The Validity Problem for Extended Regular Expressions

by

Daniele Micciancio

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author .....
Department of Electrical Engineering and Computer Science
May, 1996

Certified by .....
Albert R. Meyer
Hitachi America Professor of Engineering
Thesis Supervisor

Accepted by .....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

The Validity Problem for Extended Regular Expressions

by

Daniele Micciancio

Submitted to the Department of Electrical Engineering and Computer Science
on May, 1996, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

The study of regular expressions, possibly extended with operators other than the usual union, concatenation and star, has traditionally focused on closed expressions, i.e. expressions not containing free language variables. In this thesis we consider open expressions (i.e. expressions that may contain language variables) and the problem of deciding valid equations between them.

In particular, we study constant-free shuffle-intersection regular expressions, that is, expressions containing only language variables and using the shuffle and intersection operators in addition to the usual union, concatenation and star.

We prove the decidability of valid equations and in-equations between such expressions. This result is achieved introducing a new semantics for constant-free shuffle-intersection regular expressions. We prove that our semantics is correct and fully abstract. We then reduce the problem of deciding if two expressions have the same semantics to the equivalence problem of finite state automata.

Thesis Supervisor: Albert R. Meyer
Title: Hitachi America Professor of Engineering

Acknowledgments

I would like to thank all the people who will ever read this thesis. Among these (I hope), I would especially like to thank Albert Meyer, my thesis supervisor. Working with him has been stimulating, rewarding, and, most of all, fun.

I want also to express my deepest gratitude to Sanjoy Mitter for bringing me at MIT and helping me during my first years at MIT. I want also to thank my former supervisor, Giorgio Levi, for his guidance, encouragement and advice, even once I had left the university of Pisa.

In addition, I want to thank all the people who showed some interest in my work, including Jens Palsberg, Antonio Fernandez, Trevor Jim, Jakov Kucan, Sabrina Mantaci, and many others. Special thanks to Sandeep Shukla and Richard Beigel, who asked me to send them a copy of my thesis when I had not even started writing it. They will never imagine how much their letter helped me in getting my work done.

Thanks also to Alexander Russell and Ravi Sundaram who asked me to have their names in the acknowledgments. They would have deserved my thanks anyway for the many advise they gave me since I first came here. Thanks to Roberto De Prisco, not to want his name in the acknowledgments. No thanks (to avoid the paradox) to Alessandro D'Andrea, who asked to be put among those who didn't want to be thanked.

Finally, my warmest thanks to my parents, Mariella and Stefano Micciancio, for their constant support and encouragement.

The research described in this thesis was funded by the Army Research Office under grant DAAL03-92-G-0115.

Contents

Chapter 1

Introduction

We investigate the problem of deciding valid equations and in-equations between open regular expressions extended with the shuffle and intersection operations, motivated by the use of formal languages as a model of computation in process algebra [4, 12, 11].

Regular expressions have been widely studied and considerable progress has been made in the classification of the computational complexity of decision problems involving closed regular expressions, possibly including operators on languages other than the usual union, concatenation and star [10, 6, 13, 2, 7]. All these results concern closed expressions, that is expressions not containing free language variables.

If we allow language variables in expressions, little is known about decidability problems involving extended regular expressions: for most such problems decidability is still an open question.

The main result on open expressions (i.e. expressions containing language variables) is a folk theorem on valid regular identities stating that to decide if a regular identity with free language variables is valid, one can treat the variables as symbols and decide if the closed identity is true (see chapter 3 for more details).

This folk theorem holds only for regular expressions: as soon as other operations are allowed (e.g. shuffle or intersection) the folk theorem is not longer true. The regular operations of union, concatenation and star express the programming constructs of non-deterministic choice, sequential composition and iteration, thus taking into account only the sequential aspects of computation. Algebraic approaches to modeling

concurrent computation, mainly Milner's CCS and Hoare's CSP, also include at least a parallel composition operator which allows processes to be executed concurrently, possibly interacting through communication or synchronization actions.

In this thesis we consider open regular expressions extended with shuffle and intersection. The shuffle operator [3, 7, 8, 1], also called interleaving, has been studied as a model of concurrency and represents the simplest case of parallel composition. Namely, it represents the case when two processes are executed in parallel independently of each other, i.e. without any communication between them taking place.

Also intersection can be interpreted as a special case of parallel composition, this time with interaction. Two processes being composed with the intersection operator run in parallel, but are required to synchronize on all actions they perform.

Recently, valid equations between regular expressions with shuffle have been proved to be decidable [8]. In [9] is raised the open question if valid equations between regular expressions with intersection are decidable. A partial result in this respect has been proved by [Scott 4/1994]: valid equations with concatenation, union and intersection are decidable.

In this thesis we solve the decidability problem for valid equations between constant free open regular expressions with shuffle and intersection, that is expressions containing only language variable symbols and using the usual regular operations of union, concatenation and star extended with shuffle and intersection.

In [9] it is conjectured that the solution of the validity problem for regular expressions with shuffle and intersection would solve the open question about expressions with general communicating parallel composition. In chapter 6 we show that this conjecture is indeed very close to the truth: CSP parallel composition operation can be expressed in terms of shuffle and intersection when also renaming and hiding operators are available.

The rest of this thesis is organized as follows. In chapter 2 we review some notation. In chapter 3 we recall the folk theorem on open regular equations. In chapter 4 we present a fully abstract semantics for constant-free shuffle-intersection regular expressions. In chapter 5 we prove that valid inclusions between constant-

free shuffle-intersection regular expressions are decidable. Chapter 6 concludes with a discussion of some open problems. The automata construction used in chapter 5 is described in the appendix.

Chapter 2

Notation

We use the standard definition of *regular expression* [5] extended with the *shuffle* and *intersection* operations [3, 2]. We consider open expressions, that is expressions that may contain language variables. Namely, **open shuffle-intersection regular expressions** are defined by the grammar

$$E = a \mid x \mid E + E \mid E; E \mid E^* \mid E \| E \mid E \cap E$$

where a ranges over an alphabet Σ of **constant symbols** and x ranges over a set V of **variable symbols**. Unless otherwise stated, the sets Σ and V are always assumed to be denumerable.

The set of all languages over Σ is denoted 2^{Σ^*}. The empty string is denoted ϵ. All operations on regular expressions are interpreted over formal languages in the usual way (see table in figure 2-1). Each constant symbol $a \in \Sigma$ is interpreted as the constant language $\{a\}$. A **closed** expression is an expression that does not contain variable symbols. Closed expressions evaluate to languages in 2^{Σ^*} in the obvious way.

In order to evaluate an open expression to a language, we need to fix a valuation for the language variable symbols. A **variable assignment** σ is a function from the set of variable symbols V to the set of languages over Σ:

$$\sigma : V \to 2^{\Sigma^*}.$$

REG.EXP.	OPERATION	DEFINITION
$E_1 + E_2$	$L_1 \cup L_2$ (*union*)	$\{w \mid w \in L_1 \text{ or } w \in L_2\}$
$E_1; E_2$	$L_1 \cdot L_2$ (*concatenation*)	$\{w_1 w_2 \mid w_1 \in L_1 \text{ and } w_2 \in L_2\}$
$E_1 \| E_2$	$L_1 \| L_2$ (*shuffle*)	$\left\{ w_1^1 w_2^1 \cdots w_1^n w_2^n \;\middle\vert\; \begin{array}{l} w_1^1 \cdots w_1^n \in L_1, \\ w_2^1 \cdots w_2^n \in L_2 \end{array} \right\}$
E^*	L^* (*star*)	$L^0 = \{\varepsilon\},\ L^{n+1} = L^n \cdot L$ $L^* = \bigcup_{n \geq 0} L^n$
$E_1 \cap E_2$	$L_1 \cap L_2$ (*intersection*)	$\{w \mid w \in L_1 \text{ and } w \in L_2\}$

Figure 2-1: Operations on languages

The result of evaluating an open expression E under the variable assignment σ is denoted $E(\sigma)$. The notation $E(\sigma)$ emphasizes the fact that open expressions may be regarded as language operators acting over the language variables.

An **equation** (resp. **in-equation**) between shuffle-intersection regular expressions is a pair of open shuffle-intersection regular expressions and is written $E_1 = E_2$ (resp. $E_1 \subseteq E_2$). An equation (resp. in-equation) $E_1 = E_2$ (resp. $E_1 \subseteq E_2$) is **valid** iff for any variable assignment $\sigma: V \to 2^{\Sigma^*}$ we have $E_1(\sigma) = E_2(\sigma)$ (resp. $E_1(\sigma) \subseteq E_2(\sigma)$).

It is easy to see that the equation $E_1 = E_2$ is valid iff the two in-equations $E_1 \subseteq E_2$ and $E_2 \subseteq E_1$ are valid. Conversely, the validity of the in-equation $E_1 \subseteq E_2$ is equivalent to the validity of the equation $E_1 + E_2 = E_2$. Therefore studying the validity of equations or in-equations are two perfectly equivalent problems. In the following we will concentrate our attention on in-equations.

A **constant free** shuffle-intersection regular expression (CFSI-expression) is an open shuffle-intersection regular expression which contains as basic symbols only language variables. In other words, CFSI-expressions are defined by the grammar

$$E = x \mid E + E \mid E;E \mid E^* \mid E \| E \mid E \cap E$$

where x ranges over the set of variable symbols V.

Chapter 3

Valid Regular Identities

In this chapter we recall a well known theorem about valid equations between open regular expressions and shows that it fails to hold if other operations, such as shuffle and intersection, are taken into account.

Theorem 1 *Valid equations between open regular expressions are decidable.*

The proof of this "folk" theorem is straightforward. Let E_1 and E_2 be two regular expressions with free language variables. To decide if $E_1 = E_2$ is a valid identity, treat the variables as symbols and decide if the closed identity is true:

- If $E_1 = E_2$ is not a closed identity when the variables are treated as constant symbols, we have $E_1(\sigma) \neq E_2(\sigma)$ for the variable assignment $\sigma(x) = \{x\}$ and therefore $E_1 = E_2$ is not valid.

- On the other hand, if E_1 and E_2 (interpreted as closed expressions) represent the same language L, than for any variable assignment σ we have $E_1(\sigma) = L[\sigma] = E_2(\sigma)$ where $L[\sigma]$ denotes the language obtained from L replacing each occurrence of the symbol associated to the variable x with the strings in $\sigma(x)$ in all possible ways. Therefore $E_1 = E_2$ is valid.

For example the identity $X;(Y + Z) = X;Z + X;Z$ is valid because $a;(b + c) = a;b + a;c$.

Notice that if we allow other operations (e.g. $\|$ or $\cap$) in the expressions, the instantiation of variables by symbols fails to imply valid identities. For example, although $a\|b = a;b + b;a$ is true, the equation $X\|Y = X;Y + Y;X$ is not valid.

We make the following remarks on how the folk theorem has been proved.

1. We have associated to each open regular expression E a language $L(E)$, obtained by treating the variables in E as ordinary symbols. This language $L(E)$ may be regarded as a semantics for open regular expressions.

2. This semantics is proved correct by showing that for any variable assignment σ we can compute $E(\sigma)$ starting from $L(E)$. In other words, the information given by the semantics $L(\cdot)$ is enough to decide open equations.

3. If two expressions E_1 and E_2 are mapped to different languages $L(E_1) \neq L(E_2)$, then we can find a counterexample to the validity of the equation $E_1 = E_2$, namely $\sigma(x) = \{x\}$. Thus, the semantics $L(\cdot)$ is also fully abstract for open regular expressions with respect to the validity equivalence relation.

Notice that the proof of full abstraction relies on the assumption that there are enough constant symbols to represent all the language variables. We will come back to this point in chapter 6.

Chapter 4

Semantics

In this chapter a semantics for constant-free shuffle-intersection regular expressions is defined and proved fully abstract for valid inequalities.

Definition 1 *Let V be a set of variable symbols. A* word constraint *over V is a pair $C = (n, \rho)$ where n is a non-negative integer and ρ is a subset of $2^{\{1,\ldots,n\}} \times V$. The integer n is called the* length *of C and is denoted by $|C|$. The elements of ρ are called* simple constraints.

A constraint family *is a set $\mathcal{F}$ of word constraints. The set of all word constraints over V is denoted by $\mathcal{C}_V$.*

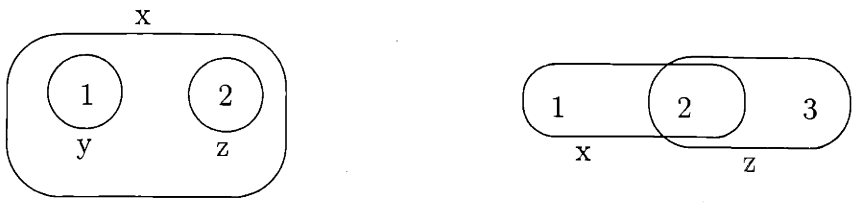
Definition 2 *Let $\sigma: V \to \mathcal{L}(\Sigma)$ be a variable assignment. A word $W \in \Sigma^*$ satisfies a word constraint $C = (n, \rho)$ under the assignment σ, written $W \models_\sigma C$, if the following two conditions hold:*

1. *$|W| = n$*

2. *for all simple constraints $(S, x) \in \rho$, $W[S] \in \sigma(x)$.*

For example, if Q2 is the word constraint $(3, \{(\{1,2\}, x), (\{2,3\}, z)\})$ shown in figure 4-1, and σ is the assignment $\sigma(x) = \sigma(z) = \{ab, ba\}$, we have $aba \models_\sigma$ Q2 and $bab \models_\sigma$ Q2. Moreover, *aba* and *bab* are the only two words that satisfy Q2 under σ.

This notion of word constraint satisfiability can be used to associate to each variable assignment σ a function from word constraints to languages.



Q1 Q2

Figure 4-1: Two word constraints

Definition 3 *For any word constraint C we define*

$$\sigma(C) = \{W \in \Sigma^* \mid W \models_\sigma C\}.$$

This function is extended to constraint families by pointwise union:

$$\sigma(\mathcal{F}) = \bigcup_{C \in \mathcal{F}} \sigma(C).$$

We now define three fundamental operations on word constraints: shuffle, concatenation and meet.

Definition 4 *Let $C_1 = (n_1, \rho_1)$ and $C_2 = (n_2, \rho_2)$ be two word constraints.*

1. *The concatenation of C_1 and C_2, written $C_1 \cdot C_2$, is the word constraint*

$$(n_1 + n_2, \rho_1 \cup \rho_2')$$

where $\rho_2' = \{(\{i + n_1 : i \in S\}, x) : (S, x) \in \rho_2\}$ is the set of simple constraints of C_2 translated by n_1 positions to the right.

2. *The shuffle of C_1 and C_2, written $C_1 \| C_2$, is the set of all word constraints*

$$(n_1 + n_2, \{(f_i(S), x) \mid (S, x) \in \rho_i \text{ for some } i = 1, 2\})$$

where $f_i: \{1, \ldots, n_i\} \to \{1, \ldots, n_1 + n_2\}$, $i = 1, 2$, *are two monotone functions such that* $((f_1(\{1, \ldots, n_1\}), f_2(\{1, \ldots, n_2\}))$ *is a partition of* $\{1, \ldots, n_1 + n_2\}$[1].

3. *The* meet *of two word constraints* C_1 *and* C_2, *written* $C_1 \wedge C_2$, *is defined iff* $n_1 = n_2$ *and in such a case it is the word constraint*

$$C_1 \wedge C_2 = (n_1, \rho_1 \cup \rho_2).$$

As an example of concatenation, consider the word constraints Q1 and Q2 in figure 4-1. Their concatenation Q1 · Q2 is the word constraint

$$(5, \{(\{1\}, y), (\{2\}, z), (\{1, 2\}, x)(\{3, 4\}, x), (\{4, 5\}, z)\})$$

(see figure 4-2).

A possible shuffle of Q1 and Q2 is the word constraint

$$(5, \{(\{1\}, y), (\{4\}, z), (\{1, 4\}, x)(\{2, 3\}, x), (\{3, 5\}, z)\})$$

(see figure 4-3).

As an example of meet consider the word constraints Q1 · $(1, \emptyset)$ and Q2. Their meet is defined because they have both length 3, and it is the word constraint

$$(3, \{(\{1\}, y), (\{2\}, z), (\{1, 2\}, x), (\{1, 2\}, x), (\{2, 3\}, z)\})$$

(see figure 4-4).

The names of *concatenation*, *shuffle* and *meet* for the above operations on word constraints are justified by the following Proposition.

Proposition 1 *For any word constraints* C_1 *and* C_2 *and for any variable assignment* $\sigma: V \to \mathcal{L}(\Sigma)$ *we have:*

1. $\sigma(C_1) \cdot \sigma(C_2) = \sigma(C_1 \cdot C_2)$

[1] recall that for any sets $A, B \subseteq C$, (A, B) is a partition of C iff $A \cap B = \emptyset$ and $A \cup B = C$

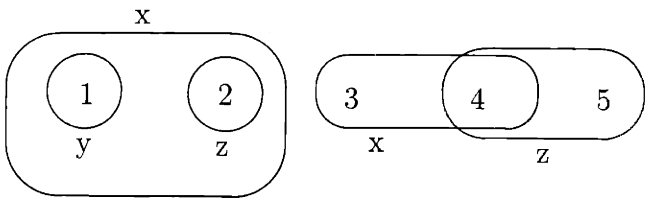


Figure 4-2: Word constraint Q1 · Q2

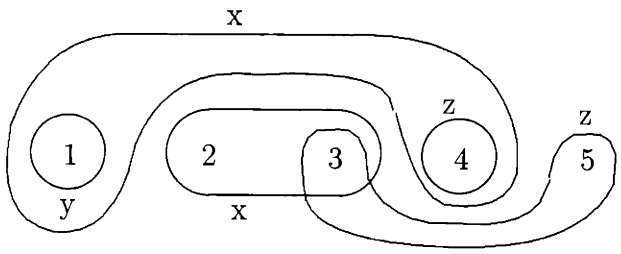


Figure 4-3: An element of Q1||Q2

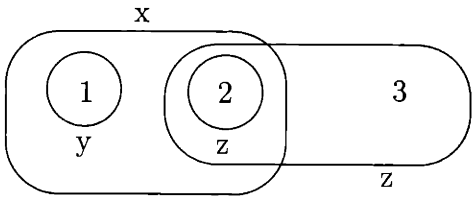


Figure 4-4: $(Q1 \cdot (1, \emptyset)) \wedge Q2$

2. $\sigma(C_1)\|\sigma(C_2) = \sigma(C_1\|C_2)$

3. $\sigma(C_1) \cap \sigma(C_2) = \begin{cases} \sigma(C_1 \wedge C_2) & \text{if } C_1 \wedge C_2 \text{ is defined,} \\ \emptyset & \text{otherwise} \end{cases}$

Proof: It easily follows from the definitions. □

All operations on word constraints can be lifted to constraint families. Also, exponentiation and star of constraint families can be defined in terms of the concatenation operation.

Definition 5 *Let $\mathcal{F}$, $\mathcal{F}_1$ and $\mathcal{F}_2$ be constraint families. We define the following operations.*

1. *Concatenation:*

$$\mathcal{F}_1 \cdot \mathcal{F}_2 = \{C_1 \cdot C_2 \mid C_1 \in \mathcal{F}_1, C_2 \in \mathcal{F}_2\}$$

2. *Shuffle:*

$$\mathcal{F}_1\|\mathcal{F}_2 = \bigcup\{C_1\|C_2 \mid C_1 \in \mathcal{F}_1, C_2 \in \mathcal{F}_2\}$$

3. *Meet:*

$$\mathcal{F}_1 \wedge \mathcal{F}_2 = \{C_1 \wedge C_2 \mid C_1 \in \mathcal{F}_1, C_2 \in \mathcal{F}_2 \text{ and } C_1 \wedge C_2 \text{ is defined}\}$$

4. *Exponentiation:*

   - $\mathcal{F}^0 = \{(0, \emptyset)\}$
   - $\mathcal{F}^{n+1} = \mathcal{F} \cdot \mathcal{F}^n$

5. *Closure:*

$$\mathcal{F}^* = \bigcup_{n \geq 0} \mathcal{F}^n$$

The equivalent of Proposition 1 still holds for constraint families.

Proposition 2 *For all constraint families $\mathcal{F}$, $\mathcal{F}_1$ and $\mathcal{F}_2$ and for any variable assignment $\sigma: V \to \mathcal{L}(\Sigma)$ we have:*

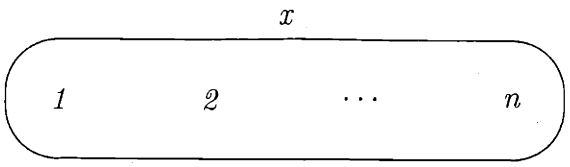


Figure 4-5: An element of $[\![x]\!]$.

1. $\sigma(\mathcal{F}_1 \cup \mathcal{F}_2) = \sigma(\mathcal{F}_1) \cup \sigma(\mathcal{F}_2)$
2. $\sigma(\mathcal{F}_1 \cdot \mathcal{F}_2) = \sigma(\mathcal{F}_1) \cdot \sigma(\mathcal{F}_2)$
3. $\sigma(\mathcal{F}_1 \| \mathcal{F}_2) = \sigma(\mathcal{F}_1) \| \sigma(\mathcal{F}_2)$
4. $\sigma(\mathcal{F}_1 \wedge \mathcal{F}_2) = \sigma(\mathcal{F}_1) \cap \sigma(\mathcal{F}_2)$
5. $\sigma(\mathcal{F}^n) = (\sigma(\mathcal{F}))^n$
6. $\sigma(\mathcal{F}^*) = (\sigma(\mathcal{F}))^*$

Proof: It easily follows from Definitions 3 and 5 and Proposition 1. □

We are now ready to define a semantics for CFSI-expressions.

Definition 6 *The* word constraint semantics *of* CFSI-*expression E is the constraint family $[\![E]\!]$ over the variables in E, defined by induction on the structure of E as follows,*

1. $[\![x]\!] = \{(n, \{(\{1, \ldots, n\}, x)\}) \mid n \geq 0\}$ *(see figure 4-5),*
2. $[\![E_1 + E_2]\!] = [\![E_1]\!] \cup [\![E_2]\!]$,
3. $[\![E_1; E_2]\!] = [\![E_1]\!] \cdot [\![E_2]\!]$,
4. $[\![E_1 \| E_2]\!] = [\![E_1]\!] \| [\![E_2]\!]$,
5. $[\![E_1 \cap E_2]\!] = [\![E_1]\!] \wedge [\![E_2]\!]$,
6. $[\![E^*]\!] = ([\![E]\!])^*$.

A fundamental property of the word constraint semantics $[\![\cdot]\!]$, is that E and $[\![E]\!]$ define the same operation over languages.

Proposition 3 *For any* CFSI*-expression* E *and variable assignment* σ*,*

$$E(\sigma) = \sigma([\![E]\!])$$

Proof: The proof is by induction on the structure of E. The base case is trivial. The other cases easily follow from Definition 6 and Proposition 2. □

We can now give a characterization of valid in-equations $E_1 \subseteq E_2$ between CFSI-expressions that does not involve any quantification over variable assignments.

First we define an closure operator *Aug* on constraint families.

Definition 7 *The* V*-augmentation closure of a constraint family* $\mathcal{F}$ *is given by*

$$\mathrm{Aug}_V(\mathcal{F}) = \mathcal{F} \wedge \mathcal{C}_V$$

where $\mathcal{C}_V$ *is the set of all word constraints over the variables* V*.*

When the set of variables V is not relevant or it is otherwise clear from the context, we omit the subscript V from the operator *Aug*.

Proposition 4 Aug *is an upper closure operator, i.e. for all constraint families* $\mathcal{F}$ *and* $\mathcal{F}'$ *it satisfies the following properties:*

1. *Monotonicity: if* $\mathcal{F} \subseteq \mathcal{F}'$ *then* $\mathrm{Aug}(\mathcal{F}) \subseteq \mathrm{Aug}(\mathcal{F}')$.
2. *Extensivity:* $\mathcal{F} \subseteq \mathrm{Aug}(\mathcal{F})$.
3. *Idempotence:* $\mathrm{Aug}(\mathcal{F}) = \mathrm{Aug}(\mathrm{Aug}(\mathcal{F}))$.

Proof: Monotonicity of *Aug* follows immediately from the definition of $\wedge$ over constraint families.

To prove extensivity, consider any word constraint $C \in \mathcal{F}$. We have $(|C|, \emptyset) \in \mathcal{C}$ and $C = C \wedge (|C|, \emptyset)$. Therefore $C \in \mathcal{F} \wedge \mathcal{C} = Aug(\mathcal{F})$.

Idempotence follows from the identity $\mathcal{C} \wedge \mathcal{C} = \mathcal{C}$ (word constraints are closed under meet). □

Another fundamental property of the augmentation operator is expressed in the following proposition.

Proposition 5 *Let $\sigma: V \to \mathcal{L}(\Sigma)$ be a variable assignment. For any constraint family $\mathcal{F}$,*

$$\sigma(\mathrm{Aug}(\mathcal{F})) = \sigma(\mathcal{F}).$$

Proof: By Proposition 4, $\mathcal{F} \subseteq Aug(\mathcal{F})$ is true. Applying σ to both sides we get $\sigma(\mathcal{F}) \subseteq \sigma(Aug(\mathcal{F}))$. Conversely, by Proposition 2 and the definition of Aug we obtain $\sigma(Aug(\mathcal{F})) = \sigma(\mathcal{F}) \cap \sigma(\mathcal{C}) \subseteq \sigma(\mathcal{F})$. This concludes the proof of $\sigma(\mathcal{F}) = \sigma(Aug(\mathcal{F}))$. □

Theorem 2 *For any pair (E_1, E_2) of constant-free shuffle-intersection regular expressions over V, $E_1 \subseteq E_2$ is valid if and only if*

$$\mathrm{Aug}_V([\![E_1]\!]) \subseteq \mathrm{Aug}_V([\![E_2]\!]).$$

In other words, the augmented word constraint semantics $\mathrm{Aug}_V([\![\cdot]\!])$ is fully abstract on CFSI*-expressions over V with respect to the valid inclusion relation.*

Proof: By Proposition 3 the in-equation $E_1 \subseteq E_2$ is valid if and only if for all variable assignment σ we have $\sigma([\![E_1]\!]) \subseteq \sigma([\![E_2]\!])$. It remains to prove that

$$Aug_V([\![E_1]\!]) \subseteq Aug_V([\![E_2]\!]) \tag{4.1}$$

is equivalent to

$$\forall\sigma.\sigma([\![E_1]\!]) \subseteq \sigma([\![E_2]\!]). \tag{4.2}$$

We will prove that (4.1) implies (4.2), (4.2) implies

$$[\![E_1]\!] \subseteq Aug_V([\![E_2]\!]) \tag{4.3}$$

and finally (4.3) implies (4.1).

(4.1) $\Rightarrow$ (4.2): Assume that (4.1) is true and let σ be a variable assignment. By Proposition 5 we have $\sigma([\![E_1]\!]) = \sigma(Aug_V([\![E_1]\!])) \subseteq \sigma(Aug_V([\![E_2]\!])) = \sigma([\![E_2]\!])$. This proves (4.1) $\Rightarrow$ (4.2).

(4.2) $\Rightarrow$ (4.3): Assume that (4.2) is true and let $C_1 = (n_1, \rho_1)$ be an element of $[\![E_1]\!]$. We want to show that $C_1 \in Aug_V([\![E_2]\!])$.

Consider a word $W = a_1 a_2 \cdots a_{n_1}$ made of all distinct symbols ($a_i \neq a_j$ for any $i \neq j$), and let σ be the variable assignment

$$\sigma(x) = \{W[S] \mid (S, x) \in \rho_1\}.$$

Clearly the word W satisfies the word constraint C_1 under the assignment σ (actually σ is the smallest variable assignment such that $W \models_\sigma C_1$). So, $W \in \sigma([\![E_1]\!]) \subseteq \sigma([\![E_2]\!])$ and there must be a word constraint $C_2 = (n_2, \rho_2)$ in $[\![E_2]\!]$ such that $W \models_\sigma C_2$.

We will show that $C_2 \wedge C_1 = C_1$. This is true iff $n_1 = n_2$ and $\rho_1 \supseteq \rho_2$. By definition of $W \models_\sigma C_2$ we have $n_2 = |W| = n_1$. Let us prove that $\rho_1 \supseteq \rho_2$. Let (S, x) be a simple constraint in ρ_2. Again, by definition of $W \models_\sigma C_2$ we have $W[S] \in \sigma(x) = \{W[S'] \mid (S', x) \in \rho_1\}$. So, $W[S] = W[S']$ for some S' such that $(S', x) \in \rho_1$. But $W[S] = W[S']$ is possible only if $S = S'$, because W is a word made of all distinct symbols. Therefore $(S, x) = (S', x) \in \rho_1$, proving that $\rho_1 \supseteq \rho_2$.

So, the word constraint C_1 equals $C_2 \wedge C_1$. Since $C_1 \in \mathcal{C}_V$ and $C_2 \in [\![E_2]\!]$, we have $C_1 = C_2 \wedge C_1 \in [\![E_2]\!] \wedge \mathcal{C}_V = Aug_V([\![E_2]\!])$.

(4.3) $\Rightarrow$ (4.1): Assume that (4.3) is true. Applying Aug_V to both sides and using Proposition 4 we get $Aug_V([\![E_1]\!]) \subseteq Aug_V(Aug_V([\![E_2]\!])) = Aug_V([\![E_2]\!])$.

This completes the proof of the Theorem. □

The following corollary will be useful to prove the decidability of valid inclusions in the next chapter.

Corollary 1 *For any pair (E_1, E_2) of constant-free shuffle-intersection regular expressions over V, $E_1 \subseteq E_2$ is valid if and only if $[\![E_1]\!] \subseteq \mathrm{Aug}_V([\![E_2]\!])$.*

Proof: This has already been proved as an intermediate step in the proof of Theorem 2. □

Chapter 5

Decidability

In this chapter we prove that valid inclusions between CFSI-expressions are decidable. We will make use of the fully abstract semantics introduced in the last chapter to reduce the problem of deciding if $E_1(\sigma) \subseteq E_2(\sigma)$ is true for all σ, to the study of a single inclusion between constraint families $[\![E_1]\!] \subseteq Aug([\![E_2]\!])$.

The set of word constraints $[\![E_1]\!]$ and $Aug([\![E_2]\!])$ are not finite and therefore they cannot be explicitly built and compared. We will try to reduce the containment $[\![E_1]\!] \subseteq Aug([\![E_2]\!])$ between sets of word constraints to an inclusion between regular languages, which can be decided with standard automata-based techniques.

So, let us define a function Φ from word constraints to words over some alphabet Δ. First of all observe that if we want to decide the validity of the equation $E_1 \subseteq E_2$, we need only to consider the variable symbols that occur in the expressions E_1 and E_2. In the rest of this chapter we assume that V is the set of variable symbols that occur in the expressions E_1 and E_2. In particular V is a finite set.

We define the alphabet Δ to be the set of relations between the set of variable symbols V and the set of five symbols $\{\mathtt{B}, \mathtt{Y}, \mathtt{N}, \mathtt{E}, \mathtt{A}\}$[1], that is $\Delta = 2^{V \times \{\mathtt{B},\mathtt{Y},\mathtt{N},\mathtt{E},\mathtt{A}\}}$. All these relations are finite and can be represented as $|V|$ by 5 matrices with 0-1 entries. We will use these finite relations as symbols of the alphabet Δ. In particular, a word over Δ is a finite sequence of (finite) relations. Notice that the set Δ, although big, still has only finitely many elements.

[1]The symbols B, E, Y, N and A stand respectively for the words *begin*, *end*, *yes*, *no*, *all*.

Definition 8 *The* meet *of any two strings α and β in Δ^*, written $\alpha \wedge \beta$, is defined iff the two strings have the same length ($|\alpha| = |\beta|$). If $\alpha = a_1 a_2 \ldots a_n$ and $\beta = b_1 b_2 \ldots b_n$ then*

$$\alpha \wedge \beta = (a_1 \cup b_1)(a_2 \cup b_2) \ldots (a_n \cup b_n).$$

We now define a function from word constraints to Δ^*.

Definition 9 *The function Φ maps word constraints to words in Δ^*. For any word constraint (n, ρ),*

- *if $\rho = \emptyset$ then $\Phi(n, \rho) = \underbrace{\emptyset \ldots \emptyset}_{n+1}$.*
- *if $\rho = \{(x, \emptyset)\}$ then $\Phi(n, \rho) = \{(x, \mathtt{A})\}\underbrace{\emptyset \ldots \emptyset}_{n}$.*
- *if $\rho = \{(x, \{j\})\}$ then $\Phi(n, \rho) = \emptyset a_1 a_2 \ldots a_n$, where*

$$a_i = \begin{cases} \{(x, \mathtt{A})\} & \text{if } i = j \\ \emptyset & \text{otherwise} \end{cases}$$

- *if $\rho = \{(x, S)\}$ and $|S| \geq 2$, then $\Phi(n, \rho) = \emptyset a_1 a_2 \ldots a_n$ where*

$$a_i = \begin{cases} \{(x, \mathtt{B})\} & \text{if } i = \min(S) \\ \{(x, \mathtt{E})\} & \text{if } i = \max(S) \\ \{(x, \mathtt{Y})\} & \text{if } \min(S) < i < \max(S) \text{ and } i \in S \\ \{(x, \mathtt{N})\} & \text{if } \min(S) < i < \max(S) \text{ and } i \notin S \\ \emptyset & \text{otherwise} \end{cases}$$

- *if ρ contains more than one simple constraint*

$$\Phi(n, \rho) = \bigwedge_{(x,S) \in \rho} \Phi(n, \{(x, S)\}).$$

For example if

$$C = (5, \{(\{1\}, y), (\{4\}, z), (\{1, 4\}, x)(\{2, 3\}, x), (\{3, 5\}, z)\})$$

is the word constraint shown in figure 4-3, then $\Phi(C)$ is the word

$$\emptyset\{(x,\mathtt{B}),(y,\mathtt{A})\}\{(x,\mathtt{N}),(x,\mathtt{B})\}\{(x,\mathtt{N}),(x,\mathtt{E}),(z,\mathtt{B})\}\{(x,\mathtt{E}),(z,\mathtt{N}),(z,\mathtt{A})\}\{(z,\mathtt{E})\}.$$

Definition 10 *Let $\alpha = a_0 \ldots a_n$ and $\beta = b_0 \ldots b_m$ be two non-empty words in Δ^*.*

1. *The Δ-concatenation of α and β, written $\alpha \triangle \beta$, is the word*

$$(a_0 \cup b_0)a_1 \ldots a_n b_1 \ldots b_m.$$

2. *For any word $\gamma \in \Delta^*$, define the relation $N(\gamma)$ as the set of pairs $(x,\mathtt{N})$ such that the last element of γ contains either $(x,\mathtt{B})$, $(x,\mathtt{Y})$ or $(x,\mathtt{N})$ (if γ is the empty word then $N(\gamma) = \emptyset$).*

   The Δ-shuffle of α and β, written $\alpha \pitchfork \beta$, is the set of words

$$(a_0 \cup b_0)\alpha'_1\beta'_1\alpha'_2\beta'_2 \ldots \alpha'_k\beta'_k$$

   such that for some k and some (possibly empty) words $\alpha_1, \ldots, \alpha_k, \beta_1, \ldots, \beta_k$,

   - $\alpha = a_0\alpha_1 \ldots \alpha_k$
   - $\beta = b_0\beta_1 \ldots \beta_k$
   - *for all $i = 1, \ldots, k$,*
     - $\alpha'_i = \alpha_i \wedge N(\beta_1 \ldots \beta_{i-1})^{|\alpha_i|}$
     - $\beta'_i = \beta_i \wedge N(\alpha_1 \ldots \alpha_i)^{|\beta_i|}$

The above operations have been called Δ-concatenation and Δ-shuffle to distinguish them from the usual concatenation and shuffle of words. However, they are closely related to the corresponding operations on word constraints, as shown in the following Proposition.

Proposition 6 *For any word constraints C_1 and C_2 we have*

1. $\Phi(C_1 \cdot C_2) = \Phi(C_1) \triangle \Phi(C_2)$

2. $\Phi(C_1 \| C_2) = \Phi(C_1) \mathbin{\Cup\!\!\!\!+} \Phi(C_2)$

3. $C_1 \wedge C_2$ *is defined iff* $\Phi(C_1) \wedge \Phi(C_2)$ *is defined, and in this case* $\Phi(C_1 \wedge C_2) = \Phi(C_1) \wedge \Phi(C_2)$

Proof: It directly follows from the definitions. □

We now define the meet, Δ-concatenation, Δ-shuffle and Δ-star of languages over Δ.

Definition 11 *Let* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *be languages over* Δ. *We define the following operations.*

1. Δ-*Concatenation:*

$$\mathcal{L}_1 \mathbin{\triangle} \mathcal{L}_2 = \{W_1 \mathbin{\triangle} W_2 \mid W_1 \in \mathcal{L}_1, W_2 \in \mathcal{L}_2\}$$

2. Δ-*Shuffle:*

$$\mathcal{L}_1 \mathbin{\Cup\!\!\!\!+} \mathcal{L}_2 = \bigcup \{W_1 \mathbin{\Cup\!\!\!\!+} W_2 \mid W_1 \in \mathcal{L}_1, W_2 \in \mathcal{L}_2\}$$

3. *Meet:*

$$\mathcal{L}_1 \wedge \mathcal{L}_2 = \{W_1 \wedge W_2 \mid W_1 \in \mathcal{L}_1, W_2 \in \mathcal{L}_2 \text{ and } W_1 \wedge W_2 \text{ is defined}\}$$

4. Δ-*Closure:*

$$(\mathcal{L}_1)^{\triangle} = \bigcup_{n \geq 0} (\mathcal{L}_1)^{\triangle n}$$

where $(\mathcal{L}_1)^{\triangle n}$ *is defined by* $(\mathcal{L}_1)^{\triangle 0} = \{\emptyset\}$, $(\mathcal{L}_1)^{\triangle n+1} = (\mathcal{L}_1)^{\triangle n} \mathbin{\triangle} \mathcal{L}_1$.

These operations are closely related to the corresponding operations on constraint families.

Proposition 7 *For all constraint families* $\mathcal{F}_1$ *and* $\mathcal{F}_2$ *we have:*

1. $\Phi(\mathcal{F}_1 \cup \mathcal{F}_2) = \Phi(\mathcal{F}_1) \cup \Phi(\mathcal{F}_2)$

2. $\Phi(\mathcal{F}_1 \cdot \mathcal{F}_2) = \Phi(\mathcal{F}_1) \triangle \Phi(\mathcal{L}_2)$

3. $\Phi(\mathcal{F}_1 \| \mathcal{F}_2) = \Phi(\mathcal{F}_1) \pitchfork \Phi(\mathcal{F}_2)$

4. $\Phi(\mathcal{F}_1 \wedge \mathcal{F}_2) = \Phi(\mathcal{F}_1) \wedge \Phi(\mathcal{F}_2)$

5. $\Phi((\mathcal{F}_1)^*) = (\Phi(\mathcal{F}_1))^{\triangle}$

Proof: It easily follows from Proposition 6. □

Moreover, all operations preserve regularity of languages.

Proposition 8 *If $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Delta^*$ are regular languages, then*

1. *$\mathcal{L}_1 \cup \mathcal{L}_2$ is regular,*

2. *$\mathcal{L}_1 \triangle \mathcal{L}_2$ is regular,*

3. *$\mathcal{L}_1 \pitchfork \mathcal{L}_2$ is regular,*

4. *$\mathcal{L}_1 \wedge \mathcal{L}_2$ is regular,*

5. *$(\mathcal{L}_1)^{\triangle}$ is regular.*

Proof: Given finite state automata recognizing $\mathcal{L}_1$ and $\mathcal{L}_2$ one can easily build finite state automata recognizing $\mathcal{L}_1 \cup \mathcal{L}_2$, $\mathcal{L}_1 \triangle \mathcal{L}_2$, $\mathcal{L}_1 \pitchfork \mathcal{L}_2$, $\mathcal{L}_1 \wedge \mathcal{L}_2$ and $\mathcal{L}_1^{\triangle}$. A possible construction is given in the appendix. □

Lemma 1 *For any* CFSI*-expression E, $\Phi([\![E]\!])$ and $\Phi(\mathrm{Aug}_V([\![E]\!]))$ are regular languages.*

Proof: First we prove that $\Phi([\![E]\!])$ is regular. The proof is by induction on the structure of E. If $E = x$ then

$$\Phi([\![E]\!]) = \{(x, \mathtt{A})\} + \emptyset\{(x, \mathtt{A})\} + \emptyset\{(x, \mathtt{B})\}\{(x, \mathtt{Y})\}^*\{(x, \mathtt{E})\}$$

which is regular. The other cases easily follow from Proposition 8.

Now consider the language $\Phi(Aug_V([\![E]\!]))$. Both $\Phi([\![E]\!])$ and $\Phi(\mathcal{C}_V)$ are regular (an automaton recognizing $\Phi(\mathcal{C}_V)$ is given in the appendix). Therefore also their intersection $\Phi(Aug_V([\![E]\!])) = \Phi([\![E]\!] \wedge \mathcal{C}_V) = \Phi([\![E]\!]) \cap \Phi(\mathcal{C}_V)$ is regular. □

From the previous Lemma, for any two expressions E_1 and E_2, the languages $\Phi([\![E_1]\!])$ and $\Phi(Aug([\![E_2]\!]))$ are regular. Moreover, it is implicit in the proof of the Lemma that we can build finite state automata recognizing these languages, yielding a first decidability result.

Proposition 9 *The set of* CFSI*-expression pairs* (E_1, E_2) *such that*

$$\Phi([\![E_1]\!]) \subseteq \Phi(\mathrm{Aug}_V([\![E_2]\!]))$$

is decidable.

Proof: To decide $\Phi([\![E_1]\!]) \subseteq \Phi(Aug_V([\![E_2]\!]))$, build finite state automata recognizing the languages $\Phi([\![E_1]\!])$ and $\Phi(Aug_V([\![E_2]\!]))$ as shown in the proof of Lemma 1 and then compare them for language inclusion. □

The above Proposition does not imply yet the decidability of valid inclusions $E_1 \subseteq E_2$ because the function Φ is not injective and therefore $\Phi([\![E_1]\!]) \subseteq \Phi(Aug_V([\![E_2]\!]))$ is not equivalent to $[\![E_1]\!] \subseteq Aug_V([\![E_2]\!])$.

Definition 12 *A relation* $a \in \Delta$ *is conflict free iff for any* x *there exists at most one symbol* $z \in \{\mathtt{B}, \mathtt{Y}, \mathtt{N}, \mathtt{E}, \mathtt{A}\}$ *such that* $(x, z) \in a$ *(i.e.* a *is a partial function). The restricted alphabet* Δ' *is the set of elements of* Δ *that are conflict free. A word constraint* C *such that* $\Phi(C) \in (\Delta')^*$ *is said conflict free.*

Proposition 10 *For any conflict free word constraints* C_1, C_2, *if* $\Phi(C_1) = \Phi(C_2)$ *then* $C_1 = C_2$.

Proof: Assume that $\Phi(C_1) = \Phi(C_2) = a_0 a_1 \ldots a_n$ is conflict free, i.e. $a_i \in \Delta'$ for all $i = 0, \ldots, n$. Clearly $|C_1| = |C_2| = n$. Let $C_1 = (n, \rho_1)$ and $C_2 = (n, \rho_2)$. We have to prove that $\rho_1 = \rho_2$. First we show that $\rho_1 \subseteq \rho_2$. Let $(x, S) \in \rho_1$. There are three cases:

- If $S = \emptyset$ then $(x, \mathtt{A}) \in a_0$ and therefore $(x, \emptyset) \in \rho_2$.

- If $S = \{i\}$ for some $i \in \{1, \ldots, n\}$, then $(x, \mathtt{A}) \in a_i$ and therefore $(x, \{i\}) \in \rho_2$.

- If $|S| \geq 2$ then $(x, \mathtt{B}) \in a_{\min(S)}$ and there must be some $(x, S') \in \rho_2$ such that $|S'| \geq 2$ and $\min(S') = \min(S)$. We will show that $S = S'$. Assume for contradiction that $S \neq S'$ and let i the smallest index for which S and S' differ. Assume without loss of generality that $i \in S$ and $i \notin S'$. If $i \leq \max S'$ then a_i contains both the pair $(x, \mathtt{N})$ and either $(x, \mathtt{Y})$ or $(x, \mathtt{E})$. If $i > \max S'$ then $a_{\max S'}$ contains both the pair $(x, \mathtt{E})$ and either $(x, \mathtt{Y})$ or $(x, \mathtt{N})$. In both cases $a_0 a_1 \ldots a_n$ is not conflict free, contradicting our hypothesis.

This proves that $\rho_1 \subseteq \rho_2$. The converse inclusion follows by symmetry. In conclusion $\rho_1 = \rho_2$ and $C_1 = C_2$. □

Therefore the function Φ is injective on conflict free word constraints. We now give a syntactic characterization of the expressions E such that the word constraints in $[\![E]\!]$ are conflict free.

Proposition 11 *If E does not contain repeated variable symbols, then $[\![E]\!]$ is a set of conflict free word constraints.*

Proof: By induction on the structure of E. □

Lemma 2 *If $\mathcal{F}_1$ and $\mathcal{F}_2$ are constraint families and $\Phi(\mathcal{F}_1) \subseteq (\Delta')^*$, then $\mathcal{F}_1 \subseteq \mathcal{F}_2$ if and only if $\Phi(\mathcal{F}_1) \subseteq \Phi(\mathcal{F}_2)$.*

Proof: If $\mathcal{F}_1 \subseteq \mathcal{F}_2$ then $\Phi(\mathcal{F}_1) \subseteq \Phi(\mathcal{F}_2)$ is obviously true. Conversely, assume that $\Phi(\mathcal{F}_1) \subseteq \Phi(\mathcal{F}_2)$ and let C_1 be an element of $\mathcal{F}_1$. The word $\Phi(C_1)$ is in $\Phi(\mathcal{F}_1) \subseteq \Phi(\mathcal{F}_2)$. Therefore there exists a word constraint $C_2 \in \mathcal{F}_2$ such that $\Phi(C_1) = \Phi(C_2)$. By Proposition 10, the word constraint C_1 is equal to C_2, proving the containment $\mathcal{F}_1 \subseteq \mathcal{F}_2$. □

Lemma 3 *Given two expressions, E_1 and E_2, one can easily compute two other expressions, F_1 and F_2, such that F_1 does not contain repeated variables and $E_1 \subseteq E_2$ is valid iff $F_1 \subseteq F_2$ is valid.*

Proof: Let F_1 be the expression obtained from E_1 replacing each variable symbol occurrence x by a distinct variable x_i, and let F_2 be the expression obtained from E_2 replacing each occurrence of x by the subexpression $(x_1 + x_2 + \cdots + x_n)$, where $x_1, x_2, \ldots, x_n$ are all the variable symbols that have been replaced for x in F_1.

Clearly F_1 does not contain repeated variables. It remains to prove that the inclusion $E_1 \subseteq E_2$ is equivalent to $F_1 \subseteq F_2$.

First assume that $E_1 \subseteq E_2$ is valid and let σ be some variable assignment defined on the variables of F_1 and F_2. I define the assignment $\sigma'(x) = \bigcup_i \sigma(x_i)$. It is easily proved by induction on the structure of E_1 and E_2 that $F_1(\sigma) \subseteq E_1(\sigma')$ and $E_2(\sigma') = F_2(\sigma)$. Since $E_1 \subseteq E_2$ is valid, we have $F_1(\sigma) \subseteq E_1(\sigma') \subseteq E_2(\sigma') = F_2(\sigma)$, proving that also $F_1 \subseteq F_2$ is valid.

Now assume that $F_1 \subseteq F_2$ and let σ be some variable assignment on the variables of E_1 and E_2. Define the assignment $\sigma'(x_i) = \sigma(x)$. We have $E_1(\sigma) = F_1(\sigma')$ and $E_2(\sigma) = F_2(\sigma')$. Therefore $E_1(\sigma) = F_1(\sigma') \subseteq F_2(\sigma') = E_2(\sigma)$, proving the validity of $E_1 \subseteq E_2$. □

Theorem 3 *The set of valid inequalities $E_1 \subseteq E_2$ between constant-free shuffle-intersection regular expressions is decidable.*

Proof: We want to decide the validity of the inclusion $E_1 \subseteq E_2$. Let F_1 and F_2 be as in Lemma 3. The inclusion $E_1 \subseteq E_2$ is valid iff $F_1 \subseteq F_2$ does, and F_1 does not contain repeated variables. By Corollary 1 to the full abstraction theorem, $F_1 \subseteq F_2$ is valid iff $[\![F_1]\!] \subseteq Aug_V([\![F_2]\!])$, where V are the variable symbols in F_1. By Proposition 11 the language $\Phi([\![F_1]\!])$ is contained in $(\Delta')^*$ and by Lemma 2 the inclusion $[\![F_1]\!] \subseteq Aug_V([\![F_2]\!])$ between constraint families is equivalent to the inclusion $\Phi([\![F_1]\!]) \subseteq \Phi(Aug_V([\![F_2]\!]))$ between regular languages. The latter inclusion is decidable by Proposition 9. □

The following corollary says that if an in-equation is not valid then there exists a "small" counterexample to its validity.

Corollary 2 *For any* CFSI*-expressions* E_1 *and* E_2*, if* $E_1 \subseteq E_2$ *is not a valid inequality then there exists a word* W *and a variable assignment* σ *such that* $|W| \leq 2^{2^{O(n^2)}}$ *and* $W \in E_1(\sigma) \setminus E_2(\sigma)$.

Proof: This bound on the size of W is implicit in our proof of decidability: if n is the size of $E_1 \subseteq E_2$, then $F_1 \subseteq F_2$ has size $O(n^2)$. The automata construction in the appendix gives a $2^{O(n^2)}$ bound on the size of the non-deterministic finite state automata recognizing $\Phi([\![E_1]\!])$ and $\Phi([\![E_2]\!])$. The final bound $2^{2^{O(n^2)}}$ follows from the standard power-set construction to transform a nondeterministic finite state automaton into a deterministic one. □

Chapter 6

Open Problems

We have defined a fully abstract semantics for constant-free shuffle-intersection regular expressions and proved the decidability of valid in-equations between these expressions. The main limitation of our result is the inability to specify constant symbols in the expressions. An interesting question that we leave as an open problem is the following.

Open Problem 1 *Are valid equations between open regular expressions with shuffle, intersection and constants decidable?*

We believe that adding constant symbols to open regular expressions with intersection is harder than for other classes of expressions because of the following reason.

Constants and the intersection operation can be used to express validity over a fixed alphabet. For example, consider the alphabet $\Sigma = \{0, 1\}$ and let E_1 and E_2 be two open expressions. Define E_1' and E_2' as the expressions obtained from E_1 and E_2 by replacing each variable symbol x with the subexpression $(x \cap (0 + 1)^*)$. The equation $E_1 = E_2$ is valid over $2^{\{0,1\}^*}$ iff $E_1' = E_2'$ is valid for arbitrary languages.

Therefore the ability to decide valid regular equations with constants and intersection, implies the decidability of equations that are valid over 2^{Σ^*}, where Σ is a fixed finite alphabet (for example the binary alphabet $\Sigma = \{0, 1\}$).

The proof of our full abstraction theorem (Theorem 2) assumes that the alphabet of constant symbols is arbitrarily large. So, Theorem 2 and our decidability result in

Theorem 3 do not extend straightforwardly to open expressions with constants.

Interestingly, the folk theorem on valid regular identities (see Theorem 1) also is based on the availability of arbitrarily many constant symbols, and as far as we know the validity of regular equations over a binary alphabet is an open question.

Therefore, we point out as an interesting subproblem of problem 1, the following simpler question.

Open Problem 2 *Let Σ be a fixed finite alphabet (e.g. $\Sigma = \{0,1\}$). Are valid regular equations over 2^{Σ^*} decidable? Are valid regular equations with shuffle over 2^{Σ^*} decidable?*

We believe that part of the difficulty of Problem 1 is already captured in Problem 2 in its simplest version: the decidability of valid regular identities over a binary alphabet.

The set of operations considered here $(+, ;, ^*, \|, \cap)$ can be further extended to include many other operations. Two particularly interesting operations are the CSP-style synchronization-with-restriction [4] and the CCS-style synchronization-by-hiding-complementary-actions [12].

Other interesting operations are *renaming*, $[a/b]$, which replaces each occurrence of symbol b by a, and *hiding*, $-a$, which erases all occurrences of the symbol a from a word, where a and b range over the alphabet of constant symbols Σ.

Although our fully abstract semantics and decidability result can be easily extended to handle also hiding and renaming operations, we have not considered these operations here because they are not very interesting without being able to express constants.

Notice that the two operations considered here (shuffle and intersection) can be thought as special cases of the CSP synchronization operation $\|_L$. Namely, the shuffle operation $\|$ is equivalent to $\|_L$ when the synchronization set $L = \emptyset$ is empty. The intersection operation $\cap$ can be thought as the synchronization operation $\|_L$ when the synchronization set $L = \Sigma$ is the whole alphabet.

Actually, the decidability of regular expressions with shuffle, intersection, hiding and renaming would yield immediately a decidability result for CSP-style regular expressions over a finite alphabet: if Σ is a finite alphabet then for any synchronization set $L \subseteq \Sigma$ and for any languages $X, Y \in 2^{\Sigma^*}$ we have

$$(X \|_L Y) = ((X \| Y[L'/L]) \cap (\Sigma \setminus L + LL')^*) - L'$$

where L' is a disjoint copy of L and $[L'/L]$, $(\Sigma \setminus L)$, LL' and $-L'$ are shorthands for finite subexpressions or operation sequences defined as follows. If $L = \{a_1, \ldots, a_k\}$ and $\Sigma = \{a_1, \ldots, a_k, b_1, \ldots b_l\}$, then

$$\begin{aligned} L' &= \{a'_1, \ldots, a'_k\} \\ [L'/L] &= [a'_1/a_1][a'_2/a_2] \cdots [a'_k/a_k] \\ (\Sigma \setminus L) &= b_1 + b_2 + \cdots + b_l \\ LL' &= a_1 a'_1 + a_2 a'_2 + \cdots + a_k a'_k \\ -L' &= -a'_1 - a'_2 \cdots - a'_k. \end{aligned}$$

Therefore any expression with $\|_L$ can be translated into an equivalent (over the finite alphabet Σ) regular expression with shuffle, intersection, hiding and renaming.

A final question that we raise is whether the bound given in Corollary 2 on the size of the smallest counterexample to non valid in-equations is tight.

Open Problem 3 *Can the $2^{2^{O(n^2)}}$ bound in Corollary 2 be improved? Is that bound still valid if we allow constants to occur in the expressions?*

Bibliography

[1] S.L. Bloom and Z. Esik. Free shuffle algebras in language varieties. Technical Report 9407, Stevens Institute og Technology, September 1994.

[2] Martin Furer. The complexity of the inequivalence problem for regular expressions with intersection. In *Automata, Languages and Programming, 7th Colloquium*, 1980.

[3] Jay Loren Gischer. *Partial Orders and the Axiomatic Theory of Shuffle.* PhD thesis, Stanford University, 1984.

[4] C.A.R. Hoare. *Communicating Sequential Processes.* International Series in Computer Science. Prentice-Hall, 1985.

[5] J. H. Hopcroft and J. D. Ullman. *Introduction to automta theory, languages, and computation.* Addison-Wesley, 1979.

[6] H.B. Hunt III. On the time and tape complexity of languages. In *Proc. 5th ACM Symposium on Theory of Computing*, 1973.

[7] Alain J. Mayer and Larry J. Stockmeyer. The complexity of word problems – this time with interleaving. Technical Report RJ 8947, IBM Research Division, San Jose, CA, 1992.

[8] A. Meyer and A. Rabinovitch. Private Communication.

[9] A. R. Meyer. Concurrent process equivalences: Some decision problems (invited talk). In *STAC 95, Lect. Notes Comp. Sci. 900*, 1995.

[10] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on the Theory of Computing*, 1970.

[11] R. Milner. A complete inference system for a class of regular behaviors. *J. Comput. System Sci.*, 28, 1984.

[12] R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice-Hall, 1989.

[13] L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time. In *Proc. 5th ACM Symposium on Theory of Computing*, 1973.

Appendix A

The Automata Construction

In the proof of Proposition 8 we claimed that given finite state automata recognizing the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ we can build finite state automata recognizing the languages

1. $\mathcal{L}_1 \cup \mathcal{L}_2$
2. $\mathcal{L}_1 \triangle \mathcal{L}_2$
3. $\mathcal{L}_1 \barwedge \mathcal{L}_2$
4. $\mathcal{L}_1 \wedge \mathcal{L}_2$
5. $(\mathcal{L}_1)^{\triangle}$.

We now illustrate a possible way to construct finite state automata recognizing these languages.

We use the standard definition of finite state automaton: a finite state automaton is defined by a tuple $(S, \Sigma, s, F, \longrightarrow)$ where S is a finite set of states, $s \in S$ is the start state, $F \subseteq S$ is the set of accepting states, Σ is a finite alphabet and $\longrightarrow \subseteq S \times \Sigma \times S$ is a transition relation. We write $u \xrightarrow{a} v$ for $(u, a, v) \in \longrightarrow$.

Since for all automata considered here the alphabet Σ is fixed to $\Delta = 2^{V \times \{\mathtt{B},\mathtt{Y},\mathtt{N},\mathtt{E},\mathtt{A}\}}$, for brevity we will omit it in the following definitions. Let $A_1 = (S_1, s_1, F_1, \longrightarrow_1)$ and $A_2 = (S_2, s_2, F_2, \longrightarrow_2)$ be two finite state automata recognizing the languages $\mathcal{L}_1, \mathcal{L}_2 \subseteq \Delta^*$ and assume, without loss of generality, that the start states (s_1 and

s_2) have no incoming transitions, and the final states (F_1 and F_2) have no outgoing transitions (otherwise add new start and final states with the appropriate transitions).

1. An automaton recognizing the language $\mathcal{L}_1 \cup \mathcal{L}_2$ can be constructed from A_1 and A_2 in the usual way [5].

2. The language $\mathcal{L}_1 \triangle \mathcal{L}_2$ is recognized by the finite state automaton

$$(S_1 \times S_2, (s_1, s_2), F_1 \times F_2, \longrightarrow)$$

where the transition relation $\longrightarrow$ is defined by the rules

$$\frac{s_1 \xrightarrow{a_1}_1 u_1, s_2 \xrightarrow{a_2}_2 u_2}{(s_1, s_2) \xrightarrow{a} (u_1, u_2)} \quad (a = a_1 \cup a_2)$$

$$\frac{u \xrightarrow{a}_1 v}{(u, w) \xrightarrow{a} (v, w)}$$

$$\frac{u \xrightarrow{a}_2 v}{(t, u) \xrightarrow{a} (t, v)} \quad (t \in F_1).$$

3. The language $\mathcal{L}_1 \barwedge \mathcal{L}_2$ is recognized by the finite state automaton

$$(S_1 \times 2^V \times S_2 \times 2^V, (s_1, \emptyset, s_2, \emptyset), F_1 \times 2^V \times F_2 \times 2^V, \longrightarrow)$$

where the transition relation $\longrightarrow$ is defined by the rules

$$\frac{s_1 \xrightarrow{a_1}_1 u_1, s_2 \xrightarrow{a_2}_2 u_2}{(s_1, \emptyset, s_2, \emptyset) \xrightarrow{a} (u_1, \emptyset, u_2, \emptyset)} \quad (a = a_1 \cup a_2)$$

$$\frac{u \xrightarrow{a}_1 v}{(u, X, w, Y) \xrightarrow{a'} (v, X', w, Y)} \quad \left(\begin{array}{l} a' = a \cup (X \times \{\mathtt{N}\}), \\ X' = \{x \mid a \cap (\{x\} \times \{\mathtt{B}, \mathtt{Y}, \mathtt{N}\}) \neq \emptyset\} \end{array} \right)$$

$$\frac{u \xrightarrow{a}_2 v}{(u, Y, w, X) \xrightarrow{a'} (v, Y, w, X')} \quad \left(\begin{array}{l} a' = a \cup (X \times \{\mathtt{N}\}), \\ X' = \{x \mid a \cap (\{x\} \times \{\mathtt{B}, \mathtt{Y}, \mathtt{N}\}) \neq \emptyset\} \end{array} \right)$$

4. The language $\mathcal{L}_1 \wedge \mathcal{L}_2$ is recognized by the finite state automaton

$$(S_1 \times S_2, (s_1, s_2), F_1 \times F_2, \longrightarrow)$$

where the transition relation $\longrightarrow$ is defined by the rule

$$\frac{u_1 \xrightarrow{a_1}_1 v_1, u_2 \xrightarrow{a_2}_2 v_2}{(u_1, u_2) \xrightarrow{a} (v_1, v_2)} \quad (a = a_1 \cup a_2).$$

5. The language $(\mathcal{L}_1)^{\triangle}$ is recognized by the finite state automaton

$$(S_1 \times \wp(V), (s_1, \emptyset), (F_1 \cup \{s_1\}) \times \{\emptyset\}, \longrightarrow)$$

where the transition relation is defined by the rules

$$\frac{s_1 \xrightarrow{a_1}_1 u}{(s_1, \emptyset) \xrightarrow{a} (u, X)} \quad (a = a_1 \cup X \times \{\mathtt{A}\})$$

$$\frac{u \xrightarrow{a}_1 v}{(u, X) \xrightarrow{a} (v, X)}$$

$$\frac{u \xrightarrow{a_1}_1 t_1, s_1 \xrightarrow{a_2}_1 v}{(u, X) \xrightarrow{a} (v, X \setminus Y)} \quad (a = a_1 \cup a_2, Y \subseteq dom(a_2) \subseteq X)$$

Another fact claimed without proof in Lemma 1 is that the language $\Phi(\mathcal{C}_V)$ is regular.

For any finite set of variables V, the language $\Phi(\mathcal{C}_V)$ is recognized by the finite state automaton

$$(2^V \cup \{s\}, s, \{\emptyset\}, \longrightarrow)$$

where the transition relation is defined by

- $s \xrightarrow{a} \emptyset$ iff $a \subseteq V \times \{\mathtt{A}\}$.
- For all $X, Y \subseteq 2^V$, $X \xrightarrow{a} Y$ iff the following conditions hold

$$(x, \mathtt{B}) \in a \;\Rightarrow\; x \in Y$$

$$
\begin{aligned}
(x, \mathtt{E}) \in a &\Rightarrow x \in X \\
(x, \mathtt{Y}) \in a &\Rightarrow x \in X \cap Y \\
(x, \mathtt{N}) \in a &\Rightarrow x \in X \cap Y
\end{aligned}
$$

The proof that the above automata recognize the right languages is a simple exercise and it is left to the reader.