Subclasses of Formalized Data Flow Diagrams: Monogeneous, Linear, and Topologically Free Choice RDFD's

Jürgen Symanzik and Albert L. Baker

TR #96–22

December 1996

Keywords: Decidability Problem, Computation System, Isomorphism, FIFO Petri Net.

Department of Computer Science

226 Atanasoff Hall

Iowa State University

Ames, Iowa 50011–1040, USA

1 SUBCLASSES OF FORMALIZED DATA FLOW DIAGRAMS: MONOGENEOUS, LINEAR, AND TOPOLOGICALLY FREE CHOICE RDFD'S

Abstract

Formalized Data Flow Diagrams (FDFD's) and, especially, Reduced Data Flow Diagrams (RDFD's) are Turing equivalent ([SB96a]). Therefore, no decidability problem can be solved for FDFD's in general. However, it is possible to define subclasses of FDFD's for which decidability problems can be answered.

In this paper we will define certain subclasses of FDFD's, which we call Monogeneous RDFD's, Linear RDFD's, and Topologically Free Choice RDFD's. We will show that two of these three subclasses of FDFD's can be simulated via isomorphism by the correspondingly named subclasses of FIFO Petri Nets. It is known that isomorphisms between computation systems guarantee the same answers to corresponding decidability problems (e. g., reachability, deadlock, liveness) in the two systems ([KM82]). This means that problems where it is known that they can (not) be solved for a subclass of FIFO Petri Nets it follows immediately that the same problems can (not) be solved for the correspondingly named subclass of FDFD's.

1.1 Introduction

Formalized Data Flow Diagrams (FDFD's) as given in [LWBL96] are a relatively new approach to the formalization of traditional Data Flow Diagrams (DFD's). Recently it has been formally established that FDFD's are Turing equivalent ([SB96a]) and their non–atomic components, e. g., stores and persistent flows, are not essential to the expressive power of FDFD's ([SB96b]). Unfortunately, this equivalence to Turing Machines prevents the analytical solution of decidability problems (e. g., reachability, deadlock, liveness) for FDFD's.

However, there exist subclasses of another computational model with the computational power of Turing Machines, FIFO Petri Nets (introduced in [MM81]), for which decidability problems can be solved. Many variations and restrictions of the basic model of FIFO Petri Nets have been considered, e. g., in [FM82], [Sta83], [Fin84], [FR85], [MF85], [Fin86], [Rou87], [CF87], [FC88], [FR88], and [Fan92]. Probably the most important work done with respect to this current paper was the survey on decidability questions for subclasses of FIFO Petri Nets in [FR88]. There, it was established which decidability problems can be solved for which subclasses of FIFO Petri Nets typically considered in the literature, that is, Monogeneous FIFO Petri Nets, Linear FIFO Petri Nets, and Topologically Free Choice FIFO Petri Nets.

In this paper, we first summarize required definitions and main results for computation systems, FIFO Petri Nets, and decidability problems in Section 1.2. In Section 1.3, we define subclasses of Reduced Data Flow Diagrams (RDFD's), i. e., Monogeneous RDFD's, Linear RDFD's, and Topologically Free Choice RDFD's. From [SB96a] we know that every RDFD can be simulated by a FIFO Petri Net with respect to an isomorphism h. We will show that this isomorphism h actually maps Monogeneous persistent flow–free RDFD's and Linear RDFD's onto subclasses of FIFO Petri Nets of the same names. Moreover, from [KM82] we know that isomorphisms preserve many decidability problems. Therefore, we can conclude that a problem that is decidable for a subclass of FIFO Petri Nets is also decidable for the related subclass of FDFD's. Unfortunately, our isomorphism h does not map (Extended) Topologically Free Choice RDFD's to (Extended) Topologically Free Choice FIFO Petri Nets. We finish this paper with a summary on possible future research in Section 1.4.

1.2 Definitions

In the next two subsections, we summarize definitions and results from [KM82]. Please refer to this work for a more detailed explanation of symbols and for additional definitions. A short summary of [KM82] is given in [SB96a]. We assume that the reader is familiar with [SB96a] since our notations, definitions, and proofs of theorems are closely related to this reference. In Subsections 1.2.3 and 1.2.4, we summarize definitions for FIFO Petri Nets and related decidability problems. In Subsection 1.2.5, we deal with subclasses of FIFO Petri Nets.

1.2.1 Computation Systems

Definition (1.2.1.1): A *computation system* $S = (\Sigma, D, x, \overline{\quad})$ consists of a set D, an element x of D, a finite set Σ of *operations*, and a function "$\overline{\quad}$" from Σ to the set of partial functions from D to D. That is, for each $a \in \Sigma$, $\overline{a}$ is a partial function from D to D. The function "$\overline{\quad}$" is extended to Σ^* by $\overline{\lambda} =$ identity, $\overline{\alpha\beta}(y) = \overline{\alpha} \cdot \overline{\beta}(y) = \overline{\beta}(\overline{\alpha}(y))$, $\alpha \cdot \beta \in \Sigma^*, y \in D$. ■

1.2.2 Decidability Problems for Computation Systems

Definition (1.2.2.1): For a given computation system $S = (\Sigma, D, x, \overline{\quad})$, we are interested in answers to the following decidability problems:

(i) Reachability: For $y \in D$, is $y \in R_S$?

(ii) Deadlock: Does there exist an $\alpha \in C_S$ such that, for every $a \in \Sigma$, $\alpha a \notin C_S$?

(iii) Termination: Is C_S finite?

(iv) Finiteness: Is R_S finite?

(v) Equivalence of sets of computation sequences: For $y, z \in D$, is $C_S(y) = C_S(z)$?

(vi) Liveness: For any $\alpha \in C_S$ and $a \in \Sigma$, does there exist a $\beta \in \Sigma^*$ such that $\alpha\beta a \in C_S$?

(vii) Exceedability: With D a partially ordered set[1] and given $y \in D$, does there exist a $z \in R_S$ such that $z \geq y$? ■

In this definition, R_S denotes the reachability set from x, C_S the set of all finite computation sequences from x, and $C_S(y)$ the set of all finite computation sequences from y.

[1] $(D, \geq)$ can be any partial ordering on the set D.

Definition (1.2.2.2): Let $S_1 = (\Sigma_1, D_1, x_1, \overline{}^{1})$ and $S_2 = (\Sigma_2, D_2, x_2, \overline{}^{2})$ be computation systems. A *homomorphism* $h = (\tau, \rho) : S_1 \rightarrow S_2$ consists of a homomorphism $\tau : \Sigma_1^* \rightarrow \Sigma_2^*$, and an injection $\rho : D_1 \rightarrow D_2$ which satisfies the following conditions:

$$\rho(x_1) = x_2 \qquad (1.2.2.2.1)$$

$$\forall\, y, z \in R_{S_1} \,\forall\, \alpha \in \Sigma_1^* : (y \overset{\alpha}{\Leftrightarrow\!\!\rightarrow} z \;\Rightarrow\; \rho(y) \overset{\tau(\alpha)}{\Leftrightarrow\!\!\rightarrow} \rho(z)) \qquad (1.2.2.2.2)$$

■

Definition (1.2.2.3): Let $h = (\tau, \rho) : S_1 \rightarrow S_2$ be a homomorphism. h is called an *isomorphism* if there is a homomorphism $h' = (\tau', \rho') : S_2 \rightarrow S_1$ such that $hh' : S_2 \rightarrow S_2$ and $h'h : S_1 \rightarrow S_1$ are identities, i. e., $\tau\tau' : \Sigma_2^* \rightarrow \Sigma_2^*$, $\tau'\tau : \Sigma_1^* \rightarrow \Sigma_1^*$, $\rho\rho' : D_2 \rightarrow D_2$, and $\rho'\rho : D_1 \rightarrow D_1$ are identities. ■

Definition (1.2.2.4): Let H be a class of homomorphisms. Let P be a problem of the form: Given a computation system $S = (\Sigma, D, x, \overline{})$, $y_1, \ldots, y_n \in D$, whether $P(S, y_1, \ldots, y_n)$? We say that P is *preserved* by H under the following condition: For any S_1 and S_2, if there is an $h \in H$ such that $h = (\tau, \rho) : S_1 \rightarrow S_2$, then $P(S_1, y_1, \ldots, y_n)$ holds if, and only if, $P(S_2, \rho(y_1), \ldots, \rho(y_n))$ holds. ■

Theorem (1.2.2.5): Particular types of homomorphisms preserve the following decidability problems:

(i) A spanning homomorphism preserves reachability.

(ii) A spanning homomorphism preserves deadlock.

(iii) A spanning homomorphism preserves the termination property.

(iv) A surjective homomorphism preserves finiteness.

(v) A principal homomorphism preserves equivalence of sets of computation sequences.

(vi) A principal homomorphism preserves liveness.

(vii) An order preserving spanning homomorphism preserves exceedability.

Proof: Proofs are given in [KM82], Section 4. ■

It should be noted that an isomorphism h is also a bijective (hence injective, surjective, hence spanning), length preserving, and principal homomorphism. Thus, an isomorphism h preserves decidability problems (i) to (vi).

1.2.3 FIFO Petri Nets

In some sense, FIFO Petri Nets (introduced in [MM81]) are Petri Nets (see [Pet81], for example) where places contain words instead of tokens and arcs are labelled by words. More formally, we make use of the definition of FIFO Petri Nets as given in [Rou87].

Definition (1.2.3.1): A *FIFO Petri Net* is a quintuple $FPN = (P, T, B, F, Q)$ where P is a finite set of *places* (also called *queues*), T is a finite set of *transitions* (disjoint from P), Q is a finite *queue alphabet*, and $F : T \times P \to Q^*$ and $B : P \times T \to Q^*$ are two mappings called respectively *forward* and *backward incidence mappings.* ■

Definition (1.2.3.2): A *marking* M of a FIFO Petri Net is a mapping $M : P \to Q^*$.
A transition t is *fireable* in M, written $M(t >$, if $\forall p \in P : B(p,t) \leq M(p)$ (where $u \leq x$ means u is a prefix of x).

For a marking M, we define the firing of a transition t, written $M(t > M'$, if $M(t >$ and the following equation between words holds $\forall p \in P : B(p,t)M'(p) = M(p)F(t,p)$. That means, the firing of a transition t removes $B(p,t)$ from the head of $M(p)$ and appends $F(t,p)$ to the end of the resulting word. ■

Definition (1.2.3.3): A FIFO Petri Net FPN together with an initial marking $M_0 : P \to Q^*$ is called a *marked FIFO Petri Net* and is denoted by (FPN, M_0).

As usual, the firing of a transition can be extended to the firing of a sequence of transitions. We denote by $FS(FPN, M_0)$ the *set of firing sequences* of this FIFO Petri Net. The firing of a sequence u of transitions from a marking M to a marking M' is written as $M(u > M'$.

The set of markings that are reachable from M_0 is called *reachability set* and it is denoted by $Acc(FPN, M_0)$. ■

In addition, the following two definitions from [FR88] are used within this paper:

Definition (1.2.3.4): Let $R(FPN, M_0)$ denote the set of all markings that are reachable from M_0, i. e., $R(FPN, M_0) = \{M \mid \exists x \in T^* : M_0(x > M\}$. Let $L(FPN, M_0)$ denote the language of the net or the set of all sequences in T^* that are fireable from M_0, i. e., $L(FPN, M_0) = \{x \mid x \in T^* \wedge M_0(x >\}$. An element $x \in T^*$ is said to be in the center of (FPN, M_0), denoted by $C(FPN, M_0)$, if, and only if, $M_0(x > M$ and $L(FPN, M)$ is infinite. ■

However in accordance with many other references, we prefer the abbreviations FS for the set of firing sequences (language, set of computation sequences) and RS for the reachability set. Therefore, we denote by $FS(FPN, M_0)$ what is denoted by $L(FPN, M_0)$ in [FR88] and we denote by $RS(FPN, M_0)$ what is denoted by $R(FPN, M_0)$ in [FR88] and by $Acc(FPN, M_0)$ in [Rou87].

Definition (1.2.3.5): The *input language* of a place p in (FPN, M_0) is defined as $L_I(FPN, M_0, p) = h_p(FS(FPN, M_0))$ with $h_p(t) = F(t, p)$. ■

1.2.4 Decidability Problems for FIFO Petri Nets

The following definition is due to [FR88].

Definition (1.2.4.1): For a given marked FIFO Petri Net (FPN, M_0), we define the following decidability problems:

Total Deadlock Problem (TDP): Is $FS(FPN, M_0)$ finite?

Partial Deadlock Problem (PDP): Is there a finite path in (FPN, M_0) that can not be extended, i. e., does there exist an $x \in T^*$ such that $M_0(x > M$ where no transition in T is fireable from M?

Boundedness Problem (BP): Is $RS(FPN, M_0)$ finite?

Reachability Problem (RP): For a marking M, is $M \in RS(FPN, M_0)$?

Quasi–Liveness Problem (QLP): $\forall t \in T$, is there an $x \in T^*$ such that $M_0(xt >$?

Liveness Problem (LP): $\forall M \in RS(FPN, M_0)\ \ \forall t \in T$, is there an $x \in T^*$ such that $M(xt >$?

Center Problem (CP): Is there an algorithm that will generate a recursive representation of $C(FPN, M_0)$?

Regularity Problem (RegP): Is $FS(FPN, M_0)$ regular? ■

Unfortunately, names for decidability problems for computation systems in [KM82] and FIFO Petri Nets in [FR88] differ. Other terms can be found within the literature. It should be noted that the following names for decidabilty problems are identical:

Computation Sytsem	FIFO Petri Net
Reachability	RP
Deadlock	PDP
Termination	TDP
Finiteness	BP
Equivalence	
Liveness	LP
Exceedability	
	QLP
	CP
	RegP

1.2.5 Subclasses of FIFO Petri Nets

In this subsection, we summarize definitions and theorems related to Monogeneous FIFO Petri Nets (e. g., [Sta83], [Fin84], [MF85], [Fin86], [Rou87], [FR88]), Linear FIFO Petri Nets (e. g., [CF87], [FR88]), and and (Extended) Topologically Free Choice FIFO Petri Nets (e. g., [Fin86], [Rou87], [FC88], [FR88]).

Monogeneous FIFO Petri Nets

In this part, we follow the notation in [Fin86] and [FR88].

Definition (1.2.5.1): Let A be a finite alphabet. Let L be a language on A. Let x and y be words in L. x is called a *left factor* of y, $x \leq y$ in symbols, if $\exists$ word $z \in A^*$: $xz = y$.

For a language $L \subset A^*$, we denote by $LeftFactor(L)$ the set of all left factors of words in L, i. e., $LeftFactor(L) = \{x \in A^* \mid \exists y \in L : x \leq y\}$. ■

Definition (1.2.5.2): Let A be a finite alphabet.

A language $L \subset A^*$ is called *strictly monogeneous* if $\exists$ words $u, v \in A^*$: $L \subset LeftFactor(uv^*)$.

A language $L \subset A^*$ is called *monogeneous* if it is equal to a finite union of strictly monogeneous languages, i. e., $L \subset \bigcup_{i=1,\ldots,k} LeftFactor(u_i v_i^*)$, where $\forall\, i \in \{1, \ldots, k\}$: $u_i, v_i \in A^*$. ■

Definition (1.2.5.3): Let (FPN, M_0) be a marked FIFO Petri Net. Let $p \in P$ be a place of FPN.

- p is called *structurally monogeneous*, if $\exists$ word $u_p \in A^*$ such that $\forall t \in T \,:\, F(t,p) \in u_p^*$.
- p is called *strictly monogeneous* if $L_I(FPN, M_0, p)$ is strictly monogeneous.
- p is called *monogeneous* if $L_I(FPN, M_0, p)$ is monogeneous.

(FPN, M_0) is called a *Monogeneous* (*Structurally Monogeneous*, *Strictly Monogeneous*, respectively) *FIFO Petri Net* if, and only if, each of its places is monogeneous (structurally monogeneous, strictly monogeneous, respectively). ■

Unfortunately, there exists no common understanding of these terms in the literature. In [Rou87] for example, the term *monogeneous* is used instead of *strictly monogeneous* and *semi–monogeneous* is used instead of *monogeneous*. Even more confusing, in [Sta83] and [Fin84] the term *monogeneous* is used for the weaker *structurally monogeneous*.

While undecidable in the general case, [Fin86] provides many sufficient and necessary conditions for a FIFO Petri Net to be monogeneous.

Linear FIFO Petri Nets

In this part, we follow the notation in [FR88].

Definition (1.2.5.4): Let A be a finite alphabet. A language $L \subset A^*$ is called *bounded* or *linear* if L is included in $a_1^* \ldots a_n^*$ for some $a_1, \ldots, a_n \in A$ with $\forall\, i \neq j \,:\, a_i \neq a_j$. ■

Definition (1.2.5.5): Let (FPN, M_0) be a marked FIFO Petri Net. Let $p \in P$ be a place of FPN. p is called *linear* if its input language is bounded.

(FPN, M_0) is called a *Linear FIFO Petri Net* (LFPN) if, and only if, each of its places is linear and has as its initial marking an element of a_1^*. ■

Definition (1.2.5.6): Let (FPN, M_0) be a marked LFPN with $FPN = (P, T, B, F, Q)$. Let SM be a set of markings over P. SM is called a *Structured Set of Terminal Markings* (SSTM) with respect to (FPN, M_0) if, and only if:

(i) membership in SM is decidable,

(ii) $M_0 \in SM$,

(iii) $\forall\, x, y \in T^* \;:\; (M_0(xy > M \wedge M_0(x > M' \wedge M \in SM) \;\Rightarrow\; M' \in SM$ (i. e., each marking reached on a path into SM must be in SM), and

(iv) $\forall\, x \in T^* \;:\; (M \in SM \wedge M(x^i > M_i, i \geq 1 \wedge M \leq M_1 \wedge M_1 \in SM) \;\Rightarrow\; \forall i \geq 1 \;:\; M_i \in SM$ (i. e., any sequence of transitions which when applied to a marking in SM terminates at another marking in SM and can be repeated indefinitely without leaving SM). ■

The notation $M \leq M_1$ relates to the definition of left factors. $M \leq M_1$ if, and only if, for all places $p \in P$ the marking M of p is a left factor of the marking M_1 of p.

Definition (1.2.5.7): Let (FPN, M_0) be a marked LFPN with $FPN = (P, T, B, F, Q)$. Let SM be a structured set of terminal markings over P. (FPN, M_0, SM) is called a *Linear FIFO Petri Net having a Structured Set of Terminal Markings* (SSTM–LFPN). The *set of firing sequences* (language) of (FPN, M_0, SM) is $FS(FPN, M_0, SM) = \{x \mid x \in T^*, M_0(x > M, M \in SM\}$. ■

The reachability tree for (FPN, M_0, SM) is simply the reachability tree for (FPN, M_0) pruned by truncating a path whenever it leaves SM. Therefore, the following holds for the reachability set of (FPN, M_0, SM):

$$RS(FPN, M_0, SM) = RS(FPN, M_0) \cap SM$$

Topologically Free Choice FIFO Petri Nets

In this part, we follow the notation in [FC88].

Definition (1.2.5.8): Let (FPN, M_0) be a marked FIFO Petri Net. Let $p \in P$ be a place of the FIFO Petri Net.

- The *input alphabet* of p is the set of all letters that appear in the valuation of at least one input arc of p.
- The *output alphabet* of p is the set of all letters appearing in the valuations of the output arcs.
- The *alphabet* of p, denoted by A_p, is the union of the input alphabet and the output alphabet.

■

Definition (1.2.5.9): Let (FPN, M_0) be a marked FIFO Petri Net. Let $p \in P$ be a place and $t \in T$ be a transition of the FIFO Petri Net. We define:

$$\begin{array}{rcl} ,(p) & = & \{v \in T \mid B(p,v) \neq \lambda\} \\ ,(t) & = & \{v \in P \mid F(t,v) \neq \lambda\} \\ ,^-(p) & = & \{v \in T \mid F(v,p) \neq \lambda\} \\ ,^-(t) & = & \{v \in P \mid B(v,t) \neq \lambda\} \end{array}$$ ■

Definition (1.2.5.10): Let (FPN, M_0) be a marked FIFO Petri Net. (FPN, M_0) is called *normalized* if the following three conditions are satisfied:

(i) Each place $p \in P$ is balanced, i. e., the input alphabet is identical to the output alphabet.

(ii) $\forall p \in P \; \forall t \in , (p) \; : \; B(t,p) \in Q$, i. e., each place is semi–alphabetic.

(iii) $\forall p \in P \; : \; M_0(p) \in A_p^*$. ■

Definition (1.2.5.11): *Hack's condition* for *free choice Petri Nets* reads as follows: A place p in a Petri Net is free choice if, and only if, we have:

$$\mid , (p) \mid > 1 \; \Rightarrow \; \forall t \in , (p) \; : \; ,^-(t) = \{p\}$$ ■

Definition (1.2.5.12): Let (FPN, M_0) be a marked FIFO Petri Net. (FPN, M_0) is called an *Extended Topologically Free Choice FIFO Petri Net* (ETFC–FPN) if, and only if, the following two conditions are satisfied:

- (FPN, M_0) is normalized.
- $\forall p \in P \; : \mid A_p \mid > 1 \; \Rightarrow \; p$ satisfies the Hack's condition. ■

1.3 Subclasses of Formalized Data Flow Diagrams

We assume that the reader is familiar with the concept of FDFD's given in [LWBL96] and [SB96a]. A short summary of [LWBL96] and definitions of Reduced Data Flow Diagrams (RDFD's) and persistent flow–free Reduced Data Flow Diagrams (PFF–RDFD's) is given in [SB96a] as well. Here, we will only provide three basic definitions related to FDFD's.

Definition (1.3.1): A *Formalized Data Flow Diagram* (FDFD) is a quintuple

$$FDFD = (B, FLOWNAMES, TYPES, P, F),$$

where B is a set of *bubbles*, $FLOWNAMES$ is a set of *flows*, $TYPES$ is a set of *types*, P is the set $\{persistent,\ consumable\}$ and $F = B \times FLOWNAMES \times TYPES \times B \times P$. The following notational convention for members from these domains is used: $b \in B, fn \in FLOWNAMES, T \in TYPES, p \in P, f \in F$. ■

Definition (1.3.2): A *firing sequence* (*computation sequence*) of an FDFD is a possibly infinite sequence $(b_i, a_i, j_i) \in B \times \{C, P\} \times I\!N, i \geq 0$, such that, if transition (b_i, a_i, j_i) is fired in state (bm, r, fs), then

$$(fs', r') = \begin{cases} (Consume(b_i))_{j_i}(fs, r), & \text{if } a_i = C \\ (Produce(b_i))_{j_i}(fs, r), & \text{if } a_i = P \end{cases}$$

$$bm'(b_i) = \begin{cases} working, & \text{if } a_i = C \\ idle, & \text{if } a_i = P \end{cases}$$

$$bm'(b) = bm(b) \quad \forall\, b \in B \Leftrightarrow \{b_i\}$$

and

$$(bm, r, fs) \rightarrow (bm', r', fs').$$

We introduce the notation $(bm, r, fs)[(b, a, j)]$ to indicate that transition (b, a, j) is fireable in state (bm, r, fs) and $(bm, r, fs)[(b, a, j)](bm', r', fs')$ to indicate that state (bm', r', fs') is reached upon the firing of transition (b, a, j) in state (bm, r, fs).
By induction, we extend this notation for firing sequences:

$$(bm_0, r_0, fs_0)[(b_1, a_1, j_1), \ldots, (b_{n-1}, a_{n-1}, j_{n-1}), (b_n, a_n, j_n)]$$

is used to indicate that transition (b_n, a_n, j_n) is fireable in state $(bm_{n-1}, r_{n-1}, fs_{n-1})$, given that

$$(bm_0, r_0, fs_0)[(b_1, a_1, j_1), \ldots, (b_{n-1}, a_{n-1}, j_{n-1})](bm_{n-1}, r_{n-1}, fs_{n-1})$$

holds. By analogy, we use

$$(bm_0, r_0, fs_0)[(b_1, a_1, j_1), \ldots, (b_n, a_n, j_n)](bm_n, r_n, fs_n)$$

to indicate that state (bm_n, r_n, fs_n) is reached upon the firing of the sequence $(b_1, a_1, j_1), \ldots, (b_n, a_n, j_n)$. ■

Definition (1.3.3): The *set of firing sequences* (*set of computation sequences, language*) of an FDFD, denoted by $FS(FDFD, \gamma_{initial})$, is the set containing all firing sequences that are possible for this FDFD, given $\gamma_{initial} = (bm_{initial}, r_{initial}, fs_{initial}) = (bm_0, r_0, fs_0)$, i. e.,

$$FS(FDFD, \gamma_{initial}) = \{s \mid s \in (B \times \{C, P\} \times I\!N)^* \ \wedge \ \gamma_{initial}[s]\}.$$

An element $s \in (B \times \{C, P\} \times I\!N)^*$ is said to be in the center of $(FDFD, \gamma_{initial})$, denoted by $C(FDFD, \gamma_{initial})$, if, and only if, $\gamma_{initial}[s]\gamma$ and $FS(FDFD, \gamma)$ is infinite. ■

We give the next definition in analogy to Definition (1.2.4.1):

Definition (1.3.4): For a given FDFD with initial state $\gamma_{initial} = (bm_{initial}, r_{initial}, fs_{initial})$, we define the following decidability problems:

Total Deadlock Problem (TDP): Is $FS(FDFD, \gamma_{initial})$ finite?

Partial Deadlock Problem (PDP): Is there a finite path in $(FDFD, \gamma_{initial})$ that can not be extended, i. e., does there exist an $s \in (B \times \{C, P\} \times I\!N)^*$ such that $\gamma_{initial}[s]\gamma$ where no transition $(b, a, j) \in (B \times \{C, P\} \times I\!N)$ is fireable in state γ?

Boundedness Problem (BP): Is $RS(FDFD, \gamma_{initial})$ finite?

Reachability Problem (RP): For a state γ, is $\gamma \in RS(FDFD, \gamma_{initial})$?

Quasi–Liveness Problem (QLP): $\forall\, (b, a, j) \in (B \times \{C, P\} \times I\!N)$, is there an $s \in (B \times \{C, P\} \times I\!N)^*$ such that $\gamma_{initial}[s, (b, a, j)]$?

Liveness Problem (LP): $\forall\, \gamma \in RS(FDFD, \gamma_{initial}) \quad \forall\, (b, a, j) \in (B \times \{C, P\} \times I\!N)$, is there an $s \in (B \times \{C, P\} \times I\!N)^*$ such that $\gamma[s, (b, a, j)]$?

Center Problem (CP): Is there an algorithm that will generate a recursive representation of $C(FDFD, \gamma_{initial})$?

Regularity Problem (RegP): Is $FS(FDFD, \gamma_{initial})$ regular? ■

1.3.1 Monogeneous (PFF–)RDFD's

Our definitions of Monogeneous (PFF–)RDFD's are related to the definitions of monogeneous languages and Monogeneous FIFO Petri Nets as given in [Fin86] and [FR88], summarized in Subsection 1.2.5.

Definition (1.3.1.1): For each flow $f \in F$ of an FDFD, we define

$$I_f \ : (B \times \{C,P\} \times \mathbb{N}) \times (\mathit{BubbleMode} \times \mathit{Read} \times \mathit{FlowState}) \to \mathit{OBJECTS} \cup \{<>\}$$

such that

$$I_f((b,a,j),(bm,r,fs)) = \begin{cases} o, & \text{if } a = P \text{ and } (\mathit{Produce}(b))_j = \mathit{Out}(o,f,b)(fs,r) \\ <>, & \text{otherwise} \end{cases}$$

where $(b,a,j) \in (B \times \{C,P\} \times \mathbb{N})$ is a transition and $(bm,r,fs) \in (\mathit{BubbleMode} \times \mathit{Read} \times \mathit{FlowState})$ is a state of the FDFD.
By induction, we define I_f for firing sequences:

$$I_f \ : (B \times \{C,P\} \times \mathbb{N})^{n+1} \times (\mathit{BubbleMode} \times \mathit{Read} \times \mathit{FlowState}) \to (\mathit{OBJECTS} \cup \{<>\})^*$$

such that

$$\begin{aligned} &I_f(((b_0,a_0,j_0),\ldots,(b_{n-1},a_{n-1},j_{n-1}),(b_n,a_n,j_n)),(bm,r,fs)) = \\ &\qquad I_f(((b_0,a_0,j_0),\ldots,(b_{n-1},a_{n-1},j_{n-1})),(bm,r,fs)) \circ I_f((b_n,a_n,j_n),(bm',r',fs')), \end{aligned}$$

where $(bm,r,fs)[(b_0,a_0,j_0),\ldots,(b_{n-1},a_{n-1},j_{n-1})](bm',r',fs')$ and "$\circ$" means the concatenation of words.
Finally, the *input language* of a flow $f \in F$ of an FDFD with initial state $\gamma_{initial} = (bm_{initial}, r_{initial}, fs_{initial})$ is defined as

$$\begin{aligned} \tilde{I}_f &:= I_f(FS(FDFD,\gamma_{initial}),\gamma_{initial}) \\ &= \{I_f(s,\gamma_{initial}) \mid s \in FS(FDFD,\gamma_{initial})\} \end{aligned} \qquad \blacksquare$$

Definition (1.3.1.2): Let $f \in F$ be a flow of an FDFD.

- f is called *structurally monogeneous* if $\exists\, u_f \in \mathit{OBJECTS} \cup \{<>\} \ \ \forall (b,a,j) \in (B \times \{C,P\} \times \mathbb{N}) \ \ \forall (bm,r,fs) \in (\mathit{BubbleMode} \times \mathit{Read} \times \mathit{FlowState})$:
$$I_f((b,a,j),(bm,r,fs)) = \begin{cases} u_f, & \text{if } a = P \text{ and } (\mathit{Produce}(b))_j = \mathit{Out}(u_f,f,b)(fs,r) \\ <>, & \text{otherwise} \end{cases}$$
- f is called *strictly monogeneous* if $\tilde{I}_f$ is strictly monogeneous.
- f is called *monogeneous* if $\tilde{I}_f$ is monogeneous. $\blacksquare$

A structurally monogeneous flow of an FDFD is more restricted than a structurally monogenous place of a FIFO Petri Net. In the FDFD, a single object $u_f \in \mathit{OBJECTS}$ (or nothing) is appended to the flow, while in the FIFO Petri Net an entire word $u_p \in A^*$ can be appended to the place. This limited behavior of the FDFD is caused by the built–in restrictions on *Produce* (see [LWBL96]) that do not allow expressions such as $Out(u_f, f, X)(Out(u_f, f, X)(fs, r))$, i. e., each outflow f can be addressed at most once in a single *Produce* case of bubble X.

Definition (1.3.1.3): A (PFF–)RDFD is called a *Monogeneous* (*Structurally Monogeneous*, *Strictly Monogeneous*, respectively) (PFF–)RDFD if, and only if, each of its flows $f \in F$ is monogeneous (structurally monogeneous, strictly monogeneous, respectively). ■

Note that every Structurally Monogeneous (PFF–)RDFD is also a Strictly Monogeneous (PFF–) RDFD, which is also a Monogeneous (PFF–)RDFD, i. e., Monogeneous (PFF–)RDFD's are the most general of these subclasses. If we state that a condition holds for Monogeneous (PFF–)RDFD's this obviously includes Structurally Monogeneous (PFF–)RDFD's and Strictly Monogeneous (PFF–)RDFD's.

Example (1.3.1.4): This example of an PFF–RDFD presents a simple communication protocol. Each participant, A and B, can initiate the communication but then has to wait for an acknowledgement from the other participant that matches its own message. It should be obvious that in this example we always have $Head(fs(BA)) = Head(fs(last_A))$ if both are not $\perp$, and $Head(fs(AB)) = Head(fs(last_B))$ if both are not $\perp$. In a system where erraneous channels are modeled instead of flows AB and BA, the current specification of bubbles A and B will most likely produce several deadlock states.
The mappings *Enabled*, *Consume*, and *Produce* for the FDFD shown in Figure 1.1 are defined as:

$$
\begin{aligned}
&Enabled(A) = \lambda fs\ .\\
&\qquad (\neg IsEmpty(init_A) \wedge Head(fs(init_A)) = a)\\
&\qquad \vee(\neg IsEmpty(BA) \wedge Head(fs(BA)) = a\\
&\qquad\qquad \wedge \neg IsEmpty(last_A) \wedge Head(fs(last_A)) = a)\\
&\qquad \vee(\neg IsEmpty(BA) \wedge Head(fs(BA)) = b\\
&\qquad\qquad \wedge \neg IsEmpty(last_A) \wedge Head(fs(last_A)) = b)\\
&Enabled(B) = \lambda fs\ .\\
&\qquad (\neg IsEmpty(init_B) \wedge Head(fs(init_B)) = a)\\
&\qquad \vee(\neg IsEmpty(AB) \wedge Head(fs(AB)) = a\\
&\qquad\qquad \wedge \neg IsEmpty(last_B) \wedge Head(fs(last_B)) = a)\\
&\qquad \vee(\neg IsEmpty(AB) \wedge Head(fs(AB)) = b\\
&\qquad\qquad \wedge \neg IsEmpty(last_B) \wedge Head(fs(last_B)) = b)
\end{aligned}
$$

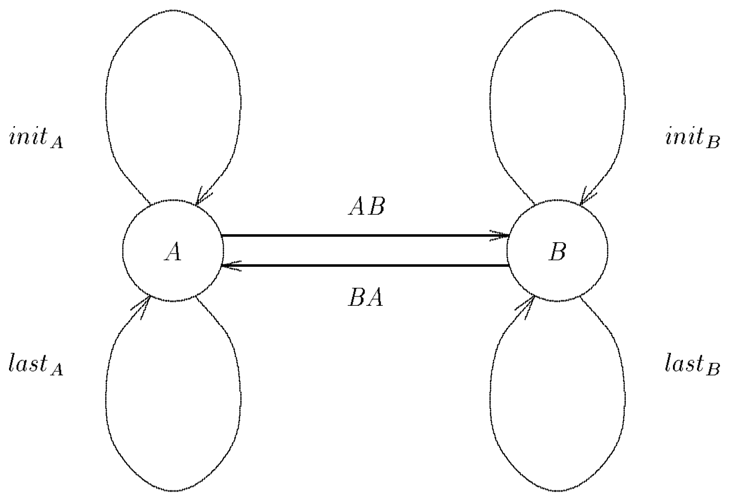

Figure 1.1: Example of a Strictly Monogeneous PFF–RDFD.

```
                                                                    Transition
$Consume(A) = \lambda(fs, r)$ .
    {if $(\neg IsEmpty(init_A) \wedge Head(fs(init_A)) = a)$
    then $In(init_A, A)(fs, r)$                                      (A, C, 1)
    fi,
    if $(\neg IsEmpty(BA) \wedge Head(fs(BA)) = a$
        $\wedge\neg IsEmpty(last_A) \wedge Head(fs(last_A)) = a)$
    then $In(BA, A)(In(last_A, A)(fs, r))$                           (A, C, 2)
    fi,
    if $(\neg IsEmpty(BA) \wedge Head(fs(BA)) = b$
        $\wedge\neg IsEmpty(last_A) \wedge Head(fs(last_A)) = b)$
    then $In(BA, A)(In(last_A, A)(fs, r))$                           (A, C, 3)
    fi
    }
$Consume(B) = \lambda(fs, r)$ .
    {if $(\neg IsEmpty(init_B) \wedge Head(fs(init_B)) = a)$
    then $In(init_B, B)(fs, r)$                                      (B, C, 1)
    fi,
    if $(\neg IsEmpty(AB) \wedge Head(fs(AB)) = a$
        $\wedge\neg IsEmpty(last_B) \wedge Head(fs(last_B)) = a)$
        then $In(AB, B)(In(last_B, B)(fs, r))$                    (B, C, 2)
        fi,
        if $(\neg IsEmpty(AB) \wedge Head(fs(AB)) = b$
            $\wedge \neg IsEmpty(last_B) \wedge Head(fs(last_B)) = b)$
        then $In(AB, B)(In(last_B, B)(fs, r))$                    (B, C, 3)
        fi
        }

    $Produce(A) = \lambda(fs, r)$ .
        {if $r(A)(init_A) = a$
        then $Out(a, AB, A)(Out(a, last_A, A)(fs, r))$            (A, P, 1)
        fi,
        if $r(A)(BA) = a \wedge r(A)(last_A) = a$
        then $Out(b, AB, A)(Out(b, last_A, A)(fs, r))$            (A, P, 2)
        fi,
        if $r(A)(BA) = b \wedge r(A)(last_A) = b$
        then $Out(a, AB, A)(Out(a, last_A, A)(fs, r))$            (A, P, 3)
        fi
        }
    $Produce(B) = \lambda(fs, r)$ .
        {if $r(B)(init_B) = a$
        then $Out(a, BA, B)(Out(a, last_B, B)(fs, r))$            (B, P, 1)
        fi,
        if $r(B)(AB) = a \wedge r(B)(last_B) = a$
        then $Out(b, BA, B)(Out(b, last_B, A)(fs, r))$            (B, P, 2)
        fi,
        if $r(B)(BA) = b \wedge r(A)(last_B) = b$
        then $Out(a, BA, B)(Out(a, last_B, B)(fs, r))$            (B, P, 3)
        fi
        }
```

Initially, $init_A$ and $init_B$ contain an a. All other flows are empty. Valid firing sequences are, for example,

$(A, C, 1), (A, P, 1), (B, C, 1), (B, P, 1),$

$$(A,C,2),(A,P,2),(B,C,2),(B,P,2),(A,C,3),(A,P,3),(B,C,3),(B,P,3),$$
$$(A,C,2),(A,P,2),(B,C,2),(B,P,2),(B,C,3),(B,P,3),(A,C,3),(A,P,3),\ldots$$

and

$$(B,C,1),(A,C,1),(A,P,1),(B,P,1),$$
$$(B,C,2),(B,P,2),(A,C,2),(A,P,2),(A,C,3),(B,C,3),(B,P,3),(A,P,3),$$
$$(A,C,2),(B,C,2),(B,P,2),(A,P,2),(B,C,3),(A,C,3),(B,P,3),(A,P,3),\ldots\ .$$

We have $\tilde{I}_{init_A} = \tilde{I}_{init_B} = \{a\}$ and $\tilde{I}_{AB} = \tilde{I}_{BA} = \tilde{I}_{last_A} = \tilde{I}_{last_B} = LeftFactor((ab)^*)$. Thus, flows $init_A$ and $init_B$ are structurally monogeneous, and flows AB, BA, $last_A$, and $last_B$ are strictly monogeneous. Overall, the PFF–RDFD is strictly monogeneous. ■

Theorem (1.3.1.5): Every Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) PFF–RDFD can be simulated by a Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) FIFO Petri Net with respect to an isomorphism h.

Proof: In [SB96a] it has been shown that every PFF–RDFD can be simulated by a FIFO Petri Net with respect to an isomorphism h. Therefore, we only have to show that this isomorphism h maps every monogeneous (structurally monogeneous, strictly monogeneous, respectively) flow $f \in F$ of the PFF–RDFD to a monogeneous (structurally monogeneous, strictly monogeneous, respectively) place of the FIFO Petri Net.

First, we want to recall from [SB96a] that the set of places P_{FPN} of the related FIFO Petri Net can be split into three disjoint subsets, (i) representing the flows of the PFF–RDFD, (ii) the *idle* working mode of the bubble, and (iii) the *working* working mode (including the values that have been read) of the bubble, i. e.,

$$\begin{aligned} P_{FPN} = \{f_1, \ldots, f_f\} & \\ & \cup \{b_{1,idle}, \ldots, b_{b,idle}\} \\ & \cup \bigcup_{i \in \{1,\ldots,b\}} \Big(\{b_{i,working:1} \mid Consume(b_i) \text{ is of type } C_1\} \\ & \qquad \cup \{b_{i,working:1}, \ldots, b_{i,working:m_i} \mid Consume(b_i) \text{ is of type } C_2, \\ & \qquad\qquad m_i = (\#\text{ of cases in } Consume(b_i))\} \Big) \end{aligned}$$

Now, we consider each of the subsets of places in P_{FPN}, with h given as in Theorem (3.1.1) in [SB96a]:

(i) $p \in \{f_1, \ldots, f_f\}$:

Since $M_{FPN}(p) = fs(p)$ by definition, the contents of each place of the FIFO Petri Net is identical to the contents of the corresponding flow of the PFF–RDFD. Also, a new value is appended to place p if, and only if, the related value is appended to the corresponding flow. Hence, since flow p is monogeneous (structurally monogeneous, strictly monogeneous, respectively), place p is monogeneous (structurally monogeneous, strictly monogeneous, respectively), too.

(ii) $p \in \{b_{1,idle}, \ldots, b_{b,idle}\}$:

The only value that is appended to place p is I. Therefore, p is structurally monogeneous (which implies that it is strictly monogeneous and monogeneous).

(iii) $p \in \{b_{1,working:1}, \ldots, b_{1,working:m_1}, \ldots, b_{b,working:1}, \ldots, b_{b,working:m_b}\}$:

The only value that is appended to place p is W. Therefore, p is structurally monogeneous (which implies that it is strictly monogeneous and monogeneous).

So, since the PFF–RDFD is monogeneous (structurally monogeneous, strictly monogeneous, respectively), i. e., each of its flows is monogeneous (structurally monogeneous, strictly monogeneous, respectively), the FIFO Petri Net is monogeneous (structurally monogeneous, strictly monogeneous, respectively), too. ■

Example (1.3.1.6): The previous Theorem does not hold in general for RDFD's with persistent flows. Consider the RDFD given in Figure 1.2.

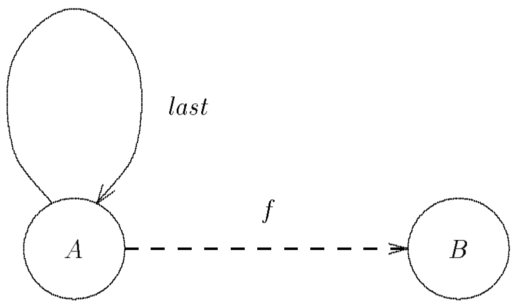

Figure 1.2: Example of a Strictly Monogeneous RDFD with Persistent Flow.

The mappings *Enabled*, *Consume*, and *Produce* are specified as follows:

$Enabled(A) = \lambda fs\ .$

$(\neg IsEmpty(last) \wedge Head(fs(last)) = 0)$

$\vee(\neg IsEmpty(last) \wedge Head(fs(last)) = 1)$

$Enabled(B) = \lambda fs\ .$

$Head(fs(f)) = 0 \vee Head(fs(f)) = 1$

```
$Consume(A) = \lambda(fs, r)$ .
    {**if** $(\neg IsEmpty(last) \wedge Head(fs(last)) = 0)$
     **then** $In(last, A)(fs, r)$
     **fi**,
     **if** $(\neg IsEmpty(last) \wedge Head(fs(last)) = 1)$
     **then** $In(last, A)(fs, r)$
     **fi**
    }
$Consume(B) = \lambda(fs, r)$ .
    {**if** $Head(fs(f)) = 0$
     **then** $In(f, B)(fs, r)$
     **fi**,
     **if** $Head(fs(f)) = 1$
     **then** $In(f, B)(fs, r)$
     **fi**
    }

$Produce(A) = \lambda(fs, r)$ .
    {**if** $r(A)(last) = 0$
     **then** $Out(1, f, A)(Out(1, last, A)(fs, r))$
     **fi**,
     **if** $r(A)(last) = 1$
     **then** $Out(0, f, A)(Out(0, last, A)(fs, r))$
     **fi**
    }
$Produce(B) = \lambda(fs, r) . \{(fs, [b_i \mapsto \lambda f . \perp]r)\}$
```

Initially, f and *last* contain a 0. Obviously, $\tilde{I}_{last} = \tilde{I}_f = LeftFactor((01)^*)$, i. e., flows *last* and f are strictly monogeneous. Overall, the RDFD is strictly monogeneous. The equivalent FIFO Petri Net constructed according to [SB96a] is given in Figure 1.3. Since $L_I(FPN, M_0, last) = LeftFactor((01)^*)$, place *last* is strictly monogeneous, but since $L_I(FPN, M_0, f) = LeftFactor((0^+1^+)^*)$, place f is not strictly monogeneous (it is not even monogeneous). ■

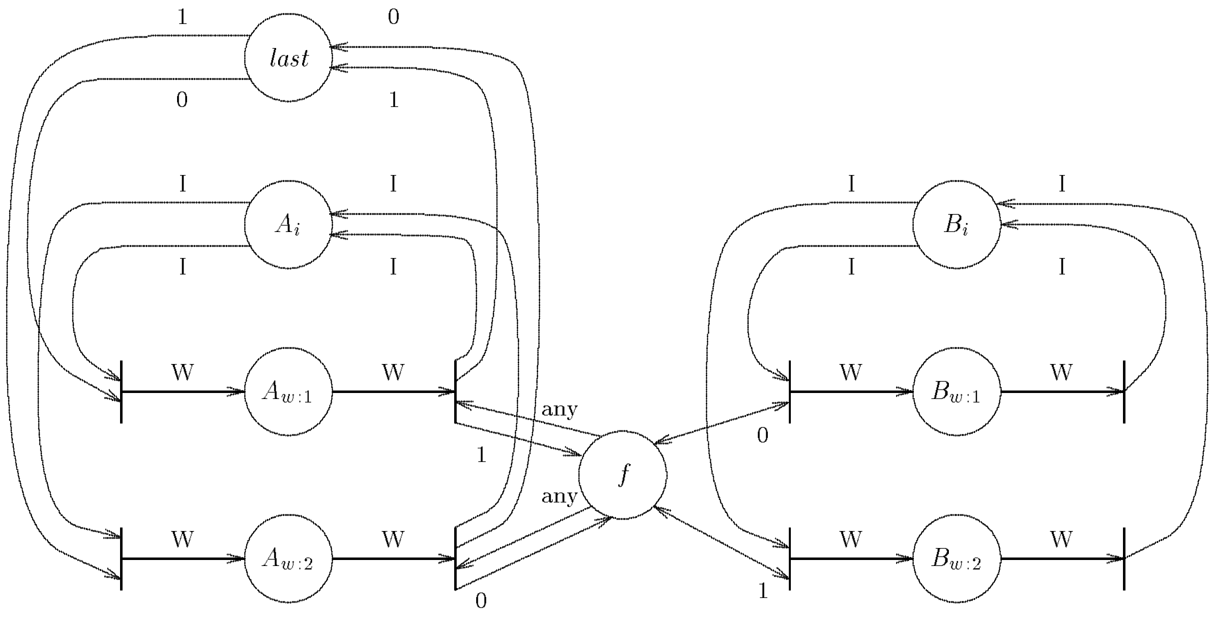

Figure 1.3: FIFO Petri Net Equivalent to a Monogeneous RDFD with Persistent Flow.

Corollary (1.3.1.7): The following problems are decidable for Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) PFF–RDFD's: TDP, PDP, BP, RP, QLP, LP, and RegP. The center of a Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) PFF–RDFD is effectively realizable, i. e., the CP is decidable.

Proof: All problems are decidable with respect to Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) FIFO Petri Nets ([Fin86], [FR88]). We have shown that there exists an isomorphism h between Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) PFF–RDFD's and Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) FIFO Petri Nets.

- According to the note following Theorem (1.2.2.5), h preserves TDP, PDP, BP, RP, and LP ([KM82]).
- QLP is decidable since LP is decidable with $\gamma = \gamma_{initial}$.
- CP is decidable for Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) FIFO Petri Nets ([FR88]).

  Since ρ is bijective, $x \in C(\textit{PFF-RDFD}, \gamma_{initial}) \Leftrightarrow \rho(x) \in C(FPN, M_0)$ where $M_0 = \rho(\gamma_{initial})$.

- RegP is decidable for Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) FIFO Petri Nets ([FR88]).

  Since τ is bijective, $FS(\mathit{PFF\text{-}RDFD}, \gamma_{initial})$ is regular $\Leftrightarrow$ $\tau(FS(FPN, M_0))$ is regular where $M_0 = \rho(\gamma_{initial})$. ■

Without giving a definition of a Petri Net (see [Pet81], for example), we state the next corollary:

Corollary (1.3.1.8): Every Monogeneous (Structurally Monogeneous, Strictly Monogeneous, respectively) PFF–RDFD can be simulated by a deterministic Petri Net.

Proof: In [Sta83] and [Fin84] it is shown that every Structurally Monogeneous FIFO Petri Net can be simulated by a labelled Petri Net.[2] In [FR88] it is shown that every Monogeneous FIFO Petri Net can be simulated by a deterministic Petri Net. Therefore, we can simulate any given Monongeneous PFF–RDFD by a Monogeneous FIFO Petri Net which is then simulated by a Petri Net. ■

The key point in this series of simulations is that every solvable decidability problem for Petri Nets remains decidable for Monogeneous FIFO Petri Nets ([Fin84]) and for Monogeneous PFF–RDFD's. Solution techniques such as the reachability tree and matrix equation approaches can be used to determine other properties such as safeness, boundedness, conservation, and coverability for Petri Nets ([Pet81]). Therefore, we immediately have solution techniques to answer related questions for Monogeneous PFF–RDFD's.

1.3.2 Linear RDFD's

Our definitions of Linear RDFD's are related to the definitions of Linear FIFO Petri Nets as given in [FR88], summarized in Subsection 1.2.5.

Definition (1.3.2.1): Let $f \in F$ be a flow of an FDFD. f is called *linear* if its input language is bounded. ■

Definition (1.3.2.2): An RDFD is called a *Linear RDFD* (L–RDFD) if, and only if, each of its flows is linear and has as its initial flow state an element of a_1^*, where $a_1 \in \mathit{OBJECTS}$. ■

[2] Note that in these two references the term *monogeneous* is used instead of the term *structurally monogeneous* which is used within this paper.

Definition (1.3.2.3): Let $\gamma_0 \in$, be the initial state of an L–RDFD. Let $S,$ be a set of states over , $=$ (*BubbleMode* $\times$ *Read* $\times$ *FlowState*). $S,$ is called a *Structured Set of Terminal States* (SSTS) with respect to $(L\text{-}RDFD, \gamma_0)$ if, and only if:

(i) membership in $S,$ is decidable,

(ii) $\gamma_0 \in S,$,

(iii) $\forall\, x, y \in (B \times \{C, P\} \times I\!N)^* \,:\, (\gamma_0[x, y]\gamma \wedge \gamma_0[x]\gamma' \wedge \gamma \in S,) \Rightarrow \gamma' \in S,$ (i. e., each state reached on a path into $S,$ must be in $S,$), and

(iv) $\forall\, x \in (B \times \{C, P\} \times I\!N)^* \,:\, (\gamma \in S, \wedge \gamma[x^i]\gamma_i, i \geq 1 \wedge \gamma \leq \gamma_1 \wedge \gamma_1 \in S,) \Rightarrow \forall i \geq 1 \,:\, \gamma_i \in S,$ (i. e., any sequence of transitions which when applied to a state in $S,$ terminates at another state in $S,$ and can be repeated indefinitely without leaving $S,$). ■

Definition (1.3.2.4): Let $\gamma_1 = (bm_1, r_1, fs_1), \gamma_2 = (bm_2, r_2, fs_2) \in$, be states of an FDFD. We say that $\gamma_1 \leq \gamma_2$ if, and only if, the following three conditions hold:

- $\forall b \in B \,:\, bm_1(b) = bm_2(b)$
- $\forall b \in B\, \forall f \in F \,:\, r_1(b)(f) = r_2(b)(f)$
- $\forall f \in F \,:\, fs_1(f) \leq fs_2(f)$, i. e., $fs_1(f)$ is a left factor of $fs_2(f)$. ■

Definition (1.3.2.5): Let $\gamma_0 \in$, be the initial state of an L–RDFD. Let $S,$ be a set of states over , $=$ (*BubbleMode* $\times$ *Read* $\times$ *FlowState*). $(L\text{-}RDFD, \gamma_0, S,)$ is called a *Linear RDFD having a Structured Set of Terminal States* (SSTS–L–RDFD). The *set of firing sequences* of $(L\text{-}RDFD, \gamma_0, S,)$ is $FS(L\text{-}RDFD, \gamma_0, S,) = \{s \mid s \in (B \times \{C, P\} \times I\!N)^* \wedge \gamma_0[s]\gamma \wedge \gamma \in S, \}$. ■

Theorem (1.3.2.6): Every (SSTS–)L–RDFD (with a Structured Set of Terminal States $S,$) with initial state $\gamma_{initial}$ can be simulated by a Linear FIFO Petri Net (with a Structured Set of Terminal Markings $\rho(S,)$) with respect to an isomorphism h.

Proof: Similar to the proof of Theorem (1.3.1.5), we distinguish among four different types of places in P_{FPN}:

(i) $p \in \{f_1, \ldots, f_f\} \wedge Consumable(p)$:

Since $M_{FPN}(p) = fs(p)$ by definition, the contents of each place of the FIFO Petri Net is identical to the contents of the corresponding flow of the L–RDFD. Also, a new value is appended to place p if, and only if, the related value is appended to the corresponding flow. Hence, since flow p is linear, place p is linear, too.

(ii) $p \in \{f_1, \ldots, f_f\} \wedge \neg Consumable(p)$:
Since $M_{FPN}(p) = fs(p)$ by definition, the contents of each place of the FIFO Petri Net is identical to the contents of the corresponding flow of the L–RDFD. A new value is appended to place p if, and only if, one of two possible cases occurs:

(a) The related value is appended to the corresponding flow. Then, since flow p is linear, place p is linear, too.

(b) A value is read from the corresponding persistent flow (but it is not removed from this flow). This relates to removing the head element and appending the new value (which is the same as the value which has been removed) to this place upon firing of a transition of the FIFO Petri Net. Since our mapping from RDFD's to FIFO Petri Nets guarantees that places representing persistent flows contain exactly one token at a time, this new value appended to place p is automatically the head element of this place. Therefore, if in the L–RDFD the word $\ldots a_i^{n_i} \ldots$ occurs as input to flow p, the word $\ldots a_i^{n_{i1}} a_i a_i^{n_{i2}} \ldots$, where $n_{i1} \geq 1, n_i = n_{i1} + n_{i2}$, will occur as input to place p in the FIFO Petri Net. Hence, place p is linear.

(iii) $p \in \{b_{1,idle}, \ldots, b_{b,idle}\}$:
The only value that is appended to place p is I. The input language of p is I^* with initial marking I. Therefore, p is linear.

(iv) $p \in \{b_{1,working:1}, \ldots, b_{1,working:m_1}, \ldots, b_{b,working:1}, \ldots, b_{b,working:m_b}\}$:
The only value that is appended to place p is W. The input language of p is W^* with initial marking $<>$. Therefore, p is linear.

So, since the L–RDFD is linear, i. e., each of its flows is linear, the FIFO Petri Net is linear, too. Now, we still have to show that $\rho(S,)$ is a Structured Set of Terminal Markings with respect to $(FPN, \rho(\gamma_{initial}))$. Since ρ is bijective, $\gamma \in S, \Leftrightarrow \rho(\gamma) \in \rho(S,)$. Since τ is bijective, $s \in FS(L\text{-}RDFD, \gamma_{initial}, S,) \Leftrightarrow \tau(s) \in FS(FPN, \rho(\gamma_{initial}), \rho(S,))$. Therefore, $\rho(S,)$ is a SSTM of the FIFO Petri Net since $S,$ is a SSTS of the L–RDFD. ■

Formally, we can incorporate the notation of a Structured Set of Terminal States $S,$ into the transitions rules (see [LWBL96], [SB96a]) that are allowed between configurations of FDFD's. The modified transition rules now read as follows:

$$\frac{\begin{gathered} bm(b) = idle, \\ Enabled(b)(fs) = true, \\ bm' = [b \mapsto working]bm, \\ (fs', r') \in Consume(b)(fs, r) \\ (bm', r', fs') \in S, \end{gathered}}{(bm, r, fs) \Leftrightarrow\!\rightarrow (bm', r', fs')}$$

and

$$\frac{\begin{gathered} bm(b) = working, \\ bm' = [b \mapsto idle]bm, \\ (fs', r') \in Produce(b)(fs, r) \\ (bm', r', fs') \in S, \end{gathered}}{(bm, r, fs) \Leftrightarrow\!\rightarrow (bm', r', fs')}$$

Of course, it must be decidable whether $(bm', r', fs') \in S,$ holds.

There are two obvious advantages of having a SSTS for FDFD's (and not only for L–RDFD's):

- A computerized evaluation of a given FDFD, for example by using the software described in [Wah95], may be restricted to those states that are of particular interest to the system analyst.
- The introduction of an SSTS is an additional approach to modify the qualitative behavior of an FDFD. For example, consider an FDFD where a communication protocol with erraneous channels has been modeled. Assume we also have been able to identify a set of error states, ES. Then, if we want to analyze a similar communication protocol where no erraneous channels occur, we do not have to modify the FDFD itself, but just have to introduce the SSTS $S, = RS(FDFD, \gamma_{initial}) \Leftrightarrow ES$, such that it is impossible for the system to enter any of the error states.

Corollary (1.3.2.7): The following problems are decidable for (SSTS–)L–RDFD's: TDP, PDP, BP, RP, and QLP.

Proof: All problems are decidable with respect to (SSTM–)LFPN's ([FR88]). We have shown that there exists an isomorphism h between (SSTS–)L–RDFD's and SSTM–LFPN's.

- According to the note following Theorem (1.2.2.5), h preserves TDP, PDP, BP, and RP ([KM82]).
- QLP is decidable for (SSTM–)LFPN's ([FR88]).

  Since τ and ρ are bijective, $\forall (b,a,j) \in (B_{RDFD} \times \{C,P\} \times I\!N)$ $\exists s \in (B_{RDFD} \times \{C,P\} \times I\!N)^*$: $\gamma_{initial}[s,(b,a,j)]$ holds $\Leftrightarrow$ $\forall t \in T_{FPN}$ $\exists \tilde{x} \in T^*_{FPN}$: $M_0(\tilde{x}t >$ holds, where $M_0 = \rho(\gamma_{initial})$, $t = \tau((b,a,j))$, and $\tilde{x} = \tau(s)$. ∎

1.3.3 Topologically Free Choice RDFD's

Our definitions of Topologically Free Choice RDFD's are related to the definitions of Topologically Free Choice FIFO Petri Nets as given in [FC88], summarized in Subsection 1.2.5.

Definition (1.3.3.1): Let $f \in F$ be a flow of an FDFD.
The *output alphabet* AO_f of a flow f is defined as

$$AO_f = \{o \mid \exists b \in B\ \exists j \in I\!N\ :\ (Consume(b))_j = \ldots Head(fs(f)) = o \ldots\}.$$

The *input alphabet* AI_f of a flow f is defined as

$$AI_f = \{i \mid \exists b \in B\ \exists j \in I\!N\ :\ (Produce(b))_j = \ldots Out(i,f,b) \ldots\}.$$

The *alphabet* A_f of a flow f is defined as $A_f = AO_f \cup AI_f$. ∎

The output alphabet AO_f is a subset of *OBJECTS* that might be read from a flow f in accordance with the mapping *Consume*. The input alphabet AI_f is a subset of *OBJECTS* that might be written to a flow f in accordance with the mapping *Produce*. These definitions are only related to the static structure of the FDFD. It is not necessarily required that all *OBJECTS* $a \in A_f$ will actually appear on this flow for any firing sequence or any initial state $\gamma_{initial}$.

Definition (1.3.3.2): The set of flows F of an FDFD with initial state $\gamma_{initial} = (bm_{initial}, r_{initial}, fs_{initial})$ is called *normalized*, if the following two conditions are satisfied:

- each flow $f \in F$ is balanced, i. e., $AI_f = AO_f = A_f$ (the input alphabet is equal to the output alphabet),
- $\forall f \in F\ :\ fs_{initial}(f) \in A^*_f$. ∎

The definition of a normalized FIFO Petri Net requires that each place $p \in P$ is semi–alphabetic, i. e., at most one element of the alphabet A is consumed from p in each step. However, this is already part of our definition of RDFD's which states that for a bubble $b \in B$, the mappings $\mathit{Enabled}(b)$ and $\mathit{Consume}(b)$ only make use of the head element of a flow $f \in \mathit{Inputs}(b)$. Hence, each flow is semi–alphabetic in an RDFD. Actually, it is even alphabetic since a similiar restriction prevents $\mathit{Produce}(b)$ to write more than one element at a time to a flow $f \in \mathit{Outputs}(b)$.

In particular, the restriction to a set of normalized flows is no restriction of the power of RDFD's but guarantees, a priori, that there will never be an object $o \in \mathit{OBJECTS}$ which can not even potentially be removed from a flow $f \in F$, in at least one $\mathit{Consume}(b)$ case. Of course, it must hold that all other flows in this $\mathit{Consume}(b)$ case have the appropriate head element before this object actually can be removed.

In analogy to Hack's definition for free choice Petri Nets we extend this definition for RDFD's:

Definition (1.3.3.3): Let $f \in F$ be a flow of an FDFD and $b \in B$ the bubble where $f \in \mathit{Inputs}(b)$. f is called *free choice* (it satisfies the Hack condition) if, and only if, it fulfills one of two possible conditions:

- A statement of the form "$\ldots \mathit{Head}(\mathit{fs}(f)) \ldots$" occurs only in one single case in $\mathit{Enabled}/\mathit{Consume}$ of bubble b, or
- for all cases in $\mathit{Enabled}/\mathit{Consume}$ of bubble b that contain "$\ldots \mathit{Head}(\mathit{fs}(f)) \ldots$", f is the only flow that is used for this case (throughout the statement we have "$\neg \mathit{IsEmpty}(\mathit{fs}(f)) \wedge \mathit{Head}(\mathit{fs}(f)) = i$" for some i's). ■

The main idea of this definition is to allow only controlled conflict. In general, conflict occurs when several cases in bubble $b \in B$ could potentially read from the same flow $f \in F$. By the definition of free choice RDFD's, if a flow f occurs in several cases in $\mathit{Enabled}/\mathit{Consume}$ of bubble b (potential conflict), then it is the only flow accessed in any of these cases. Therefore, all of the conflicting cases that require "$\mathit{Head}(\mathit{fs}(f)) = i$" are simultaneously activated, or none of them is activated since the flow is empty. This allows the choice (conflict resolution) to be made freely which case is to be selected. It does not depend on the presence of other $\mathit{OBJECTS}$ on other flows.

Definition (1.3.3.4): An RDFD is called an *Extended Topologically Free Choice RDFD* (ETFC–RDFD), if, and only if, the following two conditions are satisfied:

- the set of flows F is normalized, and
- $\forall f \in F : | A_f | > 1 \Rightarrow f$ is free choice (it satisfies the Hack condition). ■

Since the set of flows is normalized, there exists for each flow $f \in F$ of an FDFD and $b \in B$ the bubble where $f \in \mathit{Inputs}(b)$ at least one case in *Enabled/Consume* that can make use of the head element of f, thus potentially go from *idle* to *working*, provided f and, if f occurs only in a single case, all other flows that occur in this case, are not empty.

Unfortunately, our construction of FIFO Petri Nets based on a given EFCT–RDFD fails to provide an EFCT–FIFO Petri Net. The problem is structurally inherited from the definition of the isomorphism h. For each bubble in the RDFD, we introduce additional places in the FIFO Petri Net to store the bubble's working mode and the values that have been read ([SB96a]). The place of the FIFO Petri Net that represents the *idle* working mode of the RDFD causes the problem since it typically is not the only input to serveral transitions of the FIFO Petri Net. Consider the following example:

Example (1.3.3.5): A simple FDFD with only two bubbles A and B connected by a flow f.

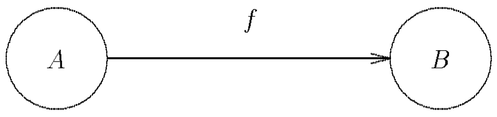

Figure 1.4: EFCT–RDFD.

The mappings *Enabled*, *Consume*, and *Produce* for the FDFD shown in Figure 1.4 are specified as follows:

$$\mathit{Enabled}(A) = \lambda \mathit{fs} \,.\, \mathit{true}$$

$$\begin{aligned} \mathit{Enabled}(B) = \lambda \mathit{fs} \,.\, & \\ & (\neg \mathit{IsEmpty}(f) \wedge \mathit{Head}(\mathit{fs}(f)) = 0) \\ & \vee(\neg \mathit{IsEmpty}(f) \wedge \mathit{Head}(\mathit{fs}(f)) = 1) \end{aligned}$$

$$\mathit{Consume}(A) = \lambda(\mathit{fs}, r) \,.\, \{(\mathit{fs}, r)\}$$

$$\begin{aligned} \mathit{Consume}(B) = \lambda(\mathit{fs}, r) \,.\, & \\ & \{\textbf{if } (\neg \mathit{IsEmpty}(f) \wedge \mathit{Head}(\mathit{fs}(f)) = 0) \\ & \ \textbf{then } \mathit{In}(f, B)(\mathit{fs}, r) \end{aligned}$$

```
        fi,
        if $(\neg IsEmpty(f) \wedge Head(fs(f)) = 1)$
        then $In(f, B)(fs, r)$
        fi
        }
```

$Produce(A) = \lambda(fs, r)\,.$

$\{Out(0, f, A)(fs, r),$

$Out(1, f, A)(fs, r)\}$

$Produce(B) = \lambda(fs, r)\,.\,\{(fs, [B \mapsto \lambda f\,.\,\bot]r)\}$

Initially, flow f is empty. According to [SB96a], the given RDFD transforms into the following marked FIFO Petri Net $FPN = ((P_{FPN}, T_{FPN}, B_{FPN}, F_{FPN}, Q_{FPN}), M_{0,FPN})$:

$P_{FPN} = \{f\} \cup \{A_i, A_{w:1}, B_i, B_{w:1}, B_{w:2}\}$

$T_{FPN} = \{C_{A1}, C_{B1}, C_{B2}\} \cup \{P_{A1}, P_{A2}, P_{B1}, P_{B2}\}$

The initial marking $M_{0,FPN}$ is such that:

$M_{0,FPN}(A_i) = M_{0,FPN}(B_i) = I$

$M_{0,FPN}(A_{w:1}) = M_{0,FPN}(B_{w:1}) = M_{0,FPN}(B_{w:2}) = <>$

$M_{0,FPN}(f) = <>$

B_{FPN}, F_{FPN}, and Q_{FPN} can be gained from Figure 1.5.

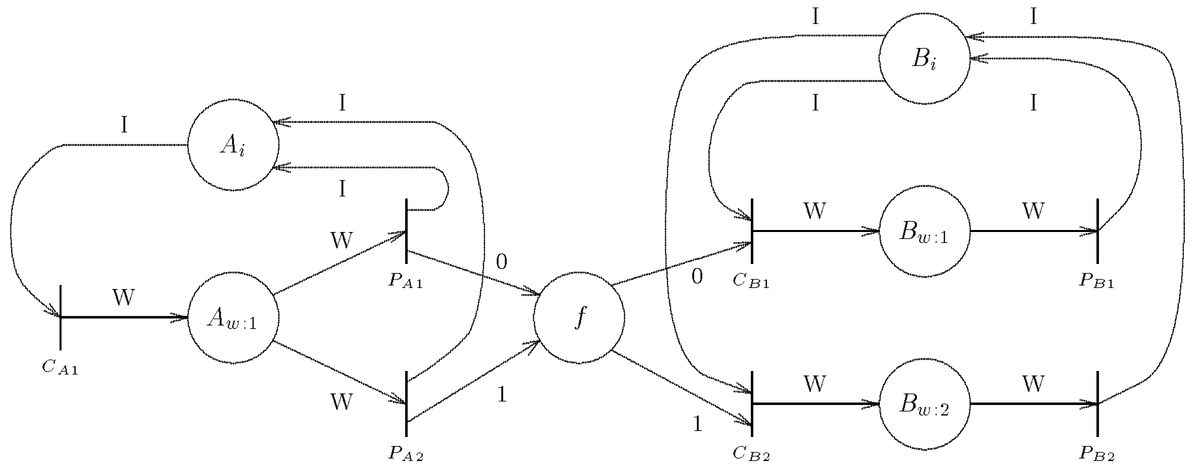

Figure 1.5: FIFO Petri Net.

The resulting FIFO Petri Net is normalized. We have:

$A_f = \{0, 1\}$

$A_{A_i} = A_{B_i} = \{I\}$

$A_{A_{w:1}} = A_{B_{w:1}} = A_{B_{w:2}} = \{W\}$

Each place is semi–alphabetic and the condition for the initial marking $M_{0,FPN}$ is fulfilled. Since $\mid A_f \mid= 2$, we have to verify that f satisfies the Hack condition. Unfortunately, it does not. We have , $(f) = \{C_{B_1}, C_{B_2}\}$ and $\mid$, $(f) \mid= 2 > 1$, but , $^{-1}(C_{B_1}) =$, $^{-1}(C_{B_2}) = \{f, B_i\} \neq \{f\}$. ■

1.4 Summary

The basic idea of this article was not to define completely new subclasses of RDFD's, but to extend known subclasses of FIFO Petri Nets towards RDFD's. Once defined, we have seen that Monogeneous PFF–RDFD's and Linear RDFD's are related to Monogeneous FIFO Petri Nets and Linear FIFO Petri Nets, respectively, through isomorphisms. These isomorphisms maintain solutions of decidability problems, thus allowing us to answer problems such as TDP, PDP, BP, RP, QLP, LP, RegP, and CP for Monogeneous PFF–RDFD's and problems such as TDP, PDP, BP, RP, and QLP for Linear RDFD's, based on methods and algorithms already available for FIFO Petri Nets. Unfortunately, our mapping from RDFD's to FIFO Petri Nets fails for ETFC–RDFD's. We are working on a different homomorphism h' between ETFC–RDFD's and ETFC–FIFO Petri Nets that hopefully will allow us to answer decidability problems for ETFC–RDFD's based on their solution for ETFC–FIFO Petri Nets.

Future work is expected to move in the following directions: It is desirable to identify further subclasses of RDFD's that allow the solution of (some) decidability questions. These new subclasses of RDFD's will also relate to additional subclasses of FIFO Petri Nets. Therefore, it would be reasonable to join research efforts on FDFD's and on FIFO Petri Nets.

So far, there remain several open decidability problems for subclasses of RDFD's, since the related problem is open for the corresponding subclass of FIFO Petri Nets. It is conjectured ([FR88]) that most of these problems are decidable even though no proof or algorithm exists at this time. Further work has to be done to identify which problem is (or is not) decidable for which subclass of RDFD's/FIFO Petri Nets. Another interesting approach would be the extension of a method well–known for Petri Nets — the reduction of the number of places of the Petri Net (e. g., [BR76]) — towards subclasses of RDFD's/FIFO Petri Nets with the intent to solve decidability questions more efficiently.

Finally, we must admit that there was no effort made so far that deals with the complexity and efficiency of the decidability algorithms. Even though many problems have been identified as decidable for particular subclasses of RDFD's, no efficient algorithm has yet been given. It is desireable to determine (lower and upper) bounds for (time and space) complexity of possible algorithms and evaluate given current (and future) algorithms with respect to these bounds.

Acknowledgements

Symanzik's research was partially supported by a German "DAAD–Doktorandenstipendium aus Mitteln des zweiten Hochschulsonderprogramms".

Bibliography

[BR76] G. Berthelot and G. Roucairol. Reduction of Petri–Nets. In A. Mazurkiewicz, editor, *Lecture Notes in Computer Science Vol. 45: Mathematical Foundations of Computer Science 1976: Proceedings, 5th Symposium, Gdańsk, September 1976*, pages 202–207, Springer–Verlag, Berlin, Heidelberg, 1976.

[CF87] A. Choquet and A. Finkel. *Simulation of Linear FIFO Nets by a new Class of Petri Nets.* Universite de Paris–Sud, Centre d'Orsay, Laboratoire de Recherche en Informatique, Bat. 490, 91405 Orsay (France), Rapport de Recherche No. 324, Jan 1987.

[Fan92] J. Fanchon. A FIFO–Net Model for Processes with Asynchronous Communication. In G. Rozenberg, editor, *Lecture Notes in Computer Science Vol. 609: Advances in Petri Nets 1992*, pages 152–178, Springer–Verlag, Berlin, Heidelberg, 1992.

[FC88] A. Finkel and A. Choquet. FIFO Nets Without Order Deadlock. *Acta Informatica*, 25(1):15–36, 1988.

[Fin84] A. Finkel. Petri Nets and Monogeneous FIFO Nets. *Bulletin of the European Association of Theoretical Computer Science*, 23:28–31, 1984.

[Fin86] A. Finkel. *Structuration des Systemes de Transitions — Applications au Controle du Parallelisme par Files FIFO.* These Science, Universite de Paris–Sud, Centre d'Orsay, 1986.

[FM82] A. Finkel and G. Memmi. FIFO Nets: A New Model of Parallel Computation. In A.B. Cremers and H.P. Kriegel, editors, *Lecture Notes in Computer Science Vol. 145: Theoretical Computer Science: 6th GI–Conference, Dortmund, January 1983*, pages 111–121, Springer–Verlag, Berlin, Heidelberg, 1982.

[FR85] M.P. Flé and G. Roucairol. Fair Serializability of Iterated Transactions Using FIFO-Nets. In G. Rozenberg, editor, *Lecture Notes in Computer Science Vol. 188: Advances in Petri Nets 1984*, pages 154–168, Springer–Verlag, Berlin, Heidelberg, 1985.

[FR88] A. Finkel and L. Rosier. A Survey on the Decidability Questions for Classes of FIFO Nets. In G. Rozenberg, editor, *Lecture Notes in Computer Science Vol. 340: Advances in Petri Nets 1988*, pages 106–132, Springer–Verlag, Berlin, Heidelberg, 1988.

[KM82] T. Kasai and R.E. Miller. Homomorphisms between Models of Parallel Computation. *Journal of Computer and System Sciences*, 25:285–331, 1982.

[LWBL96] G.T. Leavens, T. Wahls, A.L. Baker, and K. Lyle. An Operational Semantics of Firing Rules for Structured Analysis Style Data Flow Diagrams. Technical Report 93–28d, Iowa State University, Department of Computer Science, 226 Atanasoff Hall, Ames, Iowa 50011, December 1993, revised, July 1996. Available by anonymous ftp from ftp.cs.iastate.edu or by e–mail from almanac@cs.iastate.edu.

[MF85] G. Memmi and A. Finkel. An Introduction to FIFO Nets — Monogeneous Nets: A Subclass of FIFO Nets. *Theoretical Computer Science*, 35(2–3):191–214, 1985.

[MM81] R. Martin and G. Memmi. Specification and Validation of Sequential Processes Communicating by FIFO Channels. *I.E.E. Conference Publication No. 198: Fourth International Conference on Software Engineering for Telecommunication Switching Systems, Warwick, July 1981*, pages 54–57, 1981.

[Pet81] J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice–Hall, Inc., Englewood Cliffs, New Jersey, 1981.

[Rou87] G. Roucairol. FIFO–Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Lecture Notes in Computer Science Vol. 254: Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, September 1986*, pages 436–459, Springer–Verlag, Berlin, Heidelberg, 1987.

[SB96a] J. Symanzik and A.L. Baker. Formalized Data Flow Diagrams and Their Relation to Other Computational Models. Technical Report 96–20, Iowa State University, Department of Computer Science, 226 Atanasoff Hall, Ames, Iowa 50011, December 1996. Available by anonymous ftp from ftp.cs.iastate.edu or by e–mail from almanac@cs.iastate.edu.

[SB96b] J. Symanzik and A.L. Baker. Non–Atomic Components of Data Flow Diagrams: Stores, Persistent Flows, and Tests for Empty Flows. Technical Report 96–21, Iowa State University, Department of Computer Science, 226 Atanasoff Hall, Ames, Iowa 50011, December 1996. Available by anonymous ftp from ftp.cs.iastate.edu or by e–mail from almanac@cs.iastate.edu.

[Sta83] P.H. Starke. Monogenous FIFO–Nets and Petri–Nets are Equivalent. *Bulletin of the European Association of Theoretical Computer Science*, 21:68–77, 1983.

[Wah95] T. Wahls. *On the Execution of High Level Formal Specifications.* PhD Thesis, Iowa State University, Ames, Iowa, 50011, 1995.