Translation from Workflow Nets to MSVL*

Ya Shi, Zhenhua Duan**, and Cong Tian

ICTT and ISN Lab, Xidian University, Xi'an, China, 710071
y.shi@stu.xidian.edu.cn, zhenhua_duan@126.com,
tico_tools@163.com

Abstract. An automatic translation from Workflow nets (WFNs) to Modeling, Simulation and Verification Language (MSVL) is presented in this paper. As a result, WFNs can be simulated and verified through the well developed supporting tool named MSV for MSVL programs. To do so, annotations are added to WFNs first. Further, translating rules are presented w.r.t regular structures for the translation from Annotated WFNs to MSVL programs. Finally, a tool called PN2MSVL has been implemented for the automatic translation from WFNs to MSVL.

Keywords: Workflow Nets, MSVL, Modeling, Verification, Simulation.

1 Introduction

As a subset of Petri nets [16], Workflow Nets (WFNs) [1] have been widely adopted in modeling business processes. WFNs combine an intuitive graphical formalism with a mathematical sound foundation. The graphical representation is useful in capturing the intuition of a modeler faithfully while the formal foundation allows the verification of a variety of properties [8, 16]. However, as an abstract model, WFNs are not implementable although some tools, e.g. CPN [4], can support the simulation and verification of WFNs.

To implement WFNs, several transformations from WFNs to executable codes in Java, Ada, and BPEL have been investigated. Considering readability, extensibility and efficiency of the generated codes, structured translations [2, 10, 13–15, 17] draw much attention. Within structured translations, behavioral constructs, such as sequences, choices, and loops, are mapped into the corresponding structured statements of programming languages. The challenge is that some complex constructs are difficult to be translated into conventional structured statements directly. In the translations from WFNs to BPEL in [2, 15], some complex constructs are mapped to flow activities with control links to indicate the excepted execution order. These methods are unsuitable for common programming languages because of the usage of flow statements. In [2], manual translation is employed when a complex construct cannot be translated automatically. There are also other approaches [9, 12, 18, 19] through structured graph-oriented

* This research is supported by NSFC Grant Nos. 61133001, 61003078, 61272117, 61272118, and 61202038, National Program on Key Basic Research Project of China (973 Program) Grant No. 2010CB328102, and ISN Lab Grant No. ISN1102001.

** Corresponding author.

L. Groves and J. Sun (Eds.): ICFEM 2013, LNCS 8144, pp. 281–296, 2013.

models, since the structured translation from structured WFNs to programming languages is smooth. However, it is declared in [12] that there exist WFNs without any equivalent structured models. To sum up, there are still lots of difficulties in the structured transformation from WFNs to programming languages. In addition, most of the existing structured translations just aim at the implementation of WFNs without considering verification of correctness of the models.

Modeling, Simulation and Verification Language (MSVL) [6] is an executable subset of Projection Temporal Logic (PTL) [20]. In addition to common statements, eg, *assignment*, *sequence*, *condition* and *loop*, in C, C++, and Java, etc., concurrent statements like *await*, *parallel*, as well as *projection* are also provided in MSVL. As a modeling, simulation and verification language, MSVL can perform as a programming language for simulation, model a concurrent system like PROMELA [11], and verify critical properties of a system through model checking approaches. Meanwhile, supporting tool for simulation, modeling and verification with MSVL has been well developed [6].

Therefore, we are motivated to implement WFNs with MSVL such that WFNs models can be not only implemented but also verified with properties specified by Propositional Projection Temporal Logic formulas (PPTL) [6]. To the end of the translation from WFNs to MSVL programs, annotations are added to WFNs first. Further, translating rules are presented for the translation from Annotated WFNs (AWFNs) to MSVL programs and a tool called PN2MSVL is implemented for the automatic translation from WFNs to MSVL. The merits of the translation presented in this paper are in two folds: (1) the translation is structured and easy to be extended to other general programming languages; (2) when transformed as MSVL programs, critical properties of the original WFN models can be verified via model checking approaches.

The rest of the paper is organized as follows. In Section 2, preliminaries about WFNs and MSVL are presented. AWFNs and regular structures are presented in Section 3, and translating rules and algorithm from WFNs to MSVL are discussed in Section 4. In Section 5, translating tool PN2MSVL as well as case studies are presented. Finally, the conclusion is drawn and the future research directions are pointed out in Section 6.

2 Preliminaries

This section briefly presents the definitions of Workflow nets as well as MSVL.

2.1 Workflow Nets

Definition 1 (Workflow Nets). *A Petri net $N = (P, T, F)$ is a Workflow net if and only if: (1) there is one source place $i \in P$ such that ${}^\bullet i = \emptyset$; (2) there is one sink place $o \in P$ such that $o^\bullet = \emptyset$; (3) the net $\overline{N} = (P, T \cup \{t_N\}, F \cup \{(o, t_N), (t_N, i)\})$, $t_N \notin T$, is strongly connected.*

As usual, a WFN system $\Sigma = (P, T, F, M_0)$ always initially puts only one token into the source place, i.e. $M_0 = \{i\}$. A WFN $N = (P, T, F)$ is *sound* [3] iff (1) $\forall M$, $(\{i\} \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} \{o\})$; (2) $\forall M$, $(\{i\} \xrightarrow{*} M \wedge M \geq \{o\}) \Rightarrow (M = \{o\})$; (3) $\forall t \in T$, $\exists M, M'$, $\{i\} \xrightarrow{*} M \xrightarrow{t} M'$. A net $N = (P, T, F)$ is a free-choice net [5] iff $\forall t_1, t_2 \in T$, if

$^\bullet t_1 \cap {^\bullet t_2} \neq \emptyset$, then $^\bullet t_1 = {^\bullet t_2}$. Given a free-choice net N, a *complete choice* is a maximal subset of transitions with the same input places, and the set of all complete choices is denoted by CC_N.

In general only the models that have been verified will be useful in practice, and soundness is widely accepted as an essential attribute of well designed models. Further, free-choice nets are an important subclass of Petri nets. As a good compromise between expressive power and analyzability, free-choice nets have strong theoretical results and efficient analysis algorithms [5]. With these considerations, in this paper we only deal with sound free-choice WFNs.

2.2 MSVL

MSVL is an executable subset of PTL. Expressions of MSVL are presented below:

$$e ::= n|s|x$$
$$b ::= \texttt{true}|\texttt{false}|e_0 = e_1|\neg b|b_0 \texttt{ and } b_1$$

where n is an integer, s a string, and x a variable. The following are the statements in MSVL:

`Assignment` :	$x_1 <= e$	`Sequential` :	$p_1; p_2$
`Conditional` :	$\texttt{if } b_1 \texttt{ then } \{p_1\}\ else\{p_2\}$	`While` :	$\texttt{while } b_1 \texttt{ do}\{p_1\}$
`Guarded Conditional` :	$(b_1 \rightarrow p_1)[]\ldots[](b_m \rightarrow p_m)$	`Selection` :	$(p_1)\texttt{or}(p_2)$
`Interval Frame` :	$\texttt{frame}(x_1, \ldots, x_n)$	`Parallel` :	$(p_1)\|\|(p_2)$
`Await` :	$\texttt{await}(b_1)$	`Skip` :	`skip`

where e stands for an arbitrary expression, each b_i a boolean expression, each p_i a statement, $i \in \{1, 2, \ldots, m\}$, and each x_j a statement of MSVL, $j \in \{1, 2, \ldots, n\}$. $x_1 <= e$ means that the value of variable x_1 is assigned to the value of expression e_1 and a proposition p_{x_1} combined with x_1, in the mean time, is set to true. The sequential, conditional, and while statements are the same as that of conventional imperative languages. $(b_1 \rightarrow p_1)[]\ldots[](b_m \rightarrow p_m)$ means that if none of the conditions is true, the program will abort; otherwise an arbitrary program p_i with a true guard b_i will be selected for execution. $(p_1)or(p_2)$ means that either p_1 or p_2 is executed. $frame(x_1, \ldots, x_n)$ indicates that for every variable x_j the value of it keeps unchanged over an interval if no assignment to it is encountered. $(p_1)||(p_2)$ means that p_1 and p_2 start simultaneously, execute parallelly, and can end asynchronously. $await(b_1)$ does not change any variable, but waits until the condition b_1 becomes true, at which point it terminates. *skip* specifies an interval of unit length.

Currently, a tool named MSV has been developed for MSVL. MSV can work in three modes: simulation, modeling, and verification. In the simulation mode, an MSVL program is executed with an interpreter; in the modeling mode, the whole state space of the program can be illustrated in terms of Normal Form Graph (NFG) [7]; and in the verification mode, a Propositional Projection Temporal Logic (PPTL) formula is used to specify the desired property of the model, then the unified model checking approach [6] is utilized to check whether the MSVL model can satisfy the PPTL formula.

3 Annotated Workflow Nets and Regular Structures

For the fluent translation from WFNs to MSVL, we introduce Annotated Workflow Nets (AWFNs) and regular structures in AWFNs first.

3.1 Annotated Workflow Nets

Definition 2. *An* Annotated WFN (AWFN) *is a tuple* (P, T, F, G, L)*, where* $N = (P, T, F)$ *is a WFN,* G *and* L *are condition and statement annotations on a transition* $t \in T$*, respectively. For each transition* $t \in T$*, the condition annotation* $G(t)$ *of* t *is a boolean expression, and the statement annotation* $L(t)$ *of* t *is a statement in MSVL.*

In an AWFN, a transition t is *enabled* iff for each place $p \in {}^{\bullet}t$, $M(p) > 0$ and $G(t)$ is true. The statement annotation $L(t)$ will be executed while t is fired. AWFNs serve as an intermediary in the translation from WFNs to MSVL.

Given a sound free-choice WFN $N = (P, T, F)$, an AWFN $AN = (P, T, F, G, L)$ can be obtained by:

- $G(t) = \texttt{true}$ for each $t \in T$,
- $L(t) = (p_1 \texttt{ <= } 0) \texttt{ and } \ldots \texttt{ and } (p_m \texttt{ <= } 0) \texttt{ and skip}; (q_1 \texttt{ <= } 1) \texttt{ and } \ldots \texttt{ and } (q_n \texttt{ <= } 1) \texttt{ and skip}$, where ${}^{\bullet}t = \{p_1, \ldots, p_m\}$ and $t^{\bullet} = \{q_1, \ldots, q_n\}$.

Intuitively, for each place p of N, a variable p is utilized to record the number of tokens in it, and for each transition t, a statement $L(t)$ is employed to describe the effect of the transition t on places.

3.2 Regular Structures

Let $N = (P, T, F)$ be a Petri net. For two nodes $x, y \in P \cup T$, there exists a path from x to y iff $(x, y) \in F^+$. N is *standard* iff there exist two distinct nodes, denoted by st_N and end_N (means starting and ending node, respectively), such that every node appears on a path from st_N to end_N. Let $X \subseteq P \cup T$ be a set of nodes. The *projection* of N to X is a subnet $N|_X = (P|_X, T|_X, F|_X)$ of N, where $P|_X = P \cap X$, $T|_X = T \cap X$, $F|_X = F \cap ((P|_X \times T|_X) \cup (T|_X \times P|_X))$, and for each node $x \in X$, ${}^{\bullet}x|_X = {}^{\bullet}x \cap X$ and $x^{\bullet}|_X = x^{\bullet} \cap X$. $N|_X$ is *autonomous* iff $N|_X$ is standard, $({}^{\bullet}st_{N|_X} \cup end_{N|_X}{}^{\bullet}) \cap X = \emptyset$, and ${}^{\bullet}X' \cup X'^{\bullet} = X$, where $X' = X \backslash \{st_{N|_X}, end_{N|_X}\}$.

Now given a sound free-choice AWFN $N = (P, T, F, G, T)$ as well as a set of nodes $X \subseteq P \cup T$. Some regular structures in AWFNs are defined as follows:

1. Redundant Place Structure: $N|_X$ is a *redundant place structure (RPS)* iff $X = \{p_1, p_2\} \subset P$, ${}^{\bullet}p_1 = {}^{\bullet}p_2$, and $p_1^{\bullet} = p_2^{\bullet}$. Fig. 1 (a) shows a RPS, where $M(q_1) = M(q_2)$ for any reachable marking M.
2. Sequence Structure: $N|_X$ is a *sequence structure (SS)* iff $T|_X = \{t_1, t_2\}$, $P|_X = \{p\} = t_1{}^{\bullet} = {}^{\bullet}t_2$, ${}^{\bullet}p = \{t_1\}$, and $p^{\bullet} = \{t_2\}$. Obviously, a sequence structure is a standard net. Fig. 2 (a) shows an SS, where t_1 and t_2 always occur sequentially.
3. Explicit Choice Structure: $N|_X$ is an *explicit choice structure (ECS)* iff $P|_X = P_1 \cup P_2$, $P_1 \cap P_2 = \emptyset$, $|P_1| > 0$, $|P_2| > 0$, $|T|_X| > 1$, and $\forall t \in T|_X$, ${}^{\bullet}t = P_1$, $t^{\bullet} = P_2$. Fig. 3 (a) shows an ECS, where the occurrences of t_1 and t_2 are mutually exclusive.

4. Simple Loop Structure: $N|_X$ is a *simple loop structure (SLS)* iff $T|_X = \{t\}$ and $P|_X = {}^\bullet t = t^\bullet$. Fig. 4 (a) shows a SLS, where t_0 can occur repeatedly.
5. Complex Loop Structure: $N|_X$ is a *complex loop structure (CLS)* iff $N|_X$ is connected, $|T|_X| > 1$, $\forall t \in T|_X$, $|{}^\bullet t| = |t^\bullet| = 1$, and $\forall p \in P|_X$, $|{}^\bullet p|_X| = |p^\bullet|_X| = 1$. Since N is a free-choice WFN, we have $\forall p \in P|_X$, $\forall t \in p^\bullet$, ${}^\bullet t = \{p\}$. The dashed rectangle in Fig. 5 (a) shows a CLS, where once one transition in the loop occurs, all transitions in the loop will occur iteratively.
6. Complex Choice Structure: $N|_X$ is a *complex choice structure (CCS)* iff $N|_X$ is a standard net, $st_{N|_X}, end_{N|_X} \in P|_X$, $\forall t \in T|_X$, $|t^\bullet| = |{}^\bullet t| = 1$, $\forall p \in P|_X \backslash \{st_{N|_X}, end_{N|_X}\}$, $|p^\bullet|_X| = |{}^\bullet p|_X| = 1$, $|st_{N|_X}{}^\bullet|_X| = |{}^\bullet end_{N|_X}|_X| = 2$, ${}^\bullet st_{N|_X} \neq \emptyset$, $end_{N|_X}{}^\bullet \neq \emptyset$, and $\forall t \in end_{N|_X}{}^\bullet$, $|{}^\bullet t| = 1$. Similarly, since N is a free-choice net, for all $p \in P|_X$ and $t \in p^\bullet$, we have ${}^\bullet t = \{p\}$. The dashed rectangle in Fig. 6 (a) presents a CCS where four cases: t_1 and t_2, or t_3 and t_4 fire together, or only t_1 or t_4 fires.
7. Concurrent Structure: $N|_X$ is a *concurrent structure (CoS)* iff $N|_X$ is autonomous, $st_{N|_X}, end_{N|_X} \in T|_X$, $\forall p \in P|_X$, $|{}^\bullet p| = |p^\bullet| = 1$, and $T|_X = {}^\bullet(P|_X) \cup (P|_X)^\bullet$. A *minimal CoS* is a CoS where no smaller CoS can be contained. Fig. 7 (a) presents a CoS, where after the occurrence of t_0, all the rest transitions in it will occur.
8. Irregular Structure: $N|_X$ is an *irregular structure (IS)* iff X is autonomous and containing no regular structures defined above. A *minimal IS* is an IS where no smaller IS can be contained. Since N is a sound net, once a transition in an IS $N|_X$ occurs, the occurrence of $N|_X$ will eventually end with no tokens left. Fig. 8 (a) shows an IS where once t_0 or t_1 occurs, t_3 will eventually occur.

4 Translation from AWFNs to MSVL

With respect to each regular structure, a translating rule is given for the transformation from AWFNs to the eventually MSVL programs.

4.1 Rule RRP: Removal of Redundant Places

For each reachable marking of a sound free-choice AWFN, the numbers of tokens contained in all places of a redundant place structure are the same. Thus, control-flow of the structure will not be changed in case any one of the places is removed. Accordingly, for a redundant place structure $N|_X$, we remove one of the two places in X.

Formally, for a redundant place structure $N|_X$ of a sound free-choice AWFN $N = (P, T, F, G, L)$, by Rule RRP, a new sound free-choice AWFN $N' = (P', T, F', G, L)$ is obtained, where $P' = P \setminus \{p\}$, $p \in X$, and $F' = (F \cap ((P' \times T) \cup (T \times P')))$.

As an example, for the redundant place structure illustrated in Fig. 1 (a), by Rule RRP, place q_2 is removed as depicted in Fig. 1 (b).

4.2 Rule FSS: Folding Sequence Structures

In a sequence structure, two transitions always occur sequentially. We fold them into one transition with statement annotation being the sequential composition of the two statement annotations on the two transitions.

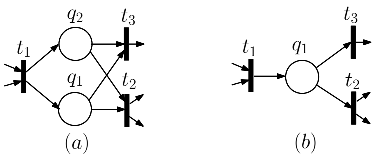

Fig. 1. Removal of Redundant Places

Formally, for a sequence structure $N|_X$ of a sound free-choice AWFN $N = (P, T, F, G, L)$, by RULE FSS, a new sound free-choice AWFN $N' = (P', T', F', G', L')$ is generated, where

- $P' = P \backslash X$;
- $T' = (T \backslash X) \cup \{d_X\}$, $d_X \notin P \cup T$;
- $F' = (F \cap ((P' \times T') \cup (T' \times P'))) \cup ({}^{\bullet}st_{N|_X} \times \{d_X\}) \cup (\{d_X\} \times end_{N|_X}{}^{\bullet})$;
- $\forall t \in T' \backslash \{d_X\}$, $G'(t) = G(t)$, and $G'(d_X) = G(st_{N|_X})$;
- $\forall t \in T' \backslash \{d_X\}$, $L'(t) = L(t)$, and $L'(d_X) = L(st_{N|_X}); L(end_{N|_X})$.

For instance, by RULE FSS, the sequence structure in Fig. 2 (a) can be transformed as d_X in Fig. 2 (b) where $G'(d_X) = G(t_1)$, and $L'(d_X) = L(t_1); L(t_2)$.

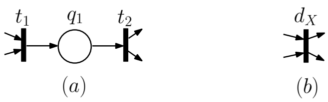

Fig. 2. Folding Sequence Structures

4.3 RULE FECS: Folding Explicit Choice Structures

In an explicit choice structure, the occurrences of transitions are mutually exclusive. The transitions are folded into one transition with a guarded conditional statement as the statement annotation.

Formally, for an explicit choice structure $N|_X$ of a sound free-choice AWFN $N = (P, T, F, G, L)$, by RULE FECS, a new sound free-choice AWFN $N' = (P, T', F', G', L')$ is produced, where

- $T' = (T \backslash X) \cup \{d_X\}$, $d_X \notin P \cup T$;
- $F' = (F \cap ((P \times T') \cup (T' \times P))) \cup ({}^{\bullet}(T|_X) \times \{d_X\}) \cup (\{d_X\} \times T|_X{}^{\bullet})$;
- $\forall t \in T' \backslash \{d_X\}$, $G'(t) = G(t)$, and $G'(d_X) = \mathtt{OR}_{d \in T|_X} G(d)$;
- $\forall t \in T' \backslash \{d_X\}$, $L'(t) = L(t)$, and $L'(d_X) = (G(t_0) \rightarrow L(t_0))[] \ldots [] G(t_n) \rightarrow L(t_n))$, where $T|_X = \{t_0, \ldots, t_n\}$.

For example, by RULE FECS, the explicit choice structure in Fig. 3 (a) is folded as the transition d_X in Fig. 3 (b) where $G'(d_X) = G(t_1)$ $\mathtt{or}$ $G(t_2)$ and $L'(d_X) = (G(t_1) \rightarrow L(t_1))[](G(t_2) \rightarrow L(t_2))$.

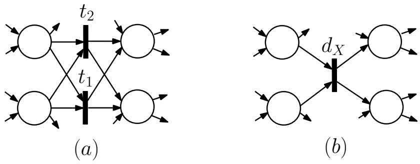

Fig. 3. Folding Explicit Choice Structures

4.4 Rule FSLS: Folding Simple Loop Structures

In a simple loop structure, the transition could occur repeatedly. We fold this structure into one transition with a while statement as the statement annotation.

Formally, for a simple loop structure $N|_X$ of a sound free-choice AWFN $N = (P, T, F, G, L)$, by Rule FSLS, a new sound free-choice AWFN $N' = (P', T', F', G', L')$ is obtained, where

- $P' = (P \backslash X) \cup P_0 \cup P_1$, where $P_0 \cap P_1 = \emptyset$, $(P_0 \cup P_1) \cap (P \cup T) = \emptyset$, and there exit bijective mappings μ_0, $P|_X \rightarrow P_0$, and μ_1, $P|_X \rightarrow P_1$;
- $T' = (T \backslash X) \cup \{d_X\}$, $d_X \notin P \cup T \cup P_0 \cup P_1$;
- $F' = (F \cap ((P' \times T') \cup (T' \times P'))) \cup (P_0 \times \{d_X\}) \cup (\{d_X\} \times P_1) \cup F_0 \cup F_1$, where $F_0 = \{(x, \mu_0(p)) | \forall p \in P|_X, \forall (x, p) \in F|_X\}$ and $F_1 = \{(\mu_1(p), x) | \forall p \in P|_X, \forall (p, x) \in F|_X\}$;
- $\forall t \in T' \backslash \{d_X\}$, $G'(t) = G(t)$, and $G'(d_X) = \texttt{true}$;
- $\forall t \in T' \backslash \{d_X\}$, $L'(t) = L(t)$ and $L'(d_X) = over_X <= 0$ and skip; while$(over_X = 0$ and $G(t))\{(over_X <= 1$ and skip) or $(L(t))\}$, where $T|_X = \{t\}$.

For instance, by Rule FSLS, the simple loop structure in Fig. 4 (a) is folded as transition d_X in Fig. 4 (b), where $G'(d_X) = \texttt{true}$ and $L'(d_X) = over_X <= 0$ and skip; while $(over_X = 0$ and $G(t_0))\{(over_X <= 1$ and skip) or $(L(t_0))\}$.

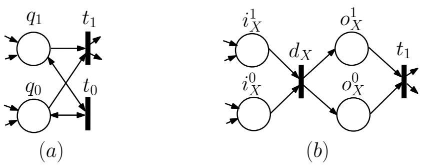

Fig. 4. Folding Simple Loop Structures

4.5 Rule FCLS: Folding Complex Loop Structures

In a complex loop structure, whenever a transition occurs, all transitions in it will occur iteratively. Accordingly, we fold a complex loop structure as a transition with a while statement as the statement annotation. Further, since it may have more than one entries or exits, two auxiliary variables $LPNext_X$ and $LPOut_X$ are utilized to mark the actual entry and exit in one occurrence.

Let $N|_X$ be a complex loop structure of a sound free-choice AWFN $N = (P, T, F, G, L)$. $entry_X$ and $exit_X$ denotes the set of entries and exits of $N|_X$, respectively. Formally, $entry_X = \{p | \forall p \in P|_X, {}^{\bullet}p \backslash X \neq \emptyset\}$ and $exit_X = \{p | \forall p \in P|_X, p^{\bullet} \backslash X \neq \emptyset\}$. Once a transition in $N|_X$ with an entry of $N|_X$ as input place occurs, the occurrence of $N|_X$ starts. While in case a transition outside of $N|_X$ with an exit of $N|_X$ as input place occurs, the occurrence of $N|_X$ ends. Therefore, when an exit q of $N|_X$ gets the token, the occurrence of $N|_X$ can end if one transition d, outside of $N|_X$, with q as input is enabled, i.e. $\mathtt{OR}_{t \in p^\bullet \setminus X} G(t)$ holds. For convenience, we use $EG_X(p)$ to indicate $\mathtt{OR}_{t \in p^\bullet \setminus X} G(t)$ for each place p in $exit_X$. To formally present the details of the transformation, notations below are defined first.

$LN_X \subseteq entry_X \times entry_X$ is a binary relation such that for any $(p_0, p_1) \in LN_X$, there exits a path in $N|_X$ from p_0 to p_1 containing no other entries of $N|_X$. $N|_Y$ is a segment of N_X iff $Y \subseteq X$, $N|_Y$ is a standard net, and $(st_{N|_Y}, end_{N|_Y}) \in LN_X$. Now we extend relation LN_X to segments, namely segments LN_X (SLN_X). Let $N|_{Y_1}$ and $N|_{Y_2}$ be two distinct segments. $(N|_{Y_1}, N|_{Y_2}) \in SLN_X$ iff $(st_{N|_{Y_1}}, st_{N|_{Y_2}}) \in LN_X$. For instance, in the complex loop structure $N|_X$ illustrated in the dashed rectangle of Fig. 5 (a), $entry_X = \{q_1, q_2\}$, $exit_X = \{q_1, q_3\}$, $EG_X(q_1) = G(t_5)$, $EG_X(q_3) = G(t_7)$, and $LN_X = \{(q_1, q_2), (q_2, q_1)\}$. There are two segments $N|_{X_1}$ and $N|_{X_2}$ of loop structure $N|_X$, where $X_1 = \{q_1, q_2, t_0\}$, $X_2 = \{q_2, q_3, q_1, t_1, t_2\}$, and $(N|_{X_1}, N|_{X_2}), (N|_{X_2}, N|_{X_1}) \in SLN_X$.

To construct the statement annotation of the transition formed by Rule FCLS, each segment of a complex structure is mapped into a conditional statement, and then a while statement is constructed such that an arbitrary conditional statement among them serves as the starting, and all conditional statements occur w.r.t to the relations of segments described in SLN_X.

Specifically, given a complex loop structure $N|_X$ of a sound free-choice AWFN $N = (P, T, F, G, L)$, a new sound free-choice AWFN $N' = (P', T', F', G', L')$ is obtained by Rule FCLS, where

- $P' = (P \setminus X) \cup \{i_X, o_X\}$ with $\{i_X, o_X\} \cap (P \cup T) = \emptyset$;
- $T' = (T \setminus X) \cup \{d_X\}$, where $d_X \notin P \cup T \cup \{i_X, o_X\}$;
- $F' = (F \cap ((P' \times T') \cup (T' \times P'))) \cup ((^\bullet entry_X \setminus X) \times \{i_X\}) \cup \{(i_X, d_X), (d_X, o_X)\} \cup (\{o_X\} \times (exit_X{}^\bullet \setminus X))$;
- $G'(t) = G(t)$, for each $t \in T' \setminus (exit_X{}^\bullet \cup \{d_X\})$; $G'(t) = G(t)$ $\mathtt{and}$ $LPOut_X =$ "p", for each $t \in exit_X{}^\bullet \setminus X$, where $^\bullet t = \{p\}$; and $G'(d_X) = \mathtt{true}$;
- $L'(t) = L(t)$, for each $t \in T' \setminus (^\bullet entry_X \cup exit_X{}^\bullet \cup \{d_X\})$, $L'(t) = L(t); LPNext_X <=$ "p" $\mathtt{and}$ $\mathtt{skip}$, for each $t \in {}^\bullet entry_X \setminus X$, where $t^\bullet \cap X = \{p\}$, $L'(t) = LPOut_X <=$ "$NULL$" $\mathtt{and}$ $\mathtt{skip}; L(t)$, for each $t \in exit_X{}^\bullet \setminus X$, and $L'(d_X)$ is shown in Statement Annotation 1.

In Statement Annotation 1, for each segment $N|_Y$ of $N|_X$ there exists a conditional statement from line 3 to 16. The condition in line 3 is used to check whether the starting place $st_{N|_Y}$ of $N|_Y$ gets a token. If the condition does not hold, the conditional statement is skipped. Otherwise, $LPNext_X$ is assigned as "$end_{N|_Y}$", which makes the conditional statement of the subsequent segment executable. Thus, the statement annotation of transition t_Y^1 between $st_{N|_Y}$ and r_Y^1 is executed subsequently. Here r_Y^1 is the first exit in $N|_Y$. When r_Y^1 gets a token, a guarded conditional occurs. In case the assignment statement in line 5 is executed, the occurrence of $N|_X$ ends and the actual exit of this occurrence is marked by $LPOut_X$. Otherwise, the occurrence of $N|_X$ goes on. t_Y^2 is the transition between r_Y^1 and the successive exit r_Y^2. t_Y^3 is the transition between r_Y^2 and the subsequent exit. r_Y^k is the last exit in $N|_Y$ and t_Y^v is the transition between r_Y^k and $end_{N|_Y}$. All conditional statements occur w.r.t to the relations of segments described in SLN_X.

Statement Annotation 1.

```
while(¬(LPNext_X = "NULL")) do{
  ...
  if(LPNext_X = "st_{N|Y}") then {
    LPNext_X <= "end_{N|Y}" and skip;  L(t_Y^1);
    (EG_X(r_Y^1)  → LPNext_X <= "NULL" and LPOut_X <= "r_Y^1" and skip)[]
    (G(t_Y^2)  → L(t_Y^2);
      (EG_X(r_Y^2)  → LPNext_X <= "NULL" and LPOut_X <= "r_Y^2" and skip)[]
      (G(t_Y^3)  → L(t_Y^3);
          ...
            (EG_X(r_Y^k)  →
              LPNext_X <= "NULL" and LPOut_X <= "r_Y^k" and skip
            )[](G(t_Y^v)  →  L(t_Y^v))
          ...
      )
    )
  }
  ...
}
```

By Rule FCLS, the complex loop structure shown in Fig. 5 (a) is transformed as d_X in Fig. 5 (b), where

$G'(t_5) = G(t_5)$ and $LPOut_X = \text{“}q_1\text{”}, L'(t_5) = LPOut_X <= \text{“}NULL\text{”}$ and skip; $L(t_5)$,
$G'(t_7) = G(t_7)$ and $LPOut_X = \text{“}q_3\text{”}, L'(t_7) = LPOut_X <= \text{“}NULL\text{”}$ and skip; $L(t_7)$,
$L'(t_3) = L(t_3); LPNext_X <= \text{“}q_1\text{”}$ and skip,
$L'(t_4) = L(t_4); LPNext_X <= \text{“}q_1\text{”}$ and skip,
$L'(t_6) = L(t_6); LPNext_X <= \text{“}q_2\text{”}$ and skip,
$G'(d_X) =$ true,
$L'(d_X) =$ while($\neg(LPNext_X = \text{“}NULL\text{”})$) do{
  if($LPNext_X = \text{“}q_1\text{”}$) then{$LPNext_X <= \text{“}q_2\text{”}$ and skip;
    ($G(t_5) \rightarrow LPNext_X <= \text{“}NULL\text{”}$ and $LPOut_X <= \text{“}q_1\text{”}$ and skip)[]($G(t_0) \rightarrow L(t_0)$)}
  if($LPNext_X = \text{“}q_2\text{”}$) then{$LPNext_X <= \text{“}q_1\text{”}$ and skip; $L(t_1)$;
    ($G(t_7) \rightarrow LPNext_X <= \text{“}NULL\text{”}$ and $LPOut_X <= \text{“}q_3\text{”}$ and skip)[]($G(t_2) \rightarrow L(t_2)$)}}

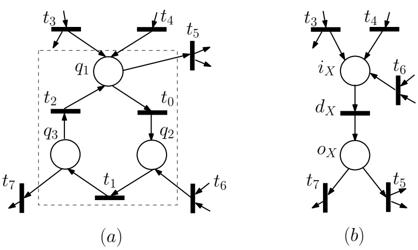

Fig. 5. Folding Complex Loop Structures

4.6 Rule FCCS: Folding Complex Choice Structure

We fold a complex choice structure $N|_X$ as a transition with a conditional statement as the statement annotation. Similar to complex loop structures, there may be more than one entries and exits in a complex choice structure. Thus, two auxiliary variables

$CCNext_X$ and $CCOut_X$ are utilized to mark the position where an occurrence of $N|_X$ starts and stops, respectively.

Let $N|_X$ be a complex choice of a sound free-choice AWFN $N = (P, T, F, G, L)$. The definitions of $entry_X$, $exit_X$, EG_X, and LN_X are the same as that of complex loop structures. $N|_Y$ is a *segment* of $N|_X$ iff $Y \subset X$, $N|_Y$ is a standard net, and (1) $(st_{N|_Y}, end_{N|_Y}) \in LN_X$, or (2) $Y \cap entry_X = st_{N|_X}$ and $end_{N|_Y} = end_{N|_X}$. Note that among the segments of $N|_X$, there are two starting at $st_{N|_X}$ as well as two ending at $end_{N|_X}$. The definition of SLN_X is the same as that of complex loop structures. For instance, for the complex choice structure $N|_X$ as shown in the dashed rectangle of Fig. 6 (a), $entry_X = \{q_0, q_2\}$, $exit_X = \{q_1, q_3\}$, $EG_X(q_1) = G(t_7)$, $EG_X(q_3) = G(t_5)$, $LN_X = \{(q_0, q_2), (q_2, q_3), (q_0, q_3)\}$. There are three segments $N|_{X_1}$, $N|_{X_2}$, and $N|_{X_3}$ of $N|_X$, where $X_1 = \{q_0, q_2, t_3\}$, $X_2 = \{q_2, q_3, t_4\}$, $X_3 = \{q_0, q_1, q_3, t_1, t_2\}$, and $SLN_X = \{(N|_{X_1}, N|_{X_2})\}$.

According to the definition of complex choice structure $N|_X$, there are two different pathes from $st_{N|_X}$ to $end_{N|_X}$. To construct the statement annotation of the transition formed by Rule FCCS, (1) for each path from $st_{N|_X}$ to $end_{N|_X}$, a sequential statement is constructed with conditional statements obtained from segments in it; (2) in each sequential statement, all conditional statements occur w.r.t to the relations of segments described in SLN_X; and (3) a new guarded conditional statement with these two sequential statements as branches is constructed.

Specifically, given a complex choice structure $N|_X$ of a sound free-choice AWFN $N = (P, T, F, G, L)$, a new sound free-choice AWFN $N' = (P', T', F', G', L')$ is obtained by Rule FCCS, where

- $P' = (P \backslash X) \cup \{i_X, o_X\}$, and $\{i_X, o_X\} \cap (P \cup T) = \emptyset$;
- $T' = (T \backslash X) \cup \{d_X\}$, where $d_X \notin P \cup T \cup \{i_X, o_X\}$;
- $F' = (F \cap ((P' \times T') \cup (T' \times P'))) \cup ((^\bullet entry_X \backslash X) \times \{i_X\}) \cup \{(i_X, d_X), (d_X, o_X)\} \cup (\{o_X\} \times (exit_X{}^\bullet \backslash X))$;
- $G'(t) = G(t)$, for each $t \in T' \backslash (exit_X{}^\bullet \cup \{d_X\})$; $G'(t) = G(t)$ $\texttt{and}$ $CCOut_X =$ "p", for each $t \in exit_X{}^\bullet \backslash X$, where $^\bullet t = \{p\}$; and $G'(d_X) = \texttt{true}$;
- $L'(t) = L(t)$, for each $t \in T' \backslash (^\bullet entry_X \cup exit_X{}^\bullet \cup \{d_X\})$; $L'(t) = L(t); CCNext_X <=$ "p" $\texttt{and skip}$, for each $\forall t \in {}^\bullet entry_X \backslash X$, where $t^\bullet \cap X = \{p\}$; $L'(t) = CCOut_X <=$ "$NULL$" $\texttt{and skip}; L(t)$, for each $t \in exit_X{}^\bullet \backslash X$; and $L'(d_X) =$

```
1 : (CCNext_X = "a_0" and (G(t_0) or EG_X(a_0)) or CCNext_X = "a_1" or ... or CCNext_X = "a_m" →
2 :    ...)[]
3 : (CCNext_X = "b_0" and (G(t_1) or EG_X(b_0)) or CCNext_X = "b_1" or ... or CCNext_X = "b_n" →
4 :    ...)
```

In $L'(d_X)$, each a_i and b_j, $i \in \{0, 1, \ldots, m\}$, $j \in \{0, 1, \ldots, n\}$, is an entry of the complex choice structure appearing in the two pathes from $st_{N|_X}$ to $end_{N|_X}$. Note that $a_0 = b_0 = st_{N|_X}$. t_0 and t_1 are the first transitions in the two pathes from $st_{N|_X}$ to $end_{N|_X}$. The two conditions are used to select a path from $st_{N|_X}$ to $end_{N|_X}$ for executing. For each segment of a path, in line 2 (or 4) there exists a conditional statement similar to the one from line 3 to 16 in Statement Annotation 1 with $LPNext_X$ and $LPOut_X$ being replaced by $CCNext_X$ and $CCOut_X$. All conditional statements in line 2 and 4 occur w.r.t to the relations of segments described in SLN_X. For segments $N|_Y$ ending at $end_{N|_X}$, the statement $CCNext_X <=$ "$end_{N|_Y}$" in line 4 of Statement Annotation 1 is replaced by $CCNext_X <=$ "$NULL$", and the codes in line 10 and 12 of Statement Annotation 1 are removed.

By RULE FCCS, the complex choice structure in Fig. 6 (a) is transformed as transition d_X in Fig. 6 (b), where

$G'(t_5) = G(t_5)$ and $CCOut_X$ = "q_3", $L'(t_5) = CCOut_X$ <= "$NULL$" and skip; $L(t_5)$,
$G'(t_7) = G(t_7)$ and $CCOut_X$ = "q_1", $L'(t_7) = CCOut_X$ <= "$NULL$" and skip; $L(t_7)$,
$L'(t_0) = L(t_0)$; $CCNext_X$ <= "q_0" and skip,
$L'(t_6) = L(t_6)$; $CCNext_X$ <= "q_2" and skip,
$G'(d_X)$ = true,
$L'(d_X)$ = ($CCNext_X$ = "q_0" and $G(t_1) \rightarrow$
if($CCNext_X$ = "q_0") then {$CCNext_X$ <= "$NULL$" and skip; $L(t_1)$;
($G(t_7) \rightarrow CCNext_X$ <= "$NULL$" and $CCOut_X$ <= "q_1" and skip)[]
($G(t_2) \rightarrow L(t_2)$; $CCNext_X$ <= "$NULL$" and $CCOut_X$ <= "q_3" and skip)})[]
($CCNext_X$ = "q_0" and $G(t_3)$ or $CCNext_X$ = "q_2" $\rightarrow$
if($CCNext_X$ = "q_0") then {$CCNext_X$ <= "q_2" and skip; $L(t_3)$};
if($CCNext_X$ = "q_2") then {$L(t_4)$; $CCNext_X$ <= "$NULL$" and $CCOut_X$ <= "q_3" and skip})

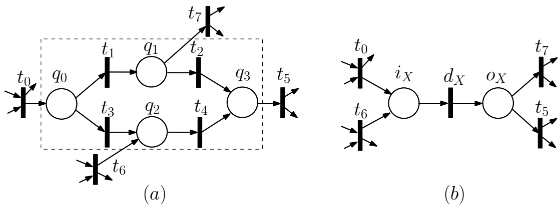

Fig. 6. Folding Complex Choice Structures

4.7 RULE FCoS: Folding Concurrent Structure

In a concurrent structure $N|_X$, once a transition $st_{N|_X}$ occurs, all other transitions will occur. Thus, it is folded into a transition with a parallel statement as the statement annotation. Furthermore, auxiliary variables and await statements are needed to control the occurrence order of transitions in it.

Given a sound free-choice AWFN $N = (P, T, F, G, L)$ with a minimal concurrent structure $N|_X$, for each $t \in T|_X$, $B_X(t)$ is used to indicate $L(t); (B_X(t_1)|| \ldots ||B_X(t_m))$, where $\{t_1, \ldots, t_m\} = t^{\bullet\bullet} \cap (T|_X \backslash \{end_{N|_X}\})$. We first update $L(t)$ as await($vp_1 = 1$ and ... and $vp_m = 1$); vp_1 <= 0 and ... and vp_m <= 0 and skip; $L(t)$, for each transition $t \in T|_X \backslash \{st_{N|_X}, end_{N|_X}\}$ with $|{}^\bullet t| > 1$, where $\{p_1, \ldots, p_m\} = {}^\bullet t \backslash \{p\}$, $p \in {}^\bullet t$, and remove input arcs of t excepting for (p, t). Secondly, for each transition $t \in T|_X$ with $\exists q \in t^\bullet$, $q^\bullet = \emptyset$, we update $L(t)$ as $L(t)$; vq_1 <= 1 and ... and vq_n <= 1 and skip, where $\{q_1, \ldots, q_n\} = \{q | \forall q \in t^\bullet, q^\bullet = \emptyset\}$. Then it has $\forall t \in T|_X \backslash \{st_{N|_X}, end_{N|_X}\}$, $|{}^\bullet t| = 1$. Finally, we fold $N|_X$ and generate a new sound free-choice AWFN $N' = (P', T', F', G', L')$, where

- $P' = (P \backslash X) \cup \{i_X, o_X\}$, $\{i_X, o_X\} \cap (P \cup T) = \emptyset$;
- $T' = (T \backslash (X \backslash \{st_{N|_X}, end_{N|_X}\})) \cup \{d_X\}$, $d_X \notin P \cup T \cup \{i_X, o_X\}$;
- $F' = (F \cap ((P' \times T') \cup (T' \times P'))) \cup \{(st_{N|_X}, i_X), (i_X, d_X), (d_X, o_X), (o_X, end_{N|_X})\}$;
- $\forall t \in T' \backslash \{d_X\}$, $G'(t) = G(t)$, and $G'(d_X)$ = true;
- $\forall t \in T \backslash \{d_X\}$, $L'(t) = L(t)$, and $L'(d_X)$ = frame($vr_1, \ldots, vr_i$) and vr_1 <= 0 and... and vr_i <= 0 and(($B_X(t_1)$))||...||($B_X(t_n)$))). Note that $\{t_1, \ldots, t_n\} \subseteq T|_X$ are the transitions that still have input places formed by output places of $st_{N|_X}$ after the first step. $vr_1, \ldots, vr_i$ are the auxiliary variables added in the rule.

By Rule FCoS, the concurrent structure in Fig. 7 (a) is transformed as transition d_X in Fig. 7 (b), where $G'(d_X) = \texttt{true}$ and $L'(d_X) = frame(vq_4)$ and $vq_4 <= 0$ and $((L(t_1); (L(t_3) || (\texttt{await}(vq_4 = 1); vq_4 <= 0$ and $\texttt{skip}; L(t_4)))) || (L(t_2); vq_4 <= 1$ and $\texttt{skip}))$.

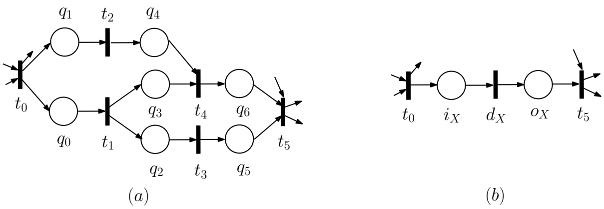

Fig. 7. Folding Concurrent Structures

4.8 Rule FIS: Folding Irregular Structures

In an irregular structure $N|_X$, once a transition occurs, the occurrence of $N|_X$ will eventually ends with no tokens left. Although the control-flow of an irregular structure is complicated, it could still be folded as a transition with a parallel statement as the statement annotation, where the occurring order of the transitions is organized by auxiliary variables and await statements.

Let $N|_X$ be an irregular structure in a sound free-choice AWFN $N = (P, T, F, G, L)$. We define $stp_X = \{st_{N|_X}\} \cap P$, $endp_X = \{end_{N|_X}\} \cap P$, $stt_X = \{st_{N|_X}\} \cap T$, and $endt_X = \{end_{N|_X}\} \cap T$. Let $N|_X$ be a minimal irregular structure of N. We first add an auxiliary transition t_{over} to N such that $t_{over}^\bullet = \emptyset$, $L(t_{over}) = over_X <= 1$ and $\texttt{skip}$, ${}^\bullet t_{over} = {}^\bullet end_{N|_X} \cap X$ and $G(t_{over}) = G(end_{N|_X})$ if $endt_X \neq \emptyset$, ${}^\bullet t_{over} = \{end_{N|_X}\}$ and $G(t_{over}) = \texttt{OR}_{t \in end_{N|_X}{}^\bullet \setminus X} G(t)$ otherwise. Accordingly, a new sound free-choice AWFN $N' = (P', T', F', G', L')$ is obtained where,

- $P' = (P \setminus X) \cup \{i_X, o_X\}$, $\{i_X, o_X\} \cap (P \cup T \cup \{t_{over}\}) = \emptyset$,
- $T' = (T \setminus ((X \cup \{t_{over}\}) \setminus (stt_X \cup endt_X))) \cup \{d_X\}$, $d_X \notin P \cup T \cup \{t_{over}, i_X, o_X\}$,
- $F' = (F \cap ((P' \times T') \cup (T' \times P'))) \cup \{(i_X, d_X), (d_X, o_X)\} \cup F_i \cup F_o$, where $F_i = ({}^\bullet stp_X \times \{i_X\}) \cup (\{i_X\} \times (stp_X{}^\bullet \setminus X))$, if $stp_X \neq \emptyset$, otherwise $F_i = stt_X \times \{i_X\}$; $F_o = (({}^\bullet endp_X \setminus X) \times \{o_X\}) \cup (\{o_X\} \times endp_X{}^\bullet)$, if $endp_X \neq \emptyset$, otherwise $F_o = \{o_X\} \times endt_X$;
- $G'(t) = G(t)$, for each $t \in T' \setminus \{d_X\}$; $G'(d_X) = \texttt{true}$, if $stp_X = \emptyset$, otherwise, $G'(d_X) = \texttt{OR}_{d \in stp_X{}^\bullet | X} G(d)$;
- $\forall t \in T' \setminus \{d_X\}$, $L'(t) = L(t)$, and $L'(d_X) =$

```
...
pro D() = {
  while(over_X = 0){await(vr_1 = 1 and ... and vr_i = 1 or over_X = 1);
    if(over_X = 0) then{vr_1 <= 0 and ... and vr_i <= 0 and skip;
      (G(t_1) → L(t_1); vs_1 <= 1 and ... and vs_{j_1} <= 1 and skip)[]
      ...
      [](G(t_k) → L(t_k); vs_k <= 1 and ... and vs_{j_k} <= 1 and skip)}}};
...
define start_X() = {vp_{j_1} <= 1 and ... and vp_{j_k} <= 1 and skip};
define end_X() = {over_X <= 0 and skip};
frame(vp_1, ..., vp_n, over_X) and vp_1 <= 0 and ... and vp_n <= 0 and over_X <= 0 and
(start_X(); (D^1_X()|| ... ||D^h_X()); end_X())
```

In $L'(d_X)$, for each place $p \in P|_X$, an auxiliary variable vp is used to record the tokens in it. And $st^{\bullet}_{N|_X} = \{p_{j_1}, \ldots, p_{j_k}\}$, if $stt_X \neq \emptyset$, otherwise $k = 1$ and $p_{j_1} = st_{N|_X}$. For each complete choice $\{t_1, \ldots, t_k\} = D \in CC_{N|_X} = \{D^1_X, \ldots, D^h_X\}$, there exits a process $D()$ containing a while statement where transitions in D are executed repeatedly. In a complete choice D, all transitions have $\{r_1, \ldots, r_i\}$ as input. Thus, all of the transitions have to wait until $N|_X$ ends or each common input place gets a token. Once one of the transitions occurs, all tokens in the input are consumed, and each output place gets a token.

By Rule FIS, the irregular structure in Fig. 8 (a) is transformed as a transition in Fig. 8 (b), where $G'(d_X) = \texttt{true}$ and $L'(d_X) =$

```
pro D_X^1() = {
  while(over_X = 0){await(vq_1 = 1 and vq_2 = 1 or over_X = 1);
    if(over_X = 0) then{vq_1 <= 0 and vq_2 <= 0 and skip;(G(t_3) → over_X <= 1 and skip)}}};
pro D_X^2() = {
  while(over_X = 0){await(vq_3 = 1 or over_X = 1);
    if(over_X = 0) then{vq_3 <= 0 and skip; (G(t_2) → L(t_2); vq_1 <= 1 and skip)}}};
pro D_X^3() = {
  while(over_X = 0){await(vq_0 = 1 or over_X = 1);
    if(over_X = 0) then{vq_0 <= 0 and skip;
      (G(t_0) → L(t_0); vq_1 <= 1 and vq_2 <= 1 and skip)[]
      (G(t_1) → L(t_1); vq_2 <= 1 and vq_3 <= 1 and skip)}}};
define start_X() = {vq_0 <= 1 and skip};
define end_X() = {over_X <= 0 and skip};
frame(vq_3, vq_2, vq_1, vq_0, over_X) and vq_3 <= 0 and vq_2 <= 0 and vq_1 <= 0 and vq_0 <= 0
and over_X <= 0 and (start_X(); (D_X^1()||D_X^2()||D_X^3()); end_X())
```

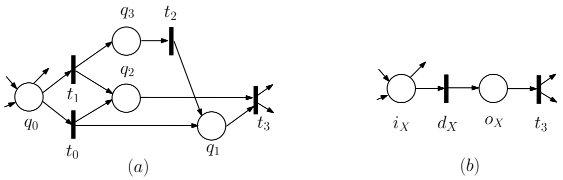

Fig. 8. Folding Irregular Structures

4.9 Translation Algorithm

Based on the translating rules, Algorithm PN2MSVL is presented for the translation from sound free-choice WFNs to MSVL programs.

In the MSVL program generated by Algorithm PN2MSVL, for each place $p_h \in P = \{p_1, \ldots, p_a\}$, a variable p_h is introduced in the final MSVL program. Especially, p_1 specifies the source place of N. $v_1, \ldots, v_b$ are the extra variables introduced by translating rules where $v^1_i, \ldots, v^j_i$ (or $v^1_s, \ldots, v^k_s$) are all integer (string) variables among them. For each transition t of T there exits a definition statement, where ${}^{\bullet}t = \{p^1_t, \ldots p^m_t\}$ and $t^{\bullet} = \{q^1_t, \ldots, q^n_t\}$.

Algorithm PN2MSVL:
Translation from sound free-choice WFNs to MSVL programs

Input: A sound free-choice WFN $N = (P, T, F)$;
Output: A MSVL program;

Translate N to the corresponding AWFN $AN = (P, T, F, G, L)$;
while $|T| > 1$
  if AN has a redundant place structure **then** Apply RULE RRP to AN; continue;
  if AN has a sequence structure **then** Apply RULE FSS to AN; continue;
  if AN has an explicit choice structure **then** Apply RULE FECS to AN; continue;
  if AN has a simple loop structure **then** Apply RULE FSLS to AN; continue;
  if AN has a complex loop structure **then** Apply RULE FCLS to AN; continue;
  if AN has a complex choice structure **then** Apply RULE FCCS to AN; continue;
  if AN has a concurrent structure **then** Apply RULE FCoS to AN; continue;
  if AN has an irregular structure **then** Apply RULE FIS to AN; continue;
Add initialization, frame and definition statements to $L(t)$; /* Let $T = \{d\}$ */
return the final MSVL program shown as follows.
  frame($p_1, \ldots, p_a, v_1, \ldots, v_b$) and p_1 <= 1 and p_2 <= 0 and ... and p_a <= 0 and
  v_i^1 <= 0 and ... and v_i^j <= 0 and v_s^1 <= "*NULL*" and ... and v_s^k <= "*NULL*" and (
   define t() = {p_t^1 <= 0 and ... and p_t^m <= 0 and skip;
          q_t^1 <= 1 and ... and q_t^n <= 1 and skip};
   ...
   $L(d)$)

5 Experiments

We have realized Algorithm PN2MSVL as a tool named PN2MSVL (`http://ictt.xidian.edu.cn/toolkit/`). PN2MVSL gets a WFN in .g format that can be produced by Workcraft [21] and outputs an MSVL program.

For the sound free-choice WFN in Fig. 9, a MSVL program can be obtained by PN2MSVL as below.

frame($p9, p8, p7, p6, p5, p4, p3, p2, p1, p0, LPNext0, LPOut0$) and $p0$ <= 1 and $p9$ <= 0
and $p8$ <= 0 and $p7$ <= 0 and $p6$ <= 0 and $p5$ <= 0 and $p4$ <= 0 and $p3$ <= 0 and $p2$ <= 0
and $p1$ <= 0 and $LPNext0$ <= "*NULL*" and $LPOut0$ <= "*NULL*" and (
 define $t8$() = {$p8$ <= 0 and $p6$ <= 0 and skip; $p9$ <= 1 and skip};
 ...
 $t0$();
 frame($vp7$) and $vp7$ <= 0 and (
  (await($vp7$ = 1); $vp7$ <= 0 and skip; $t6$())||
  (
   (true → $t5$(); $LPNext0$ <= "$p4$" and skip)[](true → $t1$(); $LPNext0$ <= "$p3$" and skip);
   while(¬($LPNext0$ = "*NULL*")){
    if($LPNext0$ = "$p3$")then{$LPNext0$ <= "$p4$" and skip; $t2$()}
    if($LPNext0$ = "$p4$")then{$LPNext0$ <= "$p3$" and skip; $t3$();
     (true → $LPNext0$ <= "*NULL*" and $LPOut0$ <= "$p5$" and skip)[](true → $t7$())}};
   $LPOut0$ <= "*NULL*" and skip; $t4$(); $vp7$ <= 1 and skip));
 $t8$())

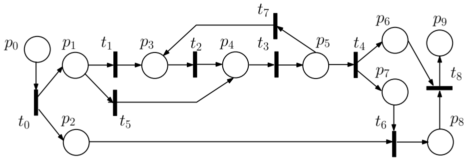

Fig. 9. A Workflow net

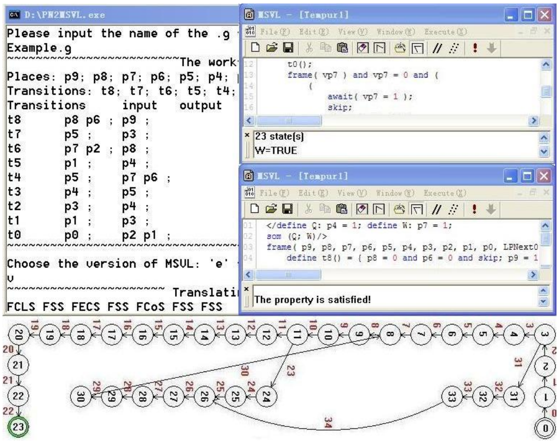

Fig. 10. Results in MSV

When implementing the MSVL program above within MSV, simulation result can be obtained as depicted in the upper right of Fig. 10, where $W = \texttt{true}$ means a successful execution of this program. Now we specify the desired property of the model in PPTL:

$$\Diamond(Q; W)$$

where Q and W means $p4 = 1$ and $p7 = 1$, respectively. The intuition of the formula is that if $p4 = 1$ holds sometimes, $p7 = 1$ will holds eventually. The verification result is shown in the lower right of Fig. 10. In addition, the model of the MSVL program can also be explored as illustrated at the bottom of Fig. 10.

6 Conclusion

An automatic translation from WFNs to MSVL is presented in this paper. The translation is structured and easy to be extended to other general programming languages. In the near future, we are going to further prove the completeness as well as the soundness of the translation. Also, we will try to improve readability of the generated MSVL programs by exploring more regular structures in AWFNs. As applications, we will translate some big systems modeled by WFNs to MSVL and verify the correctness of these systems with MSV.

References

1. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
2. van der Aalst, W.M.P., Lassen, K.B.: Translating unstructured workflow processes to readable bpel: Theory and implementation. Information & Software Technology 50(3), 131–159 (2008)

3. van der Aalst, W.M.P.: Structural characterizations of sound workflow nets. Computing Science Reports 96/23, Eindhoven University of Technology (1996)
4. CPN, http://cpntools.org/
5. Desel, J., Esparza, J.: Free choice Petri nets. Cambridge tracts in theoretical computer science, vol. 40. Cambridge University Press (1995)
6. Duan, Z., Tian, C.: A unified model checking approach with projection temporal logic. In: Liu, S., Araki, K. (eds.) ICFEM 2008. LNCS, vol. 5256, pp. 167–186. Springer, Heidelberg (2008)
7. Duan, Z., Tian, C., Zhang, L.: A decision procedure for propositional projection temporal logic with infinite models. Acta Informatica 45(1), 43–78 (2008)
8. Esparza, J., Heljanko, K.: Implementing ltl model checking with net unfoldings. In: Dwyer, M.B. (ed.) SPIN 2001. LNCS, vol. 2057, pp. 37–56. Springer, Heidelberg (2001)
9. Hauser, R.: Analysis and transformation of behavioral models containing overlapped patterns. Journal of Object Technology 9(3), 105–124 (2010)
10. Hauser, R.: Automatic transformation from graphical process models to executable code. Eidgenössische Technische Hochschule Zürich (2010)
11. Holzmann, G.J.: Design and validation of protocols: A tutorial. Computer networks and ISDN systems 25(9), 981–1017 (1993)
12. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J.: On structured workflow modeling. In: Wangler, B., Bergman, L.D. (eds.) CAiSE 2000. LNCS, vol. 1789, pp. 431–445. Springer, Heidelberg (2000)
13. Kleine, J., Reisig, W.: Transformation von offenen Workflow-Netzen zu abstrakten WS-BPEL-Prozessen (in German). Ph.D. thesis, Humboldt-Universität zu Berlin, Berlin (2007)
14. Lassen, K.B., Tjell, S.: Translating colored control flow nets into readable java via annotated java workflow nets. In: Jensen, K. (ed.) 8th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, pp. 127–146. Aarhus University, Aarhus (2007)
15. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract bpel processes. In: Kühne, T., Reisig, W., Steimann, F. (eds.) Modellierung 2008. Lecture Notes in Informatics, vol. 127, pp. 57–72. Gesellschaft für Informatik, Bonn (2008)
16. Murata, T.: Petri nets: properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (1989)
17. Ouyang, C., Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Mendling, J.: From business process models to process-oriented software systems. ACM Transactions on Software Engineering and Methodology 19(1) (2009)
18. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Information Systems 37(6), 518–538 (2012)
19. Polyvyanyy, A., García-Bañuelos, L., Fahland, D., Weske, M.: Maximal structuring of acyclic process models. The Computing Research Repository abs/1108.2384 (2011)
20. Tian, C., Duan, Z.: Propositional projection temporal logic, buchi automata and ω-regular expressions. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 47–58. Springer, Heidelberg (2008)
21. Workcraft, http://www.workcraft.org/wiki/