Scheduled protocol programming

Dokter, K.P.C.

Chapter 6

Protocols as Petri nets

Our constraint-based approach to the specification of interaction protocols is very flexible, because many syntactically different constraints have the same meaning. While such flexibility is useful for development of new compilation techniques, it distracts attention from the fundamental principle that leads to the exponential improvement of compilation in Figure 5.3. The most important concept in Chapter 5 is that rule-based stream constraints expose concurrency in constraints that define sequential behavior.

In the current chapter, we streamline this concept by expressing protocols in terms of an enhanced version Petri nets called multilabeled Petri nets[1]. The explicit concurrency in Petri nets helps us to develop alternative compilers that respect the available concurrency in a protocol.

Although we desire the protocol specification as an orchestration, we want a protocol implementation as a choreography. Indeed, implementing the protocol with a single central protocol component can easily introduce a performance bottleneck. To prevent the bottleneck, we aim for distributed protocols, i.e., protocols implemented as multiple parallel components.

It is the responsibility of the compiler of the coordination language to produce efficient protocol implementations. Jongmans [Jon16] developed a compiler that generates protocol implementations based on constraint automata [BSAR06]. A constraint automaton is a pair (P, A) consisting of a set of variables P and a state machine A, whose transition labels are pairs (N, g) consisting of a set of variables $N \subseteq P$ and a constraint g on their values. The elements P, N, and g are respectively called interface, synchronization constraint, and data constraint.

Constraint automata model protocols in a simple and intuitive manner. However, being a state machine, a constraint automaton is inherently sequential. As we desire distributed protocols, the constraint automaton representation of protocols is not completely adequate. State-space explosions for constraint automata serve as evidence for this mismatch. Alternative algorithms to compute the composite constraint automaton have only partial success [JKA17, Fig. 17(a)].

In contrast to state machines, Petri nets [Rei13] are inherently parallel. Moreover, a state machine can be viewed as a Petri net for which every transition has

[1] The work in this chapter is based on [Dok19]

a single input place and a single output place. It seems natural to generalize constraint automata by replacing the underlying state machine by a Petri net.

In the current chapter, we introduce multilabeled Petri nets as an inherently parallel extension to constraint automata. After stating some basic results on monoids and multisets (Section 6.1), we view a constraint automaton as a *multilabeled Petri net* (Section 6.2), which is an ordinary Petri net whose transitions are labeled by multisets of actions. If multiple (not necessarily distinct) transitions in a multilabeled Petri net fire in parallel, the composite transition is labeled by the union of the labels of its constituent transitions.

We generalize constraint automaton composition to composition of multilabeled Petri nets (Section 6.3). While the composition of arbitrary multilabeled Petri nets seems hard to compute, we develop an efficient algorithm that composes *square-free* nets. Intuitively, a multilabeled Petri net is squarefree iff any action occurs at most once in every (parallel) execution step of the net.

The number of places in the composite Petri net grows linearly, which prevents the state-space explosion. Therefore, multilabeled Petri nets are an adequate intermediate representation of protocols. Since a transition-space explosion is still possible, multilabeled Petri nets are not a silver bullet.

Multilabeled Petri nets can contain silent transitions, which have no observable behavior. Such silent transitions can be the result of hiding irrelevant actions. In protocol implementations based on multilabeled Petri nets, silent transitions do not perform any I/O-operation and delay the throughput of the protocol. We define an abstraction operator for multilabeled nets (Section 6.4) that removes silent transitions. We develop an algorithm that computes the abstraction of a multilabeled net.

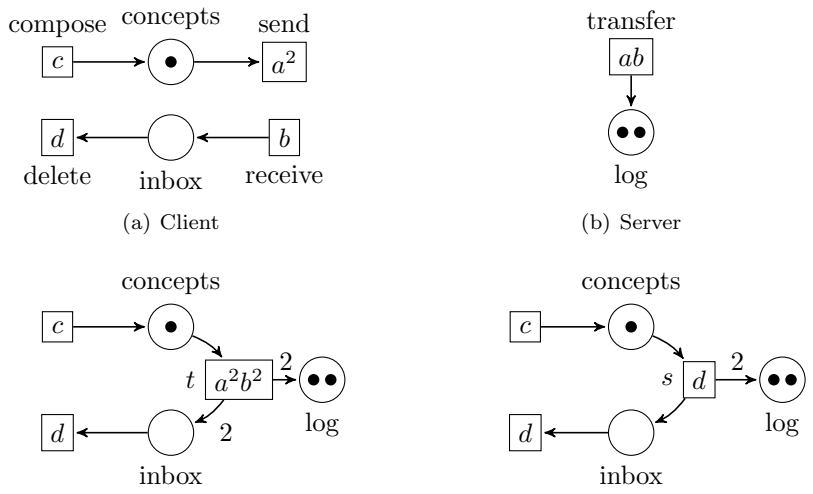
Finally, we summarize the results (Section 6.5) and point out future work.

Running example We illustrate the composition and abstraction of multilabeled Petri nets by an example on a mail server and a client.

Figure 6.1(a) shows the Petri net of a client that can compose, send, receive, and delete messages. Composed messages are stored as concepts, and received messages end up in the inbox. The client can concurrently send and receive messages, as is the case for a large company with internal mail between different departments. We want every message to be transferred to two recipients (e.g., adding a recipient in CC). We represent this intend labeling the send transition with the expression a^2, which denotes a multiset that contains a twice.

Figure 6.1(b) shows the Petri net of a server that can transfer messages. A fingerprint of each transferred message is logged. We assume that message transferal is not buffered: a send message is immediately received. We represent this by labeling the transfer transition with the expression ab that denotes the multiset that contains a and b. Note that the order of a and b is irrelevant, as ab and ba denote the same multiset.

Figure 6.1(c) shows the composition of the client and server, as defined by the composition operator presented in the current chapter. Composition of multilabeled nets synchronize transitions that agree on shared actions. Observe that the transfer transition must fire twice in order to agree on a with the send transition. Consequently, the receive transition must fire twice in order to agree on b with the



(a) Client (b) Server

(c) Composition (d) Abstraction

Figure 6.1: Multilabeled nets of a client (a) and a mail server (b). Their composition (c) synchronizes a single send transition with two transfer and two receive transitions (t equals send | transfer^2 | receive^2). The abstraction (d) hides the a and b actions and eliminates the resulting silent transition (s equals t; delete).

transfer^2 transition. Hence, the send, transfer, and receive transitions synchronize into a single parallel transition send | transfer^2 | receive^2, which we denote as t. The actions c (compose) and d (delete) of the client are not shared with the server. Therefore, the client-server composition allows the client to compose and delete a message, without synchronizing with the mail server.

Figure 6.1(d) shows the abstraction of the composite system, where message transferal (actions a and b) is hidden. After hiding a and b, transition t becomes silent. We eliminate this silent transition by *sequentially* composing t with the (observable) delete transition. In other words, we replace transition t by the sequential composition, s, of t followed by delete (i.e., $s = t$; delete). Note that s deletes only one of the two messages that are sent. The remaining message can be deleted by the usual delete transition.

Related work The literature offers a wide variety of composition operators on Petri nets, which can be divided in two categories, namely *topological* and *parallel* compositions.

By topological compositions, we mean those operators that 'glue Petri nets along their places and transitions'. For example, Mazurkiewicz [Maz95, Section 8] defines a composition operator that glues transitions with the same name. Bernardinello and De Cindio [BdC92, Part II] define a composition that glues Petri nets on places and transitions. Kotov [Kot78] defines multiple composition operators, including superposition, which glue two Petri nets along shared places and transitions. Hierarchical composition [Feh91] is a topological composition that identifies a complete Petri net with a single transition of another Petri net.

By parallel compositions, we mean those operators that 'synchronize transitions of the constituent Petri nets'. A single transition of one Petri net can synchronize with multiple transitions of the other net. Hence, parallel composition can be seen as duplication plus gluing, which distinguishes it from topological composition.

Parallel composition is called synchronous, if subsystems must 'run at the same speed'. That is, both Petri nets in a composition must simultaneously fire a transition. Examples of these operators are those that are defined as a product in a category of Petri nets [vGV87, Win87, MM90, Gol88]. Such composition operators are not always convenient for specification.

In asynchronous parallel composition, subsystems can run at different speeds. That is, one Petri net can fire a transition, while the other net does nothing. Examples of asynchronous compositions include the composition of Petri Boxes [BDH92], Signal Transition Graphs [VW02], extended safe nets [Tau89, Chapter 4], zero safe nets [BM97], and general Petri nets [Gol88].

The asynchronous composition operator defined by Anisimov, Golenkov, and Kharitonov [AGK01] comes closest to our composition operator. Their parallel composition ${}_\alpha\|_\beta$ of Petri nets is relative to some transition (multi)labelings α and β of its operands. Unlike our composition operator, Anisimov et al. suppose a CCS-like synchronization that synchronizes an action a with its conjugate $\overline{a}$. As such, their composition cannot be used as a formal semantics for Reo circuits.

Silent transitions in safe Petri nets can be removed in at least two different ways. Vogler and Wollowski [VW02] use contraction, which deletes a silent transition and merges all its input and output places. In general, contraction does not yield a safe net.

Wimmel [Wim04] eliminates silent transitions from safe Petri nets in three steps: First, he unfolds a safe Petri net into an occurrence net, whose graph structure is acyclic. Next, he finds the process [Pra86] (i.e., the set of all accepted pomsets) of the occurrence net, while ignoring all silent transitions. Finally, he constructs a suitable finite quotient of the process. By construction, the resulting Petri net has no silent transitions and is pomset-equivalent to the original net.

6.1 Preliminaries

6.1.1 Graded monoids

A **monoid** is a triple $(M, +, 0)$, where $+ : M \times M \to M$ is an associative binary operation, with identity element 0. A **submonoid** of $(M, +, 0)$ is a monoid of the form $(S, +, 0)$, where $S \subseteq M$ is a subset of M that contains $0 \in S$ and is closed under addition.

An element x in a monoid M is **invertible** iff there exists some $y \in M$, such that $x + y = 0$. An element x in a monoid M is **irreducible** iff $x = a + b$ implies that a or b is invertible, for all $a, b \in M$.

A monoid $(M, +, 0)$ is **graded** if there exists a map $g : M \to \mathbb{N}$, such that, for all $x, y \in M$, we have $g(x + y) = g(x) + g(y)$ and if $g(x) = 0$ then x is invertible.

Lemma 6.1.1. *Every element in a graded monoid is a sum of irreducibles.*

Proof. Let $x \in M$ be arbitrary. We show by induction on the grading $g(x) \in \mathbb{N}$ that x is a sum of irreducibles.

If $g(x) = 0$, then x is invertible. Hence, $x + y = 0$, for some $y \in M$. If $x = a + b$, for some $a, b \in M$, then $a + (b + y) = 0$ and a is invertible. Thus, x is irreducibile. In particular, x is a sum of irreducibles.

Suppose that every $y \in M$, with $g(y) < g(x)$, is a sum of irreducibles. If x is irreducible, then x is a sum of irreducibles. Suppose that x is reducible, i.e., $x = a + b$, for some non-invertible $a, b \in M$. By definition of the grading, $g(a), g(b) > 0$. Hence, $g(a), g(b) < g(a) + g(b) = g(x)$. By the induction hypothesis, a and b are sums of irreducibles, and so is x.

By induction, every element in a graded monoid is a sum of irreducibles. □

A **right-ideal** of a monoid $(M, +, 0)$ is a subset $I \subseteq M$, such that $x \in I$ and $y \in M$ implies $x + y \in I$, for all $x, y \in M$. A right-ideal I of a monoid M is **proper** iff $0 \notin I$. An element $x \in I$ is **right-irreducible (in I)** iff $x = a + b$ and $a \in I$ implies that b is invertible, for all $a, b \in M$.

Lemma 6.1.2. *Let I be a proper right ideal of a graded monoid M. Every element in M is a (possibly empty) sum of right-irreducibles in I plus an element $r \in M \setminus I$.*

Proof. Since the lemma holds trivially for right-irreducibles in I, let $x \in M$ be right-reducible. We show by induction on the grading $g(x) \in \mathbb{N}$ that x is a possibly empty sum of right-irreducibles in I plus an element $r \in M \setminus I$.

If $g(x) = 0$, then x is invertible. Since I is proper, we have $x \in M \setminus I$. Hence, x is an empty sum of right-irreducibles plus $r = x$.

Suppose that every y, with $g(y) < g(x)$, is a possibly empty sum of right-irreducibles in I plus an element $r \in M \setminus I$. Since x is right-reducible, we find some $a \in I$ and some non-invertible b, such that $x = a + b$. Non-invertibility of b implies that $g(b) > 0$. Hence,

$$g(a) < g(a) + g(b) = g(a + b) = g(x).$$

The induction hypothesis shows that $a = (\sum_{i=1}^{n} a_i) + r$ is a sum of $n \geq 0$ right-irreducibles $a_1, \ldots, a_n$ in I plus an element $r \in M \setminus I$. Since $a \in I$ and $r \in M \setminus I$, we have $n \geq 1$. Thus, $\sum_{i=1}^{n} a_i$ is an element of the right-ideal I. As I is proper, $\sum_{i=1}^{n} a_i$ is non-invertible, and $g(\sum_{i=1}^{n} a_i) \geq 1$. This implies that

$$g(r + b) < g(\textstyle\sum_{i=1}^{n} a_i) + g(r + b) = g((\sum_{i=1}^{n} a_i) + r + b) = g(x).$$

The induction hypothesis applied to $r + b$ yields right-irreducibles $b_1, \ldots, b_m$ and an element $r' \in M \setminus I$, such that $r + b = (\sum_{j=1}^{m} b_j) + r'$. Hence, $x = a + b = (\sum_{i=1}^{n} a_i) + r + b = (\sum_{i=1}^{n} a_i) + (\sum_{j=1}^{m} b_j) + r'$, which proves the lemma. □

6.1.2 Multisets

A **multiset** over a set X is an unordered collection of elements with duplicates, which is formally represented as a map $m : X \to \mathbb{N}$ that counts the number of occurrences of each $x \in X$ in the multiset. The set of all multisets over X is denoted as $\mathbb{N}^X$. The **cardinality** $|m|$ of a multiset m is defined as the cardinality of the set $\{(x, k) \mid x \in X, 0 \leq k < m(x)\}$. A multiset m is **non-empty** iff $0 < |m|$, and **finite** iff $|m| < \aleph_0$, where $\aleph_0$ is the first infinite cardinal number. The empty multiset is denoted as $\emptyset$. The set of all finite multisets over X is denoted as

$\mathbb{N}^{(X)} = \{m : X \to \mathbb{N} \mid |m| < \aleph_0\}$. The **free commutative monoid** is the triple $(\mathbb{N}^{(X)}, \cup, \emptyset)$.

For $k \in \mathbb{N}$, and multisets $m, m' \in \mathbb{N}^X$, the **union** $m \cup m'$, **intersection** $m \cap m'$, **difference** $m \setminus m'$, and **multiplication** km are defined, for $x \in X$, as

$$\begin{aligned}(m \cup m')(x) &= m(x) + m'(x),\\ (m \cap m')(x) &= \min(m(x), m'(x)),\\ (m \setminus m')(x) &= m(x) \dot{-} m'(x)\\ (k \cdot m)(x) &= k \cdot m(x),\end{aligned}$$

where $\dot{-}$ is monus, defined as $a \dot{-} b = \max(a - b, 0)$, for all $a, b \in \mathbb{N}$. The **subset** relation of multisets is defined as $m \subseteq m'$ iff $m(x) \leq m'(x)$, for all $x \in X$.

Multisets $m_0, m_1, m_2 \in \mathbb{N}^X$ satisfy the following identities:

$$\begin{aligned}m_0 \setminus (m_1 \cup m_2) &= (m_0 \setminus m_1) \setminus m_2\\ m_0 \cup (m_1 \setminus m_0) &= m_1 \cup (m_0 \setminus m_1)\\ (m_0 \cup m_1) \setminus m_2 &= (m_0 \setminus m_2) \cup (m_1 \setminus (m_2 \setminus m_0))\end{aligned}$$

Restriction $m|_Y$ of a multiset m on X to a subset $Y \subseteq X$ is defined, for all $y \in Y$, as $m|_Y(y) = m(y)$.

It is convenient to represent a finite multiset over a set X as (an equivalence class of) a finite sequence of elements from X. Let X^* be the free monoid on X that consists of all finite words of elements from X (including the empty word ϵ). A word $w \in X^*$ induces a multiset $w : X \to \mathbb{N}$, by defining, for all $x \in X$,

$$\epsilon(x) = 0, \quad wx(x) = w(x) + 1, \qquad wy(x) = w(x), \text{ for } y \neq x.$$

Note that different words might define the same multiset. For example, xy and yx both denote the multiset wherein both x and y occur once.

6.2 Multilabeled Petri nets

Multilabeled Petri nets are Petri nets whose transitions are labeled with a multiset of actions.

Definition 6.2.1. A **multilabeled (Petri) net** is a tuple (A, P, T, μ_0) with

1. A a set of actions,
2. P a set of places,
3. $T \subseteq \mathbb{N}^P \times \mathbb{N}^A \times \mathbb{N}^P$ a set of transitions, and
4. $\mu_0 : P \to \mathbb{N}$ an initial marking.

Inspired by Goltz [Gol88], the notation in Definition 6.2.1 slightly differs from the usual notation of Petri nets. The advantage of this presentation is that transitions (including its set of input and output places) can be studied in isolation, which allows for parallel and sequential composition of transitions.

For a transition $t = (P, \alpha, Q) \in T$, we write ${}^\bullet t = P$ for the multiset of input places of t, we write $t^\bullet = Q$ for the multiset of output places of t, and we write $\ell(t) = \alpha$ for the multiset of labels of t.

For the development of the composition operator for multilabeled nets in Section 6.3, we use the standard concurrent semantics of Petri nets, which allows multiple transitions to fire in parallel.

Definition 6.2.2. A **multitransition** of a multilabeled net (A, P, T, μ_0) is a finite multiset $\theta : T \to \mathbb{N}$ of transitions. $\mathbb{N}^{(T)}$ denotes the set of all multitransitions.

A multitransition $\theta \in \mathbb{N}^{(T)}$ of a multilabeled net $N = (A, P, T, \mu_0)$ defines a (concrete) transition $\tau(\theta) = ({}^\bullet\theta, \ell(\theta), \theta^\bullet)$ in $\mathbb{N}^P \times \mathbb{N}^A \times \mathbb{N}^P$, wherein

$$
\begin{aligned}
{}^\bullet\theta &= \textstyle\bigcup_{t \in T} \theta(t) \cdot {}^\bullet t \\
\theta^\bullet &= \textstyle\bigcup_{t \in T} \theta(t) \cdot t^\bullet \\
\ell(\theta) &= \textstyle\bigcup_{t \in T} \theta(t) \cdot \ell(t)
\end{aligned}
$$

A multitransition θ of a N is **enabled** at a marking $\mu \in \mathbb{N}^P$ iff ${}^\bullet\theta \subseteq \mu$. A marking $\mu' \in \mathbb{N}^P$ is **obtained** from a marking $\mu \in \mathbb{N}^P$ via a multitransition θ (denoted $\mu \,[\theta\rangle\, \mu'$) iff θ is enabled at μ, and $\mu' = (\mu \setminus {}^\bullet\theta) \cup \theta^\bullet$.

Definition 6.2.3. The **concurrent semantics** of a multilabeled net (A, P, T, μ_0) is a pointed, directed, labeled graph (V, E, μ_0), consisting of

1. vertices $V = \{\mu : P \to \mathbb{N}\}$, and
2. labeled edges $E = \{(\mu, \ell(\theta), \mu') \in V \times \mathbb{N}^A \times V \mid \mu \,[\theta\rangle\, \mu', |\theta| > 0\}$.

Note that the empty multitransition $\theta = \emptyset$ does not constitute a valid step in the semantics, as $\emptyset$ allows for internal divergent behavior (by always firing $\emptyset$).

For the development of the abstraction operator in Section 6.4, we rely on the interleaving semantics of nets:

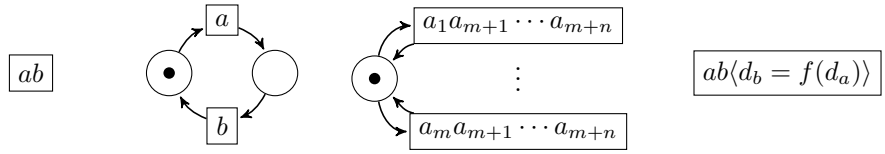
Definition 6.2.4. The **interleaving semantics** of a multilabeled net (A, P, T, μ_0) is a pointed, directed, labeled graph (V, E, μ_0), consisting of

1. vertices $V = \{\mu : P \to \mathbb{N}\}$, and
2. labeled edges $E = \{(\mu, \ell(t), \mu') \in V \times \mathbb{N}^A \times V \mid \mu \,[t\rangle\, \mu'\}$.

The only difference between the concurrent semantics and interleaving semantics of multilabeled nets is the cardinality of the multitransitions.

We introduce some terminology for a multilabeled net $N = (A, P, T, \mu_0)$. N is called **finite** iff A, P, and T are all finite. For $k \geq 1$, N is called **k-bounded** iff every reachable marking μ satisfies $\mu(p) \leq k$, for all $p \in P$. N is called **safe** iff N is 1-bounded.

A marking $\mu' \in \mathbb{N}^P$ is reachable from a marking $\mu \in \mathbb{N}^P$ via a sequence of transitions $t_1 \cdots t_n \in T^*$, with $n \geq 0$, (denoted $\mu \,[t_1 \cdots t_n\rangle\, \mu'$) iff there exists markings $\mu_1, \ldots, \mu_{n-1} \in \mathbb{N}^P$, such that $\mu \,[t_1\rangle\, \mu_1 \cdots \mu_{n-1} \,[t_n\rangle\, \mu'$. A **firing sequence** of N is a sequence of transitions $t_1 \cdots t_n \in T^*$, with $n \geq 0$ and $\mu_0 \,[t_1 \cdots t_n\rangle\, \mu'$, for some marking μ'. A marking μ is called **reachable** iff μ is reachable from the initial marking μ_0.



(a) $\mathsf{sync}(a,b)$ (b) $\mathsf{fifo}_1(a,b)$ (c) $\mathsf{node}_{m,n}(a_1,\ldots,a_{m+n})$ (d) $\mathsf{transformer}_f(a,b)$

Figure 6.2: Multilabeled nets for Reo primitives. The action $\langle d_b = f(d_a)\rangle$ in the transformer is the encoding of a data constraint.

A transition t is **dead** in N iff t does not occur on any firing sequence. A transition t is **potentially fireable** in N iff t is not dead in N. For $k \geq 1$, a transition $t \in T$ in a multilabeled net N is **k-live** iff ${}^\bullet t(p) \leq k$, for all $p \in P$. Every potentially fireable transition in a k-bounded net is k-live.

6.2.1 Constraint automata

As stated earlier, multilabeled nets generalize constraint automata [BSAR06] (without data constraints). If data constraints are ignored, the interpretation of constraint automata as multilabeled nets is rather straightforward. Figure 6.2 shows the multilabeled nets for some frequently used Reo primitives.

The $\mathsf{sync}(a,b)$ protocol in Figure 6.2(a) accepts a datum at input a and immediately offers it at output b. Since the $\mathsf{sync}(a,b)$ protocol is stateless, its multilabeled net does not contain a place. The $\mathsf{fifo}_1(a,b)$ protocol in Figure 6.2(b) accepts a datum at input a and stores it. In the next step, it offers the stored datum at output b. The $\mathsf{node}_{m,n}(a,\ldots,a_{m+n})$ protocol in Figure 6.2(c) accepts a datum at any input a_i, with $1 \leq i \leq m$, and immediately offers a copy of it at every output a_j, with $m < j \leq m+n$. The place in the $\mathsf{node}_{m,n}(a,\ldots,a_{m+n})$ protocol does not serve as memory, but encodes the conflict between the transitions. The $\mathsf{transformer}_f(a,b)$ protocol accepts a datum d_a from its input a, and simultaneously offers the datum $f(d_a)$ at its output b. The transformation of datum d_a into d_b is modeled by the data constraint $d_b = f(d_a)$.

In general, constraint automata can have non-trivial data constraints, as is the case for $\mathsf{transformer}_f(a,b)$. It is certainly possible to extend the definition of multilabeled nets (Definition 6.2.1) to include data constraints as well. However, such extension would not add any expressiveness to multilabeled nets, because data constraints can be encoded as fresh actions. For example, the data constraint $d_b = f(d_a)$ can be encoded as a fresh action $\langle d_b = f(d_a)\rangle$, which we call a **data-constraint action**. Freshness ensures that data-constraint actions are not used for synchronization (as is defined in Section 6.3). After composition, data-constraint actions can be decoded back into data constraints. If multiple data-constraint actions end up in the same transition label, their decoded data constraints are combined via conjunction. Figure 6.2(d) shows the multilabeled net for the transformer channel.

The above trick to encode data constraints as actions can be similarly applied to other semantic models [JA12].

The multilabeled nets that come from constraint automata are square-free:

Definition 6.2.5. A multitransition θ in a multilabeled net N is **square-free** iff $\ell(\theta)(a) \leq 1$, for all $a \in A(N)$. A multilabeled net N is **square-free** iff every potentially-fireable multitransition θ in N is square-free.

It is easy to verify that all multilabeled nets in Figure 6.2 are square-free: every transition is square-free and every multitransition θ of size $|\theta| > 1$ is dead.

The main reason for considering square-free nets is that their composition can be easily computed.

6.3 Composition

For this section, fix two multilabeled nets $N_i = (A_i, P_i, T_i, \mu_{0i})$, for $i \in \{0, 1\}$, which are interpreted according to the concurrent semantics in Definition 6.2.3.

We define the composition $N_0 \times N_1$ of N_0 and N_1 that synchronizes N_0 and N_1 on shared actions $A_0 \cap A_1$. We follow a standard approach to define the (parallel) composition of multilabeled nets. First, we generate all combinations of transitions in N_0 and N_1 that can fire in parallel. Next, we restrict to synchronizing combinations that 'agree on shared actions'. Finally, we restrict to a subset of combinations that generate all synchronizing combinations.

Recall that the disjoint union $X + Y$ of two sets X and Y is defined as $(X \times \{0\}) \cup (Y \times \{1\})$.

Definition 6.3.1. A **global transition** is a finite multiset $\eta : T_0 + T_1 \to \mathbb{N}$.

A global transition $\eta \in \mathbb{N}^{(T_0+T_1)}$ has, for $i \in \{0, 1\}$, a **local component** $\eta|_i \in \mathbb{N}^{(T_i)}$ defined as $\eta|_i(t) = \eta(t, i)$, for all $t \in T_i$.

The set $\mathbb{N}^{(T_0+T_1)}$ of all global transitions, endowed with the union $\cup$ of multiset and the empty multiset $\emptyset$, constitutes a monoid. Moreover, multiset cardinality, $|\cdot|$, defines a grading on $\mathbb{N}^{(T_0+T_1)}$. In particular, any submonoid of $\mathbb{N}^{(T_0+T_1)}$ is a graded monoid.

We now formalize the notion of agreement on shared actions:

Definition 6.3.2. A global transition $\eta \in \mathbb{N}^{(T_0+T_1)}$ is **S-compatible**, $S \subseteq A_0 \cap A_1$, iff $\ell(\eta|_0)(a) = \ell(\eta|_1)(a)$, for all $a \in S$. The set of S-compatible global transitions is denoted as $C(S) \subseteq \mathbb{N}^{(T_0+T_1)}$. We call η **compatible**, if η is $A_0 \cap A_1$-compatible.

Intuitively, the local components $\eta|_0$ and $\eta|_1$ of an S-compatible multitransition $\eta \in \mathbb{N}^{(T_0+T_1)}$ agree only on the shared actions in S, while they may disagree on shared actions $a \in (A_0 \cap A_1) \setminus S$ outside of S.

A compatible global transition $\eta \in C(A_0 \cap A_1)$ defines a (concrete) transition $\lambda(\eta) = ({}^\bullet\eta, \ell(\eta), \eta^\bullet) \in \mathbb{N}^{P_0+P_1} \times \mathbb{N}^{A_0 \cup A_1} \times \mathbb{N}^{P_0+P_1}$, with

$$\begin{aligned} {}^\bullet\eta(p, i) &= {}^\bullet(\eta|_i)(p) \\ \eta^\bullet(p, i) &= (\eta|_i)^\bullet(p) \\ \ell(\eta)(a) &= \ell(\eta|_i)(a) \qquad \text{if } a \in A_i \end{aligned}$$

for all $(t, i) \in T_0 + T_1$ and $a \in A_0 \cup A_1$. Note that $\ell(\eta)(a)$ is well-defined, since $\ell(\eta|_0)(a) = \ell(\eta|_1)(a)$, for $a \in A_0 \cap A_1$.

The empty global transition $\emptyset$ is trivially S-compatible, for all $S \subseteq A_0 \cap A_1$. Furthermore, the union $\alpha \cup \beta$ of two compatible global transitions is again S-compatible, for all $S \subseteq A_0 \cap A_1$. Hence, the set of S-compatibles $C(S)$ is a (graded) submonoid of the graded monoid $\mathbb{N}^{(T_0+T_1)}$. Lemma 6.1.1 shows that every compatible global transition can be decomposed into irreducibles.

Any reducible S-compatible global transition $\eta = \alpha \cup \beta$, for some global transitions α and β, is redundant, as $\lambda(\eta)$ can be simulated by firing $\lambda(\alpha)$ and $\lambda(\beta)$ in parallel. Hence, we consider the set $C_0(S) \subseteq C(S)$ of all irreducible S-compatible global transitions.

The image of $C_0(A_0 \cap A_1)$ under the map $\lambda : C(A_0 \cap A_1) \to \mathbb{N}^{P_0+P_1} \times \mathbb{N}^{A_0 \cup A_1} \times \mathbb{N}^{P_0+P_1}$ defines the set of transitions of the composition:

Definition 6.3.3. The **composition** $N_0 \times N_1$ of the multilabeled nets N_0 and N_1 is a multilabeled net with actions $A_0 \cup A_1$, places $P_0 + P_1$, transitions $\lambda(C_0(A_0 \cap A_1))$, and initial marking μ_0 defined as $\mu_0(p, i) = \mu_{0i}(p)$, for all $(p, i) \in P_0 + P_1$.

It is laborious but straightforward to verify that the composition of multilabeled Petri nets is associative. The composition is commutative only up to renaming of places, due to the index from the disjoint union. The composition is idempotent only up to semantic equivalence, as is duplicates the places.

Example 6.3.1. Consider the mail example in Figure 6.1, and suppose that the set of actions of the client equals $\{a, b, c, d\}$ and the set of actions of the server equals $\{a, b\}$. According to Definition 6.3.3, the transitions in the composition of the nets in Figures 6.1(a) and 6.1(b) consist of irreducible compatible global transitions. The compose transition in Figure 6.1(a) and the (implicit) empty multitransition $\emptyset$ in Figure 6.1(b) trivially agree on shared actions. Therefore, (compose $\mid \emptyset$) is a compatible transition. Being of length 1, the compose transition is necessarily irreducible, which implies that the compose transition is a transition of the composition in Figure 6.1(c).

Similarly, the global transition η, with components $\eta|_0 = \text{send} \mid \text{receive}^2$ and $\eta|_1 = \text{transfer}^2$, is also a compatible transition. Indeed, the label of $\eta|_0$ and $\eta|_1$ both equal a^2b^2. Clearly, η is irreducible, which shows that $\lambda(\eta)$ is a transition of the composition in Figure 6.1(c). ◊

6.3.1 Composition algorithm

Definition 6.3.3 only defines the transitions of the composition: it does not suggest a procedure on how these transitions can be found. It seems difficult to compute the composition of arbitrary multilabeled nets. Since the current work is motivated by constraint automata, we develop an algorithm that computes the composition of square-free multilabeled nets (Definition 6.2.5).

Lemma 6.3.1 shows that square-freeness must be checked only for atomic nets.

Lemma 6.3.1. *If N_0 and N_1 are square-free, then so is $N_0 \times N_1$.*

Proof. If a global transition $\eta \in \mathbb{N}^{(T_0+T_1)}$ is potentially fireable, then so are its local components $\eta|_0$ and $\eta|_1$. For $i \in \{0, 1\}$, square-freeness of N_i implies that $\ell(\eta|_i)(a) \leq 1$ and $a \in A_i$. By construction, $\ell(\eta)(a) \leq 1$, for all $a \in A_0 \cup A_1$. Hence, $N_0 \times N_1$ is square-free. □

By Definition 6.3.3, the composition $N_0 \times N_1$ can contain dead transitions. Since these dead transitions do not contribute to the behavior of the multilabeled net, it is no problem if our composition algorithm does not generate them. As the composition, $N_0 \times N_1$, is square-free, we consider only square-free transitions:

Definition 6.3.4. A global transition $\eta : T_0 + T_1 \to \mathbb{N}$ is **square-free** if $\lambda(\eta)$ is square-free. For $S \subseteq A_0 \cap A_1$, we denote the set of all square-free, irreducible S-compatible global transitions as $\overline{C_0}(S) \subseteq C_0(S)$.

We compute the composition of square-free nets N_0 and N_1 by recursion on the number of shared actions. This procedure is conveniently expressed with the following terminology:

Definition 6.3.5. The **difference** $d_a(\eta)$ of a global transition $\eta \in \mathbb{N}^{(T_0+T_1)}$ at a shared action $a \in A_0 \cap A_1$ is defined as the integer

$$d_a(\eta) = \ell(\eta|_0)(a) - \ell(\eta|_1)(a).$$

The set of all square-free, irreducible, S-compatible global transitions with difference $d \in \mathbb{Z}$ is denoted as $\overline{C_0^d}(S)$.

It is straightforward to verify that $d_a(\alpha \cup \beta) = d_a(\alpha) + d_a(\beta)$, for global transitions $\alpha, \beta \in \mathbb{N}^{(T_0+T_1)}$.

Since every global transition is $\emptyset$-composite, $C(\emptyset) = \mathbb{N}^{(T_0+T_1)}$ and $C_0(\emptyset) = T_0 + T_1$, where each $(t, i) \in T_0 + T_1$ is viewed as a singleton multiset on $T_0 + T_1$.

Lemma 6.3.2 expresses square-free, irreducible S-compatibles in terms of square-free, irreducible S'-compatibles, with $S' \subseteq S$.

Lemma 6.3.2. *If $S \subseteq A_0 \cap A_1$, and $a \in (A_0 \cap A_1) \setminus S$ then*

$$\overline{C_0}(S \cup \{a\}) \;\subseteq\; \overline{C_0^0}(S) \cup \{\alpha_{-1} \cup \alpha_1 \mid \alpha_d \in \overline{C_0^d}(S)\} \;\subseteq\; C(S \cup \{a\}).$$

Proof. For the first inclusion, let $\eta \in C_0(S \cup \{a\})$. Since every $S \cup \{a\}$-compatible is also S-compatible, we have that $\eta \in C(S)$. As $C(S)$ is a graded monoid, Lemma 6.1.1 shows that, for some $n \geq 1$ and $\beta_1, \ldots, \beta_n \in C_0(S)$, we have

$$\eta = \beta_1 \cup \cdots \cup \beta_n$$

Since η is square-free, we have,for every $1 \leq k \leq n$, that

$$\ell(\beta_k)(a) \leq \textstyle\sum_{i=1}^n \ell(\beta_i)(a) = \ell(\bigcup_{i=1}^n \beta_i)(a) = \ell(\eta)(a) \leq 1,$$

which shows that every β_i, $1 \leq i \leq n$, is square-free. In particular, the difference of β_i at a satisfies $d_a(\beta_i) \in \{-1, 0, 1\}$. We distinguish two cases:

Case 1: Suppose that $d_a(\beta_1) = 0$. Then, β_1 and $\beta_2 \cup \ldots \cup \beta_n$ are both S-compatible. Irreducibility of η shows that $n = 1$. Hence, $\eta = \beta_1 \in \overline{C_0^0}(S)$.

Case 2: Suppose that $d_a(\beta_1) = \pm 1$. Since η is S-compatible, we have that

$$\textstyle\sum_{i=1}^n d_a(\beta_i) = d_a(\bigcup_{i=1}^n \beta_i) = d_a(\eta) = 0$$

Since $d_a(\beta_i) \in \{-1, 0, 1\}$, for $1 \leq i \leq n$, we find some $1 < k \leq n$, such that $d_a(\beta_k) = -d_a(\beta_1) = \mp 1$. Without loss of generality, we assume that $k = 2$. From

Algorithm 1: Distribution

Input : Two finite, square-free multilabeled nets N_0 and N_1.
Output: $\overline{C_0}(A_0 \cap A_1) \subseteq C \subseteq C(A_0 \cap A_1)$.

```
C ← T_0 + T_1 ⊆ ℕ^(T_0+T_1);
foreach a ∈ A_0 ∩ A_1 do
    foreach d ∈ {−1, 0, 1} do
        C^d ← {η ∈ C | d_a(η) = d};
    C ← C^0 ∪ {α ∪ β | (α, β) ∈ C^{−1} × C^1, α ∪ β square-free};
```

$d_a(\beta_1 \cup \beta_2) = d_a(\beta_1) + d_a(\beta_2) = 0$, it follows that $\beta_1 \cup \beta_2$ and $\beta_3 \cup \ldots \cup \beta_n$ are both S-compatible. Irreducibility of η and non-emptyness of β_i, for $1 \leq i \leq n$, shows that $n = 2$, which implies that $\eta = \beta_1 \cup \beta_2$, with $\beta_i \in \overline{C_0^{\pm 2i \mp 3}}(S)$, for $i \in \{1, 2\}$. In both cases, $\eta \in \overline{C_0^0}(S) \cup \{\alpha_{-1} \cup \alpha_1 \mid \alpha_d \in \overline{C_0^d}(S)\}$.

For the second inclusion, let $\eta \in \overline{C_0^0}(S) \cup \{\alpha_{-1} \cup \alpha_1 \mid \alpha_d \in \overline{C_0^d}(S)\}$. Suppose that $\eta \in \overline{C_0^0}(S)$. By construction, $d_a(\eta) = 0$, which shows that $\eta \in C(S \cup \{a\})$. Every decomposition of η in $C(S \cup \{a\})$ is also a decomposition in $C(S)$. Hence, irreducibility of η in $C(S)$ implies that $\eta \in \overline{C_0}(S \cup \{a\})$.

Suppose $\eta \in \{\alpha_{-1} \cup \alpha_1 \mid \alpha_d \in \overline{C_0^d}(S)\}$. Then, we have $d_a(\eta) = d_a(\alpha_{-1} \cup \alpha_1) = d_a(\alpha_{-1}) + d_a(\alpha_1) = -1 + 1 = 0$. Hence, $\eta \in C(S \cup \{a\})$. □

Algorithm 1 computes a set $C \subseteq C(A_0 \cap A_1)$ of compatible global transitions of two square-free multilabeled nets N_0 and N_1, including all square-free, irreducible global transitions (i.e., $C \supseteq \overline{C_0}(A_0 \cap A_1)$). We conjecture that Algorithm 1 actually generates $C = \overline{C_0}(A_0 \cap A_1)$, which means that Algorithm 1 produces only irreducible global transitions.

For clarity, we present distribution (Algorithm 1) in its simplest form. Distribution can be optimized by using an appropriate data structure for set C for efficient constructions of the subsets $C^d = \{\eta \in C \mid d_a(\eta) = d\}$, for $d \in \{-1, 0, 1\}$.

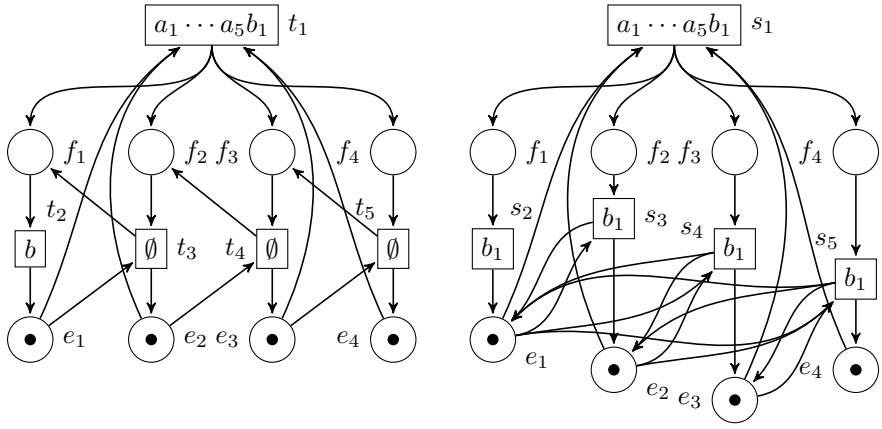
Theorem 6.3.3. *Distribution (Algorithm 1) is totally correct.*

Proof. By Lemma 6.3.2, after $S \subseteq A_0 \cap A_1$ iterations, set C consists of all square-free, irreducible S-compatibles. Finiteness of N_0 and N_1 ensures that $A_0 \cap A_1$ is finite, which implies termination. □

Example 6.3.2 (Alternator)**.** Consider the $\mathsf{alternator}_n$ protocol, $n \geq 2$, defined as the following composition of node, sync, $\mathsf{syncdrain}$, and fifo_1 components:

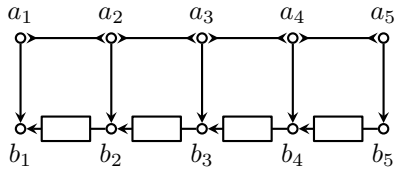
$$\begin{aligned}\mathsf{alternator}_n = {} & \textstyle\prod_{i=1}^{n} \left(\mathsf{node}_{1,3}(a_i; a_i^1, a_i^2, a_i^3) \times \mathsf{node}_{2,1}(b_i^1, b_i^2; b_i) \times \mathsf{sync}(a_i^2, b_i^1)\right) \\ & \times \textstyle\prod_{i=1}^{n-1} \left(\mathsf{syncdrain}(a_i^3, a_{i+1}^1) \times \mathsf{fifo}_1(b_{i+1}, b_i^2)\right),\end{aligned}$$

where the multilabeled net for $\mathsf{syncdrain}$ and sync are identical. For those familiar with the syntax of Reo, $\mathsf{alternator}_5$ has the following diagram:



(a) Composition (b) Abstraction

Figure 6.3: Composition and abstraction of the $\mathsf{alternator}_5$ protocol.



Using Algorithm 1 and Theorem 6.3.3, we can compute the multilabeled net of $\mathsf{alternator}_5$. The composition is shown in Figure 6.3(a). For readability, we do not draw the places induced by the node components. Their contents remains the same throughout the execution of the $\mathsf{alternator}_5$. We also hide all internal actions by removing from the transition labels all actions other than b_1 and $a_1, \ldots, a_5$. We formalize this procedure in Definition 6.4.1.

The multilabeled net in Figure 6.3(a) can be used for the implementation of the $\mathsf{alternator}_5$ protocol. In a naive implementation, a place is implemented as a variable, and a transtion is implemented as a thread. Each thread reads and writes to these variables according to the flow relation, and performs I/O-operations according to the transition label. In particular, transitions with an empty label do not perform any I/O-operation. Of course, care must be taken for variables that are shared amongst different threads. ◊

The number of places in a composition $N_0 \times N_1$ is the sum of the places in N_0 and N_1, which shows that the composition operator in Definition 6.3.3 does not suffer from state-space explosions. The total number of transitions in a composition $N_0 \times N_1$ equals the number of irreducible compatible subsets of N_0 and N_1, which can potentially grow large. The following result shows that, for 'nice compositions', the number of transitions in the composition does not blow up.

Corollary. *The composition $N_0 \times N_1$ has at most $|T_0| + |T_1|$ transitions, if for every $a \in A_0 \cap A_1$ there exists an $i \in \{0, 1\}$ with $|\{t \in T_i \mid \ell(t)(a) > 0\}| \leq 1$.*

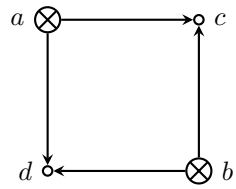


Figure 6.4: Synchronous regions lose concurrency.

Proof. The condition ensures that, in Algorithm 1, $|C^{-1}| = 1$ or $|C^1| = 1$. Hence, the size of C does not increase, which shows that the upper bound holds. □

Example 6.3.3. As shown in [JKA17, Fig. 17(a)], the constraint automaton representation of the $\mathsf{alternator}_n$ protocol does not scale well in $n \geq 1$. However, for multilabeled Petri nets the situation is completely different. Applying Line 5 to the compositions in $\mathsf{alternator}_n$ in Example 6.3.2, it can be shown that, for every $n \geq 1$, the number of transitions of the $\mathsf{alternator}_n$ is equal to n. As such, the multilabeled net representation of the $\mathsf{alternator}_n$ protocol does not suffer from a state-space or a transition-space explosion. ◊

6.3.2 Discussion on concurrency in Reo

Despite the wealth of available semantics for Reo connectors [JA12], most of them focus only on synchronization, but ignore the concurrent behavior of a Reo connector. Since these semantics form the basis of the Reo compilers, these deficiencies in the semantics of Reo leads to problems in the implementation of the Reo language.

For example, Jongmans uses constraint automata (CA) for his compiler for Reo protocols. Since CA are sequential machines, and a sequential implementation of Reo connector is not utilizing all available resources, Jongmans first decomposes the Reo protocol in synchronous regions and runs those regions in parallel with minimal synchronization overhead. In other words, Jongmans partially compensates for the loss of concurrency in CA by considering a decomposition of the CA and synchronizing these components at run time.

Unfortunately, the synchronous regions decomposition does not recover all available concurrency that was present in the original Reo protocol, as demonstrated by the following example:

Example 6.3.4. Consider the connector in Figure 6.4. It is a single synchronous region and it compiled into a single threaded program that executes the constraint automaton in Figure 6.5. The transitions ac and bd are independent and can fire concurrently in the original Reo connector. However, the sequential implementation as a CA loses this independence. ◊

In contrast with the CA semantics, our multilabeled Petri net semantics models both the concurrent and synchronous behavior of a Reo connector. It is captures all available concurrency event within a single synchronous region:

Example 6.3.5. Figure 6.4 shows the multilabeled Petri net for the synchronous regions from Example 6.3.4. The places ensure that no two adjacent transitions

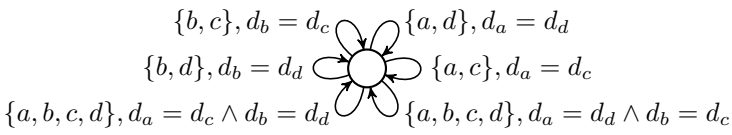


Figure 6.5: Constraint automaton of Figure 6.4

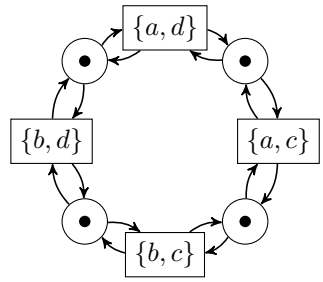


Figure 6.6: Multilabeled Petri net of Figure 6.4

can fire simultaneously. Hence, the $\{a,c\}$ and $\{b,d\}$ can fire at the same time. The same applies to the $\{a,d\}$ and $\{b,c\}$ transitions. $\Diamond$

6.4 Abstraction

The definition of multilabeled Petri nets allows for *silent* transitions, i.e., transitions t with an empty label $\ell(t) = \emptyset$. Such silent transitions can be introduced by hiding internal actions:

Definition 6.4.1 (Hiding)**.** Hiding an action $a \in A$ in a multilabeled net N yields the net $\exists_a N = (B, P, \{({}^\bullet t, \ell(t)|_B, t^\bullet) \mid t \in T\}, \mu)$, where $B = A \setminus \{a\}$ and $\ell(t)|_B$ is the restriction of $\ell(t)$ to B.

The hiding operator $\exists_a$ simply drops all occurrences of a from the label of every transition in the given Petri net.

As indicated in Example 6.3.2, the naive implementation of a silent transition is a thread that does not perform any I/O-operations. As a result, these silent transitions can delay the throughput of the protocol.

In this section, we aim to improve the generated code by transforming a fixed multilabeled net $N = (A, P, T, \mu_0)$ into an equivalent net ∂N without any silent transitions. To define the abstraction operator ∂, we follow the same strategy as for composition of multilabeled nets. First, we consider all possible sequential compositions of transitions. Next, we restrict to sequences of transitions with observable behavior. Finally, we restrict to sequences of transitions that generate all observable traces.

6.4.1 Sequential compositions

Consider the set T^* of all firing sequences of N, i.e., all finite sequences of transitions of N. Following Mazurkiewicz [Maz95], strictly different firing sequences can be considered identical up to permutation of independent transitions.

Definition 6.4.2. The **dependency relation** $D \subseteq T \times T$ is defined as

$$(s,t) \in D \quad \Leftrightarrow \quad s^\bullet \cap {}^\bullet t \neq \emptyset \;\text{ or }\; t^\bullet \cap {}^\bullet s \neq \emptyset \;\text{ or }\; \ell(s) \neq \emptyset \neq \ell(t)$$

Intuitively, transitions s and t are dependent iff one transition takes the output of the other as input, or if both transitions are observable. Note that conflicting transitions are not necessarily dependent.

The dependency relation D induces a **trace equivalence** $\equiv \subseteq T^*$ defined as the smallest congruence on T^*, such that $st \equiv ts$, for all $(s,t) \notin D$. An equivalence class $[x] = \{y \in T^* \mid y \equiv x\}$ of a firing sequence x is called a **trace**.

The **trace monoid** $T^*/\!\equiv$ is the set $\{[x] \mid x \in T^*\}$ of all traces, endowed with composition, defined, for all $x, y \in T^*$, as $[x][y] = [xy]$. Since $\equiv$ is a congruence, composition of traces is well-defined.

Observable behavior of traces is a map $o : T^*/\!\equiv \to (\mathbb{N}^A \setminus \{\emptyset\})^*$ defined, for all $x \in T^*$, as

$$o([\epsilon]) = \epsilon, \qquad o([xt]) = \begin{cases} o([x])\ell(t) & \text{if } \ell(t) \neq \emptyset \\ o([x]) & \text{otherwise} \end{cases}$$

Since observable transitions do not commute (Definition 6.4.2), o is well-defined.

Next, we define map $\sigma : T^*/\!\equiv \to \mathbb{N}^P \times \mathbb{N}^A \times \mathbb{N}^P$ that maps every trace w to a concrete transition $\sigma(w) \in \mathbb{N}^P \times \mathbb{N}^A \times \mathbb{N}^P$. The definition of this map relies on sequential composition of transitions:

Definition 6.4.3. The **sequential composition** $s;t$ of transitions s, t in a multilabeled net N is defined as $({}^\bullet(s;t), \ell(s;t), (s;t)^\bullet)$, where

$${}^\bullet(s;t) = {}^\bullet s \cup ({}^\bullet t \setminus s^\bullet), \qquad (s;t)^\bullet = t^\bullet \cup (s^\bullet \setminus {}^\bullet t), \qquad \ell(s;t) = o([st])$$

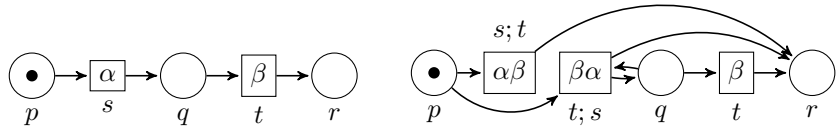
For a sequential composition $s;t$, transition t can use the tokens produced by s. Hence, t consumes only the tokens from the multiset difference ${}^\bullet t \setminus s^\bullet$. Therefore, the sequential composition $s;t$ consumes only the tokens in the multiset union ${}^\bullet s \cup ({}^\bullet t \setminus s^\bullet)$.

Example 6.4.1. Figure 6.7(b) shows some sequential compositions of transitions s and t in Figure 6.7(a). Intuitively, the sequential composition $s;t$ performs t after s. The token generated in place q by s is immediately consumed by t. Therefore, $s;t$ does not have q as input place or output place.

Note, however, that $t;s$ has q both as input place and output place, because the token produced at place q by s comes too late. ◊

Sequential composition of traces is a map $\sigma : T^*/\!\equiv \to \mathbb{N}^P \times \mathbb{N}^A \times \mathbb{N}^P$ defined, for all traces $w \in T^*/\!\equiv$ and transitions $t \in T$, as

$$\sigma([\epsilon]) = (\emptyset, \emptyset, \emptyset), \qquad \sigma(wt) = \sigma(w);t. \tag{6.1}$$



(a) Two dependent transitions (b) Sequential composition

Figure 6.7: Sequential composition of transitions with multilabels α and β.

Lemmas 6.4.1 and 6.4.2 show that σ is a well-defined homomorphism, that is, $\sigma(u)$ does not depend on the representative of $u \in T^*/\equiv$, and $\sigma(uv) = \sigma(u);\sigma(v)$, for all traces $u, v \in T^*/\equiv$.

Lemma 6.4.1. *If $(s,t) \notin D$, then $s;t = t;s$.*

Proof. If $(s,t) \notin D$, then $s^\bullet \cap {}^\bullet t = t^\bullet \cap {}^\bullet s = \emptyset$ and either $\ell(s) \neq \emptyset$ or $\ell(t) \neq \emptyset$. The latter condition implies that $\ell(s;t) = o([st]) = o([ts]) = \ell(t;s)$. The former condition implies that ${}^\bullet t \setminus s^\bullet = {}^\bullet t$ and ${}^\bullet s \setminus t^\bullet = {}^\bullet s$, which implies

$$ {}^\bullet(s;t) = {}^\bullet s \cup ({}^\bullet t \setminus s^\bullet) = ({}^\bullet s \setminus t^\bullet) \cup {}^\bullet t = {}^\bullet(t;s) $$

Similarly, it follows that $(s;t)^\bullet = (t;s)^\bullet$, which proves $s;t = t;s$. □

By construction, sequential composition of traces σ parses each trace as a left-associative composition. Thus, for traces u and v in $T^*/\equiv$, $\sigma(uv)$ and $\sigma(u);\sigma(v)$ could evaluate to different transitions. Lemma 6.4.2 shows that these expressions are equal, and that sequential composition of traces is a homomorphism.

Lemma 6.4.2. *Sequential composition of transitions is associative.*

Proof. The identities in Section 6.1 show, for all transitions $r, s, t \in T$, that

$$ \begin{aligned} {}^\bullet(r;(s;t)) &= {}^\bullet r \cup ({}^\bullet(s;t) \setminus r^\bullet) \\ &= {}^\bullet r \cup (\ (\ {}^\bullet s \cup ({}^\bullet t \setminus s^\bullet)\ ) \setminus r^\bullet\ ) \\ &= {}^\bullet r \cup ({}^\bullet s \setminus r^\bullet) \cup (\ ({}^\bullet t \setminus s^\bullet) \setminus (r^\bullet \setminus {}^\bullet s))\ ) \\ &= {}^\bullet(r;s) \cup (\ {}^\bullet t \setminus (\ s^\bullet \cup (r^\bullet \setminus {}^\bullet s)\ )\ ) \\ &= {}^\bullet(r;s) \cup ({}^\bullet t \setminus (r;s)^\bullet) \\ &= {}^\bullet((r;s);t) \end{aligned} $$

Similarly $(r;(s;t))^\bullet = ((r;s);t)^\bullet$. Since concatenation is associative,

$$ \ell(x;(y;z)) = o(x)o(y)o(z) = \ell((x;y);z), $$

which shows that $x;(y;z)$ and $(x;y);z$. □

Sequential composition of traces induces **observational equivalence** $\approx \subseteq T^*/\equiv$ defined as $v \approx w$ iff $\sigma(v) = \sigma(w)$. Since sequential composition of traces is a homomorphism, observable equivalence $\approx$ is a congruence.

6.4.2 Elimination of silent traces

Consider the set $O = \{w \in T^*/\equiv \mid o(w) \neq \epsilon\}$ of observable traces, which constitutes a proper right-ideal in the trace monoid $T^*/\equiv$. Lemma 6.1.2 shows that every observable trace w, with $o(w) \neq \epsilon$, can be decomposed as a sequence of right-irreducibles in O followed by a silent trace v, with $o(v) = \epsilon$.

Every trace w whose observable behavior $o(w)$ has length greater than one admits a decomposition uv, with $o(u) \neq \epsilon$ and $v \not\equiv \epsilon$. Hence, the observable behavior $o(w)$ of a right-irreducible trace $w \in O$ must have length one.

Nevertheless, the length $|w|$ of the trace w can be arbitrarily large. To shrink the set of traces that generate all observable behavior, we consider two additional conditions.

Definition 6.4.4. A trace w is **acyclic** iff $|w| = \min_{w' \approx w} |w'|$.

Every contiguous subtrace of an acyclic trace is acyclic.

Lemma 6.4.3. *If uvw is acyclic, then v is acyclic.*

Proof. If v is not acyclic, then there exists a trace $v' \approx v$, such that $|v'| < |v|$. Then, $|uv'w| < |uvw|$, and $\sigma(uv'w) = \sigma(u);\sigma(v');\sigma(w) = \sigma(u);\sigma(v);\sigma(w) = \sigma(uvw)$. Hence, $uv'w \approx uvw$, and uvw is not acyclic. □

Definition 6.4.5. A trace w is **k-live** iff $\sigma(w')$ is k-live, for every suffix w' of w.

Every contiguous subtrace of a k-live trace is k-live.

Lemma 6.4.4. *If uvw is k-live, then v is k-live.*

Proof. If v is not k-live, then v can be decomposed as $v = v_0v_1$, such that $\sigma(v_1)$ is not k-live. Then, uvw can be decomposed as $(uv_0)(v_1w)$. From ${}^\bullet\sigma(v_1w) = {}^\bullet(\sigma(v_1);\sigma(w)) = {}^\bullet\sigma(v_1) \cup ({}^\bullet\sigma(w) \setminus \sigma(v_1)^\bullet) \supseteq {}^\bullet\sigma(v_1)$, it follows that $\sigma(v_1w)$ is also not k-live, and that uvw is not k-live. □

The results in Lemmas 6.1.2, 6.4.3 and 6.4.4 show that the trace $w = [t_1 \dots t_n]$ of every firing sequence $t_1 \dots t_n$, $n \geq 1$, can be generated from right-irreducible, acyclic, k-live traces modulo observational equivalence $\approx$. To see this, note that w comes from a firing sequence, which means that w is k-live by construction. Now, select some acyclic $w' \approx w$ (i.e., w' is of minimal length). Lemma 6.1.2 applied for the right-ideal $O = \{u \in T^*/\equiv \mid o(u) \neq \epsilon\}$ yields a decomposition $w' = a_1 \dots a_n b$, where $a_1, \dots, a_n$ are right-irreducible in O, and $o(b) = \epsilon$ is silent. Since w is acylcic and k-live, Lemmas 6.4.3 and 6.4.4 show that the traces $a_1, \dots, a_n$ are right-irreducible, acyclic, and k-live.

Definition 6.4.6. The **abstraction** ∂N of the multilabeled net N is defined as $(A, P, \{\sigma(w) \mid w \text{ right-irreducible, acyclic, and } k\text{-live}\}, \mu)$.

Example 6.4.2. Consider the composite net N in Figure 6.1(c). Hiding actions a and b results in a net $N' = \exists_a \exists_b N$, where send | transfer^2 | receive^2 is a silent transition. From the current marking, the net N' can eventually produce observable behavior. However, to do so, N' must first firing a silent transition.

Application of the abstraction operator yields the $\partial N'$, as shown in Figure 6.1(d). In contrast with N', the abstraction $\partial N'$ can directly fire an observable transition

Algorithm 2: Abstraction

Input : A finite, k-bounded multilabeled net $N = (A, P, T, \mu_0)$.
Output: $H = \{x \in T^* \mid [x]$ right-irreducible, acyclic, and k-live$\}$.

$H \leftarrow \{t \mid t \in T,\ \ell(t) \neq \emptyset\}$;
repeat
  $h \leftarrow |H|$;
  $H \leftarrow H \cup \{sx \mid x \in H,\ s \in T,\ \ell(s) = \emptyset,\ sx \not\equiv xs,\ \sigma([sx]) \notin \sigma(H),\ [sx]\ k$-live$\}$;
until $h = |H|$;

in the current marking. As a result, the abstraction $\partial N'$ produces observable behavior faster than N', and the executable code generated from $\partial N'$ potentially optimizes the code generated from N'. ◊

6.4.3 Abstraction algorithm

The transitions of the abstraction ∂N of a multilabeled net N can be computed by recursion on the length of the underlying traces. Algorithm 2 shows a straightforward tree search that starts from the shortest possible acyclic, k-live, right-irreducible traces, namely the observable transitions.

Theorem 6.4.5. *Algorithm 2 is totally correct.*

Proof. It is routine to check that, after $n \geq 0$ iterations, H contains all acyclic, k-live, right-irreducible traces of length $n + 1$. For termination, observe that a finite, k-bounded net has only finitely many k-live transitions. □

Example 6.4.3. We use Algorithm 2 to eliminate the silent transitions $\mathsf{alternater}_5$ protocol as shown in Figure 6.3(a). Table 6.1 shows the intermediate steps

Figure 6.3(b) shows the resulting multilabeled net $\partial\mathsf{alternator}_5$. In $\partial\mathsf{alternator}_5$, a reader component at the output b_1 of $\mathsf{alternator}_5$ does not need to wait for silent transitions to move the data into the fifo_1 buffer between b_2 and b_1. Instead, the reader can immediately take the data from the first non-empty fifo_1 buffer. As such, the naive implementation of the abstraction $\partial\mathsf{alternator}_5$ should have higher throughput than $\mathsf{alternator}_5$. We did not yet verify this claim experimentally. ◊

6.5 Discussion

We introduce multilabeled Petri nets, i.e., Petri nets whose transitions are labeled by a multiset of actions. We define a binary composition operator for multilabeled nets, and construct an algorithm to compute the composition. We define a unary abstraction operator that removes silent transitions from multilabeled nets, and construct an algorithm to compute the abstraction.

The composition algorithm Algorithm 1 assumes that the multilabeled nets are square-free. Although the assumption of square-freeness is justified in the context

Table 6.1: Abstraction of the multilabeled net in Figure 6.3(a) using Algorithm 2.

w	s	${}^\bullet w$	$w^\bullet$	$\ell(w)$	acyclic	1-live	irreduc.
t_1	s_1	$e_1e_2e_3e_4$	$f_1f_2f_3f_4$	$a_1\cdots a_5b_1$	y	y	y
t_2	s_2	f_1	e_1	b	y	y	y
t_3		e_1f_2	e_2f_1	$\emptyset$	y	y	y
t_4		e_2f_3	e_3f_2	$\emptyset$	y	y	y
t_5		e_3f_4	e_4f_3	$\emptyset$	y	y	y
t_3t_1		$e_1^2e_3e_4f_2$	$f_1^2f_2f_3f_4$	$a_1\cdots a_5b_1$	y		y
t_4t_1		$e_1e_2^2e_4f_3$	$f_1f_2^2f_3f_4$	$a_1\cdots a_5b_1$	y		y
t_5t_1		$e_1e_2e_3^2f_4$	$f_1f_2f_3^2f_4$	$a_1\cdots a_5b_1$	y		y
t_3t_2	s_3	e_1f_2	e_1e_2	b	y	y	y
t_4t_2		$e_2f_1f_3$	$e_1e_3f_2$	b	y	y	
t_5t_2		$e_3f_1f_4$	$e_1e_4f_3$	b	y	y	
$t_3t_3t_2$		$e_1^2f_2^2$	$e_1e_2^2f_1$	b	y		y
$t_4t_3t_2$	s_4	$e_1e_2f_3$	$e_1e_2e_3$	b	y	y	y
$t_5t_3t_2$		$e_1e_3f_2f_4$	$e_1e_2e_4f_3$	b	y	y	
$t_3t_4t_3t_2$		$e_1^2f_2f_3$	$e_1e_2e_3f_1$	b	y		y
$t_4t_4t_3t_2$		$e_1e_2^2f_3^2$	$e_1e_2e_3^2f_2$	b	y		y
$t_5t_4t_3t_2$	s_5	$e_1e_2e_3f_4$	$e_1e_2e_3e_4$	b	y	y	y
$t_3t_5t_4t_3t_2$		$e_1^2e_3f_2f_4$	$e_1e_2e_3e_4f_1$	b	y		y
$t_4t_5t_4t_3t_2$		$e_1e_2^2f_3f_4$	$e_1e_2e_3e_4f_2$	b	y		y
$t_5t_5t_4t_3t_2$		$e_1e_2e_3^2f_4^2$	$e_1e_2e_3e_4^2f_3$	b	y		y

of constraint automata, it would be very useful to be able to automatically compose more general multilabeled nets (such as the running example).

While the definition of the composition operator relies on the concurrent semantics of nets, the definition of the abstraction operator relies on interleaving semantics. As a result, the composition and abstraction operator are not interoperable, in the sense that $\partial(N_0 \times N_1) \neq \partial N_0 \times \partial N_1$, for multilabeled nets N_0 and N_1. Such an identity would allow for simplification of intermediate compositions, which potentially speeds up the construction of the composition.