0, 1\}\} \cup \{((t, x), s) \mid ((t, s), x) \in F_1 \times \{0, 1\}\}$, $h_2 = h_1$ and $L_2 = \{((t, 1), L(t)) \mid t \in T_1\} \cup \{((t, 0), \tau) \mid t \in T_1\}$.*

It is straightforward to show that from a delay simulation between WF processes P and Q can be constructed a delay simulation between $\Phi(P)$ and $\Phi(Q)$. So the Φ operator is a *congruence* w.r.t. equivalence.

Lemma 1 *Let P, Q be WF processes. P is a reduction of Q iff there exists a delay simulation between P and $\Phi(Q)$.*

Proof: If $M \xrightarrow{\alpha} M'$ in Q, then also $M \xrightarrow{\alpha} M'$ in $\Phi(Q)$, but also $M \overset{\tau}{\Longrightarrow} M'$ in $\Phi(Q)$ whenever $M \xrightarrow{*} M'$ in Q. The rest follows from the definition of reduction.

This lemma entails that the reduction relation is transitive and thus a preorder on WF processes.

Before giving the reduction and extension rules, we give an auxiliary definition.

Definition 4 *Let t be a transition of a WF net $N = (S, T, F, h)$. Then $t\downarrow$ is defined as the smallest set of nodes satisfying the following conditions*
$t \in t\downarrow$,
$s \in S \wedge {}^\bullet s \cap t\downarrow = {}^\bullet s$,
$t \in T \wedge {}^\bullet t \cap t\downarrow \neq \emptyset$.
Informally, it consists of all places that can be marked and transitions that may fire only after t has fired.

The reduction and extension rules are now formalized as follows. Let P_1 and P_2 be defined as before. We give the ways to construct P_2 that reduces a given P_1. The inverse construction yields extension.

i) Let $t \in T_1$ such that $L_1(t) \neq \tau$. The skip construction sets $T_2 = T_1$, $S_2 = S_1$, $F_2 = F_1$, $h_2 = h_1$ and $L_2 = L_1 \setminus \{(t, L_1(t))\} \cup \{(t, \tau)\}$.

ii) Let $t \in T_1$ such that $\bigcup_{s \in {}^\bullet t} s^\bullet \neq \{t\}$. So if t can fire, some other transition can fire as well. The block construction sets $T_2 = T_1 \setminus t\downarrow$, $S_2 = S_1 \setminus t\downarrow$, $F_2 = F_1 \cap (t\downarrow \times t\downarrow)$, $h_2 = h_1$ and $L_2 = L_1 \cap T_2 \times A$. The condition is necessary and sufficient to ensure that P_2 is a WF net.

iii) Let $s \notin S_1$ and choose $T', T'' \subset T$. The addplace construction sets $T_2 = T_1$, $S_2 = S_1 \cup \{s\}$, $F_2 = F_1 \cup \{(s,t) \mid t \in T'\} \cup \{(t,s) \mid t \in T''\}$, $h_2 = h_1$ and $L_2 = L_1$, provided that N_2 is a WF net.

The following theorem states the soundness and completeness of the reduction and extension rules.

Theorem 2 *Let P, Q be WF processes. Then P is a reduction of Q iff Q can be transformed into P by the reduction and equivalence rules.*

Proof: The "only if" part (soundness) is again straightforward by establishing a reduction relation in every case. For instance suppose P is obtained from Q by the addplace construction. Every state of P corresponds to a state of Q by leaving out the added place marking. This relation between states is in fact a strong simulation, since every event possible in P is also possible in Q, leading again to related states.

The "if" part (completeness) is established as follows. Suppose a reduction relation R exists between P and Q. First transform Q into a process Q_d with twice as many transitions, by duplicating every transition, and connecting each duplicate to the same pre- and postset and giving it the same label as its original. These nets are equivalent by the addrtrans rule. Next, each duplicate transition is labeled τ instead. The resulting net Q_t is a reduction of Q_d by the skip rule. Note that Q_t equals $Phi(Q_t)$. The relation R between the states of P and Q now becomes a delay simulation between P and Q_t. Apply the interleave construction to both P and Q_t. We have S-systems P_i and Q_i with a delay simulation between their states. Mark the transitions of Q_i that lead to a place that is not related to a place of P_i, and apply the block construction to those transitions, leading to a process Q_b. The relation between the places of P_i and Q_b is now total, so it is a delay bisimulation. Hence Q_b can be transformed into P_i by the equivalence rules.

93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.
93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Komatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.
94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.
94/05	P. Zhou J. Hooman	Formal Specification and Compositional Verification of an Atomic Broadcast Protocol, p. 22.
94/06	T. Basten T. Kunz J. Black M. Coffin D. Taylor	Time and the Order of Abstract Events in Distributed Computations, p. 29.
94/07	K.R. Apt R. Bol	Logic Programming and Negation: A Survey, p. 62.
94/08	O.S. van Roosmalen	A Hierarchical Diagrammatic Representation of Class Structure, p. 22.
94/09	J.C.M. Baeten J.A. Bergstra	Process Algebra with Partial Choice, p. 16.

94/10	T. verhoeff	The testing Paradigm Applied to Network Structure. p. 31.
94/11	J. Peleska C. Huizing C. Petersohn	A Comparison of Ward & Mellor's Transformation Schema with State- & Activitycharts, p. 30.
94/12	T. Kloks D. Kratsch H. Müller	Dominoes, p. 14.
94/13	R. Seljée	A New Method for Integrity Constraint checking in Deductive Databases, p. 34.
94/14	W. Peremans	Ups and Downs of Type Theory, p. 9.
94/15	R.J.M. Vaessens E.H.L. Aarts J.K. Lenstra	Job Shop Scheduling by Local Search, p. 21.
94/16	R.C. Backhouse H. Doornbos	Mathematical Induction Made Calculational, p. 36.
94/17	S. Mauw M.A. Reniers	An Algebraic Semantics of Basic Message Sequence Charts, p. 9.
94/18	F. Kamareddine R. Nederpelt	Refining Reduction in the Lambda Calculus, p. 15.
94/19	B.W. Watson	The performance of single-keyword and multiple-keyword pattern matching algorithms, p. 46.
94/20	R. Bloo F. Kamareddine R. Nederpelt	Beyond β-Reduction in Church's $\lambda\rightarrow$, p. 22.
94/21	B.W. Watson	An introduction to the Fire engine: A C++ toolkit for Finite automata and Regular Expressions.
94/22	B.W. Watson	The design and implementation of the FIRE engine: A C++ toolkit for Finite automata and regular Expressions.
94/23	S. Mauw and M.A. Reniers	An algebraic semantics of Message Sequence Charts, p. 43.
94/24	D. Dams O. Grumberg R. Gerth	Abstract Interpretation of Reactive Systems: Abstractions Preserving $\forall$CTL*, $\exists$CTL* and CTL*, p. 28.
94/25	T. Kloks	$K_{1,3}$-free and W_4-free graphs, p. 10.
94/26	R.R. Hoogerwoord	On the foundations of functional programming: a programmer's point of view, p. 54.
94/27	S. Mauw and H. Mulder	Regularity of BPA-Systems is Decidable, p. 14.
94/28	C.W.A.M. van Overveld M. Verhoeven	Stars or Stripes: a comparative study of finite and transfinite techniques for surface modelling, p. 20.
94/29	J. Hooman	Correctness of Real Time Systems by Construction, p. 22.
94/30	J.C.M. Baeten J.A. Bergstra Gh. Ştefanescu	Process Algebra with Feedback, p. 22.
94/31	B.W. Watson R.E. Watson	A Boyer-Moore type algorithm for regular expression pattern matching, p. 22.
94/32	J.J. Vereijken	Fischer's Protocol in Timed Process Algebra, p. 38.
94/33	T. Laan	A formalization of the Ramified Type Theory, p.40.
94/34	R. Bloo F. Kamareddine R. Nederpelt	The Barendregt Cube with Definitions and Generalised Reduction, p. 37.
94/35	J.C.M. Baeten S. Mauw	Delayed choice: an operator for joining Message Sequence Charts, p. 15.
94/36	F. Kamareddine R. Nederpelt	Canonical typing and Π-conversion in the Barendregt Cube, p. 19.
94/37	T. Basten R. Bol M. Voorhoeve	Simulating and Analyzing Railway Interlockings in ExSpect, p. 30.
94/38	A. Bijlsma C.S. Scholten	Point-free substitution, p. 10.
94/39	A. Blokhuis T. Kloks	On the equivalence covering number of splitgraphs, p. 4.

94/40	D. Alstein	Distributed Consensus and Hard Real-Time Systems, p. 34.
94/41	T. Kloks D. Kratsch	Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph, p. 6.
94/42	J. Engelfriet J.J. Vereijken	Concatenation of Graphs, p. 7.
94/43	R.C. Backhouse M. Bijsterveld	Category Theory as Coherently Constructive Lattice Theory: An Illustration, p. 35.
94/44	E. Brinksma, J. Davies R. Gerth, S. Graf W. Janssen, B. Jonsson S. Katz, G. Lowe M. Poel, A. Pnueli C. Rump, J. Zwiers	Verifying Sequentially Consistent Memory, p. 160
94/45	G.J. Houben	Tutorial voor de ExSpect-bibliotheek voor "Administratieve Logistiek", p. 43.
94/46	R. Bloo F. Kamareddine R. Nederpelt	The λ-cube with classes of terms modulo conversion, p. 16.
94/47	R. Bloo F. Kamareddine R. Nederpelt	On Π-conversion in Type Theory, p. 12.
94/48	Mathematics of Program Construction Group	Fixed-Point Calculus, p. 11.
94/49	J.C.M. Baeten J.A. Bergstra	Process Algebra with Propositional Signals, p. 25.
94/50	H. Geuvers	A short and flexible proof of Strong Normalazation for the Calculus of Constructions, p. 27.
94/51	T. Kloks D. Kratsch H. Müller	Listing simplicial vertices and recognizing diamond-free graphs, p. 4.
94/52	W. Penczek R. Kuiper	Traces and Logic, p. 81
94/53	R. Gerth R. Kuiper D. Peled W. Penczek	A Partial Order Approach to Branching Time Logic Model Checking, p. 20.
95/01	J.J. Lukkien	The Construction of a small CommunicationLibrary, p.16.
95/02	M. Bezem R. Bol J.F. Groote	Formalizing Process Algebraic Verifications in the Calculus of Constructions, p.49.
95/03	J.C.M. Baeten C. Verhoef	Concrete process algebra, p. 134.
95/04	J. Hidders	An Isotopic Invariant for Planar Drawings of Connected Planar Graphs, p. 9.
95/05	P. Severi	A Type Inference Algorithm for Pure Type Systems, p.20.
95/06	T.W.M. Vossen M.G.A. Verhoeven H.M.M. ten Eikelder E.H.L. Aarts	A Quantitative Analysis of Iterated Local Search, p.23.
95/07	G.A.M. de Bruyn O.S. van Roosmalen	Drawing Execution Graphs by Parsing, p. 10.
95/08	R. Bloo	Preservation of Strong Normalisation for Explicit Substitution, p. 12.
95/09	J.C.M. Baeten J.A. Bergstra	Discrete Time Process Algebra, p. 20
95/10	R.C. Backhouse R. Verhoeven O. Weber	Math∫pad: A System for On-Line Prepararation of Mathematical Documents, p. 15
95/11	R. Seljée	Deductive Database Systems and integrity constraint checking, p. 36.
95/12	S. Mauw and M. Reniers	Empty Interworkings and Refinement

		Semantics of Interworkings Revised, p. 19.
95/13	B.W. Watson and G. Zwaan	A taxonomy of sublinear multiple keyword pattern matching algorithms, p. 26.
95/14	A. Ponse, C. Verhoef, S.F.M. Vlijmen (eds.)	De proceedings: ACP'95, p.
95/15	P. Niebert and W. Penczek	On the Connection of Partial Order Logics and Partial Order Reduction Methods, p. 12.
95/16	D. Dams, O. Grumberg, R. Gerth	Abstract Interpretation of Reactive Systems: Preservation of CTL*, p. 27.
95/17	S. Mauw and E.A. van der Meulen	Specification of tools for Message Sequence Charts, p. 36.
95/18	F. Kamareddine and T. Laan	A Reflection on Russell's Ramified Types and Kripke's Hierarchy of Truths, p. 14.
95/19	J.C.M. Baeten and J.A. Bergstra	Discrete Time Process Algebra with Abstraction, p. 15.
95/20	F. van Raamsdonk and P. Severi	On Normalisation, p. 33.
95/21	A. van Deursen	Axiomatizing Early and Late Input by Variable Elimination, p. 44.
95/22	B. Arnold, A. v. Deursen, M. Res	An Algebraic Specification of a Language for Describing Financial Products, p. 11.
95/23	W.M.P. van der Aalst	Petri net based scheduling, p. 20.
95/24	F.P.M. Dignum, W.P.M. Nuijten, L.M.A. Janssen	Solving a Time Tabling Problem by Constraint Satisfaction, p. 14.
95/25	L. Feijs	Synchronous Sequence Charts In Action, p. 36.
95/26	W.M.P. van der Aalst	A Class of Petri nets for modeling and analyzing business processes, p. 24.
95/27	P.D.V. van der Stok, J. van der Wal	Proceedings of the Real-Time Database Workshop, p. 106.
95/28	W. Fokkink, C. Verhoef	A Conservative Look at term Deduction Systems with Variable Binding, p. 29.
95/29	H. Jurjus	On Nesting of a Nonmonotonic Conditional, p. 14
95/30	J. Hidders, C. Hoskens, J. Paredaens	The Formal Model of a Pattern Browsing Technique, p.24.
95/31	P. Kelb, D. Dams and R. Gerth	Practical Symbolic Model Checking of the full μ-calculus using Compositional Abstractions, p. 17.
95/32	W.M.P. van der Aalst	Handboek simulatie, p. 51.
95/33	J. Engelfriet and JJ. Vereijken	Context-Free Graph Grammars and Concatenation of Graphs, p. 35.
95/34	J. Zwanenburg	Record concatenation with intersection types, p. 46.
95/35	T. Basten and M. Voorhoeve	An algebraic semantics for hierarchical P/T Nets, p. 32.
96/01	M. Voorhoeve and T. Basten	Process Algebra with Autonomous Actions, p. 12.
96/02	P. de Bra and A. Aerts	Multi-User Publishing in the Web: DreSS, A Document Repository Service Station, p. 12
96/03	W.M.P. van der Aalst	Parallel Computation of Reachable Dead States in a Free-choice Petri Net, p. 26.
96/04	S. Mauw	Example specifications in phi-SDL.
96/05	T. Basten and W.M.P. v.d. Aalst	A Process-Algebraic Approach to Life-Cycle Inheritance Inheritance = Encapsulation + Abstraction, p. 15.
96/06	W.M.P. van der Aalst and T. Basten	Life-Cycle Inheritance A Petri-Net-Based Approach, p. 18.
96/07	M. Voorhoeve	Structural Petri Net Equivalence, p. 16.
96/08	A.T.M. Aerts, P.M.E. De Bra, J.T. de Munk	OODB Support for WWW Applications: Disclosing the internal structure of Hyperdocuments, p. 14.
96/09	F. Dignum, H. Weigand, E. Verharen	A Formal Specification of Deadlines using Dynamic Deontic Logic, p. 18.
96/10	R. Bloo, H. Geuvers	Explicit Substitution: on the Edge of Strong Normalisation, p. 13.
96/11	T. Laan	AUTOMATH and Pure Type Systems, p. 30.
96/12	F. Kamareddine and T. Laan	A Correspondence between Nuprl and the Ramified Theory of Types, p. 12.
96/13	T. Borghuis	Priorean Tense Logics in Modal Pure Type Systems, p. 61
96/14	S.H.J. Bos and M.A. Reniers	The I^2C-bus in Discrete-Time Process Algebra, p. 25.
96/15	M.A. Reniers and J.J. Vereijken	Completeness in Discrete-Time Process Algebra, p. 139.
96/16	P. Hoogendijk and O. de Moor	What is a data type?, p. 29.