Chapter 2
Equations Over Languages and Finite Automata

2.1 Preliminaries on Languages and Finite Automata

2.1.1 Languages and Operators

Definition 2.1.1. An alphabet is a finite set of symbols. The set of all finite strings over a fixed alphabet X is denoted by $X^\star$. $X^\star$ includes the empty string ϵ. A subset $L \subseteq X^\star$ is called a **language** over alphabet X.

A language L over the alphabet $X \times V$ is a language over the alphabet $A = \{(x, v) \mid x \in X, v \in V\}$. Standard set-theoretic operations are defined on alphabets, e.g., union and intersection.

Some standard operations on languages are:

1. Given languages L_1 and L_2, respectively over alphabets X_1 and X_2, the language $L_1 \cup L_2$ over alphabet $X_1 \cup X_2$ is the **union** of languages L_1 and L_2.
2. Given languages L_1 and L_2, respectively over alphabets X_1 and X_2, the language $L_1L_2 = \{\alpha\beta \mid \alpha \in L_1, \beta \in L_2\}$ over alphabet $X_1 \cup X_2$ is the **concatenation** of languages L_1 and L_2. Define $L^0 = \{\epsilon\}$, $L^i = LL^{i-1}$. The **Kleene closure** of L is the set $L^\star = \cup_{i=0}^{\infty} L^i$ and the **positive Kleene closure** of L is $L^+ = \cup_{i=1}^{\infty} L^i$. Finally, the l**-bounded Kleene closure** of L is set $L^{\leq l} = \cup_{i=0}^{l} L^i$.
3. Given languages L_1 and L_2, respectively over alphabets X_1 and X_2, the language $L_1 \cap L_2$ over alphabet $X_1 \cap X_2$ is the **intersection** of languages L_1 and L_2. If $X_1 \cap X_2 = \emptyset$ then $L_1 \cap L_2 = \emptyset$.
4. Given a language L over alphabet X, the language $\overline{L} = X^\star \setminus L$ over alphabet X is the **complement** of language L. Similarly, given languages L_1 and L_2, respectively over alphabets X_1 and X_2, the language $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$ over alphabet X_1 is the **difference** of languages L_1 and L_2.
5. Given a language L over alphabet X, the language $Pref(L) = \{x \in X^\star \mid \exists y \in X^\star, xy \in L\}$ is the **prefix-closure** of L, i.e., the language whose words are all the prefixes of words in L.

T. Villa et al., *The Unknown Component Problem: Theory and Applications*,
DOI 10.1007/978-0-387-68759-9_2, © Springer Science+Business Media, LLC 2012

6. Given a language L over alphabet X, L^{Pref} is the **largest prefix-closed language** L' with $L' \subseteq L$.

It is useful to recall the notions of substitution and homomorphism of languages [63]. A **substitution** f is a mapping of an alphabet X onto subsets of $Y^\star$ for some alphabet Y. The substitution f is extended to strings by setting $f(\epsilon) = \{\epsilon\}$ and $f(xa) = f(x)f(a)$. An **homomorphism** h is a substitution such that $h(a)$ is a singleton string for each symbol a in the alphabet X. We introduce some useful operations on languages. The first two are associated with synchronous composition, while the last two are associated with parallel composition. These operations are integral parts of constructing the most general solution.

1. Given a language L over alphabet $X \times V$, consider the homomorphism $p : X \times V \to V^\star$ defined as
$$p((x, v)) = v,$$
then the language
$$L_{\downarrow V} = \{p(\alpha) \mid \alpha \in L\}$$
over alphabet V is the **projection** of language L to alphabet V, or V-projection of L. By definition of substitution $p(\epsilon) = \epsilon$.
2. Given a language L over alphabet X and an alphabet V, consider the substitution $l : X \to 2^{(X\times V)^\star}$ defined as
$$l(x) = \{(x, v) \mid v \in V\},$$
then the language
$$L_{\uparrow V} = \{l(\alpha) \mid \alpha \in L\}$$
over alphabet $X \times V$ is the **lifting** of language L to alphabet V, or V-lifting of L. By definition of substitution $l(\epsilon) = \{\epsilon\}$.
3. Given a language L over alphabet $X \cup V$, consider the homomorphism $r : X \cup V \to V^\star$ defined as
$$r(y) = \begin{cases} y \text{ if } y \in V \\ \epsilon \text{ if } y \in X \setminus V \end{cases},$$
then the language
$$L_{\Downarrow V} = \{r(\alpha) \mid \alpha \in L\}$$
over alphabet V is the **restriction** of language L to alphabet V, or V-restriction of L, i.e., words in $L_{\Downarrow V}$ are obtained from those in L by deleting all the symbols in X that are not in V. By definition of substitution $r(\epsilon) = \epsilon$.
4. Given a language L over alphabet X and an alphabet V, consider the mapping $e : X \to 2^{(X\cup V)^\star}$ defined as
$$e(x) = \{\alpha x \beta \mid \alpha, \beta \in (V \setminus X)^\star\},$$

then the language

$$L_{\Uparrow V} = \{e(\alpha) \mid \alpha \in L\}$$

over alphabet $X \cup V$ is the **expansion** of language L to alphabet V, or V-expansion of L, i.e., words in $L_{\Uparrow V}$ are obtained from those in L by inserting anywhere in them words from $(V \setminus X)^\star$. Notice that e is not a substitution and that $e(\epsilon) = \{\alpha \mid \alpha \in (V \setminus X)^\star\}$.

Given a language L over alphabet X, an alphabet V, and a natural number l, consider the mapping $e_l : X \to 2^{(X \cup V)^l}$ defined as

$$e_l(x) = \{\alpha x \beta \mid \alpha, \beta \in (V \setminus X)^{\leq l}\},$$

then the language

$$L_{\Uparrow (V,l)} = \{e_l(\alpha) \mid \alpha \in L\}$$

over alphabet $X \cup V$ is the ***l*-bounded expansion** of language L over alphabet V, or (V, l)-expansion of L, i.e., words in $L_{\Uparrow V}$ are obtained from those in L by inserting anywhere in them words from $(V \setminus X)^{\leq l}$. Notice that e_l is not a substitution and that $e_l(\epsilon) = \{\alpha \mid \alpha \in (V \setminus X)^{\leq l}\}$.

By definition $\emptyset_{\downarrow V} = \emptyset, \emptyset_{\uparrow V} = \emptyset, \emptyset_{\Downarrow V} = \emptyset, \emptyset_{\Uparrow V} = \emptyset, \emptyset_{\Uparrow (V,l)} = \emptyset$.

The four previous operators change a language and its alphabet of definition; in particular the operators $\uparrow$ and $\downarrow$ change what components are present in the Cartesian product that defines the language alphabet. We assume that each component has a fixed position in the Cartesian product. For instance, let language L_1 be defined over alphabet I and language L_2 be defined over alphabet O, then language $L_{1\uparrow O}$ is defined over alphabet $I \times O$ and also language $L_{2\uparrow I}$ is defined over alphabet $I \times O$, if by assumption I precedes O in the Cartesian product. More precisely, say that we introduce an ordering of alphabets, i, by which I is mapped to index $i(I)$ and O is mapped to $i(O)$, then $i(I) < i(O)$ implies that I precedes O in any Cartesian product of alphabets. The ordering is arbitrary, but, once chosen, it holds through the sequence of language operations.

The following straightforward facts hold between the projection and lifting operators, and between the restriction and expansion operators.

Proposition 2.1. *The following inverse laws for $\uparrow, \downarrow$ and $\Uparrow, \Downarrow$ hold.*

(a) Let X and Y be alphabets, and let L be a language over alphabet X, then $(L_{\uparrow Y})_{\downarrow X} = L$.
(b) Let X and Y be alphabets, and let L be a language over alphabet $X \times Y$, then $(L_{\downarrow X})_{\uparrow Y} \supseteq L$.
(c) Let X and Y be disjoint alphabets, and let L be a language over alphabet X, then $(L_{\Uparrow Y})_{\Downarrow X} = L$.
(d) Let X and Y be disjoint alphabets, and let L be a language over alphabet $X \cup Y$, then $(L_{\Downarrow X})_{\Uparrow Y} \supseteq L$.

Proposition 2.2. *The following equivalences hold.*

(a) Given alphabets X and Y, a language L over alphabet X, and a string $\alpha \in (X \times Y)^\star$, then $\alpha_{\downarrow X} \in L \Leftrightarrow \alpha \in L_{\uparrow Y}$.
(b) Given disjoint alphabets X and Y, a language L over alphabet X, and a string $\alpha \in (X \cup Y)^\star$, then $\alpha_{\Downarrow X} \in L \Leftrightarrow \alpha \in L_{\Uparrow Y}$.

Proposition 2.3. *The following distributive laws for $\uparrow$ and $\downarrow$ hold.*

(a) Let L_1, L_2 be languages over alphabet U. Then $\uparrow$ commutes with $\cup$

$$(L_1 \cup L_2)_{\uparrow I} = L_{1\uparrow I} \cup L_{2\uparrow I}.$$

(b) Let L_1, L_2 be languages over alphabet U. Then $\uparrow$ commutes with $\cap$

$$(L_1 \cap L_2)_{\uparrow I} = L_{1\uparrow I} \cap L_{2\uparrow I}.$$

(c) Let M_1, M_2 be languages over alphabet $I \times U$. Then $\downarrow$ commutes with $\cup$

$$(M_1 \cup M_2)_{\downarrow U} = M_{1\downarrow U} \cup M_{2\downarrow U}.$$

(d) Let M_1, M_2 be languages over alphabet $I \times U$. If $M_2 = (M_{2\downarrow U})_{\uparrow I}$ (or $M_1 = (M_{1\downarrow U})_{\uparrow I}$), then $\downarrow$ commutes with $\cap$

$$(M_1 \cap M_2)_{\downarrow U} = M_{1\downarrow U} \cap M_{2\downarrow U}.$$

Proof. Thesis: $(L_1 \cap L_2)_{\uparrow I} = L_{1\uparrow I} \cap L_{2\uparrow I}$.

($\Rightarrow$) If the string $(i_1,u_1)\dots(i_k,u_k) \in (L_1 \cap L_2)_{\uparrow I}$, then $u_1 \dots u_k \in L_1 \cap L_2$; thus $u_1 \dots u_k \in L_1$, $u_1 \dots u_k \in L_2$, and so $(i_1,u_1)\dots(i_k,u_k) \in L_{1\uparrow I}$, $(i_1,u_1)\dots(i_k,u_k) \in L_{2\uparrow I}$, implying $(i_1,u_1)\dots(i_k,u_k) \in L_{1\uparrow I} \cap L_{2\uparrow I}$.

($\Leftarrow$) If the string $(i_1,u_1)\dots(i_k,u_k) \in L_{1\uparrow I} \cap L_{2\uparrow I}$, then $(i_1,u_1)\dots(i_k,u_k) \in L_{1\uparrow I}$, $(i_1,u_1)\dots(i_k,u_k) \in L_{2\uparrow I}$; thus $u_1 \dots u_k \in L_1$, $u_1 \dots u_k \in L_2$, implying $u_1 \dots u_k \in L_1 \cap L_2$, and so $(i_1,u_1)\dots(i_k,u_k) \in (L_1 \cap L_2)_{\uparrow I}$.

Similarly one proves the first and third identity involving $\cup$.

Thesis: If $M_2 = (M_{2\downarrow U})_{\uparrow I}$ (or $M_1 = (M_{1\downarrow U})_{\uparrow I}$), then $(M_1 \cap M_2)_{\downarrow U} = M_{1\downarrow U} \cap M_{2\downarrow U}$.

($\Rightarrow$) If the string $u_1 \dots u_k \in (M_1 \cap M_2)_{\downarrow U}$ then there exists $i_1 \dots i_k$ such that $(i_1,u_1)\dots(i_k,u_k) \in M_1 \cap M_2$, i.e., $(i_1,u_1)\dots(i_k,u_k) \in M_1$, $(i_1,u_1)\dots(i_k,u_k) \in M_2$, and so $u_1 \dots u_k \in M_{1\downarrow U}$ and $u_1 \dots u_k \in M_{2\downarrow U}$.

($\Leftarrow$) If the string $u_1 \dots u_k \in M_{1\downarrow U} \cap M_{2\downarrow U}$, i.e., $u_1 \dots u_k \in M_{1\downarrow U}$ and $u_1 \dots u_k \in M_{2\downarrow U}$, then there exists $i_1 \dots i_k$ such that $(i_1,u_1)\dots(i_k,u_k) \in M_1$. Moreover, since $M_2 = (M_{2\downarrow U})_{\uparrow I}$, from $u_1 \dots u_k \in M_{2\downarrow U}$ it follows that $(i_1,u_1)\dots(i_k,u_k) \in M_2$. In summary, $(i_1,u_1)\dots(i_k,u_k) \in M_1$ and $(i_1,u_1)\dots(i_k,u_k) \in M_2$, implying $(i_1,u_1)\dots(i_k,u_k) \in M_1 \cap M_2$, from which follows $u_1 \dots u_k \in (M_1 \cap M_2)_{\downarrow U}$. □

Corollary 2.4. *The following commutative laws for $\uparrow$ and $\downarrow$ hold.*

(a) Let $L_i, i = 1, \ldots, n$, be languages over alphabet U. Then $\uparrow$ commutes with both $\cup$ and $\cap$

$$(\cup L_i)_{\uparrow I} = \cup(L_i \uparrow_I),$$
$$(\cap L_i)_{\uparrow I} = \cap(L_i \uparrow_I).$$

(b) Let $M_i, i = 1, \ldots, n$, be languages over alphabet $I \times U$. Then $\downarrow$ commutes with $\cup$

$$(\cup M_i)_{\downarrow U} = \cup(M_i \downarrow_U).$$

(c) Let $M_i, i = 1, \ldots, n$, be languages over alphabet $I \times U$. If $M_2 = (M_2 \downarrow_U)_{\uparrow I}$, $\ldots$, $M_n = (M_n \downarrow_U)_{\uparrow I}$ (or any collection of $n-1$ languages M_i satisfies this property), then $\downarrow$ commutes with $\cap$

$$(\cap M_i)_{\downarrow U} = \cap(M_i \downarrow_U).$$

The proof is by induction based on Prop. 2.3.

Proposition 2.5. *Suppose that I and U are disjoint alphabets. The following distributive laws for $\Uparrow$ and $\Downarrow$ hold.*

(a) Let L_1, L_2 be languages over alphabet U. Then $\Uparrow$ commutes with $\cup$

$$(L_1 \cup L_2)_{\Uparrow I} = L_1 \Uparrow_I \cup L_2 \Uparrow_I.$$

(b) Let L_1, L_2 be languages over alphabet U. Then $\Uparrow$ commutes with $\cap$

$$(L_1 \cap L_2)_{\Uparrow I} = L_1 \Uparrow_I \cap L_2 \Uparrow_I.$$

(c) Let M_1, M_2 be languages over alphabet $I \cup U$. Then $\Downarrow$ commutes with $\cup$

$$(M_1 \cup M_2)_{\Downarrow U} = M_1 \Downarrow_U \cup M_2 \Downarrow_U.$$

(d) Let M_1, M_2 be languages over alphabet $I \cup U$. If $M_2 = (M_2 \Downarrow_U)_{\Uparrow I}$ (or $M_1 = (M_1 \Downarrow_U)_{\Uparrow I}$) then $\Downarrow$ commutes with $\cap$

$$(M_1 \cap M_2)_{\Downarrow U} = M_1 \Downarrow_U \cap M_2 \Downarrow_U.$$

Proof. Thesis: $(L_1 \cap L_2)_{\Uparrow I} = L_1 \Uparrow_I \cap L_2 \Uparrow_I$.

($\Rightarrow$) If the string $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in (L_1 \cap L_2)_{\Uparrow I}$ and $\alpha_1, \ldots, \alpha_k, \alpha_{k+1} \in I^\star$, then $u_1 \ldots u_k \in L_1 \cap L_2$; thus $u_1 \ldots u_k \in L_1$, $u_1 \ldots u_k \in L_2$, and so $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_1 \Uparrow_I$, $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_2 \Uparrow_I$, implying $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_1 \Uparrow_I \cap L_2 \Uparrow_I$.

($\Leftarrow$) If the string $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_1 \Uparrow_I \cap L_2 \Uparrow_I$, then it holds that $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_1 \Uparrow_I$ and $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in L_2 \Uparrow_I$; thus $u_1 \ldots u_k \in L_1$, $u_1 \ldots u_k \in L_2$, implying $u_1 \ldots u_k \in L_1 \cap L_2$, and so it is also $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in (L_1 \cap L_2)_{\Uparrow I}$.

Similarly one proves the first and third identity involving $\cup$.

Thesis: if $M_2 = (M_2 \Downarrow_U)\Uparrow_I$ (or $M_1 = (M_1 \Downarrow_U)\Uparrow_I$) then $(M_1 \cap M_2)\Downarrow_U = M_1 \Downarrow_U \cap M_2 \Downarrow_U$.

($\Rightarrow$) If the string $u_1 \ldots u_k \in (M_1 \cap M_2)\Downarrow_U$ then there exists $\alpha_1, \ldots \alpha_k, \alpha_{k+1} \in I^\star$ such that it holds that the string $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_1 \cap M_2$, i.e., $\alpha_1 u_1 \ldots \alpha_k u_k$ $\alpha_{k+1} \in M_1$, $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_2$, and so $u_1 \ldots u_k \in M_1 \Downarrow_U$ and $u_1 \ldots u_k \in M_2 \Downarrow_U$.

($\Leftarrow$) If the string $u_1 \ldots u_k \in M_1 \Downarrow_U \cap M_2 \Downarrow_U$, i.e., $u_1 \ldots u_k \in M_1 \Downarrow_U$ and $u_1 \ldots u_k \in M_2 \Downarrow_U$, then there exists $\alpha_1 \ldots \alpha_k \alpha_{k+1} \in I^\star$ such that $\alpha_1 u_1 \ldots \alpha_k u_k$ $\alpha_{k+1} \in M_1$. Moreover, since $M_2 = (M_2 \Downarrow_U)\Uparrow_I$, from $u_1 \ldots u_k \in M_2 \Downarrow_U$ it follows that $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_2$. In summary, $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_1$ and $\alpha_1 u_1 \ldots \alpha_k\ u_k \alpha_{k+1} \in M_2$, implying $\alpha_1 u_1 \ldots \alpha_k u_k \alpha_{k+1} \in M_1 \cap M_2$, from which follows $u_1 \ldots u_k \in (M_1 \cap M_2)\Downarrow_U$. □

Example 2.6. The identity $(M_1 \cap M_2)\Downarrow_U = M_1 \Downarrow_U \cap M_2 \Downarrow_U$ does not hold without the additional hypothesis in Prop. 2.5(d). Consider $I = \{a, b\}$, $U = \{u\}$, $M_1 = \{au\}$, $M_2 = \{bu\}$, then $(M_1 \cap M_2)\Downarrow_U = \emptyset\Downarrow_U = \emptyset$ and $M_1 \Downarrow_U \cap M_2 \Downarrow_U = \{u\} \cap \{u\} = \{u\}$. Notice that au and bu are words of length 2 on the alphabet $I \cup U$.

Proposition 2.7. *The following equivalences hold.*

(a) Let L be a language over alphabet I, then $L_{\uparrow O} = \emptyset \Leftrightarrow L = \emptyset$
(b) Let L be a language over alphabet $I \times O$, then $L_{\downarrow O} = \emptyset \Leftrightarrow L = \emptyset$ In the next two statements, suppose that I and O are disjoint alphabets.
(c) Let L be a language over alphabet $I \cup O$, then $L_{\Uparrow O} = \emptyset \Leftrightarrow L = \emptyset$
(d) Let L be a language over alphabet $I \cup O$, then $L_{\Downarrow O} = \emptyset \Leftrightarrow L = \emptyset$.

Proof. The proofs are straightforward; implication $\Rightarrow$ of statement (d) is true because, even in the case that all strings in L are defined only over symbols from alphabet I, their restriction to alphabet O yields the empty string ϵ (i.e., $\epsilon \in L_{\Downarrow O} \neq \emptyset$) and so from $L \neq \emptyset$ follows that $L_{\Downarrow O} \neq \emptyset$. □

2.1.2 Finite Automata and Regular Expressions

Definition 2.1.2. A **finite automaton** (FA) is a 5-tuple $F = \langle S, \Sigma, \Delta, r, Q\rangle$. S represents the finite state space, Σ represents the finite alphabet, and $\Delta \subseteq \Sigma \times S \times S$ is the next state relation, such that $(i, p, n) \in \Delta$ iff $n \in S$ is a next state of present state $p \in S$ on symbol $i \in \Sigma$. The initial or reset state is $r \in S$ and $Q \subseteq S$ is the set of final or accepting states. A variant of FAs allows the introduction of ϵ-moves, meaning that $\Delta \subseteq (\Sigma \cup \{\epsilon)\} \times S \times S$.

The next state relation Δ can be extended to have as argument strings in Σ^* (i.e., $\tilde{\Delta} \subseteq \Sigma^* \times S \times S$) as follows: $(\rho i, s, s'') \in \tilde{\Delta}$ iff there exists $s' \in S$ such that $(\rho, s, s') \in \tilde{\Delta}$ and $(i, s', s'') \in \Delta$. For ease of notation we just use the symbol Δ for both relations and let the context determine the meaning.

A string x is said to be **accepted** by the FA F if there exists a sequence of transitions corresponding to x such that there exists a state $r' \in Q$ for which $\Delta(x, r, r')$. The **language** accepted by F, designated $L_r(F)$, is the set of strings $\{x \mid \exists r' \in Q \ [\Delta(x, r, r')]\}$. The language accepted or **recognized** by $s \in S$, denoted $L_r(F|s)$ or $L_r(s)$ when F is clear from the context, is the set of strings $\{x \mid \Delta(x, r, s)\}$.

If for each present state p and symbol i there is at least one next state n such that $(i, p, n) \in \Delta$, the FA is said to be **complete**, otherwise it is **partial**. A partial automaton can be made complete by directing the unspecified transitions to a non-accepting state.

An FA is a **deterministic finite automaton** (DFA) if for each present state p and symbol i there is exactly one next state n such that $(i, p, n) \in \Delta$.[1] The relation Δ can be replaced by the next state function δ, defined as $\delta : \Sigma \times S \to S$, where $n \in S$ is the next state of present state $p \in S$ on symbol $i \in \Sigma$ iff $n = \delta(i, p)$. An FA that is not a DFA is a **non-deterministic finite automaton** (NDFA).

A string x is said to be **accepted** by the DFA F if $\delta(x, r) \in Q$. The **language** accepted by F, designated $L_r(F)$, is the set of strings $\{x \mid \delta(x, r) \in Q\}$. The language accepted or **recognized** by $s \in S$, denoted $L_r(F|s)$ or $L_r(s)$ when F is clear from the context, is the set of strings $\{x \mid \delta(x, r) = s\}$.

The languages associated with finite automata are the regular languages, defined by means of regular expressions, as established by the theorem due to Kleene [63].

Definition 2.1.3. The **regular expressions** over an alphabet Σ are defined recursively as follows:

1. $\emptyset$ is a regular expression and denotes the empty set.
2. ϵ is a regular expression and denotes the set $\{\epsilon\}$.
3. For each $a \in \Sigma$, a is a regular expression and denotes the set $\{a\}$.
4. If l_1 and l_2 are regular expressions denoting the languages L_1 and L_2, respectively, then $(l_1 + l_2)$, $(l_1 l_2)$ and $(l_1^\star)$ are regular expressions that denote the sets $L_1 \cup L_2$, $L_1 L_2$ and $L_1^\star$, respectively.

The sets denoted by regular expressions are the **regular languages**.

Regular languages are closed under union, concatenation, complementation and intersection. Also regular languages are closed under projection, lifting and restriction, because they are closed under substitution [63]. Regular languages are closed under expansion, as shown in Sect. 2.3.1 providing an algorithm that, given the finite automaton of a language, returns the finite automaton of the expanded language.

[1] By the given definition in agreement with standard textbooks (see [63]), a DFA must be a complete FA, but a complete FA does not need to be deterministic.

2.1.3 Classes of Languages

We introduce several classes of languages used later in the paper.

Definition 2.1.4. A language L over alphabet X is **prefix-closed** if $\forall \alpha \in X^\star \; \forall x \in X \; [\alpha x \in L \Rightarrow \alpha \in L]$. Equivalently, L is prefix-closed iff $L = \mathit{Pref}(L)$.

Definition 2.1.5. A language L over alphabet $X = I \times O$ is **I-progressive** if

$$\forall i \in I \; \exists o \in O \; [\alpha \in L \Rightarrow \alpha \; (i, o) \in L].$$

If L is not I-progressive, then $\mathit{Prog}(L)$ is the **largest I-progressive language** L' such that $L' \subseteq L$.

Definition 2.1.6. A language L over alphabet $I \times O$ is **$I_{\downarrow}$-defined** if $L_{\downarrow I} = I^\star$.

If a language over $X = I \times O$ is I-progressive it is also $I_{\downarrow}$-defined, but the converse does not hold.

Example 2.8. The language $L = \{\epsilon + i_1 o_1 + i_1 o_2 (i_1 o_1)^\star\}$ is $I_{\downarrow}$-defined, but not I-progressive, as witnessed by $\alpha = i_1 o_1 \in L$ and $i = i_1$ for which there is no o such that $\alpha \; i_1 o \in L$.

Definition 2.1.7. A language L over alphabet $X = I \times O$ is **Moore**[2] with respect to alphabet I, if

$$\forall \alpha \in L \; \forall (i, o) \in X \; \forall (i', o') \in X \; [\alpha \; (i, o) \in L \Rightarrow [\alpha \; (i', o') \in L \Rightarrow \alpha \; (i', o) \in L]].$$

Definition 2.1.8. A language $L \subseteq (IO)^\star$ over alphabet $I \cup O$ (I and O disjoint) is **IO-prefix-closed** if

$$\forall \alpha \in (IO)^\star \; \forall i o \in IO \; [\alpha \; i o \in L \Rightarrow \alpha \in L].$$

Definition 2.1.9. A language $L \subseteq (IO)^\star$ over alphabet $I \cup O$ (I and O disjoint) is **IO-progressive** if

$$\forall i \in I \; \exists o \in O \; [\alpha \in L \Rightarrow \alpha \; io \in L].$$

Definition 2.1.10. A language $L \subseteq (IU^\star O)^\star$ over alphabet $I \cup U \cup O$ (I, U and O pairwise disjoint) is **$IU^\star O$-progressive** if

$$\forall i \in I \; \exists \beta \in U^\star \; \exists o \in O \; [\alpha \in L \Rightarrow \alpha \; i \beta o \in L].$$

[2]This definition is an abstraction to languages of the most common definition of Moore automata/finite state machines.

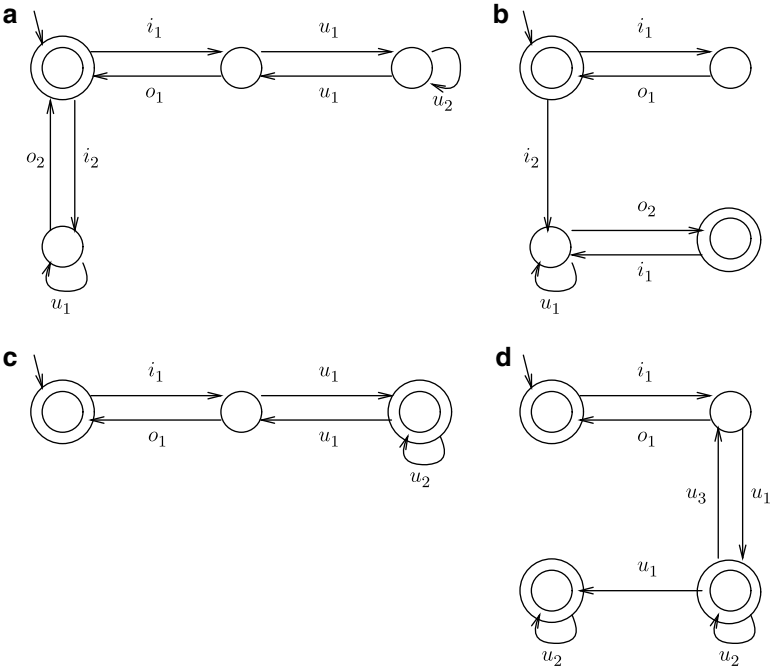

Fig. 2.1 (**a**) Finite automaton of the language described in Example 2.9-a; (**b**) Finite automaton of the language described in Example 2.9-b; (**c**) Finite automaton of the language described in Example 2.10-a; (**d**) Finite automaton of the language described in Example 2.10-b

Example 2.9. (a) Let $I = \{i_1, i_2\}$, $O = \{o_1, o_2\}$ and $U = \{u_1, u_2\}$. The language $L = \{(i_1 u_1 u_2^\star u_1 o_1 + i_2 u_1^\star o_2)^\star\}$ is $UI^\star O$-progressive, since any word in L can be extended to a word in L by suffixes starting with either i_1 or i_2. The corresponding automaton is shown in Fig. 2.1a.

(b) Let $I = \{i_1, i_2\}$, $O = \{o_1, o_2\}$ and $U = \{u_1\}$. The language $L = \{(i_1 o_1)^\star + (i_1 o_1)^\star i_2 u_1^\star o_2 (i_1 u_1^\star o_2)^\star\}$ is not $IU^\star O$-progressive, since the words in the set $\{i_2 u_1^\star o_2 (i_1 u_1^\star o_2)^\star\}$ are in L, but when $i = i_2$ there is no $\beta \in U^\star$ and no $o \in O$ such that $\alpha\, i_2 \beta o \in L$ (e.g., $\alpha = i_2 u_1 o_2$ cannot be extended by any suffix starting with i_2). The corresponding automaton is shown in Fig. 2.1b.

Definition 2.1.11. A language L over alphabet $I \cup O$ (I and O disjoint) is $I_{\Downarrow}$**-defined** if $L_{\Downarrow I} = I^\star$.

An IO-progressive language is $I_{\Downarrow}$-defined, so is an $IU^\star O$-progressive language, but the converse does not hold.

Definition 2.1.12. A language $L \neq \emptyset$ over alphabet $X \cup U$ (X and U disjoint) is U**-deadlock-free** if

$$\forall \alpha \in (X \cup U)^\star \; \forall u \in U \; \exists \beta \in U^\star \; \exists x \in X \; [\alpha u \in L \Rightarrow \alpha u \, \beta x \in L].$$

Any language $L \subseteq (IU^\star O)^\star$ is U-deadlock-free (because no word ending by a symbol $u \in U$ belongs to the language).

Example 2.10. (a) Let $X = I \cup O$, $I = \{i_1, i_2\}$, $O = \{o_1, o_2\}$ and $U = \{u_1, u_2\}$. The language $L = \{(i_1(u_1u_2^\star u_1)^\star o_1)^\star + (i_1(u_1u_2^\star u_1)^\star o_1)^\star i_1u_1u_2^\star\}$ is U-deadlock-free, because any word in the language terminating by u_1 or u_2 can be extended by suffix u_1 to a word in the language terminating by o_1. The corresponding automaton is shown in Fig. 2.1c.

(b) Let $X = I \cup O$, $I = \{i_1, i_2\}$, $O = \{o_1, o_2\}$ and $U = \{u_1, u_2, u_3\}$. The language $L = \{i_1(u_1u_2^\star u_3)^\star o_1)^\star + (i_1(u_1u_2^\star u_3)^\star o_1)^\star i_1u_1u_2^\star + (i_1(u_1u_2^\star u_3)^\star o_1)^\star i_1u_1u_2^\star\, u_1u_2^\star\}$ is not U-deadlock-free, since the words in the collection $\{(i_1(u_1u_2^\star u_3)^\star o_1)^\star i_1\, u_1u_2^\star u_1u_2^\star\}$ cannot be extended to words in L (e.g., $\alpha = i_1u_1u_2u_1$). The corresponding automaton is shown in Fig. 2.1d.

Definition 2.1.13. A language $L \neq \emptyset$ over alphabet $X \cup U$ (X and U disjoint) is **U-convergent** if $\forall \alpha \in X^\star$ the language $\alpha_{\Uparrow U} \cap L$ is finite, otherwise it is **U-divergent**.

Example 2.11. The language $L = \{iu^\star o\}$ where $X = \{i, o\}$ and $U = \{u\}$ is U-divergent, as witnessed by the string $\alpha = io \in X$ whose expansion includes the infinite set $\{iu^\star o\}$ coinciding with L: $\{\alpha_{\Uparrow U}\} = \{(io)_{\Uparrow\{u\}}\} = \{u^\star iu^\star ou^\star\} \supset \{iu^\star o\} = L$.

2.1.4 Composition of Languages

Consider two systems A and B with associated languages $L(A)$ and $L(B)$. The systems communicate with each other by a channel U and with the environment by channels I and O. We introduce two composition operators that describe the external behavior of the composition of $L(A)$ and $L(B)$.

Definition 2.1.14. Given the pairwise disjoint alphabets I, U, O, language L_1 over $I \times U$ and language L_2 over $U \times O$, the **synchronous composition** of languages L_1 and L_2 is the language[3] $[(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}]_{\downarrow I \times O}$, denoted by $L_1 \bullet_{I \times O} L_2$, defined over $I \times O$.

Definition 2.1.15. Given the pairwise disjoint alphabets I, U, O, language L_1 over $I \cup U$ and language L_2 over $U \cup O$, the **parallel composition** of languages L_1 and L_2 is the language $[(L_1)_{\Uparrow O} \cap (L_2)_{\Uparrow I}]_{\Downarrow I \cup O}$, denoted by $L_1 \diamond_{I \cup O} L_2$, defined over $I \cup O$.

Given alphabets I, U, O, language L_1 over $I \cup U$ and language L_2 over $U \cup O$, the **l-bounded parallel composition** of languages L_1 and L_2 is the language $[(L_1)_{\Uparrow O} \cap (L_2)_{\Uparrow I} \cap (I \cup O)^\star_{\Uparrow(U,l)}]_{\Downarrow I \cup O}$, denoted by $L_1 \diamond_{l_{I \cup O}} L_2$, defined over $I \cup O$.

[3] Use the same order $I \times U \times O$ in the languages $(L_1)_{\uparrow O}$ and $(L_2)_{\uparrow I}$.

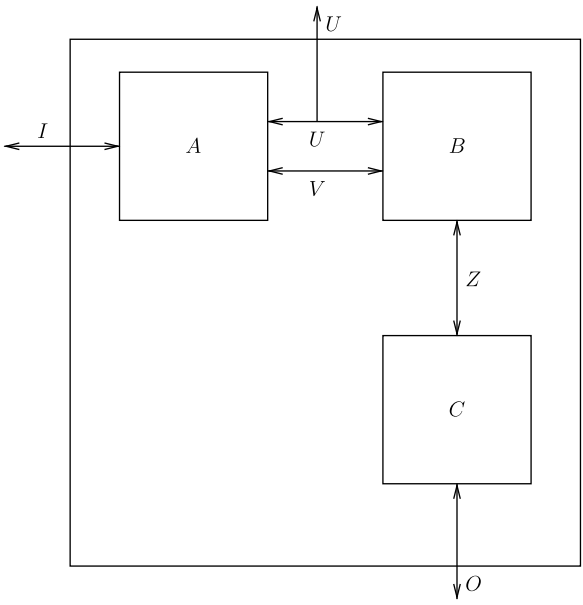

Fig. 2.2 Composition topology with three components

For ease of notation, we will omit the alphabet from the symbol of synchronous and parallel composition, unless unclear from the context. By definition of the operations $\downarrow_V$, $\uparrow_V$, $\Downarrow_V$, $\Uparrow_V$, $\Uparrow_{(V,l)}$ it follows that $\emptyset \bullet L = L \bullet \emptyset = \emptyset$, $\emptyset \diamond L = L \diamond \emptyset = \emptyset$, $\emptyset \diamond_l L = L \diamond_l \emptyset = \emptyset$.

When $l = \infty$ the definition of l-bounded parallel composition reduces to the definition of parallel composition of languages, because then $(I \cup O)^\star_{\Uparrow(U,l)}$ becomes $(I \cup O \cup U)^\star$, that is the universe over $I \cup O \cup U$, and so it can be dropped from the conjunction.

These definitions can be easily extended to more components and more complex interconnection topologies, e.g., to the topology where U is observable externally or where U is the cartesian product of two alphabets only one of which is observable.

For instance in Fig. 2.2 we show a composition topology with three components: A, B and C, which together define the composed system:

$$(A_{\uparrow Z \times O} \cap B_{\uparrow I \times O} \cap C_{\uparrow I \times U \times V})_{\downarrow I \times U \times O}.$$

We notice that I and O are external variables, U is an internal variable that is observable externally, whereas V and Z are internal variables. The composition is well-formed because the synchronous composition operator is associative [150]. Each specific topology dictates the alphabets to which projection and lifting (restriction and expansion) should be applied. In the most straightforward

composition topology with two components, we will assume that I, O are external alphabets, and U is an internal alphabet; however, U or a part of it can be made observable externally, if needed.

Comment. The definition of parallel composition justifies *a-posteriori* why the expansions operator e is not defined to be a substitution, i.e., $e(\epsilon) \neq \{\epsilon\}$. Consider a language $A = ((io)^\star(uv)^\star)^\star$ and a language B whatsoever. The parallel composition of A and B should be equal to the language $(io)^\star$, because B should not affect the $I \cup O$ behavior of A. Now suppose $B = \{\epsilon\}$. If we would define $e(\epsilon) = \{\epsilon\}$, then it would be $A \cap B_{\Uparrow I \cup O} = ((io)^\star(uv)^\star)^\star \cap \{\epsilon\}_{\Uparrow I \cup O} = ((io)^\star(uv)^\star)^\star \cap \{\epsilon\} = \{\epsilon\}$; if we define instead $\{\epsilon\}_{\Uparrow I \cup O} = (I \cup O)^\star$ then it is $A \cap B_{\Uparrow I \cup O} = ((io)^\star(uv)^\star)^\star \cap \{\epsilon\}_{\Uparrow I \cup O} = ((io)^\star(uv)^\star)^\star \cap (i \cup o)^\star = (io)^\star$, that is the expected result.

Variants of *synchronous composition* are introduced in [25] as *product*, $\times$ (with the comment *sometimes called completely synchronous composition*), and in [82] as *synchronous parallel composition*, $\otimes$. Variants of *parallel composition* are introduced in [25] as *parallel composition*, $\|$ (with the comment *often called synchronous composition*), and in [82] as *interleaving parallel composition*, $\|$; the same operator was called *asynchronous composition* in [107]. These definitions were usually introduced for regular languages; actually they were more commonly given for finite automata.

It has also been noticed by Kurshan [82] and Arnold [2] that asynchronous systems can also be modeled with the synchronous interpretation, using null transitions to keep a transition system in the same state for an arbitrary period of time. Kurshan [82] observes that: "While synchronous product often is thought to be a simple -even uninteresting!- type of coordination, it can be shown that, through use of nondeterminism, this conceptually simple coordination serves to model the most general 'asynchronous' coordination, i.e., where processes progress at arbitrary rates relative to one another. In fact the 'interleaving' model, the most common model for asynchrony in the software community, can be viewed as a special case of this synchronous product." A technical discussion can be found in [83], but the transformation is not straightforward in practice, and the matter requires further investigation.

In the sequel it will be useful to extend some properties of languages to the composition of two languages. As examples, we illustrate the extension for I-progressive and $I^\star O$-progressive languages.

Definition 2.1.16. Given a language A over alphabet $I \times U$, a language B over alphabet $U \times O$ is **A-compositionally I-progressive** if the language $L = A_{\uparrow O} \cap B_{\uparrow I}$ over alphabet $X = I \times U \times O$ is I-progressive, i.e., $\forall i \in I \; \exists (u, o) \in U \times O \; [\alpha \in L \Rightarrow \alpha\,(i, u, o) \in L]$.

Definition 2.1.17. Given a language A over alphabet $I \cup U$, a language B over alphabet $U \cup O$ is **A-compositionally $IU^\star O$-progressive** if the language $L = A_{\uparrow O} \cap B_{\uparrow I} \subseteq (IU^\star O)^\star$ over alphabet $X = I \cup U \cup O$ (I, U and O pairwise disjoint) is $IU^\star O$-progressive, i.e., $\forall i \in I \; \exists \beta \in U^\star \; \exists o \in O \; [\alpha \in L \Rightarrow \alpha\, i \beta o \in L]$.

Defns. 2.1.16 and 2.1.17 characterize compositions that do not fall into a deadlock or a livelock.

When clear from the context, instead of *A-compositionally* we will write more simply *compositionally*.

2.2 Solution of Equations Over Languages

2.2.1 Language Equations Under Synchronous Composition

Given the alphabets I, U, O, a language A over alphabet $I \times U$ and a language C over alphabet $I \times O$, consider the language equations

$$A \bullet X \subseteq C, \tag{2.1}$$

or,

$$A \bullet X = C. \tag{2.2}$$

Definition 2.2.1. Given the alphabets I, U, O, a language A over alphabet $I \times U$ and a language C over alphabet $I \times O$, language B over alphabet $U \times O$ is called a **solution** of the equation $A \bullet X \subseteq C$ iff $A \bullet B \subseteq C$.

Given the alphabets I, U, O, a language A over alphabet $I \times U$ and a language C over alphabet $I \times O$, language B over alphabet $U \times O$ is called a **solution** of the equation $A \bullet X = C$ iff $A \bullet B = C$.

A solution is called the **largest solution** if it contains any other solution. $B = \emptyset$ is the trivial solution.

Theorem 2.12. *The largest solution of the equation $A \bullet X \subseteq C$ is the language $S = \overline{A \bullet \overline{C}}$.*

If $A \bullet A \bullet \overline{C} = S$ then $\overline{A \bullet \overline{C}}$ is the largest solution of the equation $A \bullet X = C$.

Proof. Consider a string $\alpha \in (U \times O)^\star$; then α is in the largest solution of $A \bullet X \subseteq C$ iff $A \bullet \{\alpha\} \subseteq C$ and the following chain of equivalences follows:

$$\begin{aligned}
A \bullet \{\alpha\} \subseteq C \;&\Leftrightarrow \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I})_{\downarrow I \times O} \cap \overline{C} = \emptyset \;&\Leftrightarrow \textit{ by Prop. } 2.1(a)\ \overline{C} = (\overline{C}_{\uparrow U})_{\downarrow I \times O} \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I})_{\downarrow I \times O} \cap (\overline{C}_{\uparrow U})_{\downarrow I \times O} = \emptyset \;&\Leftrightarrow \textit{ by Prop. } 2.3(d) \\
&\qquad \textit{since } ((\overline{C}_{\uparrow U})_{\downarrow I \times O})_{\uparrow U} = \overline{C}_{\uparrow U} \\
(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U})_{\downarrow I \times O} = \emptyset \;&\Leftrightarrow \textit{ by Prop. } 2.7(b) \\
A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U} = \emptyset \;&\Leftrightarrow \textit{ by Prop. } 2.7(b)
\end{aligned}$$

$$(A_{\uparrow O} \cap \{\alpha\}_{\uparrow I} \cap \overline{C}_{\uparrow U})_{\downarrow U\times O} = \emptyset \Leftrightarrow \textit{by Prop. } 2.3(d)$$

$$\textit{since } \{\alpha\}_{\uparrow I} = ((\{\alpha\}_{\uparrow I})_{\downarrow U\times O})_{\uparrow I}$$

$$(\{\alpha\}_{\uparrow I})_{\downarrow U\times O} \cap (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U\times O} = \emptyset \Leftrightarrow \textit{by Prop. } 2.1(a)\ (\{\alpha\}_{\uparrow I})_{\downarrow U\times O} = \{\alpha\}$$

$$\{\alpha\} \cap (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U\times O} = \emptyset \Leftrightarrow$$

$$\alpha \notin (A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U\times O} \Leftrightarrow$$

$$\alpha \in \overline{(A_{\uparrow O} \cap \overline{C}_{\uparrow U})_{\downarrow U\times O}} \Leftrightarrow$$

$$\alpha \in \overline{A \bullet \overline{C}}$$

Therefore the largest solution of the language equation $A \bullet X \subseteq C$ is given by the language

$$S = \overline{A \bullet \overline{C}}. \tag{2.3}$$

□

Corollary 2.13. *A language B over alphabet $U \times O$ is a solution of $A \bullet X \subseteq C$ iff $B \subseteq \overline{A \bullet \overline{C}}$.*

Equations over languages can be extended to topologies with more than two components, as the one in Fig. 2.3, whose largest solution is given by:

$$S = \overline{((A_{\uparrow Z\times O} \cap C_{\uparrow I\times U\times V}) \cap \overline{S}_{\uparrow V\times Z})_{\downarrow U\times V\times Z}}$$

(see [150]).

Let S be the largest solution of the equation $A \bullet X \subseteq C$. It is of interest to investigate subsets of S that satisfy some further properties, e.g., being prefix-closed, progressive, etc.

If S is prefix-closed then S is the largest prefix-closed solution of the equation. However, not every non-empty subset of S inherits the feature of being prefix-closed. If S is not prefix-closed, then denote by S^{Pref} the set obtained from S by deleting each string that has a prefix not in S.

Proposition 2.14. *If $S^{Pref} \neq \emptyset$, then S^{Pref} is the largest prefix-closed solution of the equation $A \bullet X \subseteq C$.*

If $S^{Pref} = \emptyset$, then the equation $A \bullet X \subseteq C$ has no prefix-closed solution.

If the language S does not include the empty string, then $A \bullet X \subseteq C$ has no prefix-closed solution.

If S is U-progressive (S is a language over alphabet $U \times O$), then S is the largest U-progressive solution of the equation. However, not each non-empty subset of S

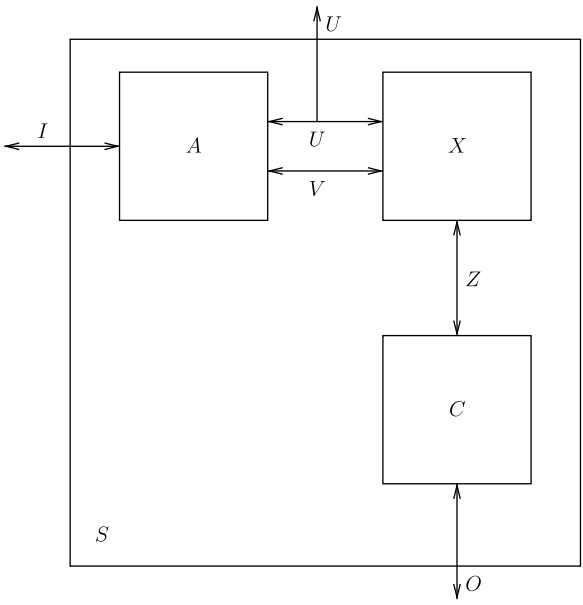

Fig. 2.3 Language equation with respect to a composition topology with three components

inherits the feature of being U-progressive. If S is not U-progressive, then denote by $Prog(S)$ the set obtained from S by deleting each string α such that, for some $u \in U$, there is no $o \in O$ for which $\alpha(u, o) \in S$.

Proposition 2.15. *If $Prog(S) \neq \emptyset$, then the language $Prog(S)$ is the largest U-progressive solution of the equation $A \bullet X \subseteq C$.*

If $Prog(S) = \emptyset$, then the equation $A \bullet X \subseteq C$ has no U-progressive solution.

2.2.2 Language Equations Under Parallel Composition

Given the pairwise disjoint alphabets I, U, O, a language A over alphabet $I \cup U$, and a language C over alphabet $I \cup O$, consider the language equation

$$A \diamond X \subseteq C, \tag{2.4}$$

or,

$$A \diamond X = C. \tag{2.5}$$

Definition 2.2.2. Given the pairwise disjoint alphabets I, U, O, a language A over alphabet $I \cup U$ and a language C over alphabet $I \cup O$, language B over alphabet $U \cup O$ is called a **solution** of the equation $A \diamond X \subseteq C$ iff $A \diamond B \subseteq C$.

Given the pairwise disjoint alphabets I, U, O, a language A over alphabet $I \cup U$ and a language C over alphabet $I \cup O$, language B over alphabet $U \cup O$ is called a **solution** of the equation $A \diamond X = C$ iff $A \diamond B = C$.

The **largest solution** is a solution that contains any other solution. $B = \emptyset$ is the trivial solution.

Theorem 2.16. *The largest solution of the equation $A \diamond X \subseteq C$ is the language $S = \overline{A \diamond \overline{C}}$.*

If $A \diamond \overline{A \diamond \overline{C}} = S$ then $\overline{A \diamond \overline{C}}$ is the largest solution of the equation $A \diamond X = C$.

Proof. Consider a string $\alpha \in (U \cup O)^\star$, then α is in the largest solution of $A \diamond X \subseteq C$ iff $A \diamond \{\alpha\} \subseteq C$ and the following chain of equivalences follows:

$$
\begin{aligned}
A \diamond \{\alpha\} \subseteq C &\Leftrightarrow \\
(A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I})_{\Downarrow I \cup O} \cap \overline{C} = \emptyset &\Leftrightarrow \textit{by Prop. 2.1(c) } \overline{C} = (\overline{C}_{\Uparrow U})_{\Downarrow I \cup O} \\
(A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I})_{\Downarrow I \cup O} \cap (\overline{C}_{\Uparrow U})_{\Downarrow I \cup O} = \emptyset &\Leftrightarrow \textit{by Prop. 2.5(d)} \\
&\quad \textit{since } ((\overline{C}_{\Uparrow U})_{\Downarrow I \cup O})_{\Uparrow U} = \overline{C}_{\Uparrow U} \\
(A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap \overline{C}_{\Uparrow U})_{\Downarrow I \cup O} = \emptyset &\Leftrightarrow \textit{by Prop. 2.7(d)} \\
A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap \overline{C}_{\Uparrow U} = \emptyset &\Leftrightarrow \textit{by Prop. 2.7(d)} \\
(A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap \overline{C}_{\Uparrow U})_{\Downarrow U \cup O} = \emptyset &\Leftrightarrow \textit{by Prop. 2.5(d)} \\
&\quad \textit{since } \{\alpha\}_{\Uparrow I} = ((\{\alpha\}_{\Uparrow I})_{\Downarrow U \cup O})_{\Uparrow I} \\
(\{\alpha\}_{\Uparrow I})_{\Downarrow U \cup O} \cap (A_{\Uparrow O} \cap \overline{C}_{\Uparrow U})_{\Downarrow U \cup O} = \emptyset &\Leftrightarrow \textit{by Prop. 2.1(c) } (\{\alpha\}_{\Uparrow I})_{\Downarrow U \cup O} = \{\alpha\} \\
\{\alpha\} \cap (A_{\Uparrow O} \cap \overline{C}_{\Uparrow U})_{\Downarrow U \cup O} = \emptyset &\Leftrightarrow \\
\alpha \notin (A_{\Uparrow O} \cap \overline{C}_{\Uparrow U})_{\Downarrow U \cup O} &\Leftrightarrow \\
\alpha \in \overline{(A_{\Uparrow O} \cap \overline{C}_{\Uparrow U})_{\Downarrow U \cup O}} &\Leftrightarrow \\
\alpha \in \overline{A \diamond \overline{C}}
\end{aligned}
$$

Therefore the largest solution of the language equation $A \diamond X \subseteq C$ is given by the language

$$S = \overline{A \diamond \overline{C}}. \tag{2.6}$$

□

Corollary 2.17. *A language B over alphabet $U \cup O$ is a solution of $A \diamond X \subseteq C$ iff $B \subseteq \overline{A \diamond \overline{C}}$.*

Proposition 2.18. *If S is U-convergent, then S is the largest U-convergent solution of the equation, and a language $B \neq \emptyset$ is a U-convergent solution iff $B \subseteq S$.*

When S is not U-convergent the largest U-convergent solution does not exist, since any finite subset of S is a U-convergent solution and therefore no string can be deleted from S without missing a solution. An analogous proposition and remark hold for S-compositionally U-convergent solutions.

2.2.3 Language Equations Under Bounded Parallel Composition

Theorem 2.19. *The largest solution of the equation $A \diamond_l X \subseteq C$ is the language*

$$S = \overline{(A_{\Uparrow O} \cap \overline{C}_{\Uparrow (U,I)})_{\Downarrow U \cup O}}.$$

Proof.

$$\begin{aligned} A \diamond_l \{\alpha\} \subseteq C \Leftrightarrow \\ (A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap (I \cup O)^{\star}_{\Uparrow (U,I)})_{\Downarrow I \cup O} \cap \overline{C} = \emptyset \Leftrightarrow \\ A_{\Uparrow O} \cap \{\alpha\}_{\Uparrow I} \cap \overline{C}_{\Uparrow (U,I)} = \emptyset \Leftrightarrow \\ \alpha \notin (A_{\Uparrow O} \cap \overline{C}_{\Uparrow (U,I)})_{\Downarrow U \cup O} \Leftrightarrow \\ \alpha \in \overline{(A_{\Uparrow O} \cap \overline{C}_{\Uparrow (U,I)})_{\Downarrow U \cup O}} \end{aligned}$$

□

2.3 Solution of Equations Over Regular Languages and Finite Automata

Language equations can be solved effectively when they are defined over languages that can be computed with finite procedures. Usually such languages are presented through their corresponding mathematical machines, e.g., finite automata for regular languages. In the following sections, equations over various classes of automata are studied, like FAs and FSMs, specializing the theory of equations to their associated languages. A key issue to investigate is the closure of the solution set with respect to a certain type of language, e.g., when dealing with FSM language equations we require that the solutions are FSM languages. This cannot be taken for granted, because the general solution of abstract language equations is expressed through the operators of complementation and composition, which do not necessarily preserve certain classes of languages.

2.3.1 An Algorithm to Solve Equations Over Regular Languages and Automata

Two well-known results [63] are that non-deterministic finite automata are equivalent (with respect to language equality) to deterministic ones and that regular expressions are equivalent to finite automata. By applying the algorithm of subset construction one converts a NDFA into an equivalent DFA (it is complete by construction). Given an NDFA $F = \langle S, \Sigma, \Delta, r, Q \rangle$, the process of subset construction

builds the DFA $F' = \langle 2^S, \Sigma, \delta, r, Q' \rangle$, where (1) the states $\tilde{s} \in 2^S$ are the subsets of S, (2) the transition relation is $\delta(i, \tilde{s}) = \cup_{s \in \tilde{s}} \{s' \mid (i, s, s') \in \Delta\}$ and (3) a state is final, i.e., $\tilde{s} \in Q' \subseteq 2^S$, iff $\tilde{s} \cap Q \neq \emptyset$. Since many of the states in 2^S are unreachable from the initial state, they can be deleted and so the determinized automaton usually has fewer states than the power set.[4] To make a NDFA complete it is not necessary to apply the full-blown subset construction, but it suffices to add a new non-accepting state s_d whose incoming transitions are (i, s, s_d) for all i, s for which there was no transition in the original automaton. By a closure construction [63], an NDFA with ϵ-moves can be converted to an NDFA without ϵ-moves; however, a subset construction must be applied at the end to determinize it.

The equivalence of regular expressions and finite automata is shown by matching each operation on regular expressions with a constructive procedure that yields the finite automaton of the result, given the finite automata of the operands. For the most common operations (union, concatenation, complementation, intersection) see [63]. Here we sketch the constructions for the less known operations of projection, lifting, restriction and expansion:

projection ($\downarrow$) Given FA F that accepts language L over $X \times V$, FA F' that accepts language $L_{\downarrow V}$ over X is obtained from F by the following procedure:

replace each edge $((x, v), s, s')$ by the edge (x, s, s').

lifting ($\uparrow$) Given FA F that accepts language L over X, FA F' that accepts language $L_{\uparrow V}$ over $X \times V$ is obtained from F by the following procedure:

replace each edge (x, s, s') by the edges $((x, v), s, s'),\ \forall v \in V$.

restriction ($\Downarrow$) Given FA F that accepts language L over $X \cup V$, FA F' that accepts language $L_{\Downarrow V}$ over V is obtained from F by the following procedure:

$\forall x \in X \setminus V$, change every edge (x, s, s') into the edge (ϵ, s, s'), i.e., replace the symbols $x \in X \setminus V$ by ϵ.[5]

expansion ($\Uparrow$) Given FA F that accepts language L over X, FA F' that accepts language $L_{\Uparrow V}$ over $X \cup V$ ($X \cap V = \emptyset$) is obtained from F by the following procedure:

for each state s, $\forall v \in V$ add the edge (self-loop) (v, s, s).

l-expansion ($\Uparrow_l$) Given FA F that accepts language L over X, FA F' that accepts language $L_{\Uparrow(V,l)}$, l integer, over $X \cup V$ ($X \cap V = \emptyset$) is obtained from F by the following procedure:

1. The set of states S' of F' is given by

$$S' = S \cup \{(s, j) \mid s \in S, 1 \leq j \leq l\}.$$

[4] It is not uncommon in practice to find that $|\tilde{S}| \leq |S|$.

[5] Apply the closure construction to obtain an equivalent deterministic finite automaton without ϵ-moves.

2. The next state relation Δ' of F' is given by

$$\begin{aligned}\Delta' = \Delta \cup \{(v, s, (s, 1)) \mid v \in V, s \in S\} \\ \cup \{(v, (s, j), (s, j+1)) \mid v \in V, s \in S, 1 \leq j < l\} \\ \cup \{(x, (s, j), s') \mid (x, s, s') \in \Delta, 1 \leq j \leq l\}.\end{aligned}$$

3. $r' = r$ and $Q' = Q$.

The procedures for projection, lifting and restriction guarantee the substitution property $f(\epsilon) = \epsilon$.

Given that all the operators used to express the solution of regular language equations have constructive counterparts on automata, we conclude that there is an effective (constructive) way to solve equations over regular languages.

As an example, given a regular language equation $A \bullet X \subseteq C$, where A is a regular language over alphabet $I \times U$, C is over $I \times O$, and the unknown regular language X is over $U \times O$, an algorithm to build X follows.

Procedure 2.3.1. *Input: Regular language equation $A \bullet X \subseteq C$; Output: Largest regular language solution X*

1. Consider the finite automata $F(A)$ and $F(C)$ corresponding, respectively, to regular languages A and C.
2. Determinize $F(C)$ by subset construction, if it is a NDFA. The automaton $F(\overline{C})$ of $\overline{C}$ is obtained by interchanging the sets of accepting and non-accepting states of the determinization of $F(C)$.
3. Lift the language A to O by replacing each label (i, u) of a transition of $F(A)$ with all triples (i, u, o), $o \in O$. Lift the language $\overline{C}$ to U by replacing each label (i, o) of a transition of $F(\overline{C})$ with all triples (i, u, o), $u \in U$.
4. Build the automaton $F(A \cap \overline{C})$ of the intersection $A \cap \overline{C}$. The states are pairs of states of the lifted automata $F(A)$ and $F(\overline{C})$, the initial state is the pair of initial states, and a state of the intersection is accepting if both states of the pair are accepting. There is a transition from the state (s_1, s_2) to the state (s_1', s_2') labeled with action (i, u, o) in $F(A \cap \overline{C})$, if there are corresponding transitions labeled with (i, u, o) from state s_1 to state s_1' in $F(A)$ and from s_2 to s_2' in $F(\overline{C})$.
5. Project $F(A \cap \overline{C})$ to $U \times O$ to obtain $F(A \bullet \overline{C})$ by deleting i from the labels (i, u, o). Projection in general makes the finite automaton non-deterministic.
6. Determinize $F(A \bullet \overline{C})$ by subset construction, if it is a NDFA. The automaton $F(\overline{A \bullet \overline{C}})$ corresponding to the regular language solution $X = \overline{A \bullet \overline{C}}$ is obtained by interchanging the sets of accepting and non-accepting states of the determinization of $F(A \bullet \overline{C})$.

Notice that Procedure 2.3.1 holds for any regular language, not only for prefix-closed languages as in restricted versions reported in the literature.

A companion procedure to solve the regular language equation under parallel composition $A \diamond X \subseteq C$ is obtained from Procedure 2.3.1, after replacing the

Cartesian product with union, projection with restriction and lifting with expansion. The largest solution of parallel equations for prefix-closed regular languages had been known already in the process-algebra literature [90, 94, 118].

2.3.2 An Application to Converter Synthesis: The Protocol Mismatch Problem

We apply the algorithm in Sect. 2.3.1 to an equation over finite automata to solve a problem of converter synthesis, i.e., the design of an automaton to translate between two different protocols.

A communication system has a sending part and a receiving part that exchange data through a specific protocol. A mismatch occurs when two systems with different protocols try to communicate. The mismatch problem is solved by designing a converter that translates between the receiver and the sender, while respecting the overall service specification of the behavior of the composed communication system relative to the environment. We formulate the problem as a parallel language equation: given the service specification C of a communication system, a component sender and a component receiver, find a converter X whose composition with the sender and receiver A meets the system specification after hiding the internal signals: $A \diamond X \subseteq C$.

As an example we consider the problem of designing a protocol converter to interface: an *alternating-bit* (AB) sender and a *non-sequenced* (NS) receiver. This problem is adapted from [78] and [56]. A communication system based on an alternating bit protocol is composed of two processes, a sender and a receiver, which communicate over a half duplex channel that can transfer data in either directions, but not simultaneously. Each process uses a control bit called the alternating bit, whose value is updated by each message sent over the channel in either direction. The acknowledgement is also based on the alternating bit: each message received by either process in the system corresponds to an acknowledgement message that depends on the bit value. If the acknowledgement received by a process does not correspond to the message sent originally, the message is resent until the correct acknowledgement is received. On the other hand, a communication system is non-sequenced when no distinction is made among the consecutive messages received or their corresponding acknowledgements. This means that neither messages nor their acknowledgements are distinguished by any flags such as with the alternating bit.

Figure 2.4 shows the block diagram of the composed system. Each component is represented by a rectangle with incoming and outgoing labeled arrows to indicate the inputs and outputs, respectively. The sender consists of an AB protocol sender (*PS*) and of an AB protocol channel (*PC*). Meanwhile, the receiving part includes an NS protocol receiver (*PR*). The converter X must interface the two mismatched protocols and guarantee that its composition with PS, PC and PR refines the service specification (*SS*) of the composed system. The events *Acc* (*Accept*) and *Del*

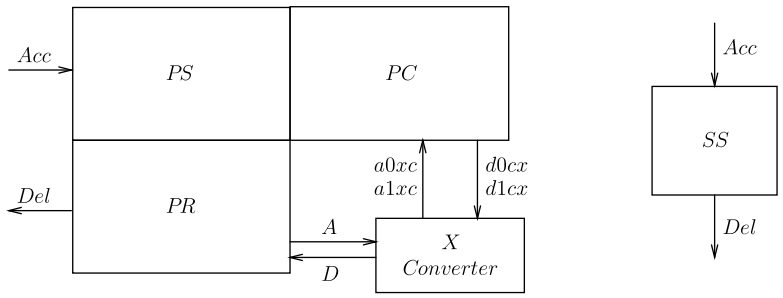

Fig. 2.4 Communication system described in Sect. 2.3.2

(*Deliver*) represent the interfaces of the communication system with the environment (the users). The converter X translates the messages delivered by the sender *PS* (using the alternating bit protocol) into a format that the receiver *PR* understands (using the non-sequenced protocol). For example, acknowledgement messages A delivered to the converter by the receiver are transformed into acknowledgements of the alternating bit protocol (*a0xc* to acknowledge a 0 bit and *a1xc* to acknowledge a 1 bit) and passed to the sender by the channel (*a0cs* to acknowledge a 0 bit and *a1cs* to acknowledge a 1 bit); data messages are passed from the sender to the channel (*d0sc* for a message controlled by a 0 bit and *d1sc* for a message controlled by a 1 bit) and then from the channel to the converter (*d0cx* for a message controlled by a 0 bit and *d1cx* for a message controlled by a 1 bit) to be transformed by the converter into a data message D for the receiver.

We model the components as I/O automata [89], which recognize prefix-closed regular languages, and we solve their language equations. Figure 2.5 shows the automata of the components of the communication system. Missing transitions go to a trap (non-accepting) state, that loops to itself under any event.

Figure 2.6 shows the largest prefix-closed solution $S = \overline{PS \diamond PC \diamond PR \diamond \overline{SS}}$ of the converter problem. All missing transitions go to an *accepting* trap state *dc* (not shown), that would loop to itself under any event; e.g., the initial state would have a transition to state *dc* under events $A, a0xc, a1xc, d1cx$. These transitions are not indicated in the state transition graph of the automaton of the solution language to avoid cluttering the picture. State *dc* can be termed the *don't care* state, because it is introduced during the determinization step to complete the automaton $PS \diamond PC \diamond PR \diamond \overline{SS}$, before the final complementation. It is reached by transitions that cannot occur due to impossible combinations of events in the composition of $PS \diamond PC \diamond PR$ and S, and so it does not matter how S behaves, once it is in state *dc* (thus the qualification *don't care* state). This makes the largest solution S non-deterministic. The solution presented in [78] and [56] does not feature this trap accepting state and so it is not complete (in [78] and [56] all missing transitions of the solution are supposed to end up in a *non-accepting* trap state, a *fail* state); without the above *dc* state, one gets only a subset of all solutions (in particular the complete solutions are missed) and this might lead to an inferior implementation.

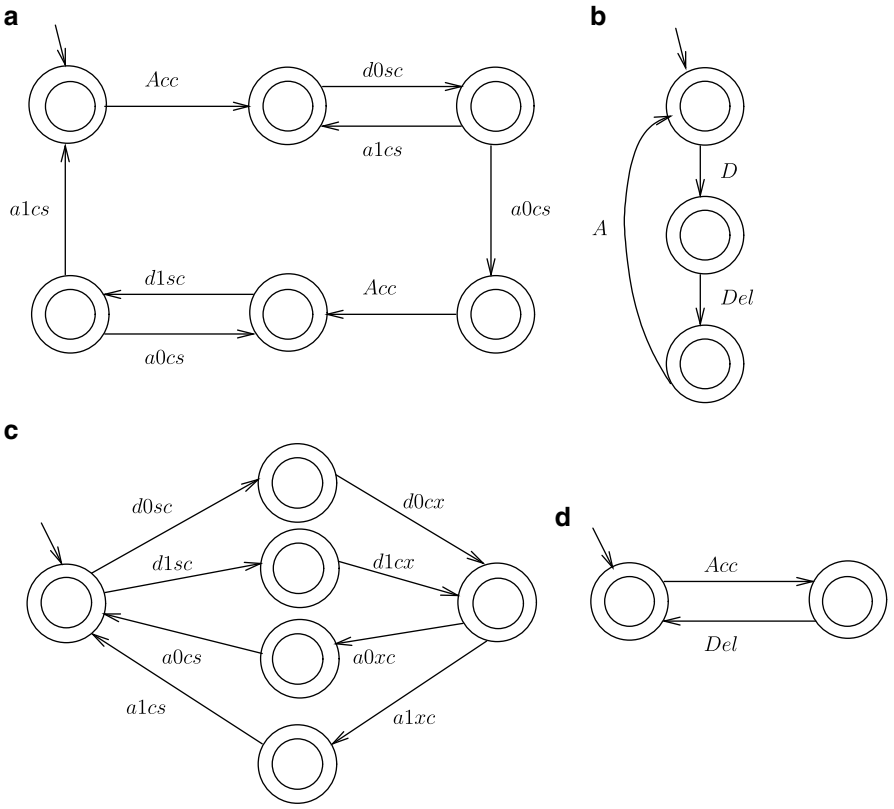

Fig. 2.5 Automata of communication system described in Sect. 2.3.2. (**a**) Automaton of PS; (**b**) automaton of PR; (**c**) automaton of PC; (**d**) automaton of SS

Figure 2.7 shows yet another view of the largest prefix-closed solution $S = \overline{PS \diamond PC \diamond PR \diamond \overline{SS}}$ of the converter problem, with the *dc* state included and the *fail* state excluded.

Figure 2.8 shows the largest prefix-closed 2-bounded solution of the converter problem, as an example of solution with bounded internal communication.

Figure 2.9 shows the composition $PS \diamond PC \diamond PR \cap S_{\Uparrow\{Acc,Del\}}$ of the communication system $PS \diamond PC \diamond PR$ and of the largest converter S. The largest prefix-closed solution S is compositionally $(I^\star O)$-progressive and compositionally prefix $(U \cup V)$-deadlock-free, but not compositionally prefix $(U \cup V)$-convergent. This means that, even if we extract from the largest solution S a complete solution $\hat{S}$ (i.e., $\hat{S}$ is defined for every internal input $A, d0cx, d1cx$), the composition of $\hat{S}$ with the context ($PS \diamond PC \diamond PR$ may deadlock, i.e., the automata can exchange internal actions without producing any output for the environment; this happens, for instance, if $\hat{S}$ after the internal input *d0cx* always selects the internal output *a1cx* (and never selects D). In general the composition of any complete solution that never produces the internal output D with the context has livelocks (loop

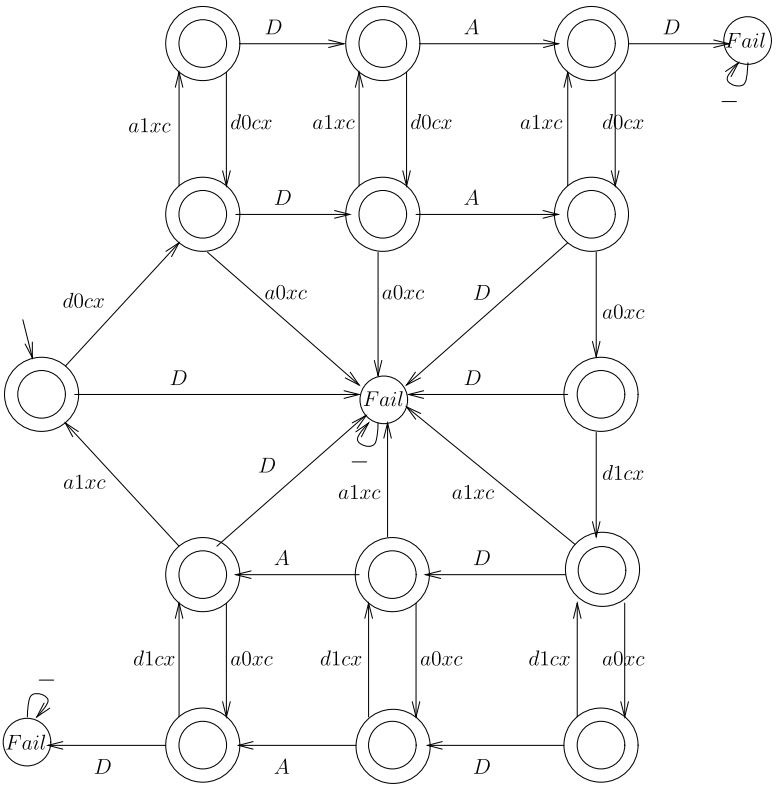

Fig. 2.6 Largest prefix-closed solution $S = \overline{PS \diamond PC \diamond PR \diamond \overline{SS}}$ of the converter problem of Sect. 2.3.2

a1xc-d0cx), which are deadlocks with respect to the external input *Acc*, i.e., the composition is not complete. So the largest solution S is not compositionally prefix $(U \cup V)$-convergent due to the existence of internal cycles, but since the latter can be exited by selecting the internal output D S is compositionally prefix $(U \cup V)$-deadlock-free (and therefore compositionally $(I^\star O)$-progressive).

The protocol conversion problem was addressed in [78], as an instance of supervisory control of discrete event systems, where the converter language is restricted to be a sublanguage of the context A, and in [56] with the formalism of input-output automata. In [78] the problem is modeled by the equation $A \diamond X = C$ over regular languages with the rectification topology (see Fig. 1d). The solution is given as a sublanguage of A of the form $A \diamond C \setminus A \diamond \overline{C}$ (not the largest solution). An algorithm to obtain the largest compositionally progressive solution is provided that first splits the states of the automaton of the unrestricted solution (refining procedure, exponential step due to the restriction operator), and then deletes the states that violate the desired requirement of progressive composition (linear step). This algorithm does not generalize as it is to topologies where the unknown component depends also on signals that do not appear in the component A.

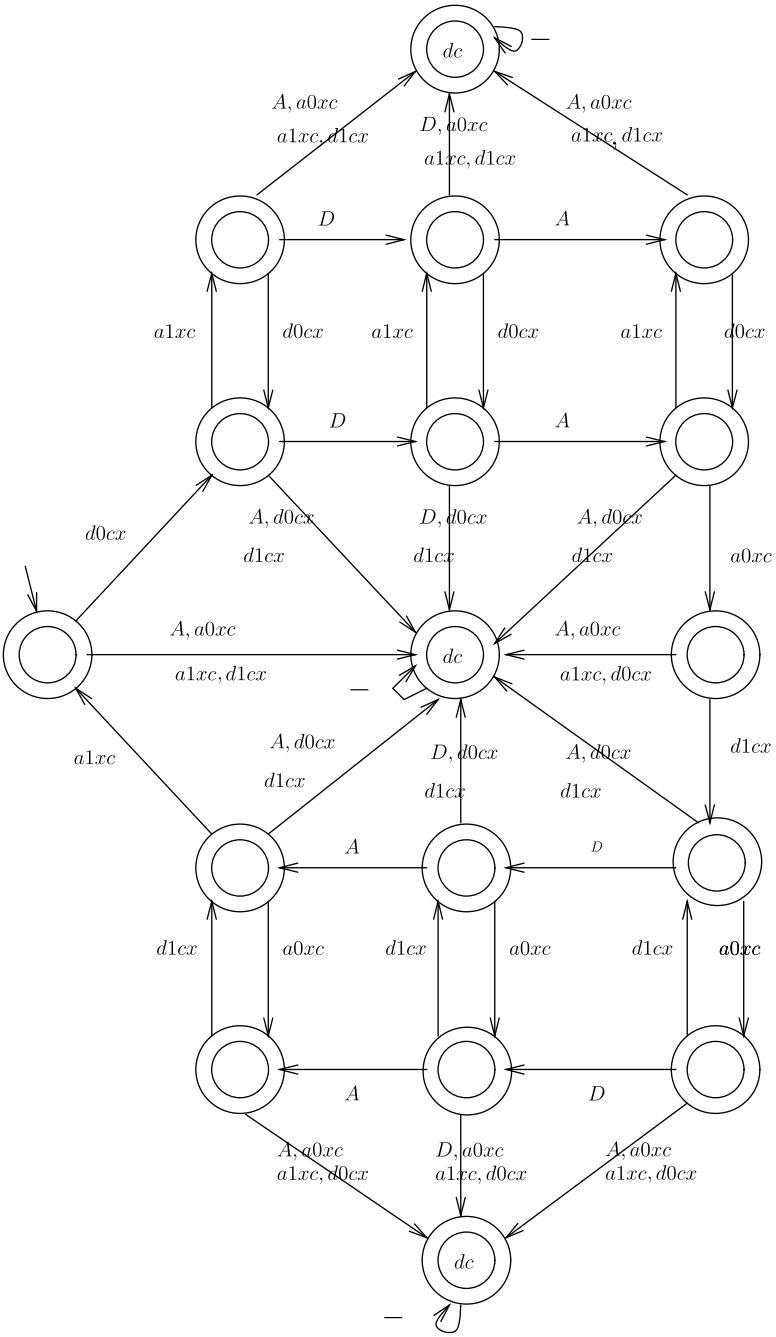

Fig. 2.7 Largest prefix-closed solution S of the converter problem of Sect. 2.3.2. It shows explicitly the transitions to the *dc* state

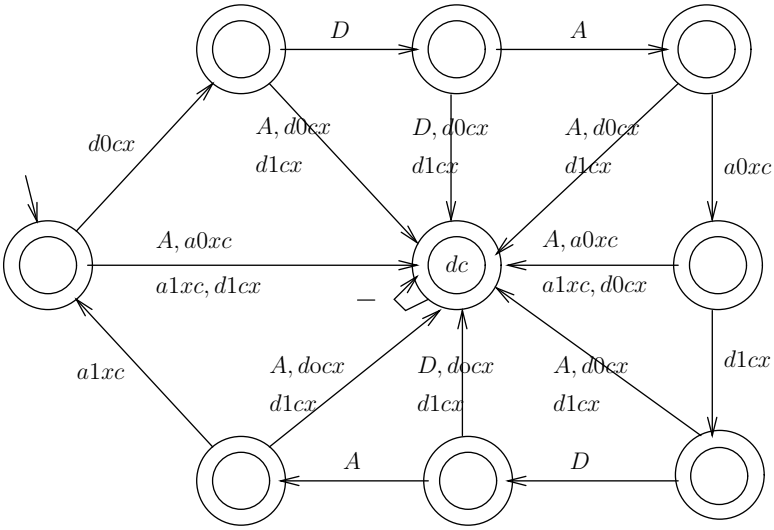

Fig. 2.8 Largest prefix-closed 2-bounded solution of the converter problem of Sect. 2.3.2

Problems

2.1. Consider the traffic controller example described in [12]. For ease of reference we show in Figs. 2.10 and 2.11 respectively the context automaton A and the specification automaton C derived from it.

Notice that the inputs of A are two binary variables v_1, v_2, and the multi-valued variable *colours* that can assume the values *green, red, yellow*. The inputs of C are two multi-valued variables i_1, i_2 that can assume each one of the three values $1, 2, 3$, and again the multi-valued variable *colours*.

Find the largest solution of the equation $A \bullet X \subseteq C$, where the input variables of X are i_1, i_2 and v_1, v_2. This is a series topology where X feeds A, and the composition of X and A yields C.

Repeat the problem assuming that X has one more input: the variable *colours* of automaton A, as in the controller's topology.

Hint.

The largest solution automaton of the series-topology equation is shown in Fig. 2.12.

If the reader will find too cumbersome carrying on by the hand the computations, it will be one more reason to appreciate the automatica tool BALM presented in the second part of the book.

2.2. Consider the alphabets $I = \{i0, i1\}$, $V = \{v0, v1\}$, $O = \{o0, o1\}$, and the automata $A = \langle S_A, \Sigma_A, \Delta_A, r_A, Q_A \rangle$ and $C = \langle S_C, \Sigma_C, \Delta_C, r_C, Q_C \rangle$. Let the automata be defined as follows: $S_A = Q_A = \{sa, sb\}$, $\Sigma_A = V \times O$, $\Delta_A = \{(v0\_o1,\ sa,\ sb),\ (v1\_o0,\ sa,\ sb),\ (v0\_o1,\ sb,\ sb),\ (v1\_o1,\ sb,\ sa)$, $r_A = sa$,

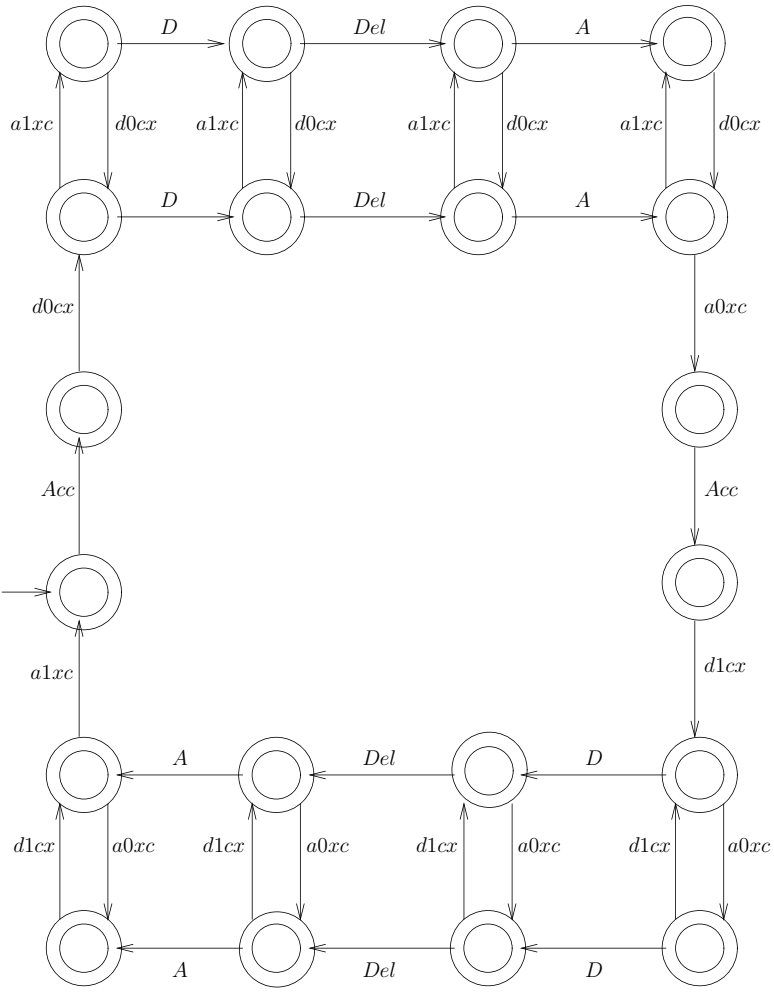

Fig. 2.9 Composition $PS \diamond PC \diamond PR \cap S_{\Uparrow\{Acc,Del\}}$ of communication system $PS \diamond PC \diamond PR$ and largest converter S of the converter problem of Sect. 2.3.2

and $S_C = Q_C = \{s1,\ s2,\ s3,\ s4\}$, $\Sigma_C = I \times O$, $\Delta_C = \{(i1\_o0,\ s1,\ s2), (i0\_o0,\ s1,\ s3), (i0\_o1,\ s2,\ s1), (i1\_01,\ s2,\ s3)\}, (i0\_o1,\ s3,\ s1), (i1\_o1,\ s3,\ s4), (i1\_o1,\ s4,\ s3)\}, (i0\_o0,\ s4,\ s3)$, $r_C = s1$.

Compute the largest solution of the equation $A \bullet X \subseteq C$, where X has inputs $I \times V$.

Verify whether in this example the largest solution is complete with respect to I.

2.3. Consider the alphabets $I = \{i1, i2\}$, $V = \{v1, v2\}$, $O = \{o1, o2\}$, and the automata $A = \langle S_A, \Sigma_A, \Delta_A, r_A, Q_A\rangle$ and $C = \langle S_C, \Sigma_C, \Delta_C, r_C, Q_C\rangle$. Let the automata be defined as follows: $S_A = Q_A = \{s1, s2\}$, $\Sigma_A = V \times O$, $\Delta_A =$

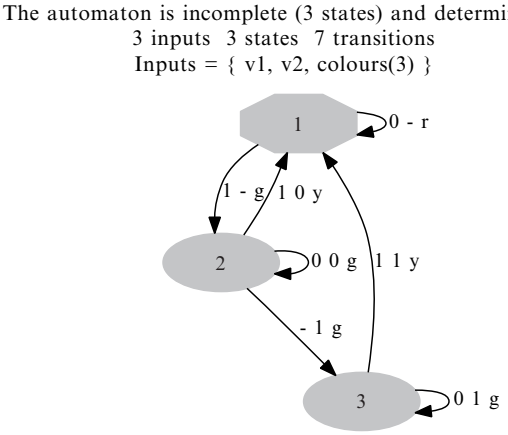

Fig. 2.10 Graphical output of BALM showing the automaton `traffic-ma.aut` describing the fixed automaton component of the traffic light controller problem. The notation of BALM will be presented formally later on, but this graph can be read as any usual FSM graphical representation

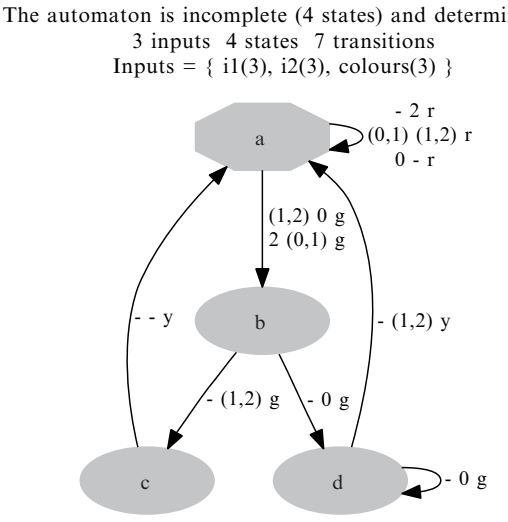

Fig. 2.11 Graphical output of BALM showing the automaton `traffic-mc.aut` describing the specification automaton of the traffic light controller problem

$\{(v1\_o1,\ s1,\ s2),\ (v2\_o2,\ s1,\ s1),\ (v1\_o2,\ s2,\ s1),\ (v2\_o1,\ s2,\ s2),\ r_A = s1$, and $S_C = Q_C = \{sa,\ sb,\ sc\}$, $\Sigma_C = I \times O$, $\Delta_C = \{(i1\_o1,\ sa,\ sb),\ (i2\_o2,\ sa,\ sc),\ (i2\_o2,\ sb,\ sa),\ (i1\_02,\ sb,\ sc)\}$, $(i1\_o1,\ sc,\ sa),\ (i2\_o1,\ sc,\ sb),\ r_C = sa$.

Compute the largest solution of the equation $A \bullet X \subseteq C$, where X has inputs $I \times V \times O$.

Verify whether in this example of controller's topology the composition of the largest solution with M_A is progressive.

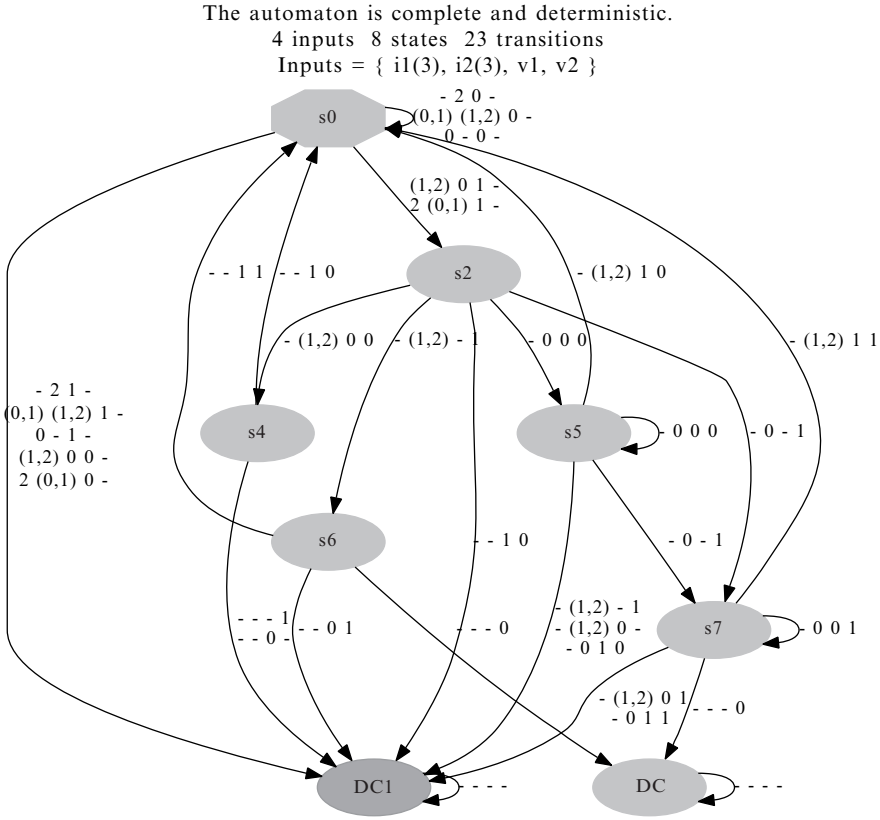

Fig. 2.12 Graphical output of BALM showing the automaton `traffic-xfsm_min.aut` describing the largest solution automaton of the traffic light controller problem

2.4. Somebody claims that there is a more efficient way to solve the language equation $F \bullet X \subseteq S$. Instead of computing the solution $\overline{(F \cap \overline{S}_{\uparrow U \times V})_{\downarrow U \times V}}$, compute as a solution $\overline{(F \cap \overline{S}_{\uparrow U \times V})}_{\downarrow U \times V}$. The advantage would be to avoid a determinization because the second complementation is performed before the projection (the latter introduces non-determinism and so the need for determinization). Is it correct? Are the two expressions equivalent ? Is one contained in the other?

http://www.springer.com/978-0-387-34532-1

The Unknown Component Problem
Theory and Applications

Villa, T.; Yevtushenko, N.; Brayton, R.K.; Mishchenko, A.;
Petrenko, A.; Sangiovanni-Vincentelli, A.

2012, XVI, 312 p., Hardcover
ISBN: 978-0-387-34532-1