

Chapter 2

Localization: Fundamental Results

2.1 Supervisory Control Theory

We start with a review of the basic concepts and results of supervisory control theory (SCT) [Won14b]. First we introduce events, strings, and languages. Let Σ be a nonempty finite alphabet of symbols $\sigma, \alpha, \beta, \dots$. These symbols will denote *events*, and Σ the *event set*. A *string* $s = \sigma_1\sigma_2 \cdots \sigma_k, k \geq 1$, is a finite sequence of events, just as an English word consists of a sequence of letters. There is a special string ϵ , the *empty string*, meaning ‘say or do nothing’. Now let Σ^* be the set of all finite-length strings including ϵ . A *language* L is an arbitrary subset of strings in Σ^* , i.e. $L \subseteq \Sigma^*$.

A string s_1 is a *prefix* of another string s , written $s_1 \leq s$, if there exists s_2 such that $s_1s_2 = s$. Both ϵ and s are prefixes of s . The *prefix closure* of a language L , written \bar{L} , is

$$\bar{L} := \{s_1 \in \Sigma^* \mid (\exists s \in L) s_1 \leq s\}. \quad (2.1)$$

It is always true that $L \subseteq \bar{L}$. We say that a language L is *closed* if $L = \bar{L}$.

For a subset $\Sigma_o \subseteq \Sigma$, the *natural projection* $P : \Sigma^* \rightarrow \Sigma_o^*$ is defined according to

$$\begin{aligned} P(\epsilon) &= \epsilon; \\ P(\sigma) &= \begin{cases} \epsilon, & \text{if } \sigma \notin \Sigma_o, \\ \sigma, & \text{if } \sigma \in \Sigma_o; \end{cases} \\ P(s\sigma) &= P(s)P(\sigma), \quad s \in \Sigma^*, \sigma \in \Sigma. \end{aligned} \quad (2.2)$$

Thus P ‘erases’ any events not in Σ_o . Extend P to $P : Pwr(\Sigma^*) \rightarrow Pwr(\Sigma_o^*)$, where $Pwr(\cdot)$ denotes *powerset*; and write $P^{-1} : Pwr(\Sigma_o^*) \rightarrow Pwr(\Sigma^*)$ for the *inverse-image function* of P . Let $L_i \subseteq \Sigma_i^*, i = 1, 2$, and bring in natural projections $P_i : (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma_i^*$. The *synchronous product* of L_1 and L_2 , written $L_1 || L_2$, is defined according to $L_1 || L_2 := P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$. Synchronous product $||$ is commutative and associative, and we use it to compose languages.

A finite-state *automaton* \mathbf{G} is a five tuple

$$\mathbf{G} := (Q, \Sigma, \delta, q_0, Q_m) \quad (2.3)$$

where Q is the finite state set, $q_0 \in Q$ the *initial state*, $Q_m \subseteq Q$ the set of *marker states*, Σ the finite *event set*, and $\delta : Q \times \Sigma \rightarrow Q$ the (partial) *state transition function*. We extend δ such that $\delta : Q \times \Sigma^* \rightarrow Q$, and write $\delta(q, s)!$ to mean that $\delta(q, s)$ is defined. There is an algebraic construction presented in Appendix A whereby an automaton serves as a *recognizer* for a given language L . The key concept, the *Nerode equivalence relation* on Σ^* , will be used in Sect. 2.6.

The *closed behavior* of \mathbf{G} is the language

$$L(\mathbf{G}) := \{s \in \Sigma^* | \delta(q_0, s)!\}. \quad (2.4)$$

As defined $L(\mathbf{G})$ is closed. The *marked behavior* of \mathbf{G} is the sublanguage

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) | \delta(q_0, s) \in Q_m\} \subseteq L(\mathbf{G}). \quad (2.5)$$

\mathbf{G} is *nonblocking* if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$, namely every string in the closed behavior may be completed to a string in the marked behavior. (The nonblocking property of \mathbf{G} may also be defined in terms of its reachability and coreachability properties.) Since \mathbf{G} may be viewed as generating its closed and marked behaviors, we shall often refer to it as a *generator*.

Let $\mathbf{G}_i := (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i}), i = 1, 2$, be two generators. Their *synchronous product* $\mathbf{G}_1 || \mathbf{G}_2 = \mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ is defined according to

$$\begin{aligned} Q &= Q_1 \times Q_2, \quad \Sigma = \Sigma_1 \cup \Sigma_2, \quad q_0 = (q_{0,1}, q_{0,2}), \quad Q_m = Q_{m,1} \times Q_{m,2}, \\ &(\forall (q_1, q_2) \in Q_1 \times Q_2, \forall \sigma \in \Sigma_1 \cup \Sigma_2) \\ \delta((q_1, q_2), \sigma) &= \begin{cases} (\delta_1(q_1, \sigma), q_2), & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ \& } \delta_1(q_1, \sigma)!\}; \\ (q_1, \delta_2(q_2, \sigma)), & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ \& } \delta_2(q_2, \sigma)!\}; \\ (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)), & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \text{ \& } \delta_1(q_1, \sigma)!\text{ \& } \delta_2(q_2, \sigma)!\}; \\ \text{not defined,} & \text{otherwise.} \end{cases} \end{aligned} \quad (2.6)$$

The closed and marked behaviors of the synchronous product \mathbf{G} satisfy

$$L(\mathbf{G}) = L(\mathbf{G}_1) || L(\mathbf{G}_2), \quad L_m(\mathbf{G}) = L_m(\mathbf{G}_1) || L_m(\mathbf{G}_2).$$

The synchronous product of more than two generators may be similarly defined, and we use it to compose generators.

Now let \mathbf{G} model the plant to be controlled. For control purposes we assume the ‘technology’ that a selected subset of events in Σ may be disabled from occurring. Thus Σ is partitioned into a *controllable* subset Σ_c and an *uncontrollable* subset Σ_u , i.e. $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$, where $\dot{\cup}$ denotes *disjoint union*. Let Γ be the collection of event subsets that always include Σ_u , i.e. $\Gamma := \{\gamma \subseteq \Sigma | \gamma \supseteq \Sigma_u\}$; we call each $\gamma \in \Gamma$ a *control pattern*. A *supervisory control* V for \mathbf{G} is any map $V : L(\mathbf{G}) \rightarrow \Gamma$, associating to each string in $L(\mathbf{G})$ a control pattern. The closed-loop system where

\mathbf{G} is under the control of V is written V/\mathbf{G} , with closed language $L(V/\mathbf{G}) \subseteq L(\mathbf{G})$ defined as follows:

- (i) $\epsilon \in L(V/\mathbf{G})$;
- (ii) if $s \in L(V/\mathbf{G})$ & $\sigma \in V(s)$ & $s\sigma \in L(\mathbf{G})$, then $s\sigma \in L(V/\mathbf{G})$;
- (iii) no other strings belong to $L(V/\mathbf{G})$.

As defined above $L(V/\mathbf{G})$ is closed. Let $M \subseteq L_m(\mathbf{G})$; V is a *marking supervisory control* for (M, \mathbf{G}) if the marked language $L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap M$. V is *nonblocking* if $\overline{L_m(V/\mathbf{G})} = \overline{L(V/\mathbf{G})}$.

Let $K \subseteq L_m(\mathbf{G})$ be a specification language that imposes some behavioral constraint on the plant \mathbf{G} . The objective of supervisory control is to synthesize V such that $L_m(V/\mathbf{G}) = K$. It turns out that *controllability* is the key concept. K is *controllable* with respect to \mathbf{G} if

$$\overline{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K}. \quad (2.7)$$

So a string cannot ‘skid out’ of \overline{K} on any uncontrollable event. The following is the main result of SCT.

Theorem 2.1 *Let $K \subseteq L_m(\mathbf{G})$, $K \neq \emptyset$. Then there exists a marking nonblocking supervisory control V for (K, \mathbf{G}) such that $L_m(V/\mathbf{G}) = K$ if and only if K is controllable.*

Whether or not a specification language K is controllable, we denote by $\mathcal{C}(K)$ the family of all controllable sublanguages of K , i.e.

$$\mathcal{C}(K) := \{K' \subseteq K \mid K' \text{ is controllable}\}. \quad (2.8)$$

$\mathcal{C}(K)$ is nonempty because the empty language \emptyset is controllable trivially. Further, controllability is closed under arbitrary set union: if K_i is controllable for all i in some index set I , then so is the union $\cup\{K_i \mid i \in I\}$. As a result, $\mathcal{C}(K)$ contains a (unique) supremal element

$$\sup \mathcal{C}(K) := \cup\{K' \mid K' \in \mathcal{C}(K)\}.$$

Theorem 2.2 *Let $K \subseteq L_m(\mathbf{G})$ and $K_{\sup} := \sup \mathcal{C}(K)$. If $K_{\sup} \neq \emptyset$ then there exists a marking nonblocking supervisory control V for (K_{\sup}, \mathbf{G}) such that $L_m(V/\mathbf{G}) = K_{\sup}$.*

Standard (polynomial) algorithms and software, based on automata, are available to compute the supremal sublanguage K_{\sup} : a generator **SUP** is computed with $L_m(\mathbf{SUP}) = K_{\sup}$ and $L(\mathbf{SUP}) = \overline{K_{\sup}}$. We use *TCT* [Won14a] for this and all subsequent computations in the monograph. The generator **SUP** is called the optimal (i.e. maximally permissive) and nonblocking *monolithic supervisor* for the pair

(G, K) ; the implementation of **SUP** indicated in Fig. 2.1 is achieved concretely by synchronous product with G .

We illustrate the monolithic supervisory control synthesis and architecture by a typical DES example: Transfer Line [Won14b, Sect.4.6]. It will be the running example for many of our subsequent theoretical developments.

Transfer Line. As displayed in Fig. 2.2 Transfer Line consists of two machines **M1**, **M2** followed by a test unit **TU**; these three agents are linked by two buffers **B1**, **B2** with capacities 3 and 1, respectively. A workpiece entering the system is first processed by **M1** and stored in **B1**, then processed by **M2** and stored in **B2**. A processed workpiece tested by **TU** may be accepted or rejected; if accepted, it is released from the system; if rejected, it is returned to **B1** for reprocessing by **M2**. Thus the structure incorporates ‘material feedback’. Controllable/uncontrollable events and their meanings are given in Fig. 2.2.

The generator models of the agents **M1**, **M2** and **TU** are displayed in Fig. 2.3. The plant G to be controlled is the synchronous product

$$G := M1 || M2 || TU.$$

The control specification is to protect the two buffers **B1** and **B2** against overflow and underflow; the generator models of **B1** and **B2** are displayed in Fig. 2.3. Let

$$SPEC := B1 || B2.$$

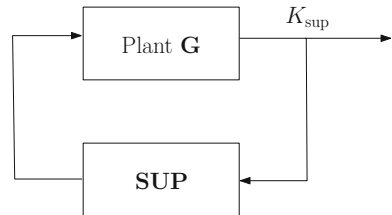
Then the specification language K is $K := L_m(SPEC || G) \subseteq L_m(G)$. Using *TCT* we compute the monolithic optimal and nonblocking supervisor **SUP**, which has 28 states; the control architecture is displayed in Fig. 2.4. This centralized architecture is to be contrasted with the distributed one (for the same Transfer Line) obtained by localization in Sect. 2.5.

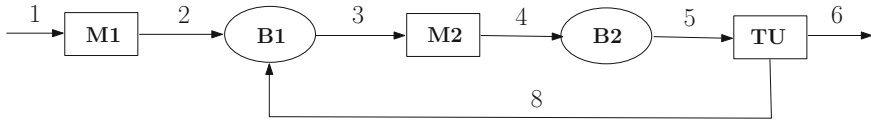
2.2 Distributed Control Problem

Let G be the plant to be controlled, consisting of N (> 1) component agents

$$G_k = (Q_k, \Sigma_k, \delta_k, q_{0,k}, Q_{m,k}), \quad k = 1, \dots, N.$$

Fig. 2.1 Monolithic supervisory control implementation





- | | |
|--|--|
| 1: M1 takes a workpiece from an input bin | 5: TU takes a workpiece from B2 and tests it |
| 2: M1 deposits a processed workpiece in B1 | 6: TU accepts a workpiece and releases it |
| 3: M2 takes a workpiece from B1 | 8: TU rejects a workpiece and returns it to B1 |
| 4: M2 deposits a processed workpiece in B2 | |

$$\Sigma_c = \{1, 3, 5\}, \quad \Sigma_u = \{2, 4, 6, 8\}$$

Fig. 2.2 Transfer Line: system configuration

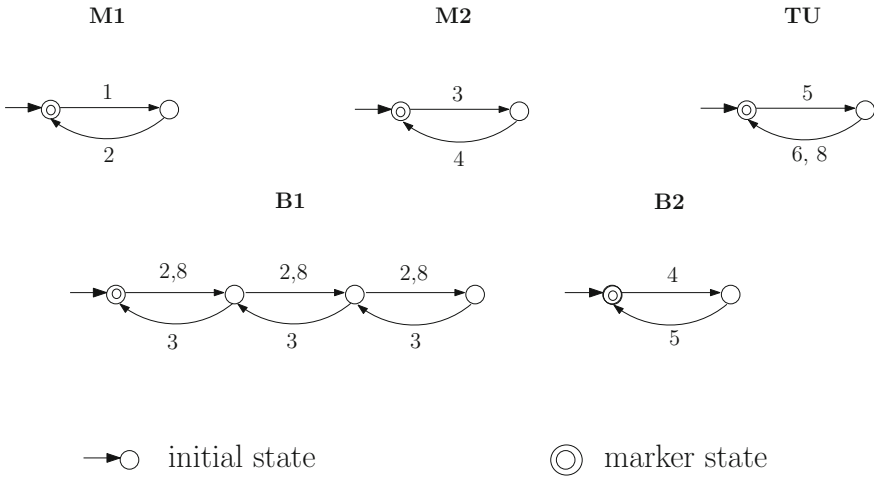


Fig. 2.3 Transfer Line: plant and specification generator models. The notation for initial state and marker state of generators will be used throughout the monograph

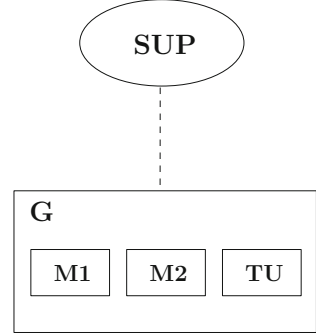
Write \underline{N} for the set of integers $\{1, \dots, N\}$. Then the closed and marked behaviors of \mathbf{G} are

$$L(\mathbf{G}) = \|\{L(\mathbf{G}_k) | k \in \underline{N}\}$$

$$L_m(\mathbf{G}) = \|\{L_m(\mathbf{G}_k) | k \in \underline{N}\}.$$

For simplicity we assume here that the agents' event sets are pairwise disjoint, i.e. $\Sigma_k \cap \Sigma_l = \emptyset$ for all $k, l \in \underline{N}$, $k \neq l$. This is the case in many distributed systems: each agent's (uncontrolled) behavior is independent of others'. In Chap. 3 we shall address the more general case where agents may share events.

Fig. 2.4 Transfer Line:
monolithic supervisory
control architecture



Each agent's event set Σ_k is partitioned into a controllable subset $\Sigma_{c,k}$ and an uncontrollable subset $\Sigma_{u,k}$, i.e. $\Sigma_k = \Sigma_{c,k} \dot{\cup} \Sigma_{u,k}$. Thus the plant \mathbf{G} is defined over $\Sigma := \dot{\cup} \{\Sigma_k | k \in \underline{N}\}$, with controllable event subset $\Sigma_c := \dot{\cup} \{\Sigma_{c,k} | k \in \underline{N}\}$ and uncontrollable subset $\Sigma_u := \dot{\cup} \{\Sigma_{u,k} | k \in \underline{N}\}$.

The component agents $\mathbf{G}_k, k = 1, \dots, N$, are implicitly coupled through a specification language $E \subseteq \Sigma^*$ that imposes a constraint on the global behavior of \mathbf{G} . (E may be the synchronous product of multiple component specifications.) For the plant \mathbf{G} and the imposed specification E , the optimal and nonblocking monolithic supervisor $\mathbf{SUP} = (X, \Sigma, \xi, x_0, X_m)$ is such that (by Theorem 2.2)

$$L_m(\mathbf{SUP}) := \sup \mathcal{C}(E \cap L_m(\mathbf{G})). \quad (2.9)$$

To rule out the trivial case, we assume that $L_m(\mathbf{SUP}) \neq \emptyset$. We also make the following practical assumption, which is the basis for our localization theory in this chapter.

Assumption 2.1 The monolithic supervisor \mathbf{SUP} is feasibly computable.

Now fix $k \in \underline{N}$. We call a generator

$$\mathbf{LOC}_k = (Y_k, \Sigma_{loc,k}, \eta_k, y_{0,k}, Y_{m,k}), \quad \Sigma_{loc,k} \subseteq \Sigma \quad (2.10)$$

a *local controller* for agent \mathbf{G}_k if \mathbf{LOC}_k enables/disables the controllable events in (and only in) $\Sigma_{c,k}$ consistently with the monolithic supervisor \mathbf{SUP} . Let $P_k : \Sigma^* \rightarrow \Sigma_{loc,k}^*$ be the natural projection. Then the above means that for all $s \in \Sigma^*$ and $\sigma \in \Sigma_{c,k}$, there holds

$$P_k(s)\sigma \in L(\mathbf{LOC}_k) \ \& \ s\sigma \in L(\mathbf{G}) \Leftrightarrow s\sigma \in L(\mathbf{SUP}). \quad (2.11)$$

The event set $\Sigma_{loc,k}$ of \mathbf{LOC}_k in general satisfies $\Sigma_k \subseteq \Sigma_{loc,k} \subseteq \Sigma$: local controller \mathbf{LOC}_k possesses all the events of agent \mathbf{G}_k , as well as (in typical cases) some events originating in other agents. Thus, while a local controller's control authority

is strictly local, its observation scope need not, and generally will not, be. Whether or not the required capability of event observation on the part of an agent is feasible in practice will evidently be case-dependent, but need not be burdensome in many applications. An instance could be nearest-neighbor observation, as for motorists maneuvering through a congested intersection. With local controllers embedded, each agent acquires strictly local control and generally non-local observation strategies; the latter are critical to achieve the necessary synchronization with other agents, thereby ensuring correct local decisions.

Of particular interest is the subset $\Sigma_{loc,k} \setminus \Sigma_k$, namely those events that appear in local controller \mathbf{LOC}_k but originate in agents other than \mathbf{G}_k . These are *communication* events, defined precisely in Sect. 2.3; they may be critical to achieve synchronization among local controllers. It is worth emphasizing that $\Sigma_{loc,k}$ is not fixed *a priori*, but will be determined systematically, as part of our localization result, to guarantee correct local control.

We are ready to formulate the (optimal and nonblocking) *Distributed Control Problem*.

Problem 2.1 Let Assumption 2.1 hold. Construct a set of local controllers $\{\mathbf{LOC}_k \mid k \in \underline{N}\}$, one for each agent, with

$$L(\mathbf{LOC}) := \bigvee_{k \in \underline{N}} L(\mathbf{LOC}_k) \quad (2.12)$$

$$L_m(\mathbf{LOC}) := \bigvee_{k \in \underline{N}} L_m(\mathbf{LOC}_k) \quad (2.13)$$

such that the following two properties hold:

$$\begin{aligned} L(\mathbf{G}) \cap L(\mathbf{LOC}) &= L(\mathbf{SUP}) \\ L_m(\mathbf{G}) \cap L_m(\mathbf{LOC}) &= L_m(\mathbf{SUP}). \end{aligned}$$

We say that \mathbf{LOC} , satisfying the above two equations, is *control equivalent* to \mathbf{SUP} with respect to \mathbf{G} . Namely we require that the collective behavior under the local controllers be identical to the monolithic controlled behavior, and therefore be globally optimal and nonblocking.

For the sake of easy implementation and comprehensibility, it would be desired in practice that the state sizes of local supervisors (controllers) be very much less than that of their parent monolithic supervisor: for each $k \in \underline{N}$, the state size $|\mathbf{LOC}_k| \ll |\mathbf{SUP}|$ where $|\cdot|$ denotes state size. Inasmuch as this property is neither precise to state nor always achievable, it must needs be omitted from the formal problem statement; in applications, nevertheless, it should be kept in mind.

2.3 Control Covers and Localization Procedure

We solve the Distributed Control Problem by developing a supervisor localization procedure, essentially decomposing the synthesized monolithic supervisor into a set of local controllers. The idea is illustrated in Fig. 2.5. The key concept of localization is *control cover*, adapted from previous work on *supervisor reduction* [VW86, SW04].

Fixing an arbitrary $k \in \underline{N}$, we shall construct a local controller \mathbf{LOC}_k for agent \mathbf{G}_k . For that, we focus on control information concerning only the controllable events in $\Sigma_{c,k}$. The control information of $\Sigma_{c,k}$ is characterized by the following four functions. Recall the plant $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ and the monolithic supervisor $\mathbf{SUP} = (X, \Sigma, \xi, x_0, X_m)$. First define $E : X \rightarrow Pwr(\Sigma)$ according to

$$E(x) = \{\sigma \in \Sigma \mid \xi(x, \sigma)!\}. \quad (2.14)$$

Thus $E(x)$ is the subset of events that are enabled at state x in \mathbf{SUP} . Next define $D_k : X \rightarrow Pwr(\Sigma_{c,k})$ according to

$$D_k(x) = \{\sigma \in \Sigma_{c,k} \mid \neg \xi(x, \sigma)! \ \& \ (\exists s \in \Sigma^*)(\xi(x_0, s) = x \ \& \ \delta(q_0, s\sigma)!\}\}. \quad (2.15)$$

In words, $D_k(x)$ is the subset of controllable events in $\Sigma_{c,k}$ that must be disabled at x in \mathbf{SUP} . This means precisely that an event $\sigma \in D_k(x)$ if and only if σ is not defined at x , but x is reached by a string s such that $s\sigma$ is in the closed behavior $L(\mathbf{G})$, namely after s , event σ is allowed by the plant \mathbf{G} but not by the supervisor \mathbf{SUP} . Note that if $\neg \xi(x, \sigma)!$ and for every $s \in \Sigma^*$, $\xi(x_0, s) = x$ implies $\neg \delta(q_0, s\sigma)!$, then it is irrelevant whether σ is enabled or disabled at x . Thus $\sigma \notin E(x)$ and $\sigma \notin D_k(x)$; in this case we say that x is a ‘don’t care’ state for σ .

Now for marking, define $M : X \rightarrow \{0, 1\}$ according to

$$M(x) = 1 \text{ if and only if } x \in X_m. \quad (2.16)$$

Thus $M(x) = 1$ means that state x is marked in \mathbf{SUP} . Also define $T : X \rightarrow \{0, 1\}$ according to

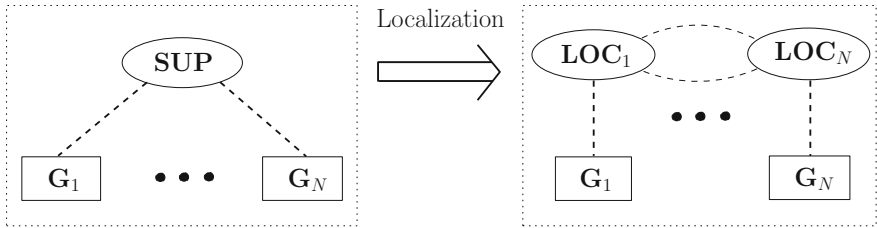


Fig. 2.5 Supervisor localization: decompose the monolithic supervisor into a set of local controllers

$$T(x) = 1 \text{ if and only if } (\exists s \in \Sigma^*) \xi(x_0, s) = x \ \& \ \delta(q_0, s) \in Q_m. \quad (2.17)$$

Hence $T(x) = 1$ means that there is a string that reaches x and also reaches some marker state in \mathbf{G} . Note that for each $x \in X$, it follows from $L_m(\mathbf{SUP}) \subseteq L_m(\mathbf{G})$ that $T(x) = 0 \Rightarrow M(x) = 0$ and $M(x) = 1 \Rightarrow T(x) = 1$.

Based on (2.14)–(2.17) above, we define the following binary relation \mathcal{R}_k on X , called *control consistency*.

Definition 2.1 Let $\mathcal{R}_k \subseteq X \times X$. We say that \mathcal{R}_k is a *control consistency relation* (with respect to $\Sigma_{c,k}$) if for every $x, x' \in X$, $(x, x') \in \mathcal{R}_k$ if and only if

$$E(x) \cap D_k(x') = \emptyset = E(x') \cap D_k(x) \quad (2.18)$$

$$T(x) = T(x') \Rightarrow M(x) = M(x'). \quad (2.19)$$

Informally, a pair of states (x, x') is in \mathcal{R}_k if (i) there is no event in $\Sigma_{c,k}$ that is enabled at x but is disabled at x' , or vice versa (consistent disablement information); and (ii) x and x' are both marked or both unmarked in \mathbf{SUP} (consistent marking information), provided either (1) there exist strings s and s' that reach x and x' , respectively, and both reach some marker state(s) in \mathbf{G} ($T(x) = T(x') = 1$), or (2) no string reaching either x or x' reaches any marker state in \mathbf{G} ($T(x) = T(x') = 0$).

It is easily verified that \mathcal{R}_k is reflexive and symmetric, but in general need not be transitive, and consequently not an *equivalence relation* (see the definition of equivalence relation in Appendix A); an example is given in Fig. 2.6. This fact leads to the following definition of *control cover* (with respect to $\Sigma_{c,k}$). Recall that a *cover* on a set X is a collection of nonempty subsets of X whose union is X .

Definition 2.2 Let $\mathcal{C}_k = \{X_i \subseteq X \mid i \in I_k\}$ be a cover on X , with I_k a suitable index set. We say that \mathcal{C}_k is a *control cover* (with respect to $\Sigma_{c,k}$) if

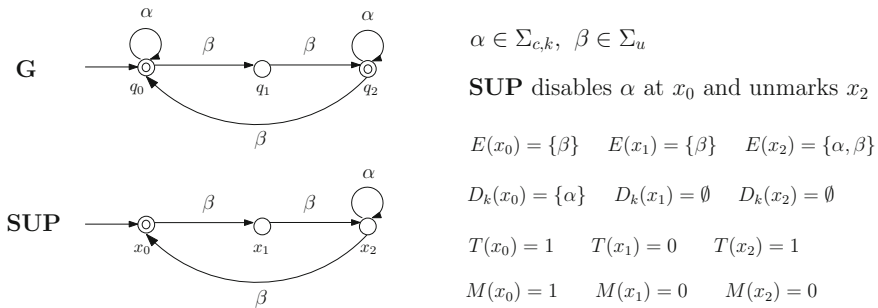


Fig. 2.6 Control consistency relation \mathcal{R}_k is not transitive: $(x_0, x_1) \in \mathcal{R}_k$, $(x_1, x_2) \in \mathcal{R}_k$, but $(x_0, x_2) \notin \mathcal{R}_k$. Indeed, both (2.18) and (2.19) fail for the pair (x_0, x_2) : $E(x_2) \cap D_k(x_0) = \{\alpha\} \neq \emptyset$; $T(x_0) = T(x_2) = 1$ but $M(x_0) \neq M(x_2)$

- (i) $(\forall i \in I_k, \forall x, x' \in X_i) (x, x') \in \mathcal{R}_k$
- (ii) $(\forall i \in I_k, \forall \sigma \in \Sigma) \left[((\exists x \in X_i) \xi(x, \sigma)!) \Rightarrow \right.$
 $\left. ((\exists j \in I_k) (\forall x' \in X_i) \xi(x', \sigma)! \Rightarrow \xi(x', \sigma) \in X_j) \right].$

A control cover \mathcal{C}_k lumps the states of **SUP** into (possibly overlapping) cells X_i , $i \in I_k$ according to (i) all states that reside in a cell X_i must be pairwise control consistent; and (ii) for every event $\sigma \in \Sigma$, all states that can be reached from any state in X_i by a one-step transition σ must be covered by the same cell X_j . Inductively, two states x, x' belong to a common cell of \mathcal{C}_k if and only if (1) x and x' are control consistent; and (2) two future states that can be reached respectively from x and x' by some string are again control consistent.

If a control cover \mathcal{C}_k happens to be a *partition* on X , it is called a *control congruence*; namely its cells are pairwise disjoint. In Sect. 2.6, we shall interpret control congruences in terms of equivalence relations on languages.

Having defined a control cover \mathcal{C}_k on X , we construct a local controller **LOC**_k = $(Y_k, \Sigma_{loc,k}, \eta_k, y_{0,k}, Y_{m,k})$ for agent **G**_k as follows.

(P1) The state set is $Y_k := I_k$, with each state $y \in Y_k$ being the label of a cell X_i of the cover \mathcal{C}_k . In particular, the initial state $y_{0,k}$ represents a cell X_{i_0} containing x_0 , i.e. $x_0 \in X_{i_0}$, and the marker state set $Y_{m,k} := \{i \in I_k | X_i \cap X_m \neq \emptyset\}$.

(P2) To determine the event set $\Sigma_{loc,k}$, first define the transition function $\eta'_k : I_k \times \Sigma \rightarrow I_k$ over the entire event set Σ by $\eta'_k(i, \sigma) = j$ if

$$(\exists x \in X_i) \xi(x, \sigma) \in X_j \ \& \ (\forall x' \in X_i) \left[\xi(x', \sigma)! \Rightarrow \xi(x', \sigma) \in X_j \right]. \quad (2.20)$$

Then choose $\Sigma_{loc,k}$ to be the union of Σ_k of agent **G**_k with events in $\Sigma \setminus \Sigma_k$ which are *not* merely selfloop transitions of η'_k , i.e.

$$\Sigma_{loc,k} := \Sigma_k \dot{\cup} \Sigma_{com,k} \quad (2.21)$$

where

$$\Sigma_{com,k} := \{\sigma \in \Sigma \setminus \Sigma_k \mid (\exists i, j \in I_k) i \neq j \ \& \ \eta'_k(i, \sigma) = j\}. \quad (2.22)$$

As defined $\Sigma_k \subseteq \Sigma_{loc,k} \subseteq \Sigma$. The subset $\Sigma_{com,k}$ determines the events of other agents that need to be communicated to agent **G**_k.

(P3) Define the transition function η_k to be the restriction of η'_k to $\Sigma_{loc,k}$; namely $\eta_k := \eta'_k|_{I_k \times \Sigma_{loc,k}} : I_k \times \Sigma_{loc,k} \rightarrow I_k$. Then η_k is extended to $\eta_k : I_k \times \Sigma_{loc,k}^* \rightarrow I_k$.

These three steps will henceforth be referred to as the *Localization Procedure*. As constructed above **LOC**_k is nonblocking, i.e. $\overline{L_m(\mathbf{LOC}_k)} = L(\mathbf{LOC}_k)$. Indeed, let $s \in L(\mathbf{LOC}_k)$ and $i := \eta_k(i_0, s)$. Then there is $x \in X_i$ such that $x = \xi(x_0, s)$. Since **SUP** is nonblocking, there is $t \in \Sigma^*$ such that $st \in L_m(\mathbf{SUP})$, i.e. $\xi(x, t) \in X_m$. It

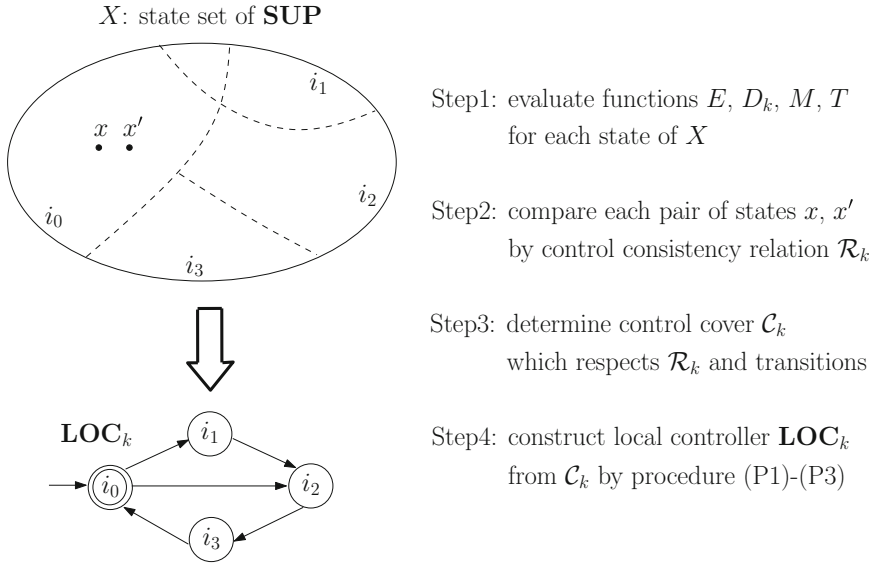


Fig. 2.7 Supervisor localization procedure

follows from \mathcal{C}_k being a control cover (condition (ii) of Definition 2.2) that $\eta'_k(i, t) \in Y_{m,k}$. Let $P_k : \Sigma^* \rightarrow \Sigma_{loc,k}^*$ be the natural projection; we derive $\eta_k(i, P_k(t)) \in Y_{m,k}$, and therefore $s \in \overline{L_m(\text{LOC}_k)}$. We have proved $L(\text{LOC}_k) \subseteq \overline{L_m(\text{LOC}_k)}$, and the reverse containment holds automatically.

Proposition 2.1 *The generator $\text{LOC}_k = (Y_k, \Sigma_{loc,k}, \eta_k, y_{0,k}, Y_{m,k})$ constructed by the Localization Procedure is a local controller for agent \mathbf{G}_k , i.e.*

$$(\forall s \in \Sigma^*, \forall \sigma \in \Sigma_{c,k}) P_k(s)\sigma \in L(\text{LOC}_k) \ \& \ s\sigma \in L(\mathbf{G}) \Leftrightarrow s\sigma \in L(\text{SUP}).$$

We postpone the proof of Proposition 2.1 after our main result below. Note that owing to the possible overlapping of cells in the cover \mathcal{C}_k , the choices of $y_{0,k}$ and η_k may not be unique, and consequently the local controller LOC_k may not be unique. In that case we pick an arbitrary instance of LOC_k . If \mathcal{C}_k happens to be a control congruence, however, then LOC_k is determined uniquely.

We have now completed the localization procedure for an arbitrarily chosen agent $\mathbf{G}_k, k \in \underline{N}$; the procedure is summarized and illustrated in Fig. 2.7. Applying the same procedure, we obtain a set of local controllers

$$\text{LOC} := \{\text{LOC}_k | k \in \underline{N}\}$$

one for each agent \mathbf{G}_k . Let $L(\text{LOC})$ and $L_m(\text{LOC})$ be defined as in (2.12) and (2.13), respectively. Then we have the following main result.

Theorem 2.3 *The set of local controllers $\mathbf{LOC} = \{\mathbf{LOC}_k | k \in \underline{N}\}$ constructed above solves the Distributed Control Problem; that is,*

$$L(\mathbf{G}) \cap L(\mathbf{LOC}) = L(\mathbf{SUP}) \quad (2.23)$$

$$L_m(\mathbf{G}) \cap L_m(\mathbf{LOC}) = L_m(\mathbf{SUP}). \quad (2.24)$$

Theorem 2.3 asserts that every set of local controllers constructed from a set of control covers is a solution to the Distributed Control Problem. Of particular interest is then a set of *state-minimal* local controllers (possibly non-unique) that can in principle be defined from a set of suitable control covers. Computation of state-minimal local controllers is, however, NP-hard,¹ and consequently we cannot expect a polynomial algorithm for it. Nevertheless, in Sect. 2.5, we shall design a polynomial algorithm to compute control congruences, instead of control covers, from which the local controllers will be generated.

We now prove Theorem 2.3, in the order first of the (\supseteq) part of (2.23) and (2.24), and then the (\subseteq) part.

Proof of Theorem 2.3. $(\supseteq, 2.24)$ Since $L_m(\mathbf{SUP}) \subseteq L_m(\mathbf{G})$, it suffices to show that $L_m(\mathbf{SUP}) \subseteq L_m(\mathbf{LOC})$. By (2.13) the latter amounts to

$$(\forall k \in \underline{N}) L_m(\mathbf{SUP}) \subseteq P_k^{-1} L_m(\mathbf{LOC}_k)$$

where $P_k : \Sigma^* \rightarrow \Sigma_{loc,k}^*$. Let $k \in \underline{N}$ and $s = \sigma_0 \sigma_1 \cdots \sigma_h \in L_m(\mathbf{SUP})$, $h \geq 0$. Then $x_1 := \xi(x_0, \sigma_0) \in X, \dots, x_h := \xi(x_0, \sigma_0 \cdots \sigma_{h-1}) \in X, x_{h+1} := \xi(x_0, s) \in X_m$. By the construction of \mathbf{LOC}_k , in particular the transition function η'_k over Σ according to (2.20), there exist i_0, i_1, \dots, i_{h+1} (with $i_0 = y_{0,k}$) such that

$$\begin{aligned} x_0 &\in X_{i_0} \quad \& \quad \eta'_k(i_0, \sigma_0) = i_1, \\ x_1 &\in X_{i_1} \quad \& \quad \eta'_k(i_1, \sigma_1) = i_2, \\ &\vdots \\ x_h &\in X_{i_h} \quad \& \quad \eta'_k(i_h, \sigma_h) = i_{h+1}, \end{aligned} \quad (2.25)$$

and $x_{h+1} \in X_{i_{h+1}}$. So $\eta'_k(i_0, \sigma_0 \sigma_1 \cdots \sigma_h) = \eta'_k(i_0, s)!$, and belongs to $Y_{m,k}$ because $X_{i_{h+1}} \cap X_m \neq \emptyset$ (x_{h+1} belongs). Moreover since each $\sigma \in \Sigma \setminus \Sigma_{loc,k}$ (defined in (2.21)) is merely a selfloop transition of η'_k , we derive $\eta_k(i_0, P_k(s)) \in Y_{m,k}$. Hence $P_k(s) \in L_m(\mathbf{LOC}_k)$, i.e. $s \in P_k^{-1} L_m(\mathbf{LOC}_k)$.

$(\supseteq, 2.23)$ This is an easy consequence of $(\supseteq, 2.24)$:

¹In Appendix B we recall the concept of NP-hardness, and prove that the computation of state-minimal local controllers is indeed as stated.

$$\begin{aligned}
L(\mathbf{SUP}) &= \overline{L_m(\mathbf{SUP})} \subseteq \overline{L_m(\mathbf{G})} \cap \overline{L_m(\mathbf{LOC})} \\
&\subseteq \overline{L_m(\mathbf{G})} \cap \overline{L_m(\mathbf{LOC})} \\
&\subseteq L(\mathbf{G}) \cap L(\mathbf{LOC}).
\end{aligned}$$

(\subseteq , 2.23) We show this by induction on the length of strings. First, as it was assumed *a priori* that $L_m(\mathbf{SUP})$ is nonempty, it follows that the languages $L(\mathbf{G})$, $L(\mathbf{LOC})$, and $L(\mathbf{SUP})$ are all nonempty, and as they are closed, the empty string ϵ belongs to each. Now suppose that $s \in L(\mathbf{G}) \cap L(\mathbf{LOC})$ implies $s \in L(\mathbf{SUP})$, and $s\sigma \in L(\mathbf{G}) \cap L(\mathbf{LOC})$, $\sigma \in \Sigma$. It will be proved that $s\sigma \in L(\mathbf{SUP})$. This is immediate when $\sigma \in \Sigma_u$, since $L_m(\mathbf{SUP})$ is controllable. Now let $\sigma \in \Sigma_c$. Since Σ_c is the disjoint union of the $\Sigma_{c,k}$, $k \in \underline{N}$, there exists k such that $\sigma \in \Sigma_{c,k}$. Hence by hypothesis and (2.12), $s, s\sigma \in P_k^{-1}(L(\mathbf{LOC}_k))$, i.e. $P_k(s), P_k(s)\sigma \in L(\mathbf{LOC}_k)$. Write $i := \eta_k(y_{0,k}, P_k(s))$ and $j := \eta_k(i, \sigma)$. By the definition of η_k (and η'_k), there exist $x \in X_i$ and $x' \in X_j$ such that $\xi(x, \sigma) = x'$; hence $\sigma \in E(x)$ (defined in (2.14)). On the other hand, by hypothesis we have $s \in L(\mathbf{SUP})$, i.e. $\xi(x_0, s)!$. As with (2.25) in (\supseteq , 2.24) we derive $\xi(x_0, s) \in X_i$, and by Definition 2.2 of control cover $(x, \xi(x_0, s)) \in \mathcal{R}_k$. It then follows from Definition 2.1 that $\sigma \notin D_k(\xi(x_0, s))$. Since $s\sigma \in L(\mathbf{G})$, i.e. $\delta(\delta(q_0, s), \sigma)!$, we conclude that $\xi(\xi(x_0, s), \sigma)!$, namely $s\sigma \in L(\mathbf{SUP})$.

(\subseteq , 2.24) Let $s \in L_m(\mathbf{G}) \cap L_m(\mathbf{LOC})$; by (2.13), $s \in P_k^{-1}(L_m(\mathbf{LOC}_k))$ for all k , i.e. $P_k(s) \in L_m(\mathbf{LOC}_k)$. Fix $k \in \underline{N}$ and write $i_m := \eta_k(y_0, P_k(s))$. Then there exists $x \in X_{i_m} \cap X_m$; therefore $M(x) = 1$ (defined in (2.16)), which implies $T(x) = 1$ (defined in (2.17)). On the other hand, since $L_m(\mathbf{G}) \cap L_m(\mathbf{LOC}) \subseteq L(\mathbf{G}) \cap L(\mathbf{LOC}) = L(\mathbf{SUP})$ (the last equality has been shown above), we have $s \in L(\mathbf{SUP})$. That is, $\xi(x_0, s)!$; again as with (2.25) in (\supseteq , 2.24) we derive $\xi(x_0, s) \in X_{i_m}$, and by Definition 2.2 of control cover $(x, \xi(x_0, s)) \in \mathcal{R}_k$. Since $s \in L_m(\mathbf{G})$, i.e. $\delta(q_0, s) \in Q_m$, we have $T(\xi(x_0, s)) = 1$. Therefore by Definition 2.1, $M(\xi(x_0, s)) = 1$, i.e. $s \in L_m(\mathbf{SUP})$. \square

We present the proof of Proposition 2.1.

Proof of Proposition 2.1. Let $s \in \Sigma^*$ and $\sigma \in \Sigma_{c,k}$. We must prove (2.11). The (\Leftarrow) direction of (2.11) follows straightforwardly from (2.23) established in Theorem 2.3. For the (\Rightarrow) direction of (2.11), let $P_k(s)\sigma \in L(\mathbf{LOC}_k)$ and $s\sigma \in L(\mathbf{G})$. Similar to (\subseteq , 2.23) in the proof of Theorem 2.3 (refer to notation therein), we have $\sigma \in E(x)$, where $x \in X_i$ and $i = \eta_k(y_{0,k}, P_k(s))$. Since $\xi(x_0, s)$ is also in X_i , we derive $(x, \xi(x_0, s)) \in \mathcal{R}_k$ and $\sigma \notin D_k(\xi(x_0, s))$. The latter means that σ is *not* disabled at $\xi(x_0, s)$ by the monolithic supervisor \mathbf{SUP} . It finally follows from $s\sigma \in L(\mathbf{G})$ that $s\sigma \in L(\mathbf{SUP})$. \square

Remark 2.1 Supervisor localization presented in this section follows a similar procedure to supervisor reduction studied in [SW04]. In reduction, D_k defined in (2.15) is concerned with the disabling action of *all* controllable events, i.e. $D_k : X \rightarrow Pwr(\Sigma_c)$; then the control consistency relation \mathcal{R}_k and control cover \mathcal{C}_k are defined accordingly, both with respect to the whole controllable event set Σ_c . In this way, reduction functions to keep just the ‘backbone’ of all control actions of the mono-

lithic supervisor **SUP**, and reduces **SUP**'s state size by projecting out the transitional constraints already imposed by the plant model **G**.

Localization, on the other hand, does more than just reduce **SUP**'s state size: it also *distributes* **SUP**'s control action to the relevant component agents, thereby creating a distributed control architecture. In this sense, therefore, reduction may be viewed as a boundary case of localization, where there is only one agent, namely the reduced supervisor. The latter may be computed by a polynomial algorithm in [SW04], from which our own localization algorithm in Sect. 2.5 is adapted. Moreover, the resulting reduced supervisor **SIM** is control equivalent to **SUP**, i.e.

$$\begin{aligned} L(\mathbf{G}) \cap L(\mathbf{SIM}) &= L(\mathbf{SUP}) \\ L_m(\mathbf{G}) \cap L_m(\mathbf{SIM}) &= L_m(\mathbf{SUP}). \end{aligned}$$

2.4 Necessity of Control Covers

We have shown that a set of control covers is sufficient to generate (by the Localization Procedure) a solution to the Distributed Control Problem. We show in this section that control covers are in fact necessary: each set of generators that is a solution to the Distributed Control Problem must be generated from a set of suitable control covers.

To establish this result, we need the following two definitions.

Definition 2.3 Let $\mathbf{Z} = (Z, \Sigma, \zeta, z_0, Z_m)$ and $\mathbf{Z}' = (Z', \Sigma, \zeta', z'_0, Z'_m)$ be two generators defined over the same event set Σ . We say that \mathbf{Z} and \mathbf{Z}' are *DES-isomorphic* if there exists a bijective map $\theta : Z \rightarrow Z'$ such that

- (i) $\theta(z_0) = z'_0$ & $\theta(Z_m) = Z'_m$;
- (ii) $(\forall z \in Z, \forall \sigma \in \Sigma) \zeta(z, \sigma)! \Rightarrow (\zeta'(\theta(z), \sigma)! \& \zeta'(\theta(z), \sigma) = \theta(\zeta(z, \sigma)))$.

The map θ is called a *DES-isomorphism*.

In the above definition, (i) specifies bijective relations of the initial and marker states between \mathbf{Z} and \mathbf{Z}' , while (ii) specifies bijective relations of the state transitions.

The next concept, *adaptedness*, is introduced in order to exclude generators that have superfluous states, transitions, and marking. We shall need this concept to prove the necessity of control covers.

Definition 2.4 Let $\Sigma_1 \subseteq \Sigma$ and $P_1 : \Sigma^* \rightarrow \Sigma_1^*$. A generator $\mathbf{Z} = (Z, \Sigma_1, \zeta, z_0, Z_m)$ is *adapted* to $\mathbf{SUP} = (X, \Sigma, \xi, x_0, X_m)$ if

- (i) $(\forall z \in Z)(\exists s \in L(\mathbf{SUP})) \zeta(z_0, P_1(s)) = z$;
- (ii) $(\forall z \in Z, \forall \sigma \in \Sigma_1) \zeta(z, \sigma)! \Rightarrow (\exists s \in L(\mathbf{SUP})) (\zeta(z_0, P_1(s)) = z \& s\sigma \in L(\mathbf{SUP}))$;
- (iii) $(\forall z \in Z_m)(\exists s \in L_m(\mathbf{SUP})) \zeta(z_0, P_1(s)) = z$.

In words, a generator is adapted to **SUP** if (i) each of its states is reachable by some string $P_1(s)$ with $s \in L(\mathbf{SUP})$; (ii) a one-step transition σ is defined at a state z

only if z is reached by a string $P_1(s)$ with s in $L(\mathbf{SUP})$ such that membership of $s\sigma$ in $L(\mathbf{SUP})$ is preserved; and (iii) each of its marker states is reachable by some string $P_1(s)$ with s in $L_m(\mathbf{SUP})$. This concept is illustrated by two examples in Fig. 2.8.

If and when a generator $\mathbf{Z} = (Z, \Sigma_1, \zeta, z_0, Z_m)$ is *not* adapted to \mathbf{SUP} , the following three *adaptation* operations will replace it by one that is.

- (A1) Delete state $z \in Z$, if there does not exist $s \in L(\mathbf{SUP})$ with $\zeta(z_0, P_1(s)) = z$.
- (A2) Delete transition $\zeta(z, \sigma)$, where $\sigma \in \Sigma_1$ and $z \in Z$, if for each $s \in L(\mathbf{SUP})$, $\zeta(z_0, P_1(s)) = z$ implies $s\sigma \notin L(\mathbf{SUP})$.
- (A3) Unmark $z \in Z_m$, if there does not exist $s \in L_m(\mathbf{SUP})$ with $\zeta(z_0, P_1(s)) = z$.

It is straightforward to verify that (A1)–(A3) will convert \mathbf{Z} into a generator adapted to \mathbf{SUP} . Note also that (A1)–(A3) will not increase the state size of \mathbf{Z} . Henceforth, without loss of generality, we shall consider only generators adapted to \mathbf{SUP} .

With adapted generators and DES-isomorphism, we state the following result.

Theorem 2.4 *Let $\mathbf{LOC} := \{\mathbf{LOC}_k = (Y_k, \Sigma_{loc,k}, \eta_k, y_{0,k}, Y_{m,k}) | k \in \underline{N}\}$ be a set of generators adapted to $\mathbf{SUP} = (X, \Sigma, \xi, x_0, X_m)$, and suppose that \mathbf{LOC} is control equivalent to \mathbf{SUP} with respect to $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, i.e. (2.23) and (2.24) are satisfied. Then there exists a set of control covers $\mathcal{C} := \{\mathcal{C}_k | k \in \underline{N}\}$ on X , with a corresponding set of generators $\{\mathbf{Z}_k = (Z_k, \Sigma_{loc,k}, \zeta_k, z_{0,k}, Z_{m,k}) | k \in \underline{N}\}$ constructed by the Localization Procedure, such that for each $k \in \underline{N}$, \mathbf{Z}_k and \mathbf{LOC}_k are DES-isomorphic.*

Theorem 2.4 asserts that every adapted solution to the Distributed Control Problem must be induced (by the Localization Procedure) from some set of control covers under the condition of DES-isomorphism. While Theorem 2.4 is not required for the subsequent computation of local controllers, it provides an important theoretical perspective: namely, it points to the *universality* of control covers as the mechanism of supervisor localization. This underlines the fact that for computational tractability, as explained in Sect. 2.5, (general) covers are traded off for (simpler) congruences.

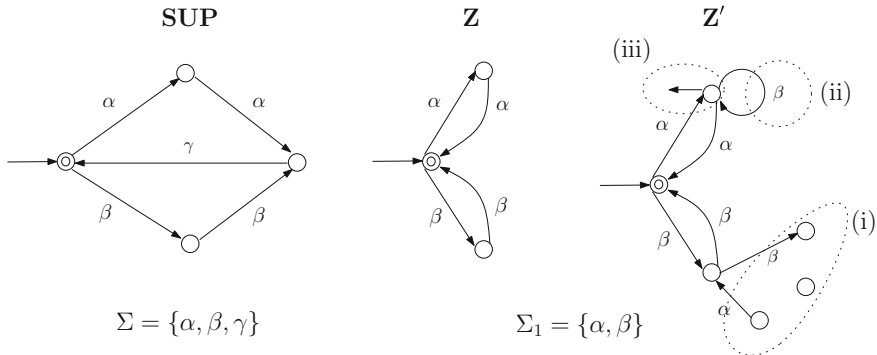


Fig. 2.8 Examples: \mathbf{Z} defined over Σ_1 is adapted to \mathbf{SUP} , but \mathbf{Z}' over Σ_1 is not. Indeed for \mathbf{Z}' , (i), (ii), and (iii) respectively display violation of the three conditions in Definition 2.4

Proof of Theorem 2.4. Fixing $k \in \underline{N}$, we show that there exists a control cover \mathcal{C}_k such that the corresponding induced generator \mathbf{Z}_k is DES-isomorphic to \mathbf{LOC}_k . Let $y \in Y_k$, $P_k : \Sigma^* \rightarrow \Sigma_{loc,k}^*$, and define

$$X(y) := \{x \in X \mid (\exists s \in L(\mathbf{SUP})) \xi(x_0, s) = x \ \& \ \eta_k(y_{0,k}, P_k(s)) = y\}.$$

Since \mathbf{LOC}_k is adapted to \mathbf{SUP} , by (i) of Definition 2.4 there is $s \in L(\mathbf{SUP})$ such that $\eta_k(y_{0,k}, P_k(s)) = y$. Hence $x := \xi(x_0, s)$ belongs to $X(y)$, and $X(y)$ is nonempty. Letting $\mathcal{C}_k := \{X(y) \mid y \in Y_k\}$, we prove three statements: (1) \mathcal{C}_k is a cover on X ; (2) \mathcal{C}_k is a control cover; and (3) \mathbf{Z}_k induced from \mathcal{C}_k by the Localization Procedure is DES-isomorphic to \mathbf{LOC}_k .

For (1), let $x \in X$, and let $s \in L(\mathbf{SUP})$ such that $\xi(x_0, s) = x$. Since \mathbf{LOC} is control equivalent to \mathbf{SUP} with respect to \mathbf{G} , in particular (2.23) holds, we derive $s \in P_k^{-1}L(\mathbf{LOC}_k)$. This implies $\eta_k(y_{0,k}, P_k(s))!$. Letting $y := \eta_k(y_{0,k}, P_k(s))$, we have $x \in X(y)$. Therefore each state in X belongs to a cell of \mathcal{C}_k .

For (2), we refer to Definition 2.2 and need to prove two statements. For the first statement, let $y \in Y_k$ and $x, x' \in X(y)$; we show that $(x, x') \in \mathcal{R}_k$, i.e. (by Definition 2.1)

$$\begin{aligned} E(x) \cap D_k(x') &= \emptyset = E(x') \cap D_k(x) \\ T(x) = T(x') &\Rightarrow M(x) = M(x'). \end{aligned}$$

To show the first condition, let $\sigma \in E(x)$, i.e. $\xi(x, \sigma)!$ (by (2.14)); it will be shown that $\sigma \notin D_k(x')$. This holds trivially, according to (2.15), if $\sigma \in E(x) \setminus \Sigma_{c,k}$. So let $\sigma \in \Sigma_{c,k}$, and assume $x = \xi(x_0, s)$ for some $s \in L(\mathbf{SUP})$. Then $s\sigma \in L(\mathbf{SUP})$, and it follows from (2.23) that $s\sigma \in P_k^{-1}L(\mathbf{LOC}_k)$. Since $P_k(\sigma) = \sigma$, we have $\eta_k(\eta_k(y_{0,k}, P_k(s)), \sigma)!$, and thus $\eta_k(y, \sigma)!$ by the definition of $X(y)$ above. On the other hand, $x' \in X(y)$ implies that there is $s' \in L(\mathbf{SUP})$ such that $\xi(x_0, s') = x'$ and $\eta_k(y_{0,k}, P_k(s')) = y$. Since $\eta_k(y, \sigma)!$, we have $s'\sigma \in P_k^{-1}L(\mathbf{LOC}_k)$. Now if $s'\sigma \notin L(\mathbf{G})$ then $\sigma \notin D_k(x')$ by (2.15). So let $s'\sigma \in L(\mathbf{G})$; according to (2.11), $s'\sigma \in P_l^{-1}L(\mathbf{LOC}_l)$ for each $l \in \underline{N}$ with $l \neq k$, because σ may be disabled only by \mathbf{LOC}_k . Hence $s'\sigma \in L(\mathbf{LOC}) \cap L(\mathbf{G})$, and by (2.23) $s'\sigma \in L(\mathbf{SUP})$, i.e. $\xi(\xi(x_0, s'), \sigma)!$. This implies $\sigma \in E(x')$, and therefore $\sigma \notin D_k(x')$. We have thus proved that $E(x) \cap D_k(x') = \emptyset$; that $E(x') \cap D_k(x) = \emptyset$ is shown by the same argument.

We proceed to verify the second condition. Suppose $M(x) \neq M(x')$; say $M(x) = 1$ and $M(x') = 0$ (the other case follows similarly), then $T(x) = 1$ and it will be shown that $T(x') = 0$. Since $x \in X(y)$, $x \in X_m$, and \mathbf{LOC}_k is adapted to \mathbf{SUP} , by (iii) of Definition 2.4 there is $s \in L_m(\mathbf{SUP})$ such that $\xi(x_0, s) = x$ and $\eta_k(y_{0,k}, P_k(s)) = y$. It follows from (2.24) that $s \in L_m(\mathbf{G}) \cap L_m(\mathbf{LOC})$, which implies $y \in Y_{m,l}$ for each $l \in \underline{N}$. On the other hand, $x' \in X(y)$ means by definition that there is $s' \in L(\mathbf{SUP})$ such that $\xi(x_0, s') = x'$ and $\eta_k(y_{0,k}, P_k(s')) = y$. Since $M(x') = 0$, we have $s' \notin L_m(\mathbf{SUP})$. By (2.24) there follows $s' \notin L_m(\mathbf{G})$, and therefore $T(x') = 0$.

Now we show the second statement of Definition 2.2:

$$(\forall y \in Y_k, \forall \sigma \in \Sigma) \left[((\exists x \in X(y)) \xi(x, \sigma)!) \Rightarrow ((\exists y' \in Y_k) (\forall x' \in X(y)) \xi(x', \sigma)!) \Rightarrow \xi(x', \sigma) \in X(y') \right].$$

Let $y \in Y_k$, $\sigma \in \Sigma$, and suppose that there exists $x \in X(y)$ such that $\xi(x, \sigma)!$. Also let $x' \in X(y)$ with $\xi(x', \sigma)!$. Then there is $s \in L(\mathbf{SUP})$ such that $\xi(x_0, s) = x'$ and $\eta_k(y_{0,k}, P_k(s)) = y$. This implies $s\sigma \in L(\mathbf{SUP})$, and by (2.23) $s\sigma \in P_k^{-1}L(\mathbf{LOC}_k)$. Let $y' := \eta_k(y_{0,k}, P_k(s\sigma))$; then $\xi(x', \sigma) = \xi(x_0, s\sigma) \in X(y')$.

Finally, for (3), we show that \mathbf{Z}_k and \mathbf{LOC}_k are indeed identical; namely they are DES-isomorphic with the identity map the DES-isomorphism. Since the control cover \mathcal{C}_k is given by $\mathcal{C}_k := \{X(y) | y \in Y_k\}$, by (P1) of the Localization Procedure we have $Z_k = Y_k$ and $z_{0,k} = y_{0,k}$. Also the marker state set $Z_{m,k}$ is given by $Z_{m,k} := \{y \in Y_k | X(y) \cap X_m \neq \emptyset\}$.

To see that $Z_{m,k} = Y_{m,k}$, first let $y \in Y_{m,k}$; since \mathbf{LOC}_k is adapted to \mathbf{SUP} , there is $s \in L_m(\mathbf{SUP})$ such that $\eta_k(y_{0,k}, s) = y$. This implies that $\xi(x_0, s)!$ and $\xi(x_0, s) \in X_m$. Hence $X(y) \cap X_m \neq \emptyset$ (because $\xi(x_0, s)$ belongs to both sets), and $y \in Z_{m,k}$. Conversely, let $y \in Z_{m,k}$; then $X(y) \cap X_m \neq \emptyset$. Let $x \in X(y) \cap X_m$; then there is $s \in L_m(\mathbf{SUP})$ such that $\xi(x_0, s) = x$ and $\eta_k(y_{0,k}, s) = y$. Hence by (2.24) $s \in L_m(\mathbf{LOC}_k)$, and $y \in Y_{m,k}$.

It is left to show $\zeta_k = \eta_k$, i.e. for each $y \in Y_k$ and $\sigma \in \Sigma_{loc,k}$, $\zeta_k(y, \sigma) = \eta_k(y, \sigma)$; here ζ_k is defined in (P2) and (P3) of the Localization Procedure. First let $y' = \zeta_k(y, \sigma)$; then there is $x \in X(y)$ such that $\xi(x, t) \in X(y')$ for some $t \in \Sigma^*$ with $P_k(t) = \sigma$. Hence there is $s \in L(\mathbf{SUP})$ such that $\xi(x_0, s) = x$, $\eta_k(y_{0,k}, P_k(s)) = y$, and $\eta_k(y_{0,k}, P_k(s)\sigma) = y'$. Therefore $\eta_k(y, \sigma) = y'$. Conversely, let $y' = \eta_k(y, \sigma)$. Since \mathbf{LOC}_k is adapted to \mathbf{SUP} , by (ii) of Definition 2.4 there is $s \in L(\mathbf{SUP})$ such that $\eta_k(y_{0,k}, P_k(s)) = y$ and $s\sigma \in L(\mathbf{SUP})$. It follows that $\xi(x_0, s) \in X(y)$, $\eta_k(y_{0,k}, P_k(s)\sigma) = y'$. Therefore $\xi(\xi(x_0, s), \sigma) \in X(y')$, and $\zeta_k(y, \sigma) = y'$. \square

2.5 Localization Algorithm

In this section we present a polynomial algorithm to compute a control congruence, rather than a control cover, from which a local controller is generated. We call this algorithm the *Localization Algorithm*; it is a generalization of the reduction algorithm in [SW04].

We sketch the idea of the Localization Algorithm as follows. Given the supervisor $\mathbf{SUP} = (X, \Sigma_c \dot{\cup} \Sigma_u, \xi, x_0, X_m)$, let the states in X be labeled as $X = \{x_0, \dots, x_{n-1}\}$ and $\Sigma_{c,k} \subseteq \Sigma_c$ be the set of controllable events of agent \mathbf{G}_k ($k \in \underline{N}$ fixed). The algorithm will generate a control congruence \mathcal{C}_k on X with respect to $\Sigma_{c,k}$.

Initially \mathcal{C}_k is set to be the singleton partition on X , i.e.,

$$\mathcal{C}_k := \{[x] \subseteq X \mid [x] = \{x\}\} = \{[x_0], \dots, [x_{n-1}]\}$$

where $[x]$ denotes the cell in \mathcal{C}_k to which x belongs. Then the algorithm ‘merges’ two cells $[x_i]$ and $[x_j]$ into one if for every $x'_i \in [x_i]$ and $x'_j \in [x_j]$, x'_i and x'_j , as well as all their corresponding future states reachable by identical strings, are control consistent in terms of \mathcal{R}_k . This ‘mergibility’ condition is checked by the function CHECK_MERGE in the pseudocode below: line 13 checks control consistency for the current state pair (x'_i, x'_j) and line 18 *recursively* checks for all their corresponding future states. Note that CHECK_MERGE takes as its arguments a state pair, a ‘waiting list’ W , and the index i at line 2; here the waiting list W stores all the pairs whose future states must be checked for control consistency.

Throughout the merging process, in order to generate a control congruence, it is crucial to prevent states from being shared by more than one cell. This is achieved by inserting in the algorithm three filters—at lines 3, 5, and 17—to eliminate element overlapping among cells in \mathcal{C}_k . The algorithm loops until all the states in X are checked.

The following is the pseudocode of the Localization Algorithm. Lines 1–8 comprise the MAIN procedure, which at line 7 calls the function CHECK_MERGE, lines 9–19.

```

1: procedure MAIN()
2:   for  $i := 0$  to  $n - 2$  do
3:     if  $i > \min\{m \mid x_m \in [x_i]\}$  then continue;
4:     for  $j := i + 1$  to  $n - 1$  do
5:       if  $j > \min\{m \mid x_m \in [x_j]\}$  then continue;
6:        $W = \emptyset$ ;
7:       if CHECK_MERGE( $x_i, x_j, W, i$ ) = true then
8:          $\mathcal{C}_k := \left\{ [x] \cup \bigcup \{[x'] \mid ((x, x'), (x', x)) \cap W \neq \emptyset\} \mid [x], [x'] \in \mathcal{C}_k \right\}$ .
9: function CHECK_MERGE( $x_i, x_j, W, i$ )
10:  for each  $x_p \in [x_i] \cup \bigcup \{[x] \mid ((x, x_i), (x_i, x)) \cap W \neq \emptyset\}$  do
11:    for each  $x_q \in [x_j] \cup \bigcup \{[x] \mid ((x, x_j), (x_j, x)) \cap W \neq \emptyset\}$  do
12:      if  $\{(x_p, x_q), (x_q, x_p)\} \cap W \neq \emptyset$  then continue;
13:      if  $(x_p, x_q) \notin \mathcal{R}_k$  then return false;
14:       $W = W \cup \{(x_p, x_q)\}$ ;
15:      for each  $\sigma \in \Sigma$  with  $\xi(x_p, \sigma) \neq \xi(x_q, \sigma)$  do
16:        if  $[\xi(x_p, \sigma)] = [\xi(x_q, \sigma)]$  or
17:           $\{(\xi(x_p, \sigma), \xi(x_q, \sigma)), (\xi(x_q, \sigma), \xi(x_p, \sigma))\} \cap W \neq \emptyset$  then continue;
18:        if  $\min\{m \mid x_m \in [\xi(x_p, \sigma)]\} < i$  or
19:           $\min\{m \mid x_m \in [\xi(x_q, \sigma)]\} < i$  then return false;
20:        if CHECK_MERGE( $\xi(x_p, \sigma), \xi(x_q, \sigma), W, i$ ) = false then
21:          return false;
22: return true.

```

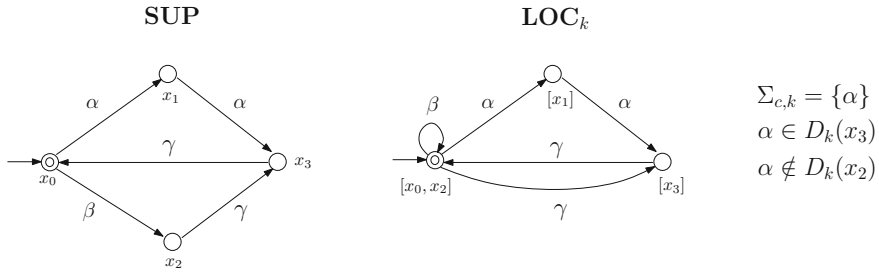


Fig. 2.9 Example: localization algorithm

Theorem 2.5 *Given the supervisor **SUP** with state set X and $|X| = n$, the Localization Algorithm terminates in finite steps, has (worst-case) time complexity $O(n^4)$, and the generated \mathcal{C}_k is a control congruence on X .*

Proof According to lines 2 and 4, the MAIN procedure can call the function CHECK_MERGE (line 7) at most $n(n-1)/2$ times. Note from line 12 that if $(x_p, x_q) \in W$ then $(x_q, x_p) \notin W$, for $p, q = 0, \dots, n-1$. Hence the size of set W (initialized at line 6) is at most $n(n-1)/2$. This implies that for fixed i, j , CHECK_MERGE(x_i, x_j, W, i) can call itself at most $|\Sigma|n(n-1)/2$ times, where $|\Sigma|$ is due to line 15. Therefore the Localization Algorithm terminates in finite steps, and has (worst-case) time complexity $O(n^4)$.

Now, since lines 3, 5, and 17 ensure no element overlapping among cells in \mathcal{C}_k , the resulting \mathcal{C}_k is a partition on X . It then suffices to show that \mathcal{C}_k is a control cover; namely the two conditions (i) and (ii) of Definition 2.2 are satisfied. That condition (i) holds is guaranteed by line 13: no two states in the same cell can be control inconsistent; and (ii) holds by lines 15–18: recursively check control consistency for future states reached by identical strings. Therefore we conclude that the resulting \mathcal{C}_k is a control congruence, as required. \square

Example 2.1 We provide an example, displayed in Fig. 2.9, to illustrate the steps of the Localization Algorithm. Notation: $\mathcal{C}_{0,k}$ denotes the initial control cover with respect to $\Sigma_{c,k}$, and $\mathcal{C}_{i,k}$ ($i = 1, 2, 3$) the resulting control cover of the i th iteration of the algorithm.

- (0) Initially, $\mathcal{C}_{0,k} = \{[x_0], [x_1], [x_2], [x_3]\}$.
- (1) (x_0, x_1) cannot be merged: they pass line 13 because $(x_0, x_1) \in \mathcal{R}_k$, but they fail at line 18 for $(\xi(x_0, \alpha), \xi(x_1, \alpha)) \notin \mathcal{R}_k$.
 (x_0, x_2) can be merged: they pass line 13 because $(x_0, x_2) \in \mathcal{R}_k$, and they trivially pass line 18 since there is no common event defined on them, so that no further control consistency needs to be verified.
 (x_0, x_3) cannot be merged: they fail at line 13, for $(x_0, x_3) \notin \mathcal{R}_k$.
 So $\mathcal{C}_{1,k} = \{[x_0, x_2], [x_1], [x_3]\}$.

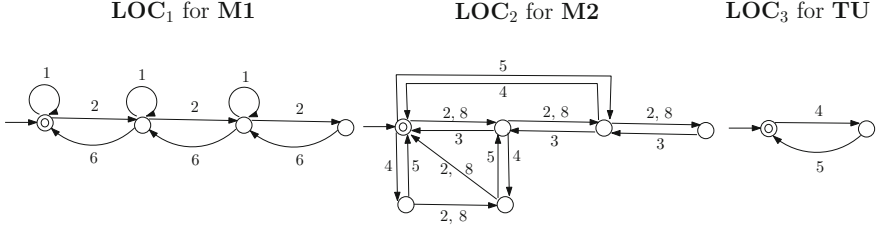


Fig. 2.10 Transfer Line: local controllers

- (2) (x_1, x_2) cannot be merged: they cannot pass line 5, because x_2 and x_0 are now in the same cell and $2 > 0$.

(x_1, x_3) cannot be merged: they fail at line 13, since $(x_1, x_3) \notin \mathcal{R}_k$.

Thus $\mathcal{C}_{2,k} = \{[x_0, x_2], [x_1], [x_3]\}$.

- (3) (x_2, x_3) cannot be merged: they fail at line 3 for, again, x_2 and x_0 are now in the same cell and $2 > 0$.

Finally, $\mathcal{C}_{3,k} = \{[x_0, x_2], [x_1], [x_3]\}$, and the induced generator \mathbf{LOC}_k (unique in this case) is displayed on the right of Fig. 2.9.

The Localization Algorithm is implemented in *TCT* [Won14a] as follows:

$$\{\mathbf{LOC}_1, \dots, \mathbf{LOC}_N\} = \text{Localize}(\mathbf{G}, \{\mathbf{G}_1, \dots, \mathbf{G}_N\}, \mathbf{SUP})$$

where plant \mathbf{G} , agents \mathbf{G}_k ($k \in \underline{N}$), and the monolithic supervisor \mathbf{SUP} are input generators, and local controllers \mathbf{LOC}_k ($k \in \underline{N}$) are output generators.

Let us now return to the Transfer Line example introduced in Sect. 2.1. Here our objective is distributed control: namely to synthesize for each agent a local controller, with no external supervisor. We apply the Localization Algorithm in *TCT* to decompose the monolithic supervisor computed in Sect. 2.1 into local controllers, one for each agent. The resulting controllers are displayed in Fig. 2.10, having 4, 6, and 2 states respectively. The desired control equivalence between these local controllers and the monolithic supervisor is confirmed also by *TCT*.

With these individual controllers, we can account for the local strategies of each agent. Machine **M1** with \mathbf{LOC}_1 , controlling event 1, ensures that no more than three workpieces can be processed simultaneously in the system, i.e. prevents choking in the material feedback loop; to achieve this goal, however, an external event 6 has to be communicated from **TU** to **M1**, i.e. $\Sigma_{com,1} = \{6\}$. Machine **M2** with \mathbf{LOC}_2 , controlling event 3, simultaneously guarantees the safety of both buffers (against underflow and overflow); in this case, external events 2, 5 and 8 must be communicated to **M2** from **M1** and **TU** respectively, i.e. $\Sigma_{com,2} = \{2, 5, 8\}$. Notice that the observed event 5, being a controllable event of **TU**, cannot be disabled by **M2**. Finally, **TU** with \mathbf{LOC}_3 , controlling event 5, is responsible only for the safety of buffer **B2**; the communication event set $\Sigma_{com,3} = \{4\}$.

Summarizing the above we display in Fig. 2.11 the distributed control architecture of Transfer Line with the communication graph of the three agents. Compared to the

centralized architecture of Fig. 2.4, the agents are now equipped with their own ‘private’ controllers but no external, high-level supervisors; yet the collective locally controlled behavior is guaranteed to be identical to the globally controlled optimal and nonblocking behavior.

2.6 Language Interpretation of Localization

We have developed supervisor localization and a localization algorithm based on the state set X of the supervisor **SUP**, where the central concept is control cover/congruence on X (Definition 2.2). In particular, the localization algorithm computes a control congruence on X so as to generate a local controller. In light of the (Nerode) identification of “states” with the cells of a *right congruence* on a language, it is of theoretical interest to identify control congruence on the state set X of the supervisor **SUP** with a corresponding construct on $L(\mathbf{SUP})$. In this way control congruence will be seen to arise “naturally” in the language framework of SCT.

In this section we relate control congruence on X to a special right congruence on $L(\mathbf{SUP})$. Several concepts to be used in this section—equivalence relation, Nerode equivalence relation, canonical recognizer, and right congruence—are summarized in Appendix A.

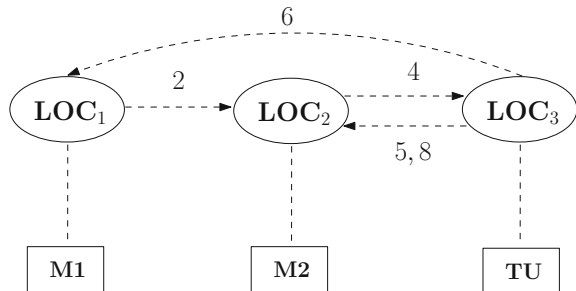
Let $\mathbf{SUP} = (X, \Sigma, \xi, x_0, X_m)$ be a supervisor, with closed language $L(\mathbf{SUP})$ and marked language $L_m(\mathbf{SUP})$. Let $\equiv_{L_m(\mathbf{SUP})}$ be the Nerode equivalence relation with respect to $L_m(\mathbf{SUP})$; that is, for arbitrary strings $s, t \in \Sigma^*$, s and t are Nerode equivalent, written $(s, t) \in \equiv_{L_m(\mathbf{SUP})}$, if and only if

$$(\forall u \in \Sigma^*) su \in L_m(\mathbf{SUP}) \Leftrightarrow tu \in L_m(\mathbf{SUP}).$$

For the subsequent development, the *dump* cell $\Sigma^* \setminus L(\mathbf{SUP})$ of $\equiv_{L_m(\mathbf{SUP})}$ plays no role, and thus we restrict $\equiv_{L_m(\mathbf{SUP})}$ to $L(\mathbf{SUP})$, namely consider $(s, t) \in \equiv_{L_m(\mathbf{SUP})}$ only for $s, t \in L(\mathbf{SUP})$.

Write $|\equiv_{L_m(\mathbf{SUP})}|$ for the cardinality (number of cells) of $\equiv_{L_m(\mathbf{SUP})}$ (excluding the dump cell). Suppose $|X| = |\equiv_{L_m(\mathbf{SUP})}|$, i.e. **SUP** is a canonical generator of the language $L_m(\mathbf{SUP})$.

Fig. 2.11 Transfer Line:
distributed control
architecture



Now fix $k \in \underline{N}$ and let $\mathcal{C}_k = \{X_i \subseteq X \mid i \in I_k\}$ be a control congruence on X with pairwise disjoint cells. Thus \mathcal{C}_k corresponds to a partition \mathcal{P}_k on $L(\mathbf{SUP})$ as follows. For each state $x \in X$ let $[x] := \{s \in L(\mathbf{SUP}) \mid \xi(x_0, s) = x\}$, and for each cell $X_i \subseteq X$ let $[X_i] := \bigcup \{[x] \mid x \in X_i\}$. Then $\mathcal{P}_k := \{[X_i] \subseteq L(\mathbf{SUP}) \mid i \in I_k\}$ is the partition corresponding to \mathcal{C}_k . The partition \mathcal{P}_k corresponds to an equivalence relation π_k on $L(\mathbf{SUP})$ according to $(s, t) \in \pi_k$ if and only if

$$(\exists i \in I_k) s \in [X_i] \ \& \ t \in [X_i].$$

The equivalence relation π_k on $L(\mathbf{SUP})$ thus corresponds to the control congruence \mathcal{C}_k on X . Our result is the following.

Theorem 2.6 *The equivalence relation π_k on $L(\mathbf{SUP})$ has the following properties:*
(i) π_k is a right congruence on $L(\mathbf{SUP})$, i.e.

$$(\forall s, t \in L(\mathbf{SUP}), \forall u \in \Sigma^*) \quad su, tu \in L(\mathbf{SUP}) \ \& \ (s, t) \in \pi_k \Rightarrow (su, tu) \in \pi_k. \quad (2.26)$$

(ii) $(\forall s, t \in L(\mathbf{SUP})) \ (s, t) \in \equiv_{L_m(\mathbf{SUP})} \Rightarrow (s, t) \in \pi_k$, i.e. π_k is ‘coarser’ than $\equiv_{L_m(\mathbf{SUP})}$.

(iii) $(\forall s, t \in L(\mathbf{SUP})) \ (s, t) \in \pi_k \Rightarrow (\xi(x_0, s), \xi(x_0, t)) \in \mathcal{R}_k$, i.e. π_k is ‘finer’ than \mathcal{R}_k .

We have thus established the correspondence of the control congruence \mathcal{C}_k on X to the special equivalence relation π_k on $L(\mathbf{SUP})$: π_k is a right congruence on $L(\mathbf{SUP})$ that is ‘coarser’ than the Nerode equivalence relation $\equiv_{L_m(\mathbf{SUP})}$ and ‘finer’ than the control consistency relation \mathcal{R}_k (Definition 2.1).

Proof of Theorem 2.6. (i) We proceed by induction on the length of string $u \in \Sigma^*$. If $u = \epsilon$, then (2.26) holds trivially. Suppose $u = \sigma$, an arbitrary event, and let $s\sigma, t\sigma \in L(\mathbf{SUP})$, $(s, t) \in \pi_k$. It will be shown that $(s\sigma, t\sigma) \in \pi_k$. From $(s, t) \in \pi_k$ we have that there exists $i \in I_k$ such that $s, t \in [X_i]$. It follows that there exist x and x' in X_i such that $\xi(x_0, s) = x$ and $\xi(x_0, t) = x'$. Since $s\sigma, t\sigma \in L(\mathbf{SUP})$, i.e. $\xi(x, \sigma)!$ and $\xi(x', \sigma)!$, by Definition 2.2(ii) that there exists $j \in I_k$ such that $\xi(x, \sigma), \xi(x', \sigma) \in [X_j]$. That is, $s\sigma, t\sigma \in [X_j]$, and therefore $(s\sigma, t\sigma) \in \pi_k$. Inductively, (2.26) holds for an arbitrary string $u \in \Sigma^*$.

(ii) Let $s, t \in L(\mathbf{SUP})$. Then

$$\begin{aligned} (s, t) \in \equiv_{L_m(\mathbf{SUP})} &\Rightarrow (\exists x \in X) \ s, t \in [x] \\ &\Rightarrow (\exists i \in I_k) \ s, t \in [x] \subseteq [X_i] \\ &\Rightarrow (s, t) \in \pi_k. \end{aligned}$$

(iii) Let $s, t \in L(\mathbf{SUP})$. Then

$$\begin{aligned}
 (s, t) \in \pi_k &\Rightarrow (\exists i \in I_k) s, t \in [X_i] \\
 &\Rightarrow (\exists x, x' \in X_i) \xi(x_0, s) = x \ \& \ \xi(x_0, t) = x' \\
 &\Rightarrow (\xi(x_0, s), \xi(x_0, t)) \in \mathcal{R}_k.
 \end{aligned}$$

□

2.7 Boundary Cases of Localization

In this section, we identify two boundary cases of supervisor localization which indicate, as a property of the localization problem itself, an extreme degree of easiness or hardness, respectively.

2.7.1 Fully-Localizable

This case is the easy situation where locally controlled agents are completely decoupled: each local controller works independently without any interaction with others through shared events. In other words, the communication event sets $\Sigma_{com,k}$ in (2.22) are empty for all $k \in \underline{N}$.

We formulate this notion of full-localizability as follows. Consider a plant \mathbf{G} over Σ composed of agents over disjoint alphabets Σ_k , $k \in \underline{N}$. For an imposed specification E , let \mathbf{SUP} be the corresponding monolithic supervisor.

Definition 2.5 \mathbf{SUP} is *fully-localizable* if there exists a set of local controllers $\{\mathbf{LOC}_k | k \in \underline{N}\}$, which is control equivalent to \mathbf{SUP} , such that for every $k \in \underline{N}$, \mathbf{LOC}_k is defined over Σ_k , i.e. $L(\mathbf{LOC}_k) \subseteq \Sigma_k^*$.

Full-localizability of \mathbf{SUP} is easily verified if one has computed a set of local controllers by the Localization Algorithm: just check if the event set of each \mathbf{LOC}_k is identical to Σ_k .² Before doing the computation, the following simple condition may be used to identify full-localizability.

Proposition 2.2 *Let the specification E be such that $E = \bigcup \{E_i | i \in \mathcal{I}\}$. If for each $i \in \mathcal{I}$ there is $k \in \underline{N}$ such that $E_i \subseteq \Sigma_k^*$, then \mathbf{SUP} is fully-localizable.*

Proof The conclusion follows immediately from the assumption that Σ_k ($k \in \underline{N}$) are pairwise disjoint and Definition 2.5 above. □

²If the check turns out negative, \mathbf{SUP} might still be fully-localizable, as conceivably there could exist other instances of local controllers (referring to the Localization Procedure in Sect. 2.3) for which the check turns out positive.

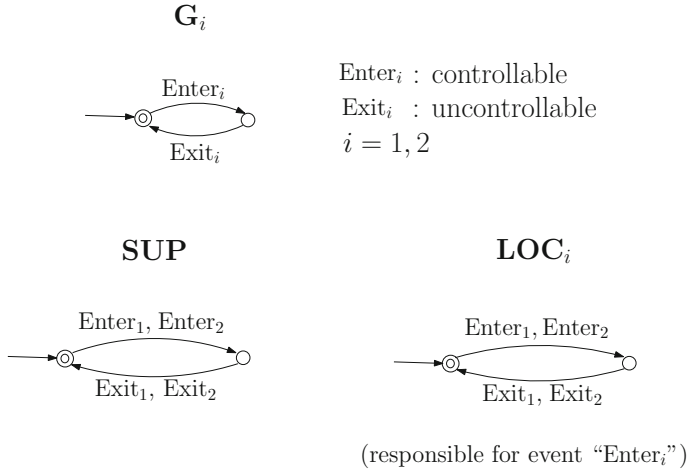


Fig. 2.12 Example: non-localizability

The assumption of Proposition 2.2 says that each component specification is imposed exclusively on some single component agent. In that case, local controllers can actually be obtained locally without going through the top-down Localization Procedure. This situation is rare in real distributed systems, inasmuch as a control specification would typically restrict the joint behavior of a group of agents. Proposition 2.2 also appeared in a different form in the decentralized control context, e.g. [WH91].

2.7.2 Non-Localizable

The other extreme of the localization problem is the hard case where component agents are coupled so tightly that each one needs to communicate with all the others. Let us introduce this situation by an example.

Example 2.2 In Fig. 2.12, two agents G_i ($i = 1, 2$) share a common resource that is not allowed to be used simultaneously, namely usage is subject to *mutual exclusion*. **SUP** is a monolithic supervisor that enforces the mutual exclusion specification. Then applying the Localization Algorithm to **SUP**, we generate for agent G_i a local controller **LOC $_i$** . However, both local controllers are nothing but the same as **SUP**; that is, our supervisor localization accomplished nothing useful.

In general, we aim to find conditions that can identify the situation where localization fails to achieve a truly local result. In that case we need only make copies of **SUP** for the relevant agents.

Definition 2.6 Let \mathbf{LOC}_k , $k \in \underline{N}$, be a local controller for agent \mathbf{G}_k computed by the Localization Algorithm. \mathbf{SUP} is *non-localizable* with respect to \mathbf{G}_k if the state sizes of \mathbf{SUP} and \mathbf{LOC}_k are the same, i.e. $|\mathbf{SUP}| = |\mathbf{LOC}_k|$.

First note that the condition $|\mathbf{SUP}| = |\mathbf{LOC}_k|$ implies that \mathbf{SUP} is in fact identical to \mathbf{LOC}_k (i.e. \mathbf{SUP} and \mathbf{LOC}_k are DES-isomorphic with the identity map as the isomorphism). This is because $|\mathbf{SUP}| = |\mathbf{LOC}_k|$ means, when running the Localization Algorithm, that no states of \mathbf{SUP} can be merged, which in turn implies that the transition structure of \mathbf{SUP} will be output unchanged.

By Theorem 2.4, \mathbf{LOC}_k is induced from some control cover, say \mathcal{C}_k . Thus to derive $|\mathbf{LOC}_k|$, it is equivalent to determine the number of cells of \mathcal{C}_k . Given $\mathbf{SUP} = (X, \Sigma, \xi, x_0, X_m)$, by the definition of control cover two states $x, x' \in X$ that belong to an identical cell must satisfy both conditions:

- (1) $(x, x') \in \mathcal{R}_k$,
- (2) $(\forall s \in \Sigma^*) \xi(x, s)! \ \& \ \xi(x', s)! \Rightarrow (\xi(x, s), \xi(x', s)) \in \mathcal{R}_k$.

Negating (1) and (2), we get

- (3) $(x, x') \notin \mathcal{R}_k$,
- (4) $(\exists s \in \Sigma^*) \xi(x, s)! \ \& \ \xi(x', s)! \ \& \ (\xi(x, s), \xi(x', s)) \notin \mathcal{R}_k$.

Hence, two states x, x' belong to different cells of \mathcal{C}_k if and only if either (3) or (4) holds. Let

$$\Omega_{\mathcal{C}_k} := \max \{l \mid (\exists X' \subseteq X) |X'| = l \ \& \ (\forall x, x' \in X') x \neq x' \Rightarrow (3) \text{ or } (4)\}.$$

The above discussion has proved the following fact.

Proposition 2.3 $|\mathbf{LOC}_k| = \Omega_{\mathcal{C}_k}$.

Now a necessary and sufficient condition for non-localizability is immediate.

Proposition 2.4 *SUP is non-localizable with respect to \mathbf{G}_k if and only if $|\mathbf{SUP}| = \Omega_{\mathcal{C}_k}$.*

Proof The conclusion follows directly from Definition 2.6 and Proposition 2.3. \square

In fact the above condition is hardly more than a restatement of the definition of non-localizability. We have still said nothing about how to check whether or not the condition holds. Nevertheless, a slight modification of $\Omega_{\mathcal{C}_k}$ will lead to a computationally verifiable sufficient condition for non-localizability.

Consider instead

$$\Omega_k := \max \{l \mid (\exists X' \subseteq X) |X'| = l \ \& \ (\forall x, x' \in X') x \neq x' \Rightarrow (x, x') \notin \mathcal{R}_k\}$$

That is, we consider only (3) above and discard (4). Thus $\Omega_k \leq \Omega_{\mathcal{C}_k}$. More importantly, if we construct an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = X$ and $\mathcal{E} = \{(x, x') \mid (x, x') \notin \mathcal{R}_k\}$, then calculating Ω_k amounts in fact to finding the *maximum clique* (i.e. largest complete subgraph) in \mathcal{G} . Although the maximum clique

problem is a well-known NP-hard problem, there exist efficient algorithms that compute sub-optimal solutions, e.g. [PX94]. In particular, the polynomial-time algorithm, implemented in *TCT*, that computes *lower bound estimate* (*lbe*) in [SW04, Sect. 4.2] can be directly employed for our purpose. Let us denote by lbe_k the outcome of the sub-optimal algorithm with respect to Ω_k . Thus we have $lbe_k \leq \Omega_k \leq \Omega_{C_k} \leq |\mathbf{SUP}|$, which gives rise to the following result.

Proposition 2.5 *If $|\mathbf{SUP}| = lbe_k$, then \mathbf{SUP} is non-localizable with respect to \mathbf{G}_k .*

Proof $|\mathbf{SUP}| = lbe_k$ implies that $|\mathbf{SUP}| = \Omega_{C_k}$, and consequently $|\mathbf{SUP}| = |\mathbf{LOC}_k|$ by Proposition 2.3. \square

The condition $|\mathbf{SUP}| = lbe_k$ is not necessary for non-localizability. If we obtain $|\mathbf{SUP}| > lbe_k$, then lbe_k tells us little about non-localizability and can only serve as a conservative lower bound estimate indicating how much localization might (conceivably) be achieved. If, however, $|\mathbf{SUP}| = lbe_k$ does hold, then the problem instance admits no useful localization solution, and we can avoid wasting further computational effort.

Continuing Example 2.2, and applying the algorithm computing lbe_k , we obtain $lbe_i = 2 = |\mathbf{SUP}|$ ($i = 1, 2$). Hence \mathbf{SUP} is non-localizable for either of the two agents, and then we simply assign the agents with copies of \mathbf{SUP} as their local controllers.

Supervisor Localization

A Top-Down Approach to Distributed Control of
Discrete-Event Systems

Cai, K.; Wonham, W.M.

2016, XV, 199 p. 136 illus., 9 illus. in color., Softcover

ISBN: 978-3-319-20495-6