ソフトウェア工学 83−20
(1992. 2. 7)

MENDELS ZONE における並行プログラムの階層的調整

内平直志　荒見美香子　本位田真一
(株) 東芝 システム・ソフトウェア技術研究所

本論文では、並行プログラム合成／再利用の新しいアプローチとして、「プログラム調整 (Program Adjustment)」を提案する。プログラム調整とは、与えられた制約を満たすように並行プログラムを自動的に修正する方法である。ここで、並行プログラムのモデルは有限状態プロセスであり、制約条件はタイミングに関するものに限定し時相命題論理で与えられる。このとき、プログラム調整は、対象プログラムが制約を満たすように制御する調整プロセス (Arbiter process) を生成、付加することで実現できる。また、対象プログラムが大規模な場合は、調整プロセスの合成に膨大な計算コストが必要になる。そこで、これを回避するために階層的調整も提案する。我々は、本階層的調整法を採用した並列計算機 Multi-PSI 上の並行プログラム開発支援環境 MENDELS ZONE を開発した。合成されたプログラムは最終的に KL1 プログラムに変換され、Multi-PSI 上で実行できる。

Compositional Adjustment of Concurrent Processes to Satisfy Temporal Logic Constraints in MENDELS ZONE

Naoshi Uchihira　Mikako Arami　Shinichi Honiden

Systems & Software Engineering Laboratory, TOSHIBA Corporation

70, Yanagi-cho, Saiwai-ku, Kawasaki-shi, Kanagawa, 210 JAPAN

E-mail: uchi@ssel.toshiba.co.jp

In this paper, we examine "program adjustment", a practical approach to the automatic programming and program synthesis for concurrent programs, which automatically reforms a roughly-made program to satisfy given constraints. The model of concurrent programs used is the finite state process, and program adjustment to satisfy temporal logic constraints is formalized as synthesis of an arbiter process which controls a target process (a roughly-made program). Compositional adjustment is also proposed for large-scale compound target processes, using process equivalence theory. We have developed a computer-aided programming environment on Multi-PSI, called MENDELS ZONE, that adopts this compositional adjustment. Adjusted programs can be compiled into KL1 programs and executed in MENDELS ZONE.

1. INTRODUCTION

As practical parallel and distributed computer systems gradually spread in the industry, there is an increasing demand for programmers who design concurrent programs. Since it is not easy for ordinary programmers to produce correct and efficient concurrent programs, several kinds of computer-aided concurrent programming environments are necessary, including tools for verification, debugging, performance evaluation, and synthesis of correct and efficient programs. MENDELS ZONE [Uchihira87, Honiden89, Uchihira90a] is a computer-aided concurrent programming environment that has been developed to make the difficult task of concurrent programming easier, especially for the Parallel Inference Machine Multi-PSI and its kernel language KL1. This paper focuses on the program synthesis feature of MENDELS ZONE.

Automatic program synthesis from some formal specification is not practical for the following reasons:
• It is not easy for ordinary programmers to write complete formal specifications.
• Automatic synthesis requires huge computing costs to produce large-scale programs.
• Synthesized programs may be inefficient.
For example, some works [Manna&Wolper84, Clarke&Emerson82] about concurrent program synthesis from temporal logic specification are very suggestive, but they can not go beyond toy program synthesis. More promising approach is the stepwise refinement which constructs (efficiently) executable programs from formal specifications through a number of provable correct development steps [de Bakker89]. However, it still has difficulties to specify a complete formal specification, and has a great gap from actual programming.

Therefore, we propose another approach "program adjustment" in place of automatic synthesis and refinement. Program adjustment means to reform a roughly-made program automatically to satisfy given constraints. Here, we consider only timing constraints for concurrent programs that can be specified by temporal logic. In this context, "a roughly-made program" is defined as a program which may be incomplete in its timing. The main idea of program adjustment is that a concurrent program may eventually satisfy some kinds of timing constraints by eliminating harmful nondeterministic alternatives (i.e., partially serializing a concurrent program). This program adjustment is practical for the following reasons:
• It is not very difficult for ordinary programmers to produce a roughly-made concurrent program, which satisfies at least functional requirements. A more difficult task is to design and debug the timing of programs.
• Many bugs derive from harmful nondeterministic alternatives.
• It is easy for ordinary programmers to write timing constraints, such as deadlock-free and starvation-free constraints.
• A roughly-made program can be intended to be efficient by a programmer.

In this paper, a concurrent program is modeled with the finite state process, which resembles the transition system in CCS and the finite automaton. A program is compositionally constructed from finite state processes with the composition operator. In the case of a finite state process, program adjustment means to adjust a roughly-made process to satisfy given constraints by adding an arbiter process which is synchronized with and controls the roughly-made process. When a target program becomes large, the arbiter synthesis may cause computing cost explosion. Therefore, we propose compositional adjustment, in which local arbiters are synthesized in each composition step. In each step, the reduction of the finite state process, based on process equivalence theory, can ease computing cost explosion. Here, we introduce a new process equivalence relation to manipulate liveness properties, because a traditional bisimulation equivalence of CCS can not. This compositional adjustment has been implemented in MENDELS ZONE.

The remainder of the paper is organized as follows. Section 2 defines Finite State Processes (FSP) and their equivalence relation and composition operator. Compositional adjustment of FSP is described in Section 3. An overview of MENDELS ZONE is briefly shown and its compositional adjustment is explained in Section 4. Finally, Section 5 shows a simple and nontrivial example of program adjustment, followed by the conclusion in Section 6.

2. FINITE STATE PROCESSES

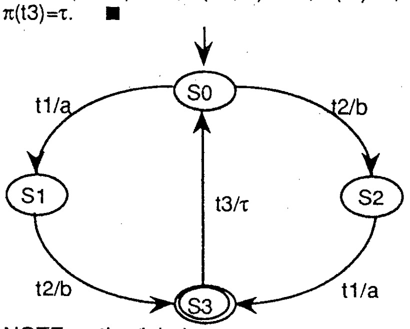
The basic model for concurrent programs is the finite state process [Kanellakis & Smolka90], which can specify the finite state transition system with liveness conditions. First, we define a Finite State Process (FSP) and an equivalence relation for FSPs. Then, several operators (composition, relabelling, and reduction) on FSPs are introduced and their properties are shown.

2.1 FINITE STATE PROCESSES

[Definition 1] (Finite State Process)
A Finite State Process (FSP) is a seventuple $P=(S,A,L,\delta,\pi,s_0,F)$, where:
• S is a finite set of states,
• A is a finite set of actions,
• L is a finite set of synchronization labels,
• δ: $S \times A \rightarrow S \cup \{\perp\}$ is a deterministic transition function (here, $\delta(s,t)=\perp$ means action $t \in A$ is disabled in state $s \in S$),
• π: $A \rightarrow (L \cup \{\tau\})$ is a labelling function, (here, τ means an invisible internal action),

- $s_0 \in S$ is an initial state, and
- $F \subset S$ is a set of designated states. ■

[Example]
P= ({s0,s1,s2,s3}, {t1,t2,t3}, {a,b}, δ, π, s_0, {s3}) where δ(s0,t1) = s1, δ(s0, t2) = s2, δ(s1,t2) = s3, δ(s2,t1)= s3, δ(s3,t3)= s0, π(t1)=a, π(t2)=b, π(t3)=τ. ■



NOTE: action/label,
a double circle means a designated state.

Fig.1 Finite State Process

To begin with, several notations are introduced. Let X be a set. The set of all finite sequences over X, with an empty sequence ε (without ε), is denoted by X^* (X^+, respectively), and the set of all infinite sequences over X is denoted by X^ω. ω means "infinitely many". X^∞ is defined by $X^\infty \equiv X^* \cup X^\omega$. For $\theta \in X^\infty$, $\theta(i)$ means the i-th element in sequence θ, $\theta|_k$ means the subsequence $\theta(1)\theta(2)\ldots\theta(k)$ of θ, and $|\theta|$ is the length of θ. Let $P=(S,A,L,\delta,\pi,s_0,F)$ be an FSP. A transition function can be extended such that $\delta : S \times A^* \rightarrow S \cup \{\perp\}$, i.e., $\delta(s,\theta a) = \delta(\delta(s,\theta),a)$. Note, $\delta(s,\varepsilon) = s$. Since a transition function is deterministic, a current state can be uniquely determined from an initial state and an action sequence. We call an action sequence a *behavior*. Similarly, we can extend a labelling function such that π: $A^* \rightarrow (L \cup \{\tau\})^*$. In addition, $\pi^\wedge(\theta)$ is defined as the sequence gained by deleting all occurrences of τ from $\pi(\theta)$. A set of reachable states from state s in P is defined as $R_P(s) \equiv \{ s' \mid \exists\theta \in A^*.\ s' = \delta(s,\theta) \}$ and $R_P{+}(s) \equiv \{ s' \mid \exists\theta \in A^+.\ s' = \delta(s,\theta) \}$. Also, a set of all possible action sequences (label sequences) of P is defined as $L(P) \equiv \{ \theta \in A^* \mid \delta(s_0,\theta) \neq \perp \}$ ($L_\pi(P) \equiv \{ \pi^\wedge(\theta) \in L^* \mid \theta \in L(P) \}$, respectively). Since interest is in the infinite behavior of FSP, we introduce a set of infinite action sequences $L_\omega(P) \subset (A^\omega \cup A^*\{\Delta\}^\omega)$ where Δ means deadlock:

$L_\omega(P) \equiv \{ \theta \in A^\omega \mid 1 \le \forall k.\delta(s_0,\theta|_k) \neq \perp \} \cup \{ \theta \in A^*\{\Delta\}^\omega \mid \exists k.1 \le \forall i \le k.\delta(s_0,\theta|_i) \neq \perp$ and $\forall a \in A.\ \delta(\delta(s_0,\theta|_k),a) = \perp$ and $\theta(j) = \Delta$ for $\forall j > k \}$

Note that if $\theta \in L(P)$ is a deadlock sequence (i.e., an inevitably finite sequence), then θ is represented as $\theta\Delta^\omega \in L_\omega(P)$. Finally, $L_\omega^{fair}(P) \subset L_\omega(P)$ is defined as $L_\omega^{fair}(P) \equiv \{ \theta \mid \theta \in L_\omega(P)$ under the *fairness condition* $\}$ where the *fairness condition* means whenever a behavior θ infinitely often passes through some state s, every action a enabled at s must appear infinitely often on θ (i.e., if $s = \delta(s_0,\theta|_i)$ for infinitely many i and $\delta(s,a) \neq \perp$, then $s = \delta(s_0,\theta|_j)$ and $\theta(j+1) = a$ for infinitely many j). Finally, L(P)/L is introduced by definition: $L(P)/L \equiv \{ \theta' \mid \exists\theta \in L(P).\forall i.(\theta'(i) = \varepsilon$ if $\theta(i) \in L$, otherwise $\theta'(i) = \theta(i))\}$ Intuitively, L(P)/L consists of a set of behaviors of P in which all elements of L are deleted.

FSP is a transition system with liveness conditions. In FSP, liveness conditions are represented by designated nodes that indicate satisfiable behavior of FSP as follows:

[Definition 2] (Satisfiable Behavior)
Let $P=(S,A,L,\delta,\pi,s_0,F)$ be an FSP. $\theta \in A^\omega$ is a satisfiable behavior, if $\delta(s_0,\theta|_k) \in F$ for infinitely many $k \ge 1$. $L_b(P) \subset A^\omega$ is a set of all satisfiable behaviors on P. ■

Note that a satisfiable behavior corresponds to an accepting run of ω-automaton.

[Definition 3] (Completeness of FSP)
Let $P=(S,A,L,\delta,\pi,s_0,F)$ be an FSP. P is *complete* if $\forall s \in R_P(s_0).\ \exists s' \in R_P{}^+(s)$ and $s' \in F$. ■

A state $s \in R_P(s_0)$, having no path to designated nodes from s, is called an unsatisfiable state. A behavior reaching to an unsatisfiable state is called an *inevitably unsatisfiable behavior.*

[Lemma 1]
If FSP P is complete,

then $L_\omega^{fair}(P) \subset L_b(P)$. ■

This lemma means that if P is complete, then a random transition over P leads to a satisfiable behavior.

2.2 EQUIVALENCE OF FINITE STATE PROCESSES

We now introduce the notion of $\pi\tau\omega$-bisimulation equivalence that is an extension of Milner's weak bisimulation equivalence [Milner89]. $\pi\tau\omega$-bisimulation equivalence has been originally developed for compositional verification [Uchihira90b]. In this paper, it is used to reduce a FSP to a smaller and equivalent one in compositional adjustment.

[Definition 4] ($\tau\omega$-divergence)
Let $P=(S,A,L,\delta,\pi,s_0,F)$ be an FSP. $s \in S$ is *$\tau\omega$-divergent* ($s\uparrow$) if $\forall n>0.\ \exists s' \in S.\ \exists\theta \in A^*.\ |\theta| = n$, $\pi^\wedge(\theta) = \varepsilon$ and $s' = \delta(s,\theta)$. ■

[Definition 5] ($\pi\tau\omega$-bisimulation Equivalence)
Let $P1=(S1,A1,L1,\delta 1,\pi 1,s_{01},F1)$ and $P2 = (S2,A2,L2,\delta 2,\pi 2,s_{02},F2)$ be FSPs. P1 and P2 are *$\pi\tau\omega$-bisimulation equivalent* ($P1 =_{\pi\tau\omega} P2$), if there is a binary relation $R \subset S1\times S2$, such that $(s_{01},s_{02})\in R$, and $\forall s1 \in S1.\ \forall s2 \in S2.\ (s_1,s_2)\in R \Leftrightarrow$
(1) $s_1 \in F1$ iff $s_2 \in F2$,
(2) $s_1\uparrow$ iff $s_2\uparrow$,
(3) $\forall t_1 \in A_1.\ \forall s_1' \in S_1.$ (if $s1' = \delta_1(s_1,t_1)$ then $\exists\theta \in A_2^*.\ \exists s_2' \in S_2.\ \pi 1^\wedge(t_1)=\pi 2^\wedge(\theta)$, $s2' = \delta_2(s_2,\theta)$, and $(s_1',s_2')\in R$,
(4) $\forall t_2 \in A_2.\ \forall s_2' \in S_2.$ (if $s2' = \delta_2(s_2,t_2)$ then $\exists\theta \in A_1^*.\ \exists s_1' \in S_1.\ \pi 2^\wedge(t_2)=\pi 1^\wedge(\theta)$, $s_1' = \delta_1(s_1,\theta)$, and $(s_1',s_2')\in R$. ■

$\pi\tau\omega$-bisimulation is extended so that it can discriminate designated states and divergence, which can not be discriminated by the weak bisimulation. The following lemma is derived from these discrimination abilities.

[Lemma 2]
If P1 is complete and $P1=_{\pi\tau\omega}P2$, then P2 is also complete. ■

[Definition 6] (Reduction)
For a given FSP $P = (S,A,L,\delta,\pi,s_0,F)$, a reduction of P, $red(P) = (Sr,Ar,Lr,\delta r,\pi r,sr_0,Fr)$, is an FSP such that $P =_{\pi\tau\omega} red(P)$ and $|Sr|\leq|S|$. ■

The smallest red(P) is constructed effectively by the relational coarsest partitioning algorithm [Paige & Tarjan87, Kanellakis & Smolka90] such that all states of P that are $\pi\tau\omega$-bisimilar to each other are brought together into a single state of red(P).

2.3 OPERATORS ON FINITE STATE PROCESSES

Concurrent programs are constructed as a composition of several FSPs that are synchronized with each other. The composition and relabelling operators for FSPs are introduced and their important properties (substitutivity and reflectivity) are shown.

[Definition 7] (Composition Operator)
For $P1=(S1,A1,L1,\delta 1,\pi 1,s1_0,F1)$ and $P2=(S2,A2,L2,\delta 2,\pi 2,s2_0,F2)$, a composition $P = P1|P2$ is defined as follows:
$B=(S1\times S2\times\{0,1\}^2, (A1\cup\{idle\}) \times (A2\cup\{idle\}), L1\cup L2,\delta,\pi,(s1_0,s2_0,0,0),F')$, where
$\delta: (S1\times S2\times\{0,1\}^2)\times(A1\cup\{idle\})\times(A2\cup\{idle\}) \rightarrow S1\times S2\times\{0,1\}^2$ such that
$\delta((s1,s2,f1,f2),(a1,a2))=$
- $(\delta 1(s1,a1),\delta 2(s2,a2),f1',f2')$ where $fi'=1$ if $\delta i(si,ai)\in Fi$, otherwise $fi'=0$ (for $i=1,2$), when $\pi 1(a1)=\pi 2(a2)\neq\tau$, and $f1=f2=1$,
- $(\delta 1(s1,a1),\delta 2(s2,a2),f1',f2')$ where $fi'=1$ if $\delta i(si,ai)\in Fi \vee fi=1$, otherwise $fi'=0$ (for $i=1,2$), when $\pi 1(a1)=\pi 2(a2)\neq\tau$, and $(f1=0\vee f2=0)$,
- $(\delta 1(s1,a1),s2,f1',0)$ where $f1'=1$ if $\delta 1(s1,a1)\in F1$, otherwise $f1'=0$, when $\pi 1(a1)\notin(L1\cap L2)$, $a2=idle$, and $f1=f2=1$,
- $(\delta 1(s1,a1),s2,f1',f2)$ where $f1'=1$ if $\delta 1(s1,a1)\in F1 \vee f1=1$, otherwise $f1'=0$, when $\pi 1(a1)\notin(L1\cap L2)$, $a2=idle$, and $(f1=0\vee f2=0)$,
- $(s1,\delta 2(s2,a2),0,f2')$ where $f2'=1$ if $\delta 2(s2,a2)\in F2$, otherwise $f2'=0$, when $\pi 2(a2)\notin L1\cap L2$, $a1=idle$, and $f1=f2=1$,
- $(s1,\delta 2(s2,a2),f1,f2')$ where $f2'=1$ if $\delta 2(s2,a2)\in F2 \vee f2=1$, otherwise $f2'=0$, when $\pi 2(a2)\notin L1\cap L2$, $a1=idle$, and $(f1=0\vee f2=0)$,
- otherwise $\perp$,

$\pi: (A_1\cup\{idle\}\times A_2\cup\{idle\}) \rightarrow L1\cup L2\cup\{\tau\}$ such that
- $\pi((a_1,a_2))=\pi_1(a_1)=\pi_2(a_2)$ if $a_1 \in A1$ and $a_2\in A2$,
- $\pi((a1,idle))=\pi 1(a1)$ if $a1 \in A1$,
- $\pi((idle,a2))=\pi 2(a2)$ if $a2 \in A2$,

and $F'=\{ (s1,s2,f1,f2) \mid f1=f2=1\}$. ■

Remark that processes are synchronized at actions with same labels. This composition is similar to composition of CCS [Milner89] except for its treatment of designated nodes. The following relabelling operators is used to relabel actions so that actions which are synchronized in composition have same labels.

[Definition 8] (Relabelling Operator)
For $P=(S,A,L,\delta,\pi,s_0,F)$ and a relabelling function $f:L\rightarrow L'\cup\{\tau\}$, $P'=P[f]$ is defined as follows:
$P'=(S,A,L',\delta,\pi',s_0,F)$, where
- $\pi'(a)=f(\pi(a))$ if $\pi(a)\neq\tau$,
- $\pi'(a)=\tau$ if $\pi(a)=\tau$ ■

[Example]
- $P1=(\{s0,s1,s2\},\{t1,t2,t3,t4,t5\},\{a1,b1,c\},\delta_1,\pi_1, s_0,\{s1\})$ where
$\delta_1(s0,t1)=s1$, $\delta_1(s0,t2)=s2$, $\delta_1(s1,t3)=s2$, $\delta_1(s2,t4)=s1$, $\delta_1(s1,t5)=s1$, $\pi_1(t1)=a1$, $\pi_1(t2)=b1$, $\pi_1(t3)=b1$, $\pi_1(t4)=a1$, $\pi_1(t5)=c$.
- $P2=(\{s0,s1,s2\},\{t1,t2,t3,t4,t5\}, \{a2,b2,d\}, \delta_2,\pi_2,s_0,\{s2\})$ where
$\delta_2(s0,t1)=s1$, $\delta_2(s0,t2)=s2$, $\delta_2(s1,t3)=s2$, $\delta_2(s2,t4)=s1$, $\delta_2(s2,t5)=s2$, $\pi_2(t1)=a2$, $\pi_2(t2)=b2$, $\pi_2(t3)=b2$, $\pi_2(t4)=a2$, $\pi_2(t5)=d$.
- relabelling functions: $fi(ai)=a$, $fi(bi)=b$, and $fi(l)=l$ for other labels ($i=1,2$).
- $P1[f1]|P2[f2] = (\{s0,s1,s2,s3,s4\},\{(t1,t1),(t2,t2),(t3,t3),(t4,t4),(t5,idle),(idle,t5)\}, \{a,b,c,d\}, \delta, \pi, s_0, \{s3,s4\})$ where

$\delta(s0,(t1,t1))=s1$, $\delta(s0,(t2,t2))=s2$,
$\delta(s1,(t3,t3))=s3$, $\delta(s1,(t5,idle))=s1$,
$\delta(s2,(t4,t4))=s4$, $\delta(s2,(idle,t5))=s2$,
$\delta(s3,(t4,t4))=s1$, $\delta(s3,(idle,t5))=s2$,
$\delta(s4,(t3,t3))=s2$, $\delta(s4,(t5,idle))=s1$,
$\pi((t1,t1))=a$, $\pi((t2,t2))=b$,
$\pi((t3,t3))=b$, $\pi((t4,t4))=a$,
$\pi((t5,idle))=c$, $\pi((idle,t5))=d$.

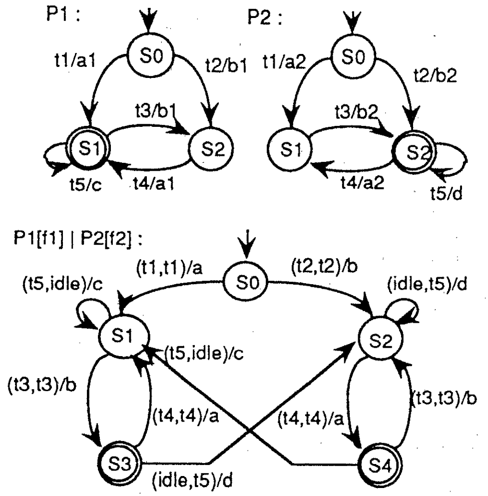


Fig2. Composition

[Definition 9] (Projection)
Let P1 and P2 be FSPs. A left projection $L(P1|P2)\downarrow left$ is defined as $L(P1|P2)\downarrow left \equiv \{ \theta 1/\{idle\} \mid \exists \theta \in L(P1|P2).\ \theta(i)=(\theta 1(i), \theta 2(i)) \}$. Similarly, a right projection $L(P1|P2)\downarrow right$ is defined. In the same way, projections of L_ω, L_ω^{fair}, and L_b are defined. ■

[Lemma 3] (Reflectivity)
Let P1 and P2 be FSPs. If P=P1|P2, then $L_b(P)\downarrow left \subset L_b(P1)$ and $L_b(P)\downarrow right \subset L_b(P2)$. ■

[Lemma 4] (Substitutivity)
$\pi\tau\omega$-bisimulation equivalence is preserved by composition and relabelling; that is, if $P =_{\pi\tau\omega} Q$, then $P|R =_{\pi\tau\omega} Q|R$, and $P[f] =_{\pi\tau\omega} Q[f]$. ■

Reflectivity and substitutivity are used in the following the basic adjustment and the compositional adjustment, respectively.

3. PROGRAM ADJUSTMENT

This section proposes compositional adjustment of FSP. Program adjustment means to adjust a roughly-made process to be complete by adding an arbiter process. First, we begin with basic adjustment.

3.1 BASIC ADJUSTMENT

[Problem]
Input: An FSP $P=(S,A,L,\delta,\pi,s_0,F)$,
Output: A maximally permissive FSP $C=(S_C,A_C,L_C,\delta_C,\pi_C,s_{0C},F_C)$ such that P|C is complete.
Here, "C is maximally permissive" means "$\forall C'$. if P|C' is complete then $L(C') \subset L(C)$". ■

Here, C is called an arbiter. The arbiter C restrains the target FSP P from falling into unsatisfiable states by eliminating harmful observable transitions.

[Algorithm 1] (Single Arbiter Synthesis)
(Step 0) P':=P.
(Step 1) Find a set of unsatisfiable states $Su \subset S'$ in $P'=(S',A',L,\delta',\pi',s_0',F')$. If there are no unsatisfiable states, go to Step 4.
(Step 2) Construct a pseudo-arbiter C' from P' as follows:
At first, τ-closure $C\tau$ is defined as
$C\tau(s,a) \equiv \{ s' \mid \exists\theta.\ (s'=\delta(s,\theta),\ \pi\hat{}(\theta)=a) \}$ for $\forall s \in S'$ and $\forall a \in L' \cup \{\varepsilon\}$,
$C\tau(Ssub,a) \equiv \bigcup_{s \in Ssub} C\tau(s,a)$ for $\forall Ssub \subset S'$ and $\forall a \in L'$,
then it is defined that $C' = (Sc', Ac', L, \delta_C', \pi_C', C\tau(s_0',\varepsilon), S_C')$, where
$Sc'=2^{S'}$, $Ac'=\{ t_a \mid a \in L\} \cup \{ t_s \mid s \in S'\}$, and
for $\forall a \in L$, $\forall s' \in Sc'$,
- $\delta_C'(s',t_a) = C\tau(s',a) \in Sc'$ if $C\tau(s',a) \cap Su = \emptyset$,
- $\delta_C'(s',t_a) = \perp$ if $C\tau(s',a) \cap Su \neq \emptyset$,
- $\delta_C'(s',t_{s'})=s'$, and

$\pi_C'(t_a)=a$ and $\pi_C'(t_{s'})=\tau$ for $\forall a \in L'$, $\forall s' \in Sc'$.
Remark that "$\delta_C'(s',t_a) = \perp$ if $C\tau(s',a) \cap Su \neq \emptyset$" means elimination of all behaviors which can not be distinguished from inevitably unsatisfiable behaviors by a label observer.
(Step 3) P':=P' | C', and return to Step 1.
(Step 4) Let a final pseudo-arbiter C' generated after applying Step 1 - Step 3 repeatedly be a arbiter C.

If C is empty (i.e., all behaviors are eliminated), C is called unrealizable, otherwise, called realizable.

[Theorem 1]
If a FSP $C=(S_C,A_C,L_C,\delta_C,\pi_C,s_{0C},F_C)$ is realizable for a given FSP $P=(S,A,L,\delta,\pi,s_0,F)$ in the above algorithm, then P|C is complete and C is maximally permissive.
(Sketch of proof) During Step 1 - Step 3, all inevitably unsatisfiable behaviors are eliminated in the final P'. Therefore, P' is complete. Since the transition function of C' is deterministic about its labels, C' restrains no satisfiable behavior of P. Therefore P|C is complete and C is maximally permissive. ■

[Corollary 1]

$L_\omega^{fair}(P|C){\downarrow}left \subset L_b(P|C){\downarrow}left \subset L_b(P)$

(Proof) It derives from Lemma 1 and Lemma 3 with Theorem 1. ■

This corollary assures that P, adjusted by C, satisfies its liveness constraints, whenever its behaviors are made by random transitions over states. Remark that an arbiter is effective in case that $L_\omega^{fair}(P) \subset L_b(P)$ does not hold.

[Example]

Fig.3 shows a simple single arbiter adjustment. In the target process P, only θ = t3t6t7 is an inevitably unsatisfiable behavior. Since {t3t6t7, t3t4} is a set of behaviors which can not be distinguished from θ (i.e. have the same label sequence "ab"), t4 and t7 are eliminated. From the reminder, the arbiter C can be constructed.

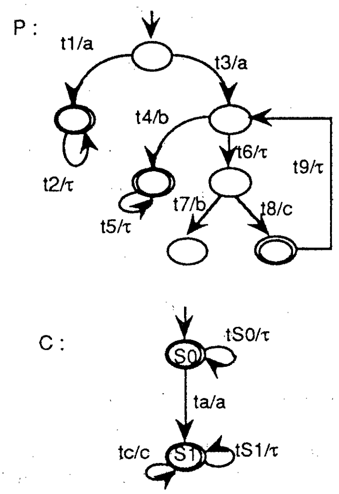


Fig.3 Single Arbiter Synthesis

3.2 COMPOSITIONAL ADJUSTMENT

When a target program is composed hierarchically with many processes and then become very large, the arbiter synthesis may cause the following problems: (1) the synthesis results in computing cost explosion, (2) a single arbiter is too restrictive to control the whole program precisely. Therefore, we propose compositional adjustment, in which local arbiters are synthesized in each composition step. The reduction of FSP can ease its computing cost explosion in each step.

[Theorem 2]

If P1 $=_{\pi\tau\omega}$ P2, then C is an arbiter of P1 iff C is an arbiter of P2.

(Proof) From Lemma 2 and Lemma 4, C|P1 is complete iff C|P2 is complete. ■

[Corollary 2]

If C is an arbiter of red(P), then C is also an arbiter of P. ■

[Algorithm 2] (Compositional Arbiter Synthesis)

For simplicity, we explain compositional adjustment for the following target program that is constructed by two-level composition (**Fig.4**). This algorithm can be extended easily to arbitrary target programs.

Target Program:

(P11[f11] | P12[f12])[f1] | (P21[f21] | P22[f22])[f2]

where P11, P12, P21, and P22 are FSPs, and f11, f12, f21, f22, f1 and f2 are relabelling functions.

The compositional arbiter synthesis is done in a bottom-up way.

(Step 1) Low level arbiters C1 and C2 are synthesized for subprocesses P11[f11] | P12[f12] and P21[f21] | P22[f22], respectively. We denote P1 ≡ (C1 | P11[f11] | P12[f12])[f1] and P2 ≡ (C2 | P21[f21] | P22[f22])[f2].

(Step 2) Reduced subprocesses red(P1) and red(P2) are made from P1 and P2.

(Step 3) A top level arbiter C0 is synthesized for a target process red(P1) | red(P2).

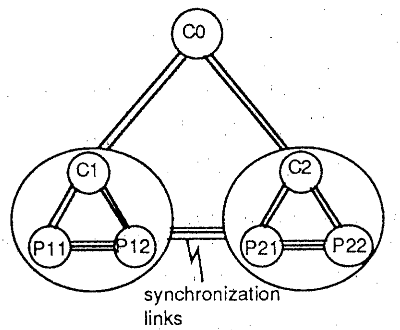


Fig.4 Compositional Adjustment

The Corollary 2 assures that reduction preserves all information necessary for each local arbiter synthesis. The reduction in each step can cut down the synthesis cost. Note that it is possible to synthesize directly a single arbiter C' for the target programs. However, C' is too restrictive because it has less visible (uncontrollable) actions compared with local arbiters, and its synthesis cost is more expensive.

4. MENDELS ZONE

4.1 OVERVIEW

MENDELS ZONE is a programming environment for concurrent programs. The target concurrent programming language is MENDEL, which is based on an extended Petri net and is then translated into the concurrent logic programming language KL1 and executed in Multi-PSI. MENDEL is regarded as a user-friendly macro language of KL1, whose purpose is similar to A'UM [Yoshida & Chikayama88] and AYA [Suzaki & Chikayama91]. However, MENDEL is more convenient for programmers to use to design a state-transition-based distributed system. MENDEL programs can also be translated into C and Occam. MENDELS ZONE supports (1) synthesis of MENDEL atomic processes, (2) graphical process interconnection, and (3) compositional adjustment of interconnected MENDEL processes based on theories described in Section 3. This adjustment procedure, which needs relatively much computing power, is implemented by KL1 and executed on Multi-PSI to achieve an effective speedup.

4.2 MENDEL NET

MENDEL is a concurrent programming language based on an extended Petri net. If a programmer constructs a program only using by MENDELS ZONE's graphic editor shown in **Fig.5**, he does not have to learn the detailed syntax of MENDEL. He is required only to know a graphical representation of the extended Petri net, called MENDEL net. Therefore, we omit an explanation of MENDEL itself. MENDEL net is extended from Petri net in the following aspects:
(1) Modularity is introduced. A module of MENDEL net represents a process.
(2) Another kind of synchronization between processes that is synchronous (i.e., hand-shake) communication is introduced, in addition to asynchronous (i.e., dataflow) communication.
(3) Each transition can have an additional enable condition, which must be satisfied when it fires, and an additional action, which is executed when it fires. Both are written by KL1.

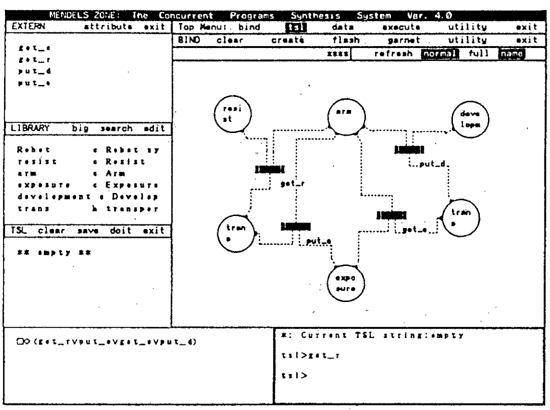


Fig.5 MENDELS ZONE

MENDEL net is graphically represented like Petri net (**Fig.6**). The basic conventions are as follows:
- Each place is represented by a circle.
- Each transition is represented by a square.
- Each process is represented by enclosing places and transitions belonging to the process with a line.
- A synchronous (hand-shake) communication is represented by a dotted line between transitions.
- An asynchronous (dataflow) communication is represented by an arrow between a transition and a place.

However, our program adjustment method is only applicable to finite state programs. When program adjustment is applied, the target MENDEL net is restricted to being a bounded one without asynchronous communications, which is able to be translated into FSPs. Furthermore, KL1 codes attached to transitions are ignored.

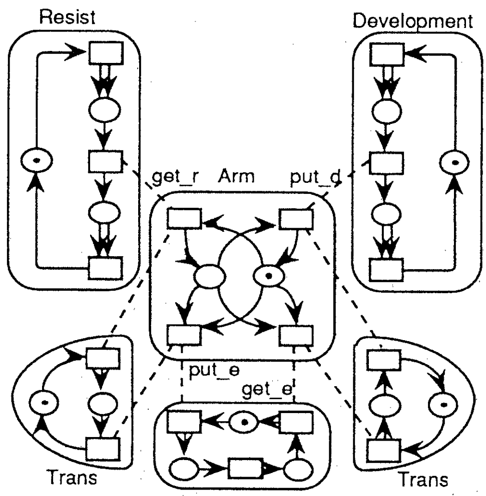


Fig.6 MENDEL NET

4.3 MENDEL NET CONSTRUCTION

A programmer can construct a MENDEL net using the graphic editor and a program library as follows:
(Step 1) Construct atomic MENDEL processes basically by software reuse [Uchihira87]. If the library has no suitable reusable MENDEL processes, MENDELS ZONE can synthesize it from a given algebraic specification [Honiden90]. It is also possible for the programmer to construct the atomic MENDEL process by himself using the graphic editor.
(Step 2) Interconnect MENDEL processes with communication links using the graphic editor to make a new compound MENDEL process. A large-scale program can be constructed in this compositional way.

Here, constructed programs are roughly-made because a programmer reuses programs whose possible behaviors he may not fully understand, and then communication links may be incomplete.

4.4 MENDEL NET VERIFICATION AND ADJUSTMENT

After constructing a roughly-made MENDEL net, the programmer specifies safety and liveness properties that must be satisfied by MENDEL net. Here, safety properties include admissible partial ordering of actions (i.e., transition firing), and liveness properties include deadlock and starvation about actions. These constraints are specified by temporal logic.

[Definition 10] (LPTL)
(1) Syntax
Linear time propositional temporal logic (LPTL) formulas are built from:
• A set of all atomic propositions: Prop={p_1, p_2, p_3,...,p_n}
• Boolean connectives: $\wedge, \neg$
• Temporal operators: X("next"), U("until")
The formation rules are:
• An atomic proposition $p \in$ Prop is a formula.
• If f1 and f2 are formulas, so are f1 $\wedge$ f2, $\neg$f1, Xf1, f1 U f2.
(2) Semantics
The operators intuitively have the following meanings:
$\neg$: NOT, $\wedge$: AND, Xf (read next f): f is true for the next state, f1 U f2 (read f1 until f2): f1 is true until f2 becomes true and f2 will eventually become true. The precise semantics are given as the Kripke structure [Manna& Wolper84]. ■

We use Ff ("eventually f") as an abbreviation for (true U f) and Gf ("always f") as an abbreviation for $\neg$F$\neg$f. Also, f1$\vee$ f2 and f1 $\supset$ f2 represent $\neg$($\neg$f1$\wedge$ $\neg$f2) and $\neg$f1 $\vee$ f2, respectively. Here, we assume a single event condition which provides that only one atomic proposition is true at any moment.

[Theorem 3]
Given an LPTL formula f under a single event condition, one can build a FSP P_f=(S,A,L, δ,π, s_0, F) such that L corresponds to a set of atomic propositions of f, and $L_b(P_f)$ is exactly the set of behaviors whose label sequences satisfy the formula f.
(Proof) It is a restriction of a general theorem [Wolper83]. ■

Remark that a label sequence of a satisfiable behavior in P_f corresponds to a model of LPTL formula.

[Example] (Temporal Logic Constraints)
Let a label set be L={a1,a2}.
(1) GF (a1 $\vee$ a2): Ether a1 or a2 must infinitely often occur.
(2) G(a1 $\supset$ XG($\neg$a2)): Whenever a1 occurs, then a2 must never occur.
FSPs which are generated from (1) and (2) are shown in **Fig.7**.

Fig.8 shows the verification and adjustment procedure: (1) The programmer can give an LPTL formula for a MENDEL net of each compound process. (2) MENDELS ZONE checks whether a MENDEL net satisfies a given LPTL formula by the model checking method for LPTL [Vardi&Wolper86]. (3) When it does not satisfy the LPTL formula, the adjustment method is invoked.

The compositional adjustment method, that is described in Section 3, can synthesize local arbiters for every compound process. Here, P_f representing temporal logic constraints is treated as one of the FSP components (i.e., a target process forms "P = Pf | P1 |...| Pn").

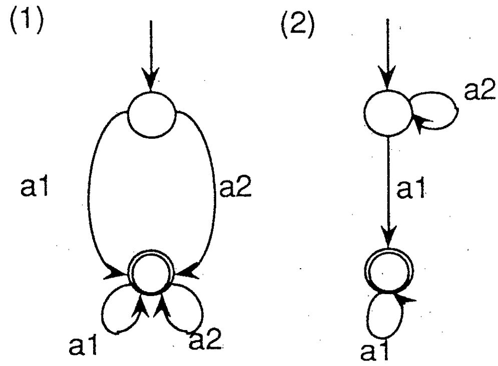


Fig.7 FSPs P_f Temporal Logic Constraints

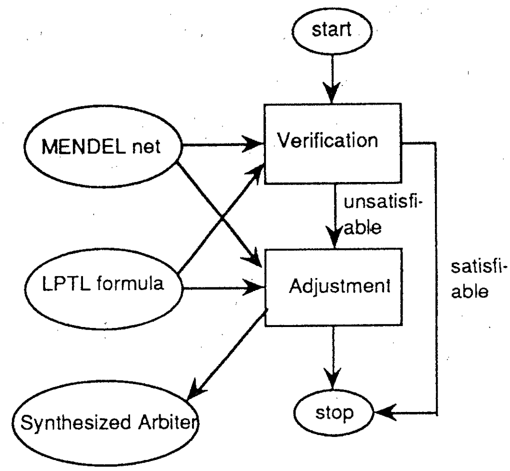


Fig.8 Verification and Adjustment

4.5 COMPILATION TO KL1 AND EXECUTION

The adjusted MENDEL program is compiled into a KL1 program, which can be executed on Multi-PSI. The programmer can check visually that the adjusted program behaves to satisfy his expectation. If not, he should consider two types of bugs: (1) Bugs of temporal logic constraints, and (2) Bugs of KL1 codes attached to transitions (i.e., its enable conditions and additional actions), which are ignored in translating to FSP.

5. EXAMPLE: THE SEQUENCE CONTROL PROGRAM

In this example we synthesize a single arbiter using MENDELS ZONE. The problem may be stated informally as follows. The target program must be designed to control machines which co-operatively process (i.e., etch) printed circuit boards (**Fig.9a**). The resist machine applies resist to boards. The exposure machine exposes boards to the light. The development machine develops boards. The arm machine moves boards from one machine to another. The target program is composed with 6 processes (Resist, Exposure, Development, Arm, and Trans × 2) which control corresponding machines. Here, Trans represents board transportation. Each process is displayed as a MENDEL net, shown in **Fig.6**. With no arbiter, this system falls into deadlock when an action label sequence of Arm "get_r → put_e → get_r" occurs. We give the following temporal logic constraints:

$$f = GF(get\_r \vee put\_e \vee get\_e \vee put\_d)$$

which means Arm never falls into deadlock. An arbiter C is synthesized as follows: First, FSPs representing 6 subprocesses are relabeled by relabelling functions fr, fe, fd, fa, ft1, and ft2, and are reduced, and FSP P_f (**Fig.9b**) representing temporal logic constraints f is generated. The target process P (**Fig.9c**) is composed from these FSPs. Finally, the arbiter C shown in **Fig.9d** is synthesized from P, according to Algorithm 1. We can see that the adjusted program "C | P_f | Resist[fr] | Exposure[fe] | Development[fd] | Arm[fa] | Trans[f_{t1}] | Trans[f_{t2}]" satisfies the above constraints.

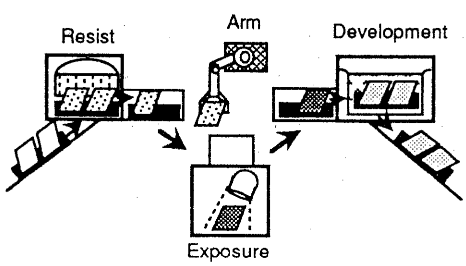


Fig. 9a Machine for Processing Printed Circuit Boards

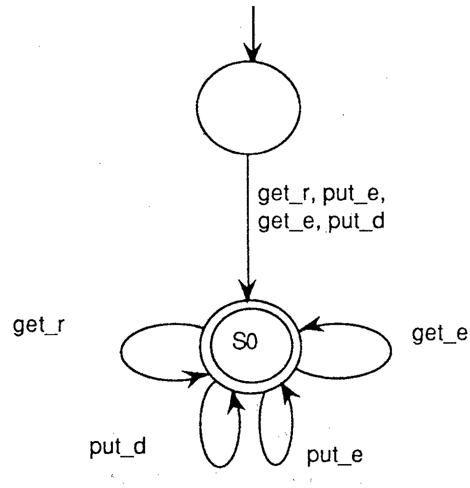


Fig.9b FSP Pf for LPTL formula

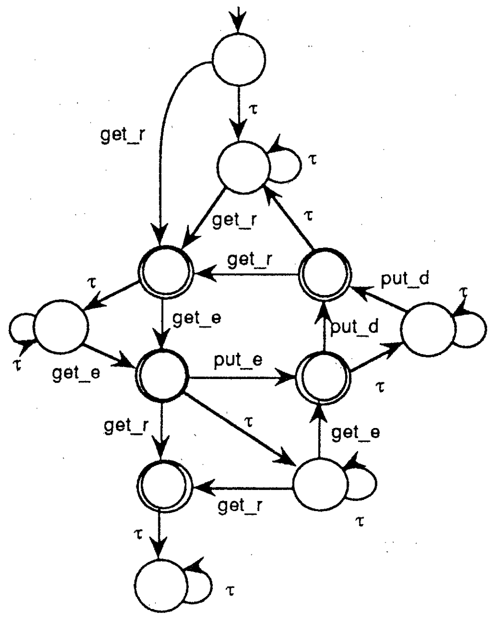


Fig.9c Target Process P (displaying only labels)

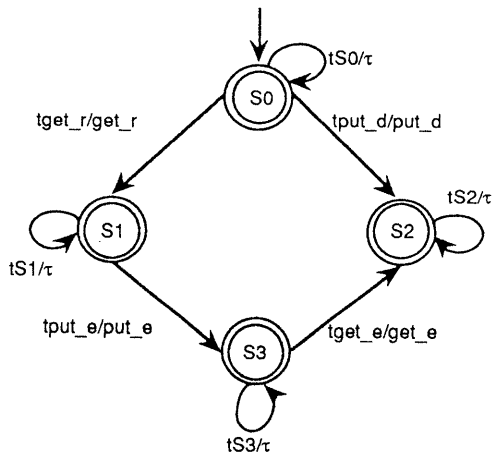


Fig.9d Synthesized Arbiter C

6. CONCLUSIONS AND RELATED WORKS

We have approached program synthesis from the viewpoint of program adjustment. In the proposed framework (i.e., FSP), program adjustment is defined as the synthesis of arbiter processes which control a target process with synchronization to satisfy their constraints. We have had some experience in state-transition-based software construction, using compositional adjustment in MENDELS ZONE.

Our previous works [Uchihira87, Uchihira90a, Uchihira&Honiden90] had proposed program synthesis methods, whose basic idea is similar to program adjustment. However, these methods are not fully compositional. In this paper, we newly introduce a CCS-like compositional framework to achieve compositional adjustment. Abadi, Lamport, and Wolper [Abadi89] proposed a compositional program synthesis using the CCS-like compositional framework, where failure equivalence is adopted instead of $\pi\tau\omega$-bisimulation equivalence. However, their approach is a top-down program refinement, which differs from our bottom-up program adjustment approach. On the other view, arbiter synthesis can be regarded as a control problem of discrete event systems which are well surveyed by Ramadge and Wonham [Ramadge&Wonham89]. However, these works showed no compositional synthesis methods satisfying liveness constraints, while they mainly consider safety properties.

ACKNOWLEDGMENT

This research has been supported by ICOT. We would like to thank Ryuzou Hasegawa of ICOT for his encouragement and support. We are also grateful to Seiichi Nishijima and Yutaka Ofude of the Systems & Software Engineering Laboratory, TOSHIBA Corporation, for providing continuous support.

REFERENCES

[Uchihira87] N. Uchihira, et al., Concurrent Program Synthesis with Reusable Components Using Temporal Logic, COMPSAC87 (1987).

[Honiden89] S. Honiden, et al., An Application of Structural Modeling and Automated Reasoning to Concurrent Program Design, 22nd HICSS (1989).

[Uchihira90a] N. Uchihira, et al., Synthesis of Concurrent Programs: Automated Reasoning Complements Software Reuse, 23nd HICSS (1990).

[Manna&Wolper84] Z.Manna and P.Wolper, Synthesis of Communicating Processes from Temporal Logic Specification, ACM Trans. Program. Lang. & Syst., **Vol. 6**, No. 1 (1984).

[Emerson&Clarke82] E.A.Emerson, E.M.Clarke, Using Branching Time Temporal Logic To Synthesize Synchronization Skeletons, Science of Computer Programming 2 (1982).

[de Bakker89] J.W. de Bakker, et al. (ed.), Stepwise Refinement of Distributed Systems, REX Workshop, LNCS 430 (1989).

[Kanellakis & Smolka90] P. C. Kanellakis, S. A. Smolka, CCS Expressions, Finite State Processes and Three Problems of Equivalence, Information and Computation 86 (1990).

[Milner89] R.Milner, Communication and Concurrency, Prentice Hall (1989).

[Uchihira90b] N. Uchihira, PQL: Modal Logic for Compositional Verification of Concurrent Programs (in Japanese), Trans. IEICE Vol.J75-DI, No.2 (1992).

[Paige&Tarjan89] R. Paige, R.E.Tarjan, Three Partition Refinement Algorithms, SIAM J. Comput. 16, No.6 (1987).

[Yoshida & Chikayama88] K. Yoshida, T. Chikayama, A'UM - Stream-Based Concurrent Object-Oriented Language -, FGCS88 (1988).

[Suzaki & Chikayama91] K.Suzaki and T.Chikayama, AYA: Process-Oriented Concurrent Programming Language on KL1 (in Japanese), KL1 Programming Workshop'91 (1991).

[Honiden90] S.Honiden, et al., A Formal Method for Real-Time SA using Algebraic and Temporal Logic Specification (in Japanese), IPSJ Tech. Rep. SE-73-3 (1990).

[Wolper83] P.Wolper, et al., Reasoning about Infinite Computation Paths, IEEE Proc. of 24th FOCS (1983).

[Vardi&Wolper86] M.Y.Vardi, P.Wolper, An Automata-Theoretic Approach To Automatic Program Verification, LICS86 (1986).

[Uchihira&Honiden90] N.Uchihira and S.Honiden, Verification and synthesis of concurrent programs using Petri nets and temporal logic, Trans. IEICE, **Vol.E73**, No.12 (1990).

[Abadi89] M. Abadi et al., Realizable and Unrealizable Specifications of Reactive Systems, 16th ICALP (1989).

[Ramadge&Wonham89] P.J.Ramadge and W.M.Wonham, The control of discrete event systems, Proc. IEEE, **Vol.77**, No.1 (1989).