Chapter 3

SURVEY OF RELATED WORK

3.1. Existing Methods for Subsystem Construction

The subsystem construction problem reported in the literature is illustrated in Fig.3.1. The problem is to find a subsystem C such that the interaction between M0 and C will provide the services specified by Sc. A major difference between this definition and the one given in Chapter 1 is the assumption of the former that observable events are controllable (to be defined in Chapter 5). This assumption makes it simpler to develop an algorithm for solving the problem. But it also limits its application since in many practical systems certain observable events are not controllable. Below, we will examine existing methods for solving this problem as reported in the literature.

$$\text{Fig.3.1}$$

3.1.1. Merlin-Bochmann’s Method: Construction under Trace Equivalence

Merlin and Bochmann [Merl83] first proposed a solution for the problem: given a partial specification $M_0 = (Q_0, \Sigma_0, \delta_0, q_{00})$ and a service specification $S_c = (Q_s, \Sigma_s, \delta_s, q_{s0})$, the solution $C = (Q_c, \Sigma_c, \delta_c, q_{c0})$ has the set of events $\Sigma_c = (\Sigma_0 \setminus \Sigma_s) \cup (\Sigma_s \setminus \Sigma_0)$, and is characterized by the set of possible traces:
\[
\text{Tr}_c = \text{Tr}_{\Sigma c}(M0\|S_c) - \text{Tr}_{\Sigma c}(M0\|\neg S_c)
\]

where \(\neg S_c\) denotes the complement of \(S_c\), i.e., the sequences of \(\neg S_c\) are those sequences over \(\Sigma_S\) that are not possible according to \(S_c\). This method considers the trace equivalence as the basis for comparing specifications. The specification obtained by the above formula gives the largest number of possible traces for the missing subsystem \(C\), i.e., for any other solution \(C'\) such that \(M0\|C'\sim S_c\), \(\text{Tr}_{\Sigma C}(C') / \text{Tr}_{\Sigma C}(C)\) must be true.

This work is the first attempt to solve the problem. The advantage of this method is that it is simple in concept and the algorithm is less complex in comparison with other algorithms. However, the trace equivalence relation is considered as not strong enough in certain applications, and further verification against the given service specification is necessary to avoid deadlocks.

An important extension of Merlin-Bochmann’s method is reported in [Kele93]. It can be summarized as follows:

1) An asymmetric finite state model is used. The asymmetric model distinguishes between sending and receiving transitions of a process, thus allowing a representation of the causality of actual communication. An advantage of this representation is to detect unspecified reception more easily in the design of communication protocols.

2) A progress algorithm is developed. An overall solution for the asymmetric model that conforms to the safety as well as progress properties is implemented by first obtaining a solution that is safe, and then running the iterative progress algorithm to remove deadlocks.

The method is applied to synthesizing protocols and protocol converters.

3.1.2. Parrow’s Method for CCS under Observational Bisimulation Relation

In [Parr89], Parrow presented a method for solving this problem based on the CCS model [Miln89]. It uses a different methodology: stepwise transformation of equations into simpler ones by using expansion theorem and congruence properties [Miln89], which Parrow called the tableau method. A tableau consists of two parts: a goal \(G\) and an environment \(E\). The intuition behind a tableau is that it represents an intermediate stage in
producing a solution: the goal indicates what remains to be done in order to maintain the equivalence of the two sides of the equation; the environment records the solutions produced so far. The initial tableau is \(<M0||C \bar{Y} Sc, E>\), where \(\bar{Y}\) denotes an observational bisimulation relation, \(M0||C \bar{Y} Sc\) is the initial goal, and the solution record \(E\) is initially empty. The key of the approach is the transformation rules which transform the equation into simpler ones and obtain simpler goals. There are two types of transformation rules:

1) **Instantiation**, which extends the environment by guessing the initial transitions of the unknown \(C\): If \(<M0||C \bar{Y} Sc, E>\), then \(C \Rightarrow \sum_{i=1}^{n} a_i C_i\) can be added to \(E\). The set \(\{a_1, ..., a_n\}\) is the initial actions of \(C\). Unfortunately, there are in general several different such sets corresponding to different solutions \(C\). If the instantiation is applied carelessly, an unsatisfiable table may be produced. In order to overcome this difficulty, Parrow introduced some heuristics for choosing this set. One such heuristic is to avoid the actions which could not possibly be initial actions of \(C\). Another heuristic is to avoid actions which will never be used in any solution \(C\).

2) **Reduction** \(<G, E> \Rightarrow <G', E>\), which strengthen the goal if \(G'\) is obtained from \(G\) in one of the following ways:
   a) **Equivalence**: an equation \(D \bar{Y} F\), where \(D\) and \(F\) are observational equivalent with respect to \(E\), can be removed from \(G\).
   b) **Splitting**, if \(G\) contains an equation \(D \bar{Y} F\), and \(D \bar{Y} \sum_{i=1}^{n} a_i D_i\), \(F \bar{Y} \sum_{i=1}^{n} a_i F_i\) for \(1 \leq i \leq n\), then the equation can be replaced by the equations

\[
\sum_{i=1}^{n} (D_i \bar{Y} F_i)
\]

The transformation procedure is repeatedly applied to the goal part until the goal part of the tableau becomes true obviously. The final result is recorded in \(E\).

Because of the heuristic feature of the transformation discussed above, the method can be implemented in a semi-automatic way. The following results are proved in [Parr89]:

29
1) If the algorithm reports a solution $C$, then $M_0 || C \uparrow \uparrow S_c$ holds.

2) If there exists a deterministic solution $C'$ such that $M_0 || C' \uparrow \uparrow S_c$, then the algorithm will report a deterministic solution $C$.

However, the solution $C$ constructed by the method may not be the most general one since the user is able to apply smarter rules to avoid unnecessary transitions in the solution.

The advantage of this approach is that the observational bisimulation is used to compare a designed system with its service specification during system construction, which guarantees functional correctness of the design. It may also give a better solution since the designer's suggestion can be integrated in the solution procedure. The disadvantage is that the method depends on the designer for choosing the transformation rules, which is sometimes not desirable. In addition, it has the limitation that $S_c$ must be deterministic.

### 3.1.3. Qin-Lews’ Method: Construction under Bisimulation

In [Qin91], Qin and Lewis presented a method under observational bisimulation relation, where a Finite State Machine (FSM) model similar to the FLTS introduced in Chapter 2 is used. The method is implemented as follows. First, it generates all possible state pairs $(p, q)$ for $p \in Q_0$ and $q \in Q_s$ by using the coupled product. This step, in fact, is to remove all the behaviours in $M_0$ that violate the safety property (the safety property is explained in Chapter 2). Then, the behaviours that violate the observational bisimulation are removed. This is implemented by constructing a new machine $M'_0$ with a set of subsets of the state pairs generated in the first step. Each subset $r$ of the state pairs must satisfy the following conditions:

1) For any $s \in Tr_{\Sigma_c}(M_0)$, if there is a state pair $(p, q)$ in $r$ and $p_0 = s > p$ in $M_0$, then for all $(p', q')$ in $r$, $p_0 = s > p'$ in $M_0$; and there does not exist another subset $r'$ such that any state pair $(p'', q'')$ of $r'$ satisfies $p_0 = s > p''$ in $M_0$.

2) For any $(p, q)$ in $r$, if $q - e \rightarrow$ in $S_c$ then $(p, q) = e \rightarrow_{\Sigma_c}$ in $M'_0$ and there is not any state pair $(p', q')$ in $r$ such that $p - e \rightarrow$ but $q - e \not\rightarrow$ for $e \in \Sigma_0 \cap \Sigma_s$.

3) For any $(p, q)$ in $r$ and $e \in (\Sigma_0 - \Sigma_s) \cup (\Sigma_s - \Sigma_0)$, if $p - e \rightarrow$ in $M_0$ but $(p, q) - e \not\rightarrow$, or, if $q - e \not\rightarrow$ for $e \in \Sigma_s - \Sigma_0$, then $e$ is inhibited in $r$.  

30
Finally, the solution C is derived from the set of subsets of state pairs by constructing a transition \( r - e \rightarrow r' \) for any two subsets \( r \) and \( r' \) if there is a state pair \((p, q)\) in \( r \) and \((p', q')\) in \( r' \) such that \((p, q) - e \rightarrow (p', q')\), where \( q \subseteq (\Sigma_0 - \Sigma_s) \cup (\Sigma_s - \Sigma_0) \).

Compared with the method presented in [Parr89], this algorithm does not need designer's participation during the system construction - it is automatic. However, it also has the limitation that Sc must be deterministic. In addition, the time complexity of the algorithm is exponential in the worst case.

3.2. Methods for Synthesizing Controllers for Discrete Event Systems

The methods of synthesizing controllers for discrete event systems are developed by the discrete system control community, where an automaton model is used, instead of the FLTS. The definition of an automaton is similar to the definition of a single labelled transition system [Rama89], including a set of states \( Q \), an alphabet \( \Sigma \) which is a set of events, a transition function \( \delta \) and an initial state \( q_0 \). One of the differences is that the automaton has a set of states \( Qm \) \subseteq Q, called marker states, which play the role of "accepting" states and mean that some tasks have been completed. State transitions are considered to occur instantaneously and asynchronously - two events can not occur simultaneously and the time between consecutive events is not fixed. The analysis and synthesis of controllers for discrete event systems are quite different from the subsystem construction reviewed in the previous sections because of the controllability and observability, which will be explained in the following section.

3.2.1. Controllability and Controller Synthesis

The basic problem in discrete event system control is to modify the open loop behavior of a given discrete event system so that it lies within some prescribed range, where open loop behaviour is the behaviour of a system G without any external control, written \( L(G) \). The prescribed range is specified by actually giving the desired closed loop behavior, or by specifying it indirectly through other qualitative performance objectives. A closed loop behaviour is the behaviour of a system under the control of an external controller \( C \), written \( L(G, C) \). The behaviour of a system is often represented as a set of all possible sequences of events of the system, called a language in automaton theory, which is the same concept as
the set of all execution sequence defined for FLTS. A natural question is: given a discrete event system $G$ with open loop language $L$, what closed loop language $K = L(G, C) \subseteq L$ can be achieved by a controller $C$?

There are two most important concepts related to this problem: controllability and observability. To control a system, it is assumed that certain events of the system can be disabled by the controller when desired. To model such a control, the set of events $\Sigma$ is partitioned into controllable and uncontrollable events: $\Sigma = \Sigma_u \cup \Sigma_c$, where the events in $\Sigma_c$ can be disabled by a controller $C$, while the events in $\Sigma_u$ can not be influenced by $C$.

**Definition (controllability [Rama89])**: For a discrete system $G$ with open loop behavior $L$ and a set of uncontrollable events $\Sigma_u$, a sublanguage $K \subseteq L$ is controllable if $K\Sigma_u \cap L \subseteq K$, where $K\Sigma_u$ denotes the concatenation of an element in $K$ with an event in $\Sigma_u$ and $K$ is the *prefix closure of* $K$ [Rama89], i.e., $K = \{k|kv \in K \text{ for } v \in \Sigma^*\}$.

This definition requires that for any $k \in K$, and an uncontrollable event $\nu \in \Sigma_u$, if $k\nu \in L$, then $k\nu \in K$. It is intuitively clear that, if an uncontrollable event occurs following an execution sequence in $K$, then the extended execution sequence must remain in $K$; otherwise, if $k \in K$ and $k\nu \notin K$, then the undesired behaviour $k\nu$ can not be prevented since $\nu$ is uncontrollable.

As introduced before, the subset of states $Q_m$ is marked as the states at which some tasks have been completed. These states are also called *accepting states* in traditional accepting automaton theory. A corresponding language, called *marked language*, written $L_m(G)$, is defined as

$$L_m(G) = \{\sigma| \sigma \in \Sigma^* \text{ and } \delta(\sigma, q) \in Q_m\}$$

There is no implication that the generating action halts after the completion of some marked sequence, i.e., marker states need not be final states. From the definition of marked language we have $L_m(G) \subseteq L(G)$. If $L_m(G) = L(G)$, then $G$ is said *nonblocking*.

[Rama89] provided the following result: For a nonblocking discrete event system $G$ with closed behavior $L$ and marked behavior $L_m$:

1) For nonempty $K \subseteq L$ there exists a supervisor $C$ such that $L(G, C) = K$ iff $K$ is prefix closed and controllable.
2) For nonempty $K \subseteq Lm$ there exists a supervisor $C$ such that $Lm(G, C) = K$, and the closed loop system is nonblocking iff $K$ is controllable, and $K \cap Lm = K$.

If $K$ does not satisfy the conditions, let $C(K)$ denote the set of controllable sublanguages of $K$. It is proved in [Rama89] that $C(K)$ is closed under set union, and there exists a unique largest controllable language $K^#$ such that $K^# \subseteq K$. The closure of $C(K)$ under set union indicates that if a given language $K$ is not controllable, then there exists a natural controllable approximation to $K$, namely the largest controllable sublanguage contained in $K$. An algorithm for the computation of $K^#$ is developed in [Rama89]. The basic idea of the algorithm is the following fixpoint characterization. Let $p(\Sigma^*)$ denote the power set of $\Sigma^*$, i.e., the set of all languages over $\Sigma^*$, and define the operator $\Omega: p(\Sigma^*) \rightarrow p(\Sigma^*)$ by $\Omega(J) = K \cap \text{sup}\{\sigma: \sigma \subseteq \Sigma^*, \sigma = \sigma, \sigma \Sigma_u \cap L \subseteq J\}$.

Then $K^#$ is the largest language $J$ such that $\Omega(J) = J$. Furthermore, if we set

$$K_0 = K$$
$$K_{i+1} = \Omega(K_i)$$

then

$$\lim_{i \rightarrow \infty} K_i = K^#.$$  

The computation of $K^#$ is of polynomial complexity.

### 3.2.2. Partial Observations and Observability

It was assumed in the last section that all of the events generated by a discrete event system $G$ can be directly observed by the controller. However, in many situations we only have partial observations. Consequently, the set $\Sigma$ may be partitioned into the disjoint sets $\Sigma = \Sigma_o \cup \Sigma_u$, where $\Sigma_o$ denotes the set of events observable and $\Sigma_u$ denotes the set of events unobservable by a controller.

We define a projection $P$ such that $P: \Sigma \rightarrow (\Sigma_o \cup \{\tau\})$. The idea is that $P(\upsilon)$ is the event observed when the system undergoes a state transition labeled by $\upsilon$. Those events $\upsilon \subseteq \Sigma$ with $P(\upsilon) = \tau$ are not observed at all, while two events $\upsilon_1, \upsilon_2 \subseteq \Sigma$ with $P(\upsilon_1) = P(\upsilon_2)$ can no longer be distinguished. The special case in which $P$ simply erases some of the events in $\Sigma$ occurs frequently, and is called a natural projection. In this case $P$ is defined as:

$$P(\upsilon) = \begin{cases} 
\upsilon, & \text{if } \upsilon \in \Sigma_o \\
\tau, & \text{if } \upsilon \notin \Sigma_o
\end{cases}$$
This function can be extended to languages by defining $P(\tau) = \tau$ and

$$P(\sigma\tau) = P(\sigma)P(\tau) \text{ for } \sigma \in (\Sigma \cup \tau)^*.$$ 

Given a prefix closed language $K \subseteq L(G)$ and a projection $P$ onto a subset of the alphabet of $L$, under what conditions does there exist a controller $C$, which only observes the projection $P(K)$, such that $L(G, C) = K$? To deal with the controller synthesis problem under partial observation, one first needs to define the observability concept.

**Definition (relation Ker-P [Rama89]):** $\text{ker-P}$ is the equivalence relation on $\Sigma^*$ defined by $(\sigma, \sigma') \in \text{ker-P}$ iff $P(\sigma) = P(\sigma')$.

This definition implies that $(\sigma, \sigma') \in \text{ker-P}$ if they have the same projection under $P$.

**Definition (relation Act_k [Rama89]):** $\text{Act}_k$ is a binary relation on $\Sigma^*$ defined by $(\sigma, \sigma') \in \text{Act}_k$ iff $\sigma, \sigma' \in K$ implies that $\forall \upsilon \in \Sigma, \sigma \upsilon \in K$ iff $\sigma' \upsilon \in K$.

In other words $(\sigma, \sigma') \in \text{Act}_k$ if all the one step continuations of $\sigma$ and $\sigma'$ that remain in $L(G)$ yield the same result with respect to membership of $K$.

**Definition (P-observability [Rama89]):** A prefix closed language $K$ is P-observable with respect to $G$ if $\text{ker-P}$ implies $\text{Act}_k$.

This definition implies the condition that if $P(\sigma) = P(\sigma')$ then $(\sigma, \sigma') \in \text{Act}_k$. In other word, this means that the projection $P$ retains sufficient information to decide whether or not, after the occurrence of some event, the resultant language is in $K$.

**Proposition [Rama89]:** Let $K \subseteq L(G)$ be prefix closed and nonempty. Then there exists a controller $C$ such that $L(G, C) = K$ iff $K$ is controllable and P-observable.

If $K$ does not satisfy the conditions of the above proposition, then a natural consideration is to try approximating $K$ by a sublanguage of $K$. Unfortunately, a unique maximal controllable and observable sublanguage of $K$ may not exist. To obtain such approximations in a reasonable and practical manner, [Rama89] considered the following P-normal sublanguages.

**Definition (P-normal language):** A closed language $K \subseteq L(G)$ is P-normal if

$$K = L(G) \cap P^{-1}(P(K))$$
where, \( P^{-1}(P(K)) \) represents the largest sublanguage of \( L \) having \( P(K) \) as its projection. This condition of a \( P \)-normal language implies that \( K \) is the largest sublanguage of \( L \) having \( P(K) \) as its projection; thus \( K \) is determined uniquely by its projection and the constraint imposed by \( L \). This kind of sublanguage has a number of special properties that make it of interest. The one that concerns us is that for \( \sigma \in L(G) \) we can decide if \( \sigma \in K \) from \( P(\sigma) \) alone.

**Proposition [Rama89]:** Let \( K \subseteq L(G) \) be closed and \( P \)-normal, then \( K \) is \( P \)-observable.

[Rand88] provided an algorithm to compute the sublanguage \( K_p \) of \( K \) that is closed, \( P \)-normal and controllable. This algorithm is presented in four steps:

1) Compute \( K' \), the largest closed \( P \)-normal sublanguage of \( K \);
2) compute \( P(L(G)), P(K') \), and let \( \Sigma u' = P(\Sigma u) - \{\tau\} \);
3) using the method described in the last section to compute \( K'' \), the largest sublanguage of \( P(K') \) that is closed and controllable with respect to \( \Sigma u' \);
4) compute \( K_p = P^{-1}(K'') \cap L(G) \).

After \( K_p \) is obtained, the controller \( C \) can be derived by constructing a deterministic machine characterized by \( P(K_p) \).

### 3.2.3 Real-Time Controller Synthesis

Taking into account that the behavior of most of the real systems depends on the quantitative passing of time, for example, using timers to solve error conditions in data communication systems, system design should consider the timing requirements. The "tick" time model is often used for modelling timed systems and controllers [Ostr90], [Bran94], which is defined as follows.

**Definition (timed transition model (TTM) [Ostr90], [Bran94]):** Given an automaton \( M = (Q, \Sigma, \delta, p_0, Q_m) \), a corresponding TTM is defined by \( M_t = (Q_t, \Sigma_t, \delta_t, p_{t0}, Q_{mt}) \) where:

1) \( Q_t = Q \cup \{T_e | e \in \Sigma \} \) and \( T_e = [0, u_e] \), where \( u_e \in \mathbb{N} \cup \{\infty\} \) denotes an upper time bound of \( e \), \( \mathbb{N} \) is the set of natural numbers, and \( [0, u_e] \) denotes a set of integers \( i \) with \( 0 \leq i \leq u_e \). Thus a state \( q \in Q_t \) is an element of the form
   \[ q = \{p, \{t_e | e \in \Sigma \}\} \]
where $t_e \in T_e$, $p \in Q$. This means that $q$ consists of an state $p \in Q$ together with a tuple assigning to each event $e \in \Sigma$ an integer in its timer interval $T_e$.

2) $\Sigma_t = \Sigma \cup \{\text{tick}\}$, where tick is introduced to represent the "tick of the global clock".

3) $\delta_t$ is the transition function: $Q \times \Sigma_t \rightarrow Q_t$.

4) $p_{t0} \in Qt$ is the initial state.

5) $Qmt / Qt$ is the set of accepting states.

In [Ostr90] and [Bran94], the rules and methods for generating the transitions and the states are developed. We use the following example to explain the definition.

**Example**: Given $M = (Q, \Sigma, \delta, p0)$ where $Q = \{0\}$, $\Sigma = \{\text{â}, \_\}$, $p0 = 0$, and $\delta(\text{â}, 0) = \delta(\_, 0) = 0$; and timed events¹ $(\text{â}, [1,1]), (\_, [2,3])$. The state set for the TTM is

$$Qt = \{0\} \cup T_{\text{â}} \cup T_{\_} = \{0\} \cup \{0, 1\} \cup \{0, 3\}$$

and the size $|Qt| = 8$. $\Sigma_t = \{\text{â}, \_, \text{tick}\}$, the initial state $p_{t0}$ is $\{0, [1, 3]\}$. The untimed automaton $M$ is shown in Fig.3.2 (a) and the TTM $G$ is shown in Fig.3.2(b), where state 1 denotes $\{0, [1, 3]\}$, state 2 denotes $\{0, [1, 2]\}$, state 3 denotes $\{0, [0, 1]\}$, state 4 denotes $\{0, [0, 3]\}$, state 5 denotes $\{0, [1, 1]\}$, state 6 denotes $\{0, [0, 0]\}$ and state 7 denotes $\{0, [1, 0]\}$.

¹Note that the definition of "timed event" in [Ostr90] and [Bran94] is different from the definition of timed event defined in Chapter 2.
With the timed model, the controller design problem is transformed to untimed system synthesis. In [Bran94], the TTM model is adjoined to the structure of [Rama89]. Within TTM, "forcible events" are introduced as the events that can prompt the timing event "tick" in competition with other events. Timed controllability is defined in analogy to its untimed counterpart described in [Rama89]. The special time event "tick" is considered controllable. If "tick" is considered as a special event, there is no significant difference between the TTM model and the untimed automaton. Therefore, the methods developed for untimed system construction can be used for timed system construction. This is the advantage of the TTM model.

The limitation of the TTM model can be observed from the definition of Qt: the state space may be too large to be processed even for a system with moderate size.

### 3.2.4. Other work on Controller Construction

For large systems, the synthesis of a controller is very complex. One method to reduce the complexity is to incorporate additional structure into the model, for example, modular specifications and modular controller synthesis. Modularity allows complex problems to be decomposed into simpler components, greater structure and flexibility to be incorporated into the controller.

Assuming that there are two constraints K1 and K2 simultaneously defined for a system G, then the overall constraint is specified by the intersection K1 ∩ K2. The key concept of modular synthesis is the property of nonconflicting languages.

**Definition** (nonconflicting languages): K1 and K2 are said to be nonconflicting when we have the equality: $K_1 \cap K_2 = K_1 \cap K_2$.

Clearly, two prefix closed languages are nonconflicting. [Rama89] has provided the following results for closed languages.

**Proposition:** Let $K_1, K_2 \subseteq \Sigma^*$ be nonconflicting. If $K_1$ and $K_2$ are both $Lm(G)$-closed and controllable, then $K_1 \cap K_2$ is $Lm(G)$ closed and controllable.
Similarly one can use the nonconflicting property to determine the condition under which the operation of taking the supremal controllable sublanguage of $K$ represented by language intersection is possible:

**Proposition:** Let $K_1, K_2 \subseteq \Sigma^*$. If $K_1$ and $K_2$ are nonconflicting, then $K_1^\# \cap K_2^\# = (K_1 \cap K_2)^\#$.

This result indicates that the supremal controllable sublanguage of $K_1 \cap K_2$ can be found by first computing $K_1^\#$ and $K_2^\#$, checking that these languages are nonconflicting, and if so, forming their intersection.

If $C_1$ and $C_2$ are controllers derived based on the constraints $K_1$ and $K_2$ respectively, then we obtain the system controller $C = C_1 \times C_2$.

**Lemma:** Let $C_1$ and $C_2$ be nonblocking controllers for $G$, and $C = C_1 \times C_2$, then:

1) $L(G, C) = L(G, C_1) \cap L(G, C_2)$;
2) $Lm(G, C) = Lm(G, C_1) \cap Lm(G, C_2)$;
3) $C$ is nonblocking for $G$ iff $Lm(G, C_1)$ and $Lm(G, C_2)$ are nonconflicting.

Assume that the system has $m$ states. Then, for a regular language $K$ specified by a $n$ state automaton, the complexity of the verification $K \Sigma_d \cap L(G) \subseteq K$ is $O(mn)$. Thus, to compute $K_1 \cap K_2$ and then to check the controllability of $K$ is of complexity $O(n^2m)$. On the other hand, to verify that $K_1$ and $K_2$ are each controllable is a computation of complexity $O(nm)$. This result indicates that the modular synthesis method can offer a significant reduction in computational complexity when it is applicable.

### 3.3. Protocol Conversion

There is a number of different networks in the world. Each of the networks uses different protocols: the form and meaning of messages are defined differently, and the exchange of messages is governed by different rules and procedures (timing, sequences ...). This is the so-called *protocol mismatch* problem as explained in [Gree86]. Heterogeneous networks are proliferating steadily as a result of many factors:

1) The competitive and proprietary nature of computer networking and the rapid pace of technical evolution in computers: Each large computer manufacturer, in attempting to gain and maintain a competitive edge for commercial reasons, has implemented, promoted and continually upgraded its own architecture and
protocols to networking. These architectures as well as their protocols show wide
differences in detail, however, they try to accomplish the same thing: reliably and
efficiently transmitting information, with little consideration of internetworking.

2) Enterprises are constantly being reorganized, which often forces independent and
heterogeneous networks to become part of new networks. Questions of
interconnection do not appear until each department has installed its subnetwork.

3) The emerging of new technologies and new version of the standards and the need
to enforce backward compatibility with the customer’s installed systems lead to
additional complexity for network interconnection.

The best solution to this problem is to re-program the protocols so that they implement the
same protocol standards, i.e., the protocols for OSI or Internet? However, it is very
difficult to do so, if not impossible:

1) International standards are seldom so rigid that interoperability is truly guaranteed.
In order to gain general agreement to the standard, the specification usually
provides several options for everything from settings of simple parameter values
to the question of which version of an entire protocol layer will be used, at the
cost of imperfect interworkability. For example, there are five protocols
developed for the transport layer in OSI.

2) In certain situations, incompatible subnetworks are more attractive, functionally or
economically. For example, both the connection-oriented protocols and the
connectionless-oriented protocols have their own applications. It is worth the
expense of providing the conversion gateway required to interconnect them
together.

3) Re-programming all of the existing protocols used by heterogeneous networks is
too expensive.

It is hard to see how the factors deriving the continued presence of computer network
heterogeneity will dissipate in the near future. Therefore, alternative solutions are sought.
One proposed alternative is to use a protocol converter embedded in an internetworking
gateway, an intermediary used to translate messages between the incompatible protocols.
Sophisticated technologies are needed for analyzing the mismatches between the
heterogeneous network architectures and protocols, and synthesizing converters that resolve these mismatches while providing good performance at low cost. In the following sections, we will discuss the existing protocol conversion methods.

3.3.1. Top-down Methods

*Top-down methods* use a global service specification as the semantic constraint. These methods can be divided into three subclasses:

a) *Service level conversion*, which only deals with the mappings of service primitives by constructing a service adapter [Boch90b].

b) *PDU level protocol conversion using a subsystem construction* method, which constructs a protocol converter at the PDU level based on a global service specification by applying subsystem construction methods. For example, the method proposed in [Calv89] and the method presented in Chapter 6 of this thesis.

c) *Transformation method*, which transforms a service adapter and the underlying protocols into a PDU level protocol converter [Boch90a], [Kris93], [Okum90].

3.3.1.1. Service Level Conversion

A typical architecture of protocol conversion at the service level is shown in Fig. 3.3, where a service adapter is used for mapping service primitives. This method results from a systematic consideration of the layered communication architecture including the service specification [Boch90b].

![Architecture of a gateway with service adapter](image)

Fig.3.3 A architecture of a gateway with service adapter

The principle of this method is as follows:
1) Construct a service adapter: given a global service specification $Sc$, the service specification $Sd$ of protocol D and the service specification $Sf$ of protocol F, the service adapter $Cx$ should be constructed such that

$Sd || Cx || Sf$ satisfy $Sc$.

2) Integrate the service adapter $Cx$ with the two protocol entities D2 and F1 to form a converter $C$ as shown in Fig.3.3. Note that the protocol entities D2 and F1 are used directly in the converter, but the derivation of the service adapter $Cx$ is independent of them.

The principle of this method implies that the protocol conversion problem relies on finding a service adapter $Cx$. It is possible to derive a service adapter formally by using a subsystem construction algorithm. For example, Okumura [Okum90] proposed to use Merlin-Bochmann’s subsystem construction method to derive a service adapter $Cx$ from the services $Sd$ and $Sf$ of the two protocols, and a given global service specification $Sc$ by using the formula described in Section 3.1.

One important problem of service level conversion is the so-called concatenation invariance property of a communication service. A global property of a communication service is concatenation invariant if it remains satisfied on an end-to-end basis over several concatenated communication services. In [Boch90b], this property is discussed in detail. The examples shown in Fig.3.4 and Fig.3.5 may be used to explain the concept [Boch90b].

![Fig. 3.4 Different types of delivery confirmation](image-url)
Fig. 3.5 Different scenarios of interconnection

Fig. 3.4 depicts three types of delivery confirmations. The solid (dashed) arrows denote the interactions of Send-data and Receive-data (Send-ack and Receive-ack). Numbers associated with arrows represent a relative temporal ordering. Fig. 3.5 represents three interconnection scenarios. Clearly, the interconnection of two type B services does not satisfy the concatenation invariance for data confirmation. For other services, it is satisfied. We note that concatenation invariance of type A confirmation is vacuous, since this confirmation has no global significance, only a local one.

Those properties of the interconnected communication services that do not satisfy concatenation invariance will not be preserved in the case of interworking. If all the services are concatenation invariant, the user processes using these services do not see any qualitative difference between the communication in the local network and throughout interconnected networks.

One of the main advantages of service level conversion is its simplicity: from the principle of the method, we know that the derivation of a service adapter is independent of the underlying protocols and channels, only the service specifications are used; no PDU level constraint is necessary for constructing a service adapter. This avoids the principal difficulty of the methods based on PDU level constraints (namely message relations). Moreover, as the original protocol implementations can be directly reused for the implementation of the converter, the implementation effort can be largely reduced.

However, compared with PDU level conversion, this method has a limited applicability, since there are cases for which no service level converter exists although a PDU level converter can be constructed. An example is the type B service concatenation shown in Fig. 3.5(b). It is clear that the acknowledgment service provided by a Receive-ack event requires the transmission of an ack-PDU. The Receive-ack event is generated by the
protocol entity D1 as soon as the ack-PDU is received. Since the D2 entity may send the ack-PDU as soon as the Receive-data event occurs at the service interface, there is no converter at the service level that would be able to give an end-to-end significance to the Receive-ack event in the case of interworking with a second type B service. In the case of conversion at the PDU level, however, end-to-end significance would be obtained if the converter does not send the ack-PDU until itself has received a corresponding ack-PDU from the final destination.

3.3.1.2. PDU Level Protocol Conversion Using the Subsystem Construction Approach

A PDU level converter can be constructed directly by using subsystem construction methods as mentioned in Chapter 1. Given the required service specification Sc and two protocols D (D1, D2) and F (F1, F2), the problem is to find a PDU level converter C such that

\[\text{D1} || \text{Chd} || \text{C} || \text{Chf} || \text{F2} \text{ satisfy } \text{Sc}\]

as depicted in Fig.1.4. Since D1, Chd, Chf, and F2 are given, this expression can be represented in the form of the subsystem construction problem discussed in Section 3.1 as:

\[\text{M0} || \text{C} \text{ satisfy } \text{Sc}\]

where \(\text{M0} = \text{D1} || \text{Chd} || \text{Chf} || \text{F2}\).

The subsystem construction method proposed in [Calv89] is specially designed for protocol conversion, since the missing subsystem (i.e., the converter) is assumed not to interact directly with the environment (the higher layer). It is postulated that M0 satisfies Sc in respect to a safety property and a progress property, which actually constitute the testing preorder defined in Chapter 2. The algorithm to find C is divided into two phases. In the first phase, the state set and transition relations of C are constructed inductively by searching M0 and Sc under the safety constraints, starting with their initial states. The result is a specification of C with the maximum trace set satisfying the safety property. In the second phase, states of C that violate the progress constraint in its interaction with M0 are iteratively removed. If C is not empty after the algorithm terminates, then C is the solution, and it is the most general solution in the sense that for any other solution C’, \(\text{Tr}_{\Sigma_C}(C') / \text{Tr}_{\Sigma_C}(C)\). If C is empty, no solution exists.

The strength of this approach is that a given desired service is guaranteed to be satisfied. In comparison with the service level conversion, a converter will be found if it exists, even if no corresponding service adapter exists. A simple example is depicted in Fig.3.5(b). If a
PDU level converter is used, the end-to-end significance of acknowledgment would be obtained if the converter does not send the ack-PDU until itself has received a corresponding ack-PDU from the final destination.

A disadvantage of this approach is the fact that the solution may contain superfluous states and transitions that may be harmful for system performance. An example given in [Calv89] shows that even when a message is received correctly, the converter may respond with a wrong acknowledgment, which forces the sender to retransmit the message. This problem is due to the fact that the solution constructed by the method is the most general, as was already observed in [Merl83]. Moreover, the states to be checked by the method may grow exponentially with the size of the state space of M0. Further, channel specifications are part of the input to the algorithm. Hence, the algorithm is more complex compared with other top-down methods. Another limitation is that the service must be deterministic.

3.3.1.3. From Service Level Conversion to PDU Level Conversion

If a service adapter has been constructed, one may derive a PDU level converter by correctness preserving transformations from the service adapter. The principle of this method is based on the observation that if the interworking system in Fig.1.4 and Fig.3.3 provide the same required service Sc, then the PDU level converter C should be equivalent to D2||Cx||F1 [Boch90b]. These transformations can be realized by composing the three subsystems D2, F1 and Cx into one module, i.e., C = D2||Cx||F1 such that the equivalence relation is retained after the transformations. The service primitives of D2 and F1 are not visible in the final result C.

Several methods have been already reported based on this principle [Boch90a, Kris93, Okum90]. In [Boch90a], an AB protocol and a NS protocol are used as an example to illustrate this idea. It is also shown that different PDU level converters can be derived for the same pair of protocols if different service adapters are used.

In [Okum90], a two-stage approach was developed to derive protocol converters. In the first stage, Merlin-Bochmann's subsystem construction algorithm [Merl83] is used to derive a service adapter from the service specifications. In the second phase, a PDU level protocol converter is constructed by directly composing the service adapter and the underlying protocol specifications. Since the trace equivalence relation is used in the first stage, the obtained conversion system may lead to a deadlock state. Okumura discussed under what
conditions the constructed conversion system will inherit the properties from the original protocols.

It is possible that the constructed converter, $D2||Cx||F1$, may contain some transitions that are never executed in the system $D1||Chd||C||Chf||F2$. An efficient algorithm was presented in [Kris93] to remove the superfluous states and transitions. The basic idea is to remove from $D2$ and $F1$ those transitions (and related states) that have unmatched service primitives as events and those that can be reached only from the unmatched service primitives; then the algorithm computes and retains the strongly connected component starting with the initial state, and discards the rest of the machine.

Since this method is based on a service adapter, simplicity is its main advantage. In addition, as presented in [Kris93], non executable states and transitions can be removed by an efficient algorithm. End-to-end semantics of control messages can be maintained under certain conditions [Kris93] by correctness preserving transformations.

This method has the same limitations as the method discussed in the last section.

### 3.3.2. Bottom-up Methods

The bottom-up methods begin by analyzing low level functions of the protocols to be converted in order to find PDU level constraints, and these constraints are used to construct a converter by the algorithms described below. These algorithms can be divided into three subclasses:

a) The methods based on a partial specification of the converter, which may be obtained by using some heuristic method.

b) The methods based on message mapping relations between the protocols.

c) The methods based on the combination of message mapping relations and a partial specification of the converter.

#### 3.3.2.1. Methods Based on a Partial Specification of the Converter

##### 3.3.2.1.1. Conversion Seed Method

The basic principle of the method was presented in [Okum87], where a partial specification of a converter is called a conversion seed. A conversion seed, given in the form of a Communication Finite State Machine (CFSM), specifies the order in which the events
should be executed by the converter. This constitutes a partial behavior of the converter. Assuming Xe is a conversion seed, the idea to construct a full converter is as follows:

- Construct the Cartesian cross-product of D2 and F1, denoted as M0. Since M0 contains all the sequences of D2 and F1, it guarantees that a converter derived from M0 is a reduced CFSM of D2 in its communication with D1, and a reduced CFSM of F1 in its communication with F2.

- The constructed converter must satisfy the semantic constraint defined by the conversion seed Xe. Therefore, the sequences of M0 that violate the constraint should be removed. This can be done by combining M0 and Xe to form a new FSM C as follows. Assuming M0 = (Q0, Σ0, δ0, p0) and Xe = (Qx, Σx, δx, q0x) with Σx/Σ0, then C = (Q0 × Qx, Σ0, δp, (p0, q0x)) where δp is the transition function defined on Q0×Qx such that for any p ∈ Q0, and qx ∈ Qx:
  1) (p, qx) − e → (p', qx) if p− e → p' , and e ∈ Σ0∪ {τ} − Σx;
  2) (p, qx) − e→ (p', qx')iff p− e → p’ and qx− e→ qx’ and e ∈ Σ0∩Σx

- The C constructed above can not guarantee the general correctness properties such as the freedom from deadlock and unspecified reception. The unspecified reception is a situation in which no reception event is specified for a message when the message appears at the head of a channel. In order to construct an error-free converter, these properties should be checked, and any error should be corrected by using the assumption that the two protocols are free from deadlock and unspecified reception.

This method provides an efficient formal procedure to construct a protocol converter if a conversion seed is given. The limitation of the method is that the algorithm needs a conversion seed to guarantee semantic correctness, as discussed above. However, the method provides no systematic procedure to find one. Hence constructing a conversion seed can be done only heuristically - a difficult task for complex protocols. If the algorithm fails in producing a converter, it is hard to know whether the problem is due to the hard-mismatch property of the two protocols or the way the conversion seed is specified. In the later case, the designer should try with an alternative conversion seed.

3.3.2.1.2. Reducing Complexity by Decomposition and Merging Functions
In [Yao90], a modular approach to construct a protocol converter was proposed which may reduce the complexity of finding a conversion seed by decomposing and merging protocol functions. The approach was triggered by the general principle of divide-and-conquer and by the observation that many communication protocols go through different phases, corresponding to distinct protocol functions. A protocol D is decomposable if D can be constructed through a series of safe merge operations on a set of distinct functions, where no two functions share any common states except the initial and final states of the two functions. The safe merge operation on the two adjacent functions Func1 and Func2 is defined as the operation which joins the final state of Func1 with the initial states of Func2, and the generated function is a safe function [Yao90]. The basic steps of the construction are as follows:

- Decomposition of the protocols into distinct functions such that the protocols can be constructed by the set of functions through a series of safe merge operations.

- Identify heuristically the distinct function pairs that are equivalent in semantics, called compatible functions. The information of how each pair of compatible functions should interoperate needs to be specified as a CFSM, which is a conversion seed for this pair of functions. For the functions that are not convertible, called local functions, the order in which they should be executed by the converter needs to be specified as well.

- Based on the intended interworking information, the algorithm proposed by Okumura [Okum86] is used for each pair of functions to derive a corresponding function converter.

- The information from the first step is used to merge the local functions selected by the designer with the function converters constructed in the last step to form a final converter. The constructed protocol converter not only includes all the function converters that provide the common subset of functions for the two protocols, but also contains the necessary local functions to guarantee correctness. The states or transitions of the final converter that cause deadlock or unspecified reception are removed.

The advantage of this method is that it reduces the complexity of finding a single conversion seed, as proposed in [Okum86]. After decomposing the protocols into separate
functions, it is easier to identify the compatible functions and the intended interworking information. If a converter does not exist, this method may be able to locate the problem.

However, the decomposition and merge operations are not completely formal, which means ingenuity and intuitive understanding of each protocol function is required. The techniques reported in [Chow85] and [Choi86] may be applied to partly solve the protocol decomposition problem in certain special cases. Moreover, the problem of formally deriving (and validating) a conversion seed has not been solved, although the application of this method may be simpler than the method of [Okum86] because the intuitive step of finding a conversion seed needs to be applied to simpler functions only.

3.3.2.2. Methods Based on Message Relations

In [Shu91], a method for indirect conversion was proposed. Its motivation is that for many practical converters, the messages produced by one protocol must first be stored in a non-FIFO queue, then reordered and converted into the messages of the other protocol. Thus a converter can be realized by combining D2 and F1 with two non-FIFO queues as shown in Fig.3.6. The Put event inserts messages successfully received by the protocol entity D2 (or F1) into the buffer at the converter; and the Get event retrieves the messages in a proper order from the buffer, when available, converts and passes them to the other protocol entity F1 (or D2). The order in which messages should be retrieved from the buffer by a Get event is decided by the message to be sent in F1 (or D2) and the message mapping relation. The basic principle that guides how to insert a Put or Get event is: a Put event should be inserted into the place where a message is successfully received by a protocol entity; a Get event should be inserted into the place where a protocol entity needs to send a message. A detailed algorithm for inserting the Get and Put events into D2 and F1 is developed in [Shu91], where some tricks are used to insure the safety of the conversion system after the Get and Put events are inserted.

![Fig.3.6 an indirect converter](image-url)
However, this construction does not guarantee general correctness properties such as the absence of: 1) buffer overflow, i.e., the number of messages stored in the buffer exceeds its capacity; 2) improper termination, i.e., the protocol terminates with some message leftover in the buffer; and 3) deadlocks. The events that violate the general correctness properties should be removed. In [Shu91] an efficient algorithm was proposed to perform this task by checking only a partial reachability graph of C.

A similar method was proposed in [Shu89]. The main difference is the fact that the Put and Get events inserted into D2 and F1 can communicate synchronously (directly coupled), instead of using a non-FIFO buffer. Hence this is called a synchronization method.

The constructed converter is composed of two CFSMs that represent concurrent processes which communicate through the non-FIFO buffers. The non FIFO feature of the buffers makes it easier to convert the messages that need reordering. The architecture of such a converter makes it convenient to split and implement the gateway into two half-gateways in order to allow each network to gain complete control over its own protocols. As other methods based on PDU level constraints, this method does not present how to obtain a proper message mapping set.

3.3.2.3. Method Based on a Message Relation and a Partial Specification of the Converter

In [Raja91], an algorithm was proposed to derive protocol converters based on executable protocol traces constrained by message mapping relations and a semantic specification. The compatible messages of the two protocol entities D2 and F1 are specified as two sets $\Sigma_r$ (the set of compatible receiving events) and $\Sigma_s$ (the set of compatible sending events), and a message mapping function is defined as $\mathcal{C}: \Sigma_r \times \Sigma_s$. The set of non compatible messages is called $\Sigma_n$. A semantic specification defines the expected execution order for certain messages from $\Sigma_r$, $\Sigma_s$ and $\Sigma_n$. This is similar to a conversion seed, but it is specified in English. The starting point of this method is to derive a set of message mapping relations and semantic specifications by identifying common protocol functions within different phases. The key concept of this method is the definition of a legal trace, denoted $s$ which is an executable trace on the Cartesian cross-product of two protocol entities D2 and F1, provided the following conditions are satisfied:
1) Events of $s$ belong to sets $\Sigma_r$, $\Sigma_s$ and $\Sigma_n$; $s$ satisfies the constraints specified by the message mapping function and the semantic specification, and is at least of length one.

2) For every event $s(i) \in \Sigma_r$ there exists one and only one event $s(j) = \mathcal{C}(s(i))$ with $j > i$, where $s(i)$ and $s(j)$ denote the $i$th and $j$th events of the trace $s$, respectively. This condition implies that in a legal trace a message from $\Sigma_r$ can not be selected more than once, thus the trace length is restricted. It also implies that a trace never ends with a message from $\Sigma_r$.

3) For events $s(i_1) \in \Sigma_r$ and $s(i_2) \in \Sigma_r$ with $i_2 > i_1$, there exists event $s(j_1) = \mathcal{C}(s(i_1))$ and $s(j_2) = \mathcal{C}(s(i_2))$ with $j_2 > j_1$. This condition implies that FIFO must be satisfied for all message mapping, hence application of the method is limited.

The algorithm used to construct a converter consists of four sub-algorithms, called Composite, Trace, Synthesize, and Verify:

- The *Composite* algorithm constructs a state space $S_0$ formed by the Cartesian cross-product of two protocol entities $D_2$ and $F_1$, which produces the maximum sequences of the converter to be constructed. However, no semantic constraints are imposed on the messages sequences of $D_2$ and $F_1$.

- The *Trace* algorithm then searches exhaustively the state space $S_0$ to construct a set of legal traces. The first legal trace is constructed starting at the initial state of $S_0$; and subsequent legal traces may begin from an end state or a begin state of a legal trace since the set of traces is prefix closed. The search is constrained by the legal trace definition, and is limited only to the reachable states of $S_0$. The messages from the set $\Sigma_n$ may be included in a legal trace if the given constraints are not violated.

- The *Synthesize* algorithm constructs a communicating finite state machine (CFSM) $C$ by synthesizing the given legal trace set.

- The CFSM $C$ derived in the previous step may violate the general correctness properties discussed in Section 2.3. The *Verify* algorithm checks that the converter is free from unspecified receptions, deadlocks, and livelocks. As proved in [Raja91], the verification can be done by checking only $C$, similar to that
proposed in [Shu91], because it is assumed that the protocols are free from unspecified reception, deadlock and livelock.

If some errors are found in C during Verify, then a valid converter is not possible with the given constraints (the semantic specification and the message mapping function), but a different choice of the constraints may still lead to a valid converter. If no error found, then C is the correct converter.

This method is conceptually similar to the method presented in [Yao90]. Both methods start with identifying protocol functions under different phases to derive the message relationships and execution orders of (certain) events, which are called the intended interworking specification in [Yao90], and the message mapping function and the semantic specification in this method. Each function converter in [Yao90] corresponds to the legal trace. Both methods need merge events: legal traces are synthesized in this method and the function converters and the local functions are merged in [Yao90]. The difference is that an intended interworking specification is specified as a CFSM and is used as a conversion seed, while the message mapping relation in this method is specified as a message mapping set, and the semantic specification is given in English, which makes the automation of the Trace algorithm impractical. The Verify algorithm is simple, as it is based on the similar idea presented in [Okum86]. Compared with the method presented in [Shu91], the semantic specification may make the design more flexible since the designer is able to impose more semantic constraints (not only the message mapping relation) on the execution order of the converter.

However, this method has the same difficulty as discussed in [Okum86]: deriving the semantic specification is not easy since the method provides no formal procedure to do it. Therefore, an intuitive understanding of the protocols is needed to find a semantic specification. In addition, as the Trace algorithm searches the whole state space S0 to find the legal traces, the complexity of computation is exponential with the size of the state space. The FIFO assumption of Condition 3 in the legal trace definition and the one-to-one correspondence of compatible messages also limit the application of the method.

3.3.2.4. Deriving Message Relations by Projection

In [Lam88], a method was proposed to find a common subset of functions from two protocols, called an image protocol, by protocol projection techniques [Lam84]. This is
achieved by projecting a protocol onto an image protocol. The projection is a way to focus on those aspects of each protocol (components) that are relevant to the properties to be proved. An image protocol is derived from a given protocol by partitioning the state set of each protocol component into a number of blocks in such a way that states considered equivalent, relevant to the properties to be proved, belong to the same block; states in the same block of the partition are indistinguishable in the image of that component and form a single state in the image protocol. This defines a mapping from each state in the component to a state in the image component. The state space partition implies an equivalence relation on the set of messages sent and received in the protocol. Messages whose receptions cause the same image state transitions are considered equivalent, and are mapped to the same image message. Messages that cause no change in the image state of their receiver do not appear in the image protocol at all, and are said to have a null image. An image protocol is specified as a CFSM, which is a sub-machine of the original protocol. Therefore the image protocol is said to have a resolution lower than that of its original protocol. It should be pointed out that, to perform a partition of the state space of the original protocol, ingenuity and intuitive understanding of each state is necessary in order to find out equivalent states and messages relevant to the properties to be proved.

Suppose protocols D and F can each be projected onto a common image protocol, say $P_{ci}$, then $P_{ci}$ embodies some functionality that is common to D and F. Each protocol has properties corresponding to those of $P_{ci}$. On the other hand, messages that have null image in the projection have no meaning with respect to the common functionality represented by $P_{ci}$. The common image $P_{ci}$ defines a semantic correspondence between states and messages of D and states and messages of F.

This concept of a common image protocol can be applied to the construction of protocol converters. If a common image with adequate functionality can be found, a converter can be specified directly, since the common image $P_{ci}$ defines a semantic correspondence between states and messages of D and states and messages of F. This equivalence can be easily implemented by a memoryless converter: whenever the converter receives a message, immediately it forwards a message of the other protocol that has the same image. Clearly null messages will be ignored. Lam proved that the conversion system has safety properties corresponding to those of the common image [Lam88]. If the image is well formed [Lam88] in each projection, then the correspondence holds for progress properties as well. Therefore the conversion problem focuses on finding a common image protocol that satisfies a given global service specification.
We may argue that only a common image protocol is *often* not enough for constructing a protocol converter. For most practical protocols, other functions should be added as pointed out in [Yao91]. However, finding a common image protocol is a good way to find a set of significant message mapping relations that may be used by other conversion methods.