A logical characterisation of availability languages

Bachelorarbeit

vorgelegt am 1. Juni 2011

am Fachbereich Informatik der TU Kaiserslautern

Name:	Susanne Göbel
Email-Adresse:	s_goebel@informatik.uni-kl.de
Fachbereich:	Informatik
Universität:	Technische Universität Kaiserslautern
Studiengang:	Bachelor Informatik
Erstgutachter:	Jun.-Prof. Dr. Roland Meyer
Zweitgutachter:	Prof. Dr. Klaus Madlener

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Zusammenfassung. Die Annahme, dass alle Komponenten fehlerfrei arbeiten und immer verfügbar sind, wird häufig in der Verifikation von Systemen getroffen, obwohl sie erwiesenermaßen falsch ist: Im Internet und in Client-Server-Anwendungen sind Paketverluste und Serverausfälle nur allzu bekannte Phänomene.

Üblicherweise wird die Verfügbarkeit eines Systems als der Quotient aus der Zeit, in der das System korrekt arbeitet, über dem Zeitraum, in dem das System betrachtet wird, definiert [Tri82]. Von diesem Begriff ausgehend, gelang es [HMO10] Verfügbarkeit unter dem Begriff der Verfügbarkeitssprachen (engl. availability languages) im Diskreten zu fassen. Es konnte eine Charakterisierung dieser Sprachen über eine Erweiterung regulärer Ausdrücke (regular availability expressions) und ein gleichmächtiges Automatenmodell angegeben werden.

Diese Arbeit stellt nun auch eine logische Charakterisierung vor. Dazu wird eine Erweiterung von MSO Logik eingeführt, die wir im Folgenden als Verfügbarkeitslogik (availability logic) bezeichnen. Das Hauptresultat der Arbeit besteht in einer effizienten Übersetzungsvorschrift, um Verfügbarkeitsausdrücke in Verfügbarkeitslogik zu übersetzen.

Die Untersuchung der Verfügbarkeitslogik gründet sich zum einen auf der Hoffnung, durch sie Antworten auf offene Entscheidbarkeitsfragen für Verfügbarkeitsausdrücke zu erhalten, zum anderen bietet sie einen Zugang zum Model Checking. Als ersten Schritt hin zu entscheidbaren Unterklassen der Verfügbarkeitslogik erläutern wir Techniken, um die Anzahl der Mengenvariablen erheblich zu reduzieren.

Abstract. Approaches to system verification usually assume that all components work perfectly and are available all the time. However, this is no valid assumption e.g. for most client-server application since packet loss and downtimes are well-known phenomena.

The availability of a system is typically defined as the ratio of correct functioning and the considered time interval [Tri82]. Inspired by this notion a discrete characterisation of availability via availability languages has been introduced by [HMO10] to include the aspect of availability in verification. The authors gave a characterisation via regular availability expressions and a corresponding automata model.

This thesis introduces a characterisation of availability languages via availability logic, an extension of MSO logic. Our main result is an effective translation algorithm from regular availability expressions into availability logic.

The study of availability logic is motivated by open decidability problems for availability languages that we hope to settle by a logical account. A second application can be seen in model checking. We present techniques to reduce the number of second-order quantifiers significantly as a first step towards decidable fragments.

Contents

1 Introduction

Today the internet is used for almost all kinds of applications. We chat and mail, do online banking and buy things in online shops. We expect all of these applications to guarantee a high reliability and a quick execution of the necessary steps. No one would use amazon if half of the purchases were dropped and no one would rely on google if they would usually need half an hour to answer a request.

Therefore, the reliability of the applications is a crucial issue. This reliability does not only depend on the work within the application but also on the reliability of the communication towards the server which performs the necessary actions and even on the availability of the server itself. A formal characterisation is necessary to take the system's availability into account when we investigate certain properties.

Availability is usually defined as the ratio of correct functioning and the considered time interval in a continuous setting[Tri82].

[HMO10] presents that availability can already be studied in the easier discrete setting of words, since availability can be expressed as the ratio of letters encoding "good" functioning to the number of letters at all.

The authors introduce the model of availability languages which allows to check specified availabilities on nearly arbitrary subparts of a word. Availability languages are characterised via an extension of regular expressions with cardinality constraints called regular availability expressions and via the corresponding availability automata.

In opposition to weighted automata [DKV09] and Presburger regular expressions [SSMH04] the measurements of the cardinality constraints in availability expressions affect the further acceptance of the word. As pointed out in [HMO10] their model also allows an unbound number of check positions whereas Presburger regular expressions only have a finite number of arithmetic constraints.

Reliability properties are usually investigated in the context of **model checking**. The model checking problem is defined as the question whether a system $\mathcal{A}$ satisfies a specification φ. Availability languages can either be used as a specification e.g. if we want to guarantee that a server is available at least 50% of its life or they can be included in the system description if we want to prove reliability properties like "each request is answered within 5 seconds" for a system like the google servers for which we know that the availability of each server is smaller than 100%.

Usually the system is given in the form of an automaton while the specification is given as a formula. Therefore a logical characterisation of availability languages is necessary to extend the model checking approach to availability languages.

Furthermore, there are open decidability problems like the emptyness problem for regular availability expressions. We hope that those problems can be settled by the help of a logical account.

This thesis establishes a logical characterisation of availability languages based on an extension of MSO logic which we call availability logic.

We recapitulate basic terms and results for formal languages and mathematical logic with a special stress on regular expressions and MSO logic on words in chapter 2. In chapter 3 regular availability expressions are investigated closer. We present definitions and properties to capture the discovered peculiarities. These notions ease the understanding of the construction in chapter 4. This chapter introduces our main result, an effective translation algorithm from regular availability expressions into availability logic. We then reduce in chapter 5 the number of second-order quantifiers in the constructed formulas using a trick which was already proposed in [EF95, page 112] and some new modifications dealing with the extensions to MSO.

As shown by Büchi in 1960 MSO logic has the same expressiveness regular expressions have [Lib04, page 124] and is thus decidable.

Decidability and undecidability results have been proven for different extensions of MSO. Weighted logic [DG07] corresponding to weighted automata has a decidable language equivalence problem. The extension of MSO with cardinality constraints on arbitrary second-order sets [KR03] has decidable and undecidable fragments depending on the number and kind of the second-order quantifiers. The decidability of their Σ_1^1 fragment of MSO with cardinalities has motivated our transformations in chapter 5.

The study of availability logic can also be seen as a first step towards a study of different extensions of MSO logic with cardinality constraints.

2 Preliminaries

We shortly recapitulate formal languages and regular expressions in section 2.1 and present some basic properties of regular expressions. Section 2.2 on page 11 introduces the basic terms of mathematical logic which are necessary to understand the definition of MSO logic given in section 2.3.

2.1 Words and languages

An **alphabet** is a non-empty finite set of symbols, which we usually denote by Σ. The elements of an alphabet are called **letters** and finite sequences of the letters are called **words**. The sequences of letters are written in one line without withespaces, just like words in natural languages. The empty sequence without any letters is called **empty word** and denoted by ε.

By using the phrase "words over an alphabet" we indicate that the word may only contain letters which are part of the specified alphabet. However, not all of these symbols have to be used in the word. An extreme example is the empty word ε, which can be defined as a word over any alphabet.

It is important to distinguish between words in a natural language and words in the way we define them. Consider the alphabet $\Sigma = \{n, o\}$, over which we may form the words *on* and *no*, but also words like *nnn* or *onnnno*.

There are some functions defined on words which we shall use later on. The **length of a word** w is the number of letters it contains. A word's length is denoted by $|w|$.

Let $w = a_1 \dots a_n$ be a word with letters a_1 to a_n, then

$$|w| = |a_1 \dots a_n| = n \quad .$$

Over $\Sigma = \{a, b\}$ we have $|\varepsilon| = 0$ and $|ab| = 2$.

Let w_1 be a word over Σ_1, w_2 a word over Σ_2. The **concatenation** of w_1 and w_2 is the sequence in which the letters of w_1 are followed by the letters of w_2. The concatenation is denoted by $w_1.w_2$ and is a word over the alphabet $\Sigma_1 \cup \Sigma_2$.

$$w_1.w_2 = w_1 w_2 \quad .$$

The concatenation of $w_1 = aba$ and $w_2 = aba$ is the word *abaaba*. A concatenation of a word w with ε is w again.

Let Σ be an alphabet, $A \subseteq \Sigma$ a subalphabet, and w a word over Σ. The **projection** of w onto the alphabet A is written $\pi_A(w)$ and defined as the word w', which is formed by concatenating all letters in w that are part of A in the order of their appearance in w.

Consider $\Sigma = \{a, b\}$ with the subalphabet $A = \{a\}$. For the word $w_1 = ababbba$, $\pi_{\{a\}}(w_1) = aaa$, while for $w_2 = bbb$, the projection $\pi_{\{a\}}(bbb)$ is ε.

A **language** $\mathcal{L}$ over Σ is defined as a set of words over a given alphabet Σ.

We characterise languages in two ways, by means of regular expressions and logical formulas.

To this end, we first introduce operations on languages. We need them to understand the semantics of regular expressions.

Let $\mathcal{L}_1, \mathcal{L}_2$ be languages. Their **concatenation** $\mathcal{L}_1.\mathcal{L}_2$ is the language consisting of all the concatenations of words from $\mathcal{L}_1$ with words from $\mathcal{L}_2$.

$$\mathcal{L}_1.\mathcal{L}_2 = \{u.v \quad | u \in \mathcal{L}_1, v \in \mathcal{L}_2\} \quad .$$

The languages $\mathcal{L}_1 = \{ab, aba\}$ and $\mathcal{L}_2 = \{c, cd\}$ form two different concatenations, depending on the order in which they are combined. While $\mathcal{L}_1.\mathcal{L}_2 = \{abc, abcd, abac, abacd\}$, $\mathcal{L}_2.\mathcal{L}_1 = \{cab, caba, cdab, cdaba\}$. We shall often omit the dot. Furthermore there are abbreviations to indicate the concatenation of a language with itself.

The multiple concatenation of a language $\mathcal{L}$ is abbreviated $\mathcal{L}^i$, where $i \in \mathbb{N}$ denotes the number of concatenations of $\mathcal{L}$ with itself. $\mathcal{L}^0$ encodes $\{\varepsilon\}$, all other concatenations are defined inductively:

$$\mathcal{L}^i = \mathcal{L}\mathcal{L}^{i-1} \quad .$$

Let $\mathcal{L}$ be a language. The **Kleene closure** or **Kleene star** of $\mathcal{L}$, $\mathcal{L}^*$, is the language which contains $\varepsilon, \mathcal{L}$, and any finite number of concatenations of $\mathcal{L}$ with itself.

$$\mathcal{L}^* = \bigcup_{i=0}^{\infty} \mathcal{L}^i \quad .$$

The **Kleene plus** $\mathcal{L}^+$ excludes $\mathcal{L}^0$:

$$\mathcal{L}^+ = \bigcup_{i=1}^{\infty} \mathcal{L}^i = \mathcal{L}\mathcal{L}^*$$

Note that $\mathcal{L}^+$ contains ε iff $\mathcal{L}$ does.

The operators * and $^+$ usually produce an infinite set of words. Only when applied to $\{\varepsilon\}$, they have no effect. The language that * produces on $\mathcal{L}_1 = \{a\}$ is $\mathcal{L}_1^* = \{\varepsilon, a, aa, aaa, \dots\}$.

Regular expressions offer an efficient way to characterise languages by describing most common matches on words. We define regular expressions in an inductive way, following [HU79, page 28].

Definition 1 (Syntax of regular expressions)**.** *Let Σ be an alphabet.*

- *$\varnothing$ is a regular expression.*
- *a with $a \in \Sigma$ is a regular expression.*
- *ϵ is one, too.*

For regular expressions ρ, τ, the following compound expressions are regular as well:

- $\rho + \tau$,

- $\rho.\tau$,
- *and* ρ^*.

We will sometimes call compound regular expressions **recursive regular expressions** to indicate that their evaluation requires recursive calls. All other regular expressions are then referred to as **non-recursive**. In order to reduce the number of necessary parentheses we give the operators $+$, ., and * a precedence. The Kleene operator * has a higher precedence than the concatenation which has a higher precedence than the choice $+$.

Definition 2 (Semantics of regular expressions)**.** *Let* Σ *be an alphabet, and* ρ, τ *be regular expressions.*

- $\mathcal{L}(\varnothing) = \varnothing$
- *For* $a \in \Sigma$, $\mathcal{L}(a) = \{a\}$.
- $\mathcal{L}(\epsilon) = \{\varepsilon\}$.
- $\mathcal{L}(\rho + \tau) = \mathcal{L}(\rho) \cup \mathcal{L}(\tau)$.
- $\mathcal{L}(\rho.\tau) = \mathcal{L}(\rho).\mathcal{L}(\tau)$.
- $\mathcal{L}(\rho^*) = (\mathcal{L}(\rho))^*$.

We say that a word w **matches** a regular expression ρ if it is $w \in \mathcal{L}(\rho)$. A language denoted by a regular expression τ is called a **regular language**, written as $\mathcal{L}(\tau)$. Two regular expressions ρ, τ are called **equivalent**, denoted by $\rho \equiv \tau$, iff $\mathcal{L}(\rho) = \mathcal{L}(\tau)$.

We will sometimes use the expression ρ^+ for a regular expression ρ. This is an abbreviation for $\rho.\rho^*$ which captures the Kleene plus.

The expression a^*b^* characterises the language $\mathcal{L}(a^*b^*)$, consisting of all words over $\{a, b\}$, in which bs are only placed after as, while all the as may only be placed before the bs and there may also be no as or no bs at all. Words in this language are for example ε, *aaaaaa*, *b*, *aaabbb* and *abbbbb*. The language $\mathcal{L}(a^+b^+)$ also consists of words in which bs follow as, but excludes the case of no bs or no as. Words are *aaabbb*, *abbbbb* or *ab*, while ε, *b* or *aaaaaa* are excluded.

We introduce some properties of regular expressions which are helpful to simplify a given expression. We will derive similar properties for regular availability expressions in chapter 3 on page 23.

Theorem 1 (Properties of regular expressions)**.** *Let* ρ, τ, *and* σ *encode regular expressions.*

Associative property *Expressions solely involving union or concatenation are invariant with respect to the order of operations:*

- $(\rho + \tau) + \sigma \equiv \rho + (\tau + \sigma)$

- $(\rho.\tau).\sigma \equiv \rho.(\tau.\sigma)$

Commutative property *The order in which two regular expressions are united does not matter:*

- $\rho + \tau \equiv \tau + \rho$
- *In general:* $\rho.\tau \not\equiv \tau.\rho$

Distributive property *The concatenation with a union may be performed as two single concatenations with the union parts:*

- $(\rho + \tau).\sigma \equiv \rho.\sigma + \tau.\sigma$
- $\rho.(\tau + \sigma) \equiv \rho.\tau + \rho.\sigma$

Zero element *An element ν is called zero element under a unary operation $^\circ$ iff $\nu^\circ = \nu$. An element ν is called zero element under a binary operation $^\circ$ if for any element ξ the equality $\nu^\circ\xi = \nu = \xi^\circ\nu$ holds.*

The expression $\varnothing$ is zero under concatenation and Kleene's $^+$, ε is zero under Kleene's * and Kleene's $^+$:

- $\varnothing.\rho \equiv \varnothing \equiv \rho.\varnothing$
- $\varnothing^+ \equiv \varnothing$
- *But:* $\varnothing^* \equiv \varepsilon \not\equiv \varnothing$
- $\varepsilon^* \equiv \varepsilon$
- $\varepsilon^+ \equiv \varepsilon$

Neutral element *An element μ is called neutral element under a binary operation $^\circ$ iff for any element ξ the equality $\mu^\circ\xi = \xi = \xi^\circ\mu$ holds. The expression $\varnothing$ is neutral under union, ε is neutral under concatenation:*

- $\varnothing + \rho \equiv \rho \equiv \rho + \varnothing$
- $\varepsilon.\rho \equiv \rho \equiv \rho.\varepsilon$

Proof. We prove the equivalences via the languages where we exploit the commutative and associative property of the set union.

- $(\rho + \tau) + \sigma \equiv \rho + (\tau + \sigma)$:
  $\mathcal{L}((\rho + \tau) + \sigma) = (\mathcal{L}(\rho) \cup \mathcal{L}(\tau)) \cup \mathcal{L}(\sigma) = \mathcal{L}(\rho) \cup (\mathcal{L}(\tau) \cup \mathcal{L}(\sigma)) = \mathcal{L}(\rho + (\tau + \sigma))$.

- $(\rho.\tau).\sigma \equiv \rho.(\tau.\sigma)$:
  $\mathcal{L}((\rho.\tau).\sigma) = \mathcal{L}(\rho.\tau).\mathcal{L}(\sigma) = (\mathcal{L}(\rho).\mathcal{L}(\tau)).\mathcal{L}(\sigma) = \{u.v.w | (u.v) \in (\mathcal{L}(\rho).\mathcal{L}(\tau)), w \in \mathcal{L}(\sigma)\} = \{u.v.w | u \in \mathcal{L}(\rho), (v.w) \in (\mathcal{L}(\tau).\mathcal{L}(\sigma))\} = \mathcal{L}(\rho).(\mathcal{L}(\tau).\mathcal{L}(\sigma)) = \mathcal{L}(\rho.(\tau.\sigma))$.

- $\rho + \tau \equiv \tau + \rho$:
  This follows immediately from the commutative property of the set union.

- In general: $\rho.\tau \not\equiv \tau.\rho$:
  This was already seen in chapter 2.1 on page 8 with $\mathcal{L}(ab + aba).\mathcal{L}(c + cd) \neq \mathcal{L}(c + cd).\mathcal{L}(ab + aba)$.

- $\varnothing.\rho \equiv \varnothing \equiv \rho.\varnothing$:
  Since there is now word v in $\mathcal{L}(\varnothing)$, there cannot be a word $v.w$ with $v \in \mathcal{L}(\varnothing), w \in \mathcal{L}(\rho)$, so $\mathcal{L}(\varnothing.\rho) = \mathcal{L}(\varnothing)$. The opposite direction can be seen with analogue arguments starting with $\mathcal{L}(\rho.\varnothing)$.

- $\varnothing^+ \equiv \varnothing$:
  $\mathcal{L}(\varnothing^+) = \mathcal{L}(\varnothing.\varnothing^*) = \mathcal{L}(\varnothing).\mathcal{L}(\varnothing^*)$ and by the previous item this is $\mathcal{L}(\varnothing)$.

- But: $\varnothing^* \equiv \varepsilon \not\equiv \varnothing$:
  $\mathcal{L}(\varnothing^*) = \{\varepsilon\} \cup \mathcal{L}(\varnothing^+) = \{\varepsilon\} \cup \varnothing = \{\varepsilon\}$.

- $\varepsilon^* \equiv \varepsilon$:
  Since $\mathcal{L}(\varepsilon).\mathcal{L}(\varepsilon) = \mathcal{L}(\varepsilon)$ any multiple concatenation is ε again. Furthermore, $\mathcal{L}(\varepsilon) = \{\varepsilon\}$ and $\mathcal{L}(\varepsilon)^0 = \{\varepsilon\}$. So, $\mathcal{L}(\varepsilon^*) = \{\varepsilon\} \cup \{\varepsilon\} \cup \{\varepsilon\} = \{\varepsilon\}$.

- $\varepsilon^+ \equiv \varepsilon$:
  $\mathcal{L}(\varepsilon^+) = \mathcal{L}(\varepsilon).\mathcal{L}(\varepsilon^*) = \mathcal{L}(\varepsilon).\mathcal{L}(\varepsilon) = \mathcal{L}(\varepsilon)$.

- $\varnothing + \rho \equiv \rho \equiv \rho + \varnothing$:
  $\mathcal{L}(\varnothing + \rho) = \mathcal{L}(\varnothing) \cup \mathcal{L}(\rho) = \varnothing \cup \mathcal{L}(\rho) = \mathcal{L}(\rho)$. Furthermore, $\mathcal{L}(\rho) = \mathcal{L}(\rho) \cup \varnothing = \mathcal{L}(\rho + \varnothing)$.

- $\varepsilon.\rho \equiv \rho \equiv \rho.\varepsilon$:
  $\mathcal{L}(\varepsilon.\rho) = \mathcal{L}(\varepsilon).\mathcal{L}(\rho) = \{u.v | u \in \mathcal{L}(\varepsilon), v \in \mathcal{L}(\rho)\} = \{\varepsilon.v | v \in \mathcal{L}(\rho)\} = \{v | v \in \mathcal{L}(\rho)\}$.
  The opposite direction can be seen with analogue arguments starting with $\mathcal{L}(\rho.\varepsilon)$.

□

There are languages which are not expressible via regular expressions. For example the language $\{a^n b^n | n \in \mathbb{N}\}$ is not regular. A proof is given in [Sch08, page 33] and in example 5.3.3 in [EF95, page 114] via a pumping argument. It can be seen with similar arguments that also **matching parentheses** $\{w \in \mathcal{L}(b + e)^* \mid$ the number of bs equals the number of es and for each prefix of w the number of bs is larger or equal to the number of es$\}$ and $\{a^n b^n c^n | n \in \mathbb{N}\}$ are not regular.

2.2 Mathematical logic

Mathematical logic has very different forms. We study binary logic which is based on the two boolean truth values true and false. Formulas are assigned a truth value depending on the interpretation of the used symbols. We follow [Lib04, page 13 and following] but concentrate on a subset of logical formulas, consisting of predicate symbols, function symbols, logical connectives and quantifiers.

While the evaluation of quantifiers and connectives is fixed by rules, the interpretation of a predicate or a function symbol is usually definable and is documented as the interpretation of the symbol. **Logical connectives** connect formulas, we use $\neg, \wedge, \vee$ and $\rightarrow$ as symbols. To avoid a lot of parenthesis, we define $\neg$ to have the highest priority, with the other symbols following in the order $\wedge, \vee$, and as last $\rightarrow$.

From the viewpoint of the evaluation function e, connectives use boolean values as input and are assigned a boolean value as output. Let φ_A and φ_B be formulas such that their evaluation encodes a boolean value $\in$ {true, false}.

$$\begin{array}{lcl} e(\neg\varphi_A) & = & \textbf{if } e(\varphi_A) \textbf{ then } \text{false } \textbf{else } \text{true} \\ e(\varphi_A \wedge \varphi_B) & = & \textbf{if } e(\varphi_A) \textbf{ then } e(\varphi_B) \textbf{ else } \text{false} \\ e(\varphi_A \vee \varphi_B) & = & \textbf{if } e(\varphi_A) \textbf{ then } \text{true } \textbf{else } e(\varphi_B) \\ e(\varphi_A \rightarrow \varphi_B) & = & \textbf{if } e(\varphi_A) \textbf{ then } e(\varphi_B) \textbf{ else } \text{true} \end{array}$$

We can now already evaluate simple formulas. The evaluation of $\varphi \vee \neg\varphi$ is true, independent of the truth value of φ, while $e(\varphi \wedge \neg\varphi)$ is always false. If we let φ_1 encode true and φ_2 encode false, we can evaluate $\varphi_0 = \varphi_1 \rightarrow ((\varphi_2 \wedge (\varphi_1 \rightarrow \varphi_1)) \vee (\varphi_1 \vee \varphi_2))$ from the outside to the inside as

$$\begin{array}{lcl} e(\varphi_0) & = & \textbf{if } e(\varphi_1) \textbf{ then } e((\varphi_2 \wedge (\varphi_1 \rightarrow \varphi_1)) \vee (\varphi_1 \vee \varphi_2)) \textbf{ else } \text{true} \\ & = & e((\varphi_2 \wedge (\varphi_1 \rightarrow \varphi_1)) \vee (\varphi_1 \vee \varphi_2)) \\ & = & \textbf{if } e(\varphi_2 \wedge (\varphi_1 \rightarrow \varphi_1)) \textbf{ then } \text{true } \textbf{else } e(\varphi_1 \vee \varphi_2) \\ & = & e(\varphi_1 \vee \varphi_2) \\ & = & \text{true} \quad . \end{array}$$

The smallest formulas are the **predicate symbols**. Predicate symbols can contain function symbols as parameters for the evaluation of the predicate towards a boolean value. This value depends on the interpretation of the predicate itself and on its input data if such data exists. When evaluating a predicate, we first evaluate its input function symbols to receive input data we then evaluate the predicate itself under. The predicate interpretation can be given in the form of a rule like for example $even(x)$ iff x is a natural number with $x \equiv 0 \bmod 2$, or as the set of input data satisfying the predicate, which would be $even = 2\mathbb{N}$. The **universe** or **domain** of an interpretation delimits the interpretation of the function symbols. When interpreting function symbols, we distinguish between **first-order symbols** abbreviated as FO symbols or and **monadic second order symbols** abbreviated as MSO symbols. The interpretation of a FO symbol is an element of the domain, while the interpretation of an MSO symbol is a set of elements of the domain.

We define a **structure** which is also often called a **model** following [Lib04, page 13].

Definition 3 (Structure)**.** *Let φ be a formula, $\mathbb{P}$ the set of predicate symbols used in φ and $\mathbb{F}$ a set containing all function symbols used. Let $\mathcal{A}$ be a domain, $i(\mathbb{P})$ be a set of interpretations for each element of $\mathbb{P}$, $i(\mathbb{F})$ be a set of interpretations which respect $\mathcal{A}$ for each element of $\mathbb{F}$. A structure $\mathfrak{A}$ is written as*

$$\mathfrak{A} =< \mathcal{A}, i(\mathbb{P}), i(\mathbb{F}) > \quad .$$

Please note that our definition allows to interpret more function symbols than actually occur in the formula. Any unnecessary interpretation is ignored when evaluating the formula. We will exploit this fact in the proof of Lemma 4 on page 32.

Consider the formula $\varphi = p(x)$. Besides the domain there is the predicate p and the function x for which we both have to name an interpretation to define a structure. We will use the natural numbers as a first domain. When we want to use p as true, if its input is an odd number, we can make φ true or false by the interpretation we choose for x: If we choose for example 3 as interpretation of x, then the formula evaluates to true, while, if we choose 4, it will be interpreted as false.

By changing the domain to $2\mathbb{N}$ but keeping the interpretation of p as it was before, we have no longer an option to satisfy φ by any interpretation of x we may choose. As x is fixed to be a part of the domain, it has to be an even number and by that cannot satisfy our predicate p.

A structure is an important element when determining a formula's truth value. Combined with the evaluation rules for connectives and quantifiers it will allow us to uniquely calculate the truth value for any formula. As the evaluation rules will be fixed, the structure will remain the only variable part. To stress the structure's impact on the formula's truth value, we will usually state that a formula is true or false under a given structure. To complete our definitions, we now define the evaluation for the quantifiers we use.

Logical quantifiers are expressed over a function symbol. This symbol is then called a **quantified** or **bound variable** and will not be interpreted. Every occurrence of the variable within the scope of the quantifier is also called bound. The scope of a quantifier can be defined by the parentheses $\{,\}$. This definition, but also the implicitly assumed scope, must respect the parentheses of the formula. Whenever a new quantifier for the same variable occurs within the formula, the scope of the old quantifier is intercepted for the scope of the new one. Variables which are not bound are called **free**.

We use the **existential quantifier** $\exists$ and the **universal quantifier** $\forall$. Their evaluation e under a given domain $\mathcal{A}$ depends on the interpretation of the formula φ over which the scope of the quantifier ranges.

$e(\exists x\, \varphi)$ = true, iff there is at least one interpretation $i(x)$ in $\mathcal{A}$ such that φ can be evaluated to true with the restriction that any occurrence of x which is free in φ is interpreted as $i(x)$.

$e(\forall x\, \varphi(x))$ = true, iff for all interpretations $i(x)$ in $\mathcal{A}$ φ can be evaluated to true with the restriction that any occurrence of x which is free in φ is interpreted as $i(x)$.

This illustrates why we cannot interpret bound variables: their values are controlled by the quantifier.

Let x and y be FO symbols, and *uneq* be a predicate which expects two FO symbols as parameters. We analyse the formula $\varphi = \forall x \forall y\, uneq(x, y)$ under the fixation that *uneq* expresses, wether two elements are unequal. As the predicate is fixed and not interpretable and there do not occur any free variables, we do not have to choose any predicate or function symbol interpretations. It is sufficient to fix the domain $\mathcal{A}$ and use $\mathfrak{A} =< \mathcal{A}, \varnothing, \varnothing >$ as a structure.

We use $\mathcal{A}_1 = \{1, 2\}$ as a first domain and determine φ's truth value. We have to evaluate the formula under any value of the domain and choose 1 first. The variable x

is interpreted as 1 in the remaining formula $\forall y\, uneq(x, y)$ so we evaluate $\forall y\, uneq(1, y)$ further. We test $y = 2$ first and receive true, as $1 \neq 2$, but when evaluating y as 1, the formula is false. We can stop our evaluation and determine the truth value of φ as false under this domain. In fact, φ is false under any domain but the empty set $\varnothing$, as y takes every domain value and therefore also uses the same value, x just has for any choice of x. Only if there is no element in the domain, then φ is true, as it states a property for all x, which always holds if there is no x.

This does not work any more if we consider the formula $\varphi_1 = \exists x \forall y\, uneq(x, y)$. It is not satisfiable as $\exists x$ is only true if there is at least one element in the domain. $\forall y$ yields that we will compare the element with itself and therefore receive false.

On the other hand $\varphi_3 = \forall x \exists y\, uneq(x, y)$ is true for any domain with two or more elements, as we can choose any element but the value behind x as value of y. Also, φ_3 is true under the domain $\varnothing$, as it starts with $\forall x$ again.

A **substitution** is a syntactical replacement of one function symbol by another one. The substitution is applied on every free occurrence of the symbol while any bound occurrence mustn't be replaced. In $\varphi = x < y \wedge \exists x x > y$ the function symbol y only occurs freely. The substitution of y by z transforms φ into $\varphi' = x < z \wedge \exists x\, x > z$. An analogue substitution of x by z does only have an effect on the first occurrence of x. The constructed formula φ'' is $z < y \wedge \exists x\, x > y$.

We have seen the impact of quantification on formulas and on the degree of freedom for our interpretation. These observations are reflected in the following definition.

Definition 4 (Sentences)**.** *A formula, in which no variable occurs freely is called a sentence. A structure $\mathfrak{A}$ for a sentence φ can always omit the interpretation of functions. Let $\mathbb{P}$ be the predicates occuring in φ.*

$$\mathfrak{A} =< \mathcal{A}, i(\mathbb{P}), \varnothing > \quad .$$

2.3 FO and MSO logic in word domains

The introduction of FO and MSO logic is based on [Tho97].

A finite word $w = a_0, a_1, \ldots, a_{n-1}$ over an alphabet Σ can be interpreted as a mapping of positions $\{0, 1, \ldots, n-1\}$ to letters of the alphabet Σ

$$w : \{0, 1, \ldots, n-1\} \to \Sigma,$$

where a position x maps to $a \in \Sigma$ ("$x \mapsto a$") iff the letter at position x is a.
The set of positions in the word is called the word's **word domain** $dom(w)$.

For example, the word *abbca* implies the mapping $(0 \mapsto a, 1 \mapsto b, 2 \mapsto b, 3 \mapsto c, 4 \mapsto a)$ with $dom(abbca) = \{0, 1, 2, 3, 4\}$.

Definition 5 (Syntax of FO logic)**.** *An FO formula for words over an alphabet Σ consists of:*

- *function symbols $(x, y, \ldots)$ as variables for positions*

- *logical connectives* $\neg, \wedge, \vee$ *and* $\rightarrow$
- *quantifiers* $\forall$ *and* $\exists$
- *predicates* $S(x,y), x < y$ *and* $Q_a(x)$ *with* $a \in \Sigma$

Further predicates may be introduced if they are abbreviations for FO formulas.

To fill this syntactical definition with semantics we first introduce the word model, which defines the interpretation of the domain and the predicates with respect to a given word.

Definition 6 (Word models)**.** *Let w be a word, x and y positions in w. The word model $\underline{w}$ of w is defined as $(dom(w), S^w, <^w, (Q_a^w)_{a\in\Sigma})$, where the predicates are fixed to the following interpretation:*

$S^w(x,y)$	*iff*	*position x is directly followed by position y in w*
$x <^w y$	*iff*	*position x is followed (but not necessarily directly followed) by position y in w*
$Q_a^w(x)$	*iff*	*the letter at position x is a*

(The extension ^{w} indicates, that the predicate is evaluated according to w.)

We complete our interpretation of FO formulas by fixing the interpretation of free variables.

Definition 7 (Satisfaction of FO formulas)**.** *Let Σ be an alphabet, and w a word over Σ with $p_1, \ldots, p_n \in dom(w)$ positions in w. Let $\varphi(x_1, \ldots, x_n)$ be a FO formula, in which x_1 to x_n occur freely. We say that $\underline{w}$ satisfies φ*

$$(\underline{w}, p_1, \ldots, p_n) \vDash \varphi(x_1, \ldots, x_n),$$

iff $\varphi(x_1, \ldots, x_n)$ is true if it is interpreted with the domain and the predicate interpretation $\underline{w}$ implies and p_1 to p_n serve as interpretation for the free variables x_1 to x_n.

We also state that a formula φ accepts a word w to express $(\underline{w}, p_1, \ldots, p_n) \vDash \varphi(x_1, \ldots, x_n)$ from the viewpoint of the formula. If there are several free variables the match of a variable x with its interpretation p will be explicitly given to avoid ambiguities.

Please note that in analogy to the example in 2.2 on the previous page any formula starting with an existentially quantified position variable is false under ε, since this formula states that a position x exists. In contradiction a universally quantified formula is true, since anything is true for all x if no x exists.

Observation 1 ($\underline{\varepsilon}$ as a model)**.** *Let $\varphi(x)$ be a formula in which at most x occurs freely.*

$(\underline{\varepsilon}) \quad \nvDash \quad \exists x \varphi(x)$
$(\underline{\varepsilon}) \quad \vDash \quad \forall x \varphi(x) \quad .$

We consider the alphabet $\{a, b\}$. The FO formula $\exists x Q_a(x)$ accepts all words, in which at least one letter a exists. These words can be summarised as $\mathcal{L}(\Sigma^* a \Sigma^*)$. We call languages which are definable via FO sentences FO definable languages.

Definition 8 (FO defined language)**.** *The language over an alphabet Σ defined by an FO sentence φ is called $\mathcal{L}(\varphi)$ and is defined as $\{w \in \Sigma^* | (\underline{w}) \vDash \varphi\}$. On the other hand, a language $\mathcal{L}$ is called FO definable iff there is an FO sentence φ, so that $\mathcal{L} = \mathcal{L}(\varphi)$.*

By extending FO logic with the ability to formulate sets of positions and quantify over them, we derive the more powerful monadic second order (short: MSO) logic.

Definition 9 (Syntax of MSO logic)**.** *An MSO formula for words over an alphabet Σ consists of:*

- *FO function symbols $(x, y, \dots)$ as variables for positions*
- *MSO function symbols $(X, Y, \dots)$ as variables for sets of positions*
- *logical connectives $\neg, \wedge, \vee$ and $\rightarrow$*
- *quantifiers $\forall$ and $\exists$*
- *predicates $S(x, y), x < y$, $Q_a(x)$ with $a \in \Sigma$ and $X(x)$*

Further predicates may be introduced if they are abbreviations for MSO formulas.

The newly introduced predicate $X(x)$ is true iff the set of positions X contains the position x.

Please note that this fixation removes $X(x)$ from the set of predicates for which we may define an interpretation. This implies that no changes in the word model will be necessary to adjust it to MSO. Furthermore, the definition of satisfaction of MSO formulas is the same as the one for FO as regards content.

Definition 10 (Satisfaction of MSO formulas)**.** *Let Σ be an alphabet, and w a word over Σ with $p_1, \dots, p_n \in dom(w)$ positions in w. Let $\varphi(x_1, \dots, x_n)$ be an MSO formula the FO and MSO formulas x_1 to x_n occur freely in. We say that φ is satisfied in $\underline{w}$*

$$(\underline{w}, p_1, \dots, p_n) \vDash \varphi(x_1, \dots, x_n),$$

iff $\varphi(x_1, \dots, x_n)$ is true if it is interpreted with the domain and the predicate interpretation $\underline{w}$ implies and p_1 to p_n serve as interpretation for the free variables x_1 to x_n.

Like already discussed for FO logic we avoid ambiguities in the match of free variables and interpretations by explicitly giving the corresponding interpretation to each free variable as soon as there are at least two.

While we have been able to determine the truth value of any formula starting with an FO quantifier under $\underline{\varepsilon}$, we cannot make equivalent statements for second-order quantification. As the empty set is a set as well, existential quantifiers and universal quantifiers both yield the evaluation of the formula with the former bound variable interpreted as the empty set $\varnothing$.

When reasoning about the decidability of MSO extensions MSO formulas in Σ_1^1 form are of special interest. Our definition follows [EF95, page 39].

Definition 11 (Σ_1^1)**.** *An MSO formula φ is called in Σ_1^1 form if it begins with a block of second-order quantification without interleaved predicates and has a body $\mathcal{B}$ only FO quantification is done in. The body may also contain predicates which range over the second-order variables.*

$$\varphi = \exists X_1 \ldots \exists X_n \, \mathcal{B}$$

In analogy to the definition of languages as all the words matching a given regular expression, we can now define languages as all the words satisfying a closed MSO formula:

Definition 12 (MSO defined language)**.** *The language over an alphabet Σ defined by an MSO sentence φ is called $\mathcal{L}(\varphi)$ and is defined as $\{w \in \Sigma^* | (\underline{w}) \vDash \varphi\}$. On the other hand, a language $\mathcal{L}$ is called MSO definable iff an MSO sentence φ exists, so that $\mathcal{L} = \mathcal{L}(\varphi)$.*

The following important result was discovered by Büchi in 1960 and characterises the expressiveness of MSO. It was the reason why we chose to define availability logic on the basis of MSO.

Theorem 2 (MSO = regular)**.** *A language is definable in MSO iff it is regular.*

Proof. Theorem and proof in [Lib04, page 124]. □

3 Availability expressions

This section establishes a formal and discrete characterisation of availability and introduces a corresponding model. Definition 15 and 16 on the following page, and the examples of expressible not regular languages are taken from [HMO10], where the model was introduced. All other results in this section are original.

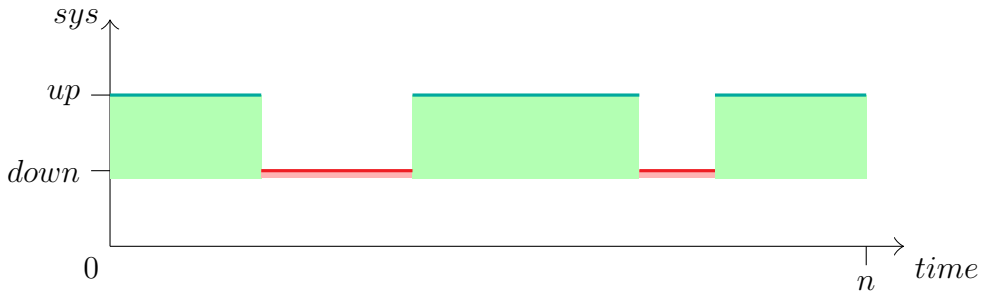
3.1 Defining availability

Imagine you would measure from time to time the functioning of a system and always consider the new measurement as the value the system had since your last measurement point. Each of these measurements is documented by adding a letter encoding the currently measured value to the finite string which encodes all your measurements so far. Once you finish measuring you have produced a finite word w.

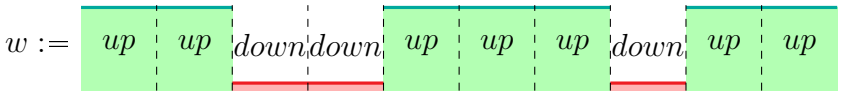
Definition 13 (availability)**.** *Let w be a finite word, in which each symbol $a \in A \subseteq \Sigma$ encodes correct functioning of the system, while each letter $b \in \Sigma \backslash A$ encodes a measurement point in which the system did not work correctly. We express availability as the ratio of letters $a \in A$ in w to the length of w. Hence, the availability of a system is represented as*

$$\frac{|\pi_A(w)|}{|w|} \quad .$$

To illustrate the abstraction, we consider the following finite run of a highly unreliable server:



With $\Sigma := \{up, down\}$ we model w as:



Now we can express the availability during the run as $\frac{|\pi_{\{up\}}(w)|}{|w|} = \frac{7}{10}$.

3.2 Regular availability expressions (rae)

Besides calculating the actual availability of a given system run we can also formulate availability properties of which we would like to express that our system satisfies – or should satisfy – them. Those properties are referred to as availability constraints.

Definition 14 (availability constraints)**.** *Let Σ be an alphabet, w a word over Σ. An availability constraint is formulated as an inequation*

$$|\pi_A(w)| \gtrsim k \cdot |w| \qquad \gtrsim \in \{>, \geqslant\}$$

where A is the part of the alphabet Σ encoding correct functioning of the system and $k \in [0,1]$ is the desired availability.

Please note that also inverse constraints like $|\pi_A(w)| \lesssim k \cdot |w|$ can be modeled by $|\pi_{\Sigma \setminus A}(w)| \gtrsim (1-k) \cdot |w|$, so we can safely use them as abbreviations.

We can use the constraints to restrict the languages accepted by a regular expression further. For example, we could wish a server availability of at least 50%. This can be modeled as $\mathcal{L} = ((up + down)^*.\checkmark)_{\{up\} \geqslant \frac{1}{2}}$, where $(\dots)_{\{up\} \geqslant \frac{1}{2}}$ checks the desired availability $|\pi_{\{up\}}(w)| \gtrsim k \cdot |w|$ at $\checkmark$. The former seen server run is in $\mathcal{L}$, while $downupdown \notin \mathcal{L}$.

Definition 15 (Syntax of raes)**.** *Let Σ be an alphabet with $a \in \Sigma$ and $A \subseteq \Sigma$, $\gtrsim \in \{>, \geqslant\}$ and $k \in [0,1]$. The set of regular availability expressions (rae) over Σ is inductively defined as: $\varnothing$, ε, a and $\checkmark$ are raes. If $\mathcal{A}$ and $\mathcal{B}$ are raes, then $\mathcal{A} + \mathcal{B}$, $\mathcal{A}.\mathcal{B}$, $\mathcal{A}^*$, and $(\mathcal{A})_{A \gtrsim k}$ are raes, too.*

The two newly introduced expressions $\checkmark$ and $(\dots)_{A \gtrsim k}$ are closely related. The constraint $(\dots)_{A \gtrsim k}$ defines by its opening bracket "(" the beginning of the subword, on which we want to check an availability constraint, and specifies by the closing bracket "$)_{A \gtrsim k}$" the availability constraint itself as well as its scope of application. Each check symbol $\checkmark$ within the constraint marks the end of a subword on which we check an availability constraint. If there is no check symbol within a constraint, the specified availability constraint is never checked. Outside a constraint a check symbol is treated like a usual letter, although $\checkmark$ is never allowed to be part of the original alphabet Σ to avoid ambiguities.

Definition 16 (Semantics of raes)**.** *Let Σ be an alphabet so that $\checkmark \notin \Sigma$. It is $\mathcal{L}(rae) \subseteq (\Sigma \cup \{\checkmark\})^*$. The semantics of $\varnothing$, ε, a with $a \in \Sigma$, $+$, . , and * are not changed for raes. We define $\mathcal{L}(\checkmark) = \{\checkmark\}$ and $\mathcal{L}((\rho)_{A \gtrsim k}) = \mathcal{L}_{A \gtrsim k}(\rho)$, where $\mathcal{L}_{A \gtrsim k}(\rho) = \{\pi_\Sigma(w) | w \in \mathcal{L}(\rho)$ and $|\pi_A(w_1)| \gtrsim k \cdot |\pi_\Sigma(w_1)|$ for all $w_1.\checkmark.w_2 = w\}$.*

The option to place a check symbol at an arbitrary position within a rae allows to define availability constraints on subparts of words which are no well-nested subwords but just prefixes. For example, $(a(\checkmark b)^+)_{\{a\} > \frac{1}{2}}$ on the word $w = a\checkmark b\checkmark b$ allows to check the availability constraint on the prefix $w = a\checkmark b\checkmark$, although the nesting is $a((\checkmark b)(\checkmark b))$.

When nesting constraints the innermost constraint is the one to be checked at the included check symbols, any surrounding constraint has to be ignored. This can be seen at the example $\sigma = ((a\checkmark)_{\{a\}\geqslant\frac{1}{2}})_{\{b\}\geqslant\frac{1}{2}}$ for the word $a\checkmark$. We first determine the word $w \in \mathcal{L}((a\checkmark)_{\{a\}\geqslant\frac{1}{2}}$ which $((a\checkmark)_{\{a\}\geqslant\frac{1}{2}}$ accepts if it is given $a\checkmark$. This word is a since it is $\pi_\Sigma(a\checkmark)$. Now we evaluate the outer constraint with $w = a$ and since there is no check symbol, we receive $a \in \mathcal{L}(\sigma)$.

The expressiveness of raes is strictly larger than the expressiveness of regular expressions as some examples of expressible languages which are not regular show:

- Matching parenthesis: $(((b + e\checkmark)^*)_{\{b\}\geqslant\frac{1}{2}}\checkmark)_{\{e\}\geqslant\frac{1}{2}}$

  Whenever we read an e we have to check that so far at least as many brackets were opened as closed. Together with the final check that all brackets were closed we characterise matching parenthesis.

- $\{a^n b^n \mid n \in \mathbb{N}\}$: $((a^*b^*\checkmark)_{\{a\}\geqslant\frac{1}{2}}\checkmark)_{\{b\}\geqslant\frac{1}{2}}$

  Any word matching $(a^*b^*\checkmark)_{\{a\}\geqslant\frac{1}{2}}$ has to consist of as and bs, where as form at least the first half of the word. Combined with the second restriction that at least half of the letters are bs, the only option left is that one half of the letters are as and the other are bs. The expression a^*b^* makes sure that as form the first and bs form the second half, which leaves us with $a^n b^n$ as the only option.

- $\{a^n b^n c^n \mid n \in \mathbb{N}\}$: $(((a^*b^*c^*\checkmark)_{\{a\}\geqslant\frac{1}{3}}\checkmark)_{\{b\}\geqslant\frac{1}{3}}\checkmark)_{\{c\}\geqslant\frac{1}{3}}$

  Extending the former construction we can also express analogously formed context-sensitive languages.

3.3 Properties of raes

This section illustrates strengths and weaknesses of the introduced model. We present a normal form and properties, which can be used to simplify raes.

First, we investigate the semantics given in definition 16 on the previous page closer. The definition of $\mathcal{L}((\rho)_{A\gtrsim k}) = \mathcal{L}_{A\gtrsim k}(\rho)$, where $\mathcal{L}_{A\gtrsim k}(\rho) = \{\pi_\Sigma(w) | w \in \mathcal{L}(\rho)$ and $|\pi_A(w_1)| \gtrsim k \cdot |\pi_\Sigma(w_1)|$ for all $w_1.\checkmark.w_2 = w\}$ introduces some dynamic aspects. As we already illustrated in Section 3.2, there are expressions ρ so that the word w' in $\mathcal{L}(\rho)$ is not the same as the word w for which we asked if it is accepted by ρ. The difference between the words lies in the position of check symbols: In w' some check symbols of w are removed via the constraints.

To capture the words w which are accepted by a rae exactly we introduce the model *abrae* as a helping construct.

Definition 17 (abrae)**.** *The syntax of an abrae is the syntax of a rae, also the semantics are equal except for $\mathcal{L}_{A\gtrsim k}(\rho)$. For an abrae ρ, $\mathcal{L}_{A\gtrsim k}(\rho) = \{w \mid w \in \mathcal{L}(\rho)$ and for each $w_1.\checkmark.w_2$, so that there was no constraint checked on the $\checkmark$ in ρ, $|\pi_A(w_1)| \gtrsim k \cdot |\pi_\Sigma(w_1)|\}$.*

The language of the corresponding abrae to a rae is exactly the set of words which are accepted by the rae.

Definition 18 (Acceptance in rae)**.** *Let ρ be a rae and w a word over $\Sigma \cup \{\checkmark\}$. The word w is accepted by the rae ρ if $w \in \mathcal{L}(\rho)$, where ρ is understood as an abrae.*

In opposition to a rae, an abrae accepts a word without changing it. The model can be imagined like a look-ahead: Before we really change the word by projecting letters away, we first want to know wether the choices we made for the constraints lead to a word in the language of the rae.

For one word which is accepted by a rae there can be several choices which lead to different words in the language of the given rae. We consider the rae $\rho = ((a^*b^*\checkmark)^*)_{\{a\}\geqslant\frac{1}{2}}b^*\checkmark^*$ and the word $aab\checkmark b\checkmark$. We can either match the whole word to the constraint, so that $aab\checkmark$ and $b\checkmark$ are two subwords in the Kleene star or we can match $aab\checkmark$ on the constraint and $b\checkmark$ on the following $b^*\checkmark^*$. In the former case, $w' \in \mathcal{L}(\rho)$ is $aabb$, while in the latter case w' is $aabb\checkmark$.

This shows that there is for a word w accepted by an rae ρ not one deterministically determined corresponding word but there may be several. The difference lies in the way the word w is matched onto the expression ρ or more precisely in the way we match the check symbols. If a check symbol of w is matched onto a check symbol in ρ which is surrounded by a constraint, then this check symbol will be projected away in $w' \in \mathcal{L}(\rho)$, otherwise the check symbol remains in the word.

Definition 19 (active check)**.** *Let ρ a rae. A check symbol $\checkmark$ in ρ is called active, if it is not surrounded by a constraint, otherwise it is called asleep.*

We consider the rae $\rho = (\sigma)_{\{a\}\geqslant\frac{1}{2}}$, where $\sigma = (c\checkmark a^*)_{\{c\}\geqslant\frac{1}{2}}\checkmark$. The last $\checkmark$ in σ is active, since it appears outside the constraint, but if we consider ρ this check is also asleep.

Definition 20 (free check)**.** *A check symbol in a word w, which matches an active check symbol in ρ in one way of ρ to accept w is called free under ρ. The set of all checks which are free under one given way to accept ρ is called checkset $free(w, \rho)$.*

When we apply the former example on the word $w = c\checkmark a\checkmark$ to determine free checks, we see that there is only one way for ρ to accept w. So $free(w, \rho)$ is uniquely defined. It is $free(w, \rho) = \varnothing$, since there are no active checks in ρ. The checkset $free(w, \sigma)$ contains the last letter in w since this letter matches the active last check symbol.

The definition gives us a handle on one concrete way to accept the word. Rather than logging the whole acceptance procedure and with that somehow the derivation we can limit our knowledge to the matches of the active check symbols since they are the only ones which would remain in a word $w' \in \mathcal{L}(\rho)$ produced out of w.

The set $free(w, \rho)$ is uniquely defined for a concrete acceptance and a pair (w, ρ), while there may be several ways to accept w under ρ, which produce the same set of free check positions. If several different sets can be produced for a given pair (w, ρ) the acceptance can lead to different words.

The non-determinism which word of the language is produced when accepting a given word can be avoided if we limit our model onto expressions ρ in which no active check symbols occur. Since there are no active checks in ρ there cannot be free checks in the word w accepted by ρ. Hence the word in the language $\mathcal{L}(\rho)$ contains no checks and is therefore uniquely defined for each accepted word as its projection to Σ:

Fact 1. *Let ρ be a expression in which no active check symbol occurs. The language $\mathcal{L}(\rho)$ is $\{\pi_\Sigma(w) \mid w$ accepted by $\rho\}$.*

It is valid to limit the words in the language of a rae to Σ, since check symbols in a word encode no system behaviour and hence they carry no information which is necessary once we know that the word is accepted by the rae. To remove them in a rae ρ, we can add a constraint, which removes all check symbols, but does not restrict the so far accepted words further. Such a constraint is $(\dots)_{\Sigma\geqslant 0}$, since it yields each projection of a word to Σ to have at least 0 letters from Σ, which is always given.

Definition 21 (Check-closed form)**.** *A rae ω is in check-closed form, iff there are no active checks in ω. Otherwise an expression is called check-open.*

For example, $(a\checkmark)_{\{a\}>0}$ is in check-closed form, since the only used check symbol is surrounded by the constraint $(\dots)_{\|a\}>0}$, while $a\checkmark$ itself is not in check-closed form. Of course, any availability expression, in which no check occurs, – especially every regular expression – is in check-closed form.

Lemma 1 (Closure of the check-closed form)**.** *Let ρ, τ be arbitrary raes. The expression $(\rho)_{A\gtrsim k}$ is always check-closed. The expressions $(\rho)^*$ and $(\rho)^+$ are check-closed iff ρ is check-closed. The expressions $\rho+\tau$ and $\rho.\tau$ are check-closed iff ρ and τ are check-closed.*

Proof. Since the check-constraint surrounds the whole expression, it also surrounds every check symbol. So the first claim is true by the definition of the check-closed form.

We proof the second claim in two implications.

"⇒" : Consider ρ not in check-closed form. This means, that there is at least one check symbol in ρ, which is not guarded by a constraint. Since neither * nor $^+$ introduce a constraint, guarding this check symbol, ρ^* and ρ^+ are also not in check-closed form.

"⇐" : Let ρ be a check-closed expression. Since neither * nor $^+$ introduce a check symbol ρ^* and ρ^+ are also check-closed.

In analogy, the third claim can be shown:

"⇒" : Consider ρ or τ not in check-closed form. This means, that there is at least one check symbol in ρ or τ, which is not guarded by a constraint. Since neither $+$ nor concatenation introduce a constraint, guarding this check symbol, $\rho+\tau$ and $\rho.\tau$ are also not in check-closed form.

"$\Leftarrow$" : Let ρ and τ be check-closed expressions. Since ρ and τ are check-closed, their union or concatenation is also check-closed as none of these operators introduces a new check symbol.

□

There are some general rules to simplify regular availability expressions, which we present as a first theorem.

Theorem 3 (Properties of regular availability expressions)**.** *Let ρ, τ and σ encode regular availability expressions.*

Associative property *Expressions solely involving union or concatenation are invariant with respect to the order of operations:*

- $(\rho + \tau) + \sigma \equiv \rho + (\tau + \sigma)$
- $(\rho.\tau).\sigma \equiv \rho.(\tau.\sigma)$

Commutative property *The order in which two availability expressions are united does not matter:*

- $\rho + \tau \equiv \tau + \rho$
- *In general:* $\rho.\tau \not\equiv \tau.\rho$

Distributive property *The concatenation with a union may be performed as two single concatenations with the union parts, in analogy to the regular case. Also, a constraint over a union is equivalent to the union of two constraints:*

- $(\rho + \tau).\sigma \equiv \rho.\sigma + \tau.\sigma$
- $\rho.(\tau + \sigma) \equiv \rho.\tau + \rho.\sigma$
- $(\rho + \tau)_{A \gtrsim k} \equiv (\rho)_{A \gtrsim k} + (\tau)_{A \gtrsim k}$

Zero element *The expression $\varnothing$ is zero under concatenation and Kleene's $^+$, ε is zero under Kleene's * and Kleene $^+$, even if the used expressions are availability expressions. In addition, $\varnothing$ and ε are both zero under check constraints:*

- $\varnothing.\rho \equiv \varnothing \equiv \rho.\varnothing$
- $\varnothing^+ \equiv \varnothing$
- *But:* $\varnothing^* \equiv \varepsilon \not\equiv \varnothing$
- $\varepsilon^* \equiv \varepsilon$
- $\varepsilon^+ \equiv \varepsilon$
- $(\varnothing)_{A \gtrsim k} \equiv \varnothing$
- $(\varepsilon)_{A \gtrsim k} \equiv \varepsilon$

Neutral element *The expression $\varnothing$ is neutral under union, ε is neutral under concatenation, just like in the regular case.*

- $\varnothing + \rho \equiv \rho \equiv \rho + \varnothing$
- $\varepsilon.\rho \equiv \rho \equiv \rho.\varepsilon$

Proof. Since the semantics of the union, the concatenation, and the Kleene expression is the same as in the regular case, we can adapt the proof of Theorem 1 on page 9. Therefore, we only prove the equations, which use special properties of the availability extensions.

$(\rho + \tau)_{A \gtrsim k} \equiv (\rho)_{A \gtrsim k} + (\tau)_{A \gtrsim k}$:
$\mathcal{L}((\rho + \tau)_{A \gtrsim k}) = \mathcal{L}_{A \gtrsim k}(\rho + \tau) = \{\pi_\Sigma(w) | w \in \mathcal{L}(\rho + \tau) \text{ and } |\pi_A(w_1)| \gtrsim k \cdot |\pi_\Sigma(w_1)| \text{ for all } w_1.\checkmark.w_2 = w\} = \{\pi_\Sigma(w) | w \in \mathcal{L}(\rho) \text{ and } |\pi_A(w_1)| \gtrsim k \cdot |\pi_\Sigma(w_1)| \text{ for all } w_1.\checkmark.w_2 = w\} \cup \{\pi_\Sigma(w) | w \in \mathcal{L}(\tau) \text{ and } |\pi_A(w_1)| \gtrsim k \cdot |\pi_\Sigma(w_1)| \text{ for all } w_1.\checkmark.w_2 = w\} = \mathcal{L}_{A \gtrsim k}(\rho) \cup \mathcal{L}_{A \gtrsim k}(\tau) = \mathcal{L}_{A \gtrsim k}(\rho) + \mathcal{L}_{A \gtrsim k}(\tau)$.

$(\varnothing)_{A \gtrsim k} \equiv \varnothing$:
$\mathcal{L}_{A \gtrsim k}(\varnothing) = \{\pi_\Sigma(w) | w \in \mathcal{L}(\varnothing) \text{ and } |\pi_A(w_1)| \gtrsim k \cdot |\pi_\Sigma(w_1)| \text{ for all } w_1.\checkmark.w_2 = w\}$. Since there is no word w in $\mathcal{L}(\varnothing) = \varnothing$, $\mathcal{L}_{A \gtrsim k}$ is $\varnothing$ as well.

$(\varepsilon)_{A \gtrsim k} \equiv \varepsilon$:
The only word in $\mathcal{L}(\varepsilon)$ is ε. Since no check symbol occurs in ε, the surrounding constraint is never checked.

□

By extending the explanation we already used for $(\varepsilon)_{A \gtrsim k} \equiv \varepsilon$, we derive the following lemma.

Lemma 2 (Check-closed expressions as zero)**.** *A check-closed expression ω is a zero under the constraint:*

$$(\omega)_{A \gtrsim k} \equiv \omega \quad .$$

Especially, we can define a simplification for check-closed expressions over concatenation:

$$(\rho.\omega)_{A \gtrsim k} \equiv (\rho)_{A \gtrsim k}.\omega \quad .$$

Proof. The expression ω is check-free and there is no expression following within the constraint, in which a check symbol occurs. This means, that the constraint information is never used in the first case and not necessary behind ρ in the second case. Also, constraint annotations and their range of application over check-free suffixes do not have any further effect on the acceptance of a word.

On the other hand, the addition of arbitrary constraints – even of those, which are not satisfiable – to a check-closed expression or in the latter case the extension over the check-free successor expression never changes the accepted language, since after analyzing and accepting the subexpressions, there are no check symbols left in the word, on which one had to test the availability constraint. □

4 Logical characterisation of availability languages

In analogy to the characterisation of availability languages by regular availability expressions we introduce a logical characterisation based on an extension of MSO logic which we call **availability logic**. We first present some abbreviations in MSO, then we give a direct translation of regular expressions into MSO. By the help of the newly introduced predicate *comp* to express counting constraints we then translate availability extensions so that we can prove the following theorem.

Theorem 4 ($\mathcal{L}(\rho) = \mathcal{L}(\varphi)$)**.** *For any check-closed availability expression ρ there is an availability sentence φ so that $\mathcal{L}(\varphi) = \mathcal{L}(\rho)$.*

The direct translation of regular expressions into MSO formulas was independently found by [EF95, page 111], who also introduced a transformation into a Σ_1^1 formula. Although we introduce our own construction as it is easier to understand, we relay on their work when we transform our formulas into Σ_1^1 formulas enriched by a slight modification of the *comp* predicate in section 5 on page 41.

4.1 Abbreviations

The formerly defined symbols for MSO logic are capable of expressing it completely [Tho97]. Still, we can ease the understandability of our formulas by defining abbreviations for often expressed predicates. For every newly defined abbreviation we will not only describe the intuitive meaning of the predicate, but also give an MSO formula expressing it. Thus we define the truth value formally and prove the predicate's expressiveness in MSO at once.

Definition 22 ($x = y$)**.**

$$x = y := \neg(x < y) \wedge \neg(y < x)$$

The predicate checks the equality of the positions not just the equality of the letters.

Definition 23 ($x \geqslant y$)**.**

$$x \geqslant y := \neg(x < y)$$

By swapping x and y, $x > y$ and $x \leqslant y$ can be defined analogously.

Definition 24 ($Q_A(x)$)**.** *Let $A \subseteq \Sigma$ with Σ the alphabet the words are formed over.*

$$Q_A(x) := \vee_{a \in A} Q_a(x) \quad .$$

The predicate allows to check whether any letter of A is placed at a given position x. Since A is finite, this predicate can be expressed using a finite number of disjunctions of position variables. This makes it expressible in MSO logic for any given alphabet A.

Definition 25 ($Q_A(X)$)**.** *Let $A \subseteq \Sigma$ with Σ the alphabet the words are formed over.*

$$Q_A(X) := \forall x X(x) \rightarrow Q_A(x) \quad .$$

This predicate allows to check wether in all positions x of X a symbol of A is placed. Relying on $Q_A(x)$ to be an MSO formula, the new formula is in MSO, too. Please note that this formula is true if X is empty.

Definition 26 ($X \subseteq Y$)**.**

$$X \subseteq Y := \forall x X(x) \rightarrow Y(x) \quad .$$

The predicate is true iff X is a subset of Y, while $\mathcal{Z}(X)$ expresses wether a given set X is connected. It states that there is no position y between two positions x_1 and x_2, such that the latter are in X while the intermediate y is not.

Definition 27.

$$\mathcal{Z}(X) := \neg\exists y \neg X(y) \wedge \exists x_1 \exists x_2 X(x_1) \wedge X(x_2) \wedge (x_1 < y) \wedge (y < x_2) \quad .$$

It is often necessary to capture the first or the last position of a set.

Definition 28 ($first_D(x)$)**.** *Let D be a set variable, x a position variable.*

$$\begin{array}{lcl} first_D(x) & := & D(x) \wedge \neg\exists y D(y) \wedge y < x \\ last_D(x) & := & D(x) \wedge \neg\exists y D(y) \wedge y > x \quad . \end{array}$$

4.2 Encoding subdomains

In an inductive approach it is important to delimit a subproblem to the necessary subpart of the input data. Our input consists of words which we delimit to subwords by formulating restrictions on the position variables of the subformula. Their position variables are no longer freely chosen but have to be part of the subword's domain, which we denote by $\underline{D}$ if no concrete set is given.

The domain for each subformula is determined by the formula for the regular availability expression above. To define the top-level domain as a set D we need to express the fact that each position is part of D, which can be done by $\exists D \forall x D(x)$. When we introduce a new MSO or FO variable in any subformula we restrict it to the current domain $\underline{D}$ by requiring it to be part of $\underline{D}$. It might happen that the name of the domain and a freshly introduced variable are the same. We call the situation a **name conflict**. It can be resolved by renaming one of the variables in conflict. Name conflicts can be more generally interpreted as the introduction of a new variable within the scope of another one that has the same name. When translating the check constraints into logic we will use a name conflict to restrict the checking onto the currently active check symbols.

Below we present the original formula on the left and the formula restricted to the domain $\underline{D}$ on the right. We do not deal with free variables, since we have limited ourselves to sentences by Theorem 4 on the previous page.

$$\begin{array}{llllll} \exists x & \varphi & \rightsquigarrow & \exists x & \underline{D}(x) & \wedge\varphi \\ \forall x & \varphi & \rightsquigarrow & \forall x & \underline{D}(x) & \rightarrow \varphi \\ \exists X & \varphi & \rightsquigarrow & \exists X & X \subseteq \underline{D} & \wedge\varphi \\ \forall X & \varphi & \rightsquigarrow & \forall X & X \subseteq \underline{D} & \rightarrow \varphi \quad . \end{array}$$

4.3 Transformation of non-recursive regular expressions

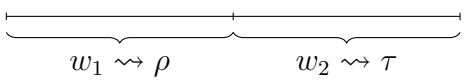
The elementary expressions are immediately translated with respect to $\underline{D}$. A formula only satisfied by ε is $\neg\exists x \underline{D}(x)$, which yields the domain to be empty. A single a is characterised by $\exists x \underline{D}(x) \wedge Q_a(x) \wedge \neg\exists y\, \underline{D}(y) \wedge \neg y = x$. This expresses that the word forming a model of the formula has one position with the letter a and no position but this one. The expression $\varnothing$ is a special case, since there is no word which satisfies this expression. We can express it by an unsatisfiable formula like $\exists x \underline{D}(x) \wedge \neg x = x$.

4.4 Transformation of recursive regular expressions

To express recursive calls with the expression ρ on the domain $\underline{D}$ we use $form\,\rho\,\underline{D}$.

The expression $\rho + \tau$ can be translated to $(form\,\rho\,\underline{D}) \vee (form\,\tau\,\underline{D})$ without changing the domain.

For a concatenation $\rho.\tau$, we need to find two subwords such that the former satisfies ρ and the latter satisfies τ.

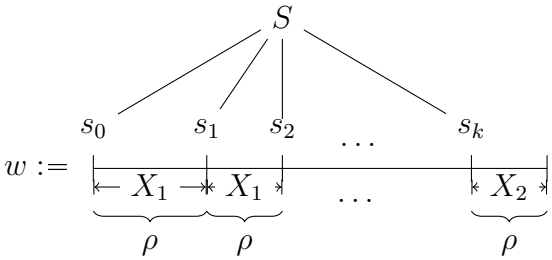
$w :=$ 

Both words have to follow directly and without overlap, so that together they form the whole word. This property is characterised by

$$
\begin{aligned}
&\exists X_1 \exists X_2\, X_1 \subseteq \underline{D} \wedge X_2 \subseteq \underline{D} \\
&\wedge (\neg\exists x X_1(x) \wedge X_2(x)) && \text{(no overlapping)} \\
&\wedge (\neg\exists x \underline{D}(x) \wedge \neg X_1(x) \wedge \neg X_2(x)) && (X_1, X_2 \text{ form the whole word}) \\
&\wedge (\forall x \forall y X_1(x) \wedge X_2(y) \rightarrow x < y) && (X_1 \text{ before } X_2) \\
&\wedge (\text{form}\,\rho\, X_1) \wedge (\text{form}\,\tau\, X_2) \quad ,
\end{aligned}
$$

where X_1 and X_2 are the domains of w_1 and w_2.

The expression ρ^* is more difficult to encode. A generalisation of the concatenation does not work, as * allows for any number of concatenations. So we can neither introduce a new set X_i for each concatenation i nor can we preliminarily limit the number of sets. By using the set's first position instead of a whole set we avoid these problems.



- S comprises each beginning of a subword.
- X_1 represents each subword $[s_i, s_{i+1}[$ between two subword beginnings s_i, s_{i+1}.

- X_2 represents the last part of the word.
- Each X_1 and also X_2 form a model of ρ.

Each beginning of a subword works as a guard to make sure that the sets following each other do neither overlap nor leave gaps. By also stating that the union of the sets is the whole word we can effectively describe the Kleene star using the three second-order variables S, X_1 and X_2.

$$
\begin{aligned}
&\exists S\, S \subseteq \underline{D} && (1)\\
&\wedge\, (\forall x\, first_{\underline{D}}(x)) \rightarrow S(x)) && (2)\\
&\wedge\, (\forall x \forall y (x < y \wedge S(x) \wedge S(y) \wedge \neg \exists z\, x < z \wedge z < y \wedge S(z)) \rightarrow && (3)\\
&\quad (\exists X_1\, X_1 \subseteq \underline{D} \wedge ((\text{form}\, \rho\, X_1) && (4)\\
&\quad \wedge\, X_1(x) \wedge \neg \exists z\, z < x \wedge X_1(z) && (5)\\
&\quad \wedge\, \neg \exists z\, \underline{D}(z) \wedge x < z \wedge z < y \wedge \neg X_1(z) && (6)\\
&\quad \wedge\, \neg \exists z\, z \geqslant y \wedge X_1(z))) && (7)\\
&\wedge\, (\forall x\, last_S(x) \rightarrow && (8)\\
&\quad (\exists X_2\, X_2 \subseteq \underline{D} \wedge (\text{form}\, \rho\, X_2) && (9)\\
&\quad \wedge\, X_2(x) \wedge \neg \exists y\, \underline{D}(y) \wedge x < y \wedge \neg X_2(y) \wedge \neg \exists z\, z < x \wedge X_2(z))) && (10)
\end{aligned}
$$

Line (2) states that the beginning of the whole word is also the beginning of a subword. The lines (3) to (7) describe the sets X_1: Under the condition that x and y are two beginnings of subwords directly following each other (3), the set X_1 between them contains x and nothing before x (5) contains every position between x and y (6) but does not contain any position from y on (7). Likewise, lines (8) to (10) describe X_2: under the condition that x is the last beginning of a subword (8), X_2 contains x, any position after x as long as this position is part of the domain, and no position before x.

Only one slight modification is necessary to translate ρ^+: we state that there is a position starting the last subword which excludes the case of zero repetitions but also excludes ε from being a model at all. Since $\mathcal{L}(\rho^+)$ contains ε if $\mathcal{L}(\rho)$ does, we have to add the formula accepting ρ as a second option. Since $\mathcal{L}(\rho) \subseteq \mathcal{L}(\rho^+)$, this introduces no further words but ε if $\varepsilon \in \mathcal{L}(\rho)$. In summary we have to replace the universal quantifier in line (8) by an existential one, so that we have "$(\exists x\, last_S(x) \wedge$" and also add "$\vee$ form $\rho\, \underline{D}$" as a second option on the top-level.

4.5 Transformation of availability extensions

Availability expressions are strictly more expressive than regular expressions as we saw in section 3.2 on page 20 by the expressibility of $\{a^n b^n\}$. This makes clear that an extension to MSO will be necessary to express them, since Büchi's theorem (2.3) states that an MSO-definable language is always regular.

We extend MSO by a new second-order predicate $comp_{A \gtrsim k}(X)$ which checks, if within the subword w encoded by X the availability constraint $|\pi_A(w)| \gtrsim k \cdot |\pi_\Sigma(w)|$ holds. We call this extension **availability logic**. To model the constraint we work with the cardinality of a set, which is not provided by MSO logic on word domains. We use two subsets Y and Z where Y is $\pi_\Sigma(w)$, while Z is a freely chosen subset of Y, only consisting of positions with an element of A. Instead of changing the word when evaluating a constraint like we do when we evaluate the word under the expression, we simply ignore the check symbols by limiting Y to Σ. This makes it necessary to use a trick, when evaluating nested constraints, but avoids the problem of changing domains or positions. We present the trick at the end of this section. To ensure that $comp_{A \gtrsim k}$ is only used on subwords and not on any sets, we also require the set to be connected ($\mathcal{Z}(X)$) in order to evaluate the predicate to true.

Definition 29 (comp)**.**

$$\begin{aligned} comp_{A \gtrsim k}(X) \textit{ iff } & \mathcal{Z}(X) \wedge \exists Y\, Y \subseteq X \wedge [\forall x(X(x) \wedge Q_\Sigma(x)) \rightarrow Y(x)] \wedge Q_\Sigma(Y) \\ & \wedge \exists Z\, Z \subseteq X \wedge [\forall x(X(x) \wedge Q_A(x)) \rightarrow Z(x)] \wedge Q_A(Z) \\ & \wedge |Z| \gtrsim k \cdot |Y| \quad . \end{aligned}$$

We have to show that the predicate indeed checks the availability constraint $|\pi_A(w)| \gtrsim k \cdot |\pi_\Sigma(w)|$ if X characterises the subword w.

Lemma 3. *Let Σ be an alphabet, w a word over $\Sigma \cup \{\checkmark\}$ and X a second-order variable which is interpreted as $dom(w)$. Then $comp_{A \gtrsim k}(X)$ holds iff $|\pi_A(w)| \gtrsim k \cdot |\pi_\Sigma(w)|$ holds.*

Proof. Since X is fixed as the domain $dom(w)$, we know that it is connected and can ignore $\mathcal{Z}(X)$ in the further proof which consists of two inclusions.

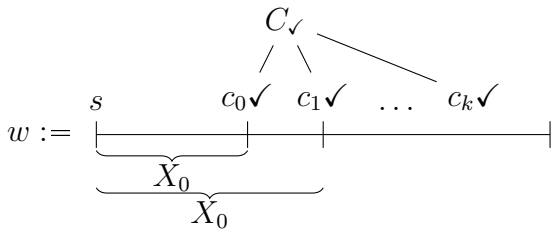
"⇒" : Since $comp_{A \gtrsim k}(X)$ holds, there is a set Y which is a subset of X and therefore of $dom(w)$. This set Y consists of all the positions in X a letter of Σ is placed at. In addition, Y does not contain any position no letter of Σ is written at. Thus Y can only be the set of all positions a letter of Σ is placed at and therefore $|Y| = |\pi_\Sigma(w)|$. The set Z is described in analogy so $|Z| = |\pi_A(w)|$. Since $|Z| \gtrsim k \cdot |Y|$ holds, $|\pi_A(w)| \gtrsim k \cdot |\pi_\Sigma(w)|$ also holds.

"⇐" : We choose Y as all positions in w which carry a letter of Σ and Z as all positions in w which carry a letter of A. As we know from "⇒" this is a valid choice for the first two lines of the formula. Since $|\pi_A(w)| \gtrsim k \cdot |\pi_\Sigma(w)|$ holds, the third line is also true.

□

We have seen that the new predicate is powerful enough to model the checking of an availability constraint. The sets we will use this predicate on are specified in the availability extensions: Each ✓ claims an availability constraint $A \gtrsim k$ on a subword v which begins with the first letter after the opening bracket of the constraint $(\dots)_{A \gtrsim k}$ and

ends with the ✓. Since each included ✓ is projected away by $\pi_\Sigma(Y)$, it is not important whether we define the subword to end before or with the check symbol.



We characterise the set X_0 related to the subword v by its borders. The first border is given by the constraint and the second one is given by ✓. This strong interconnection is represented in a set $C_\checkmark$ which is introduced whenever a new constraint $(\rho)_{A\gtrsim k}$ is translated. The set $C_\checkmark$ has some similarities with the set S used in the Kleene translation, since it contains end points of subwords, but in $C_\checkmark$ these subwords all start in the same position and overlap. In chapter 5.2 on page 43 we will see that this difference allows us to completely remove $C_\checkmark$, while S cannot be eliminated. We state that any element of $C_\checkmark$ is the end of a subword v for which the constraint holds. This is done by

$$\begin{aligned}
&\exists C_\checkmark\, C_\checkmark \subseteq \underline{D} \wedge \forall x\, C_\checkmark(x) \\
&\quad \rightarrow (\exists X_0\, X_0 \subseteq \underline{D} \wedge X_0(x) \wedge \neg\exists y\, y > x \wedge X_0(y) \\
&\qquad \wedge \forall z\, (\underline{D}(z) \wedge z < x) \rightarrow X_0(z) \\
&\qquad \wedge \mathit{comp}_{A\gtrsim k}(X_0)) \\
&\quad \wedge \operatorname{form}\rho\, \underline{D} \quad .
\end{aligned}$$

Whenever the expression ✓ is translated, we do not only express that the current letter is ✓, but also state that the position is part of $C_\checkmark$:

$$\begin{aligned}
&\exists x \underline{D}(x) \wedge Q_\checkmark(x) \wedge \neg\exists y\, \underline{D}(y) \wedge \neg y = x \\
&\qquad \wedge C_\checkmark(x) \quad .
\end{aligned}$$

This translation introduces a free variable $C_\checkmark$. Since we are limited to sentences, we have to make sure that there is a surrounding constraint binding this variable. But since we have limited the considered expressions to the check-closed form, such a surrounding constraint is always given.

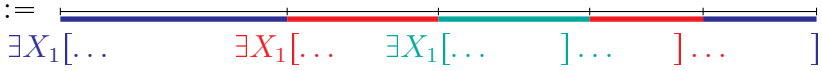
Check symbols already evaluated in underlying constraints should not be considered when evaluating the current constraint, since they would already have been projected away. Our construction ignores them using the scope of a variable: As discussed in 4.2 on page 26 name conflicts appear if we introduce a new variable within the scope of

another one with the same name. For example, the translation of form $\rho.\tau\, X_1$ causes a name conflict with

$$\begin{aligned}
&\exists X_1 \exists X_2\, X_1 \subseteq \mathbf{X_1} \wedge X_2 \subseteq \mathbf{X_1} \\
&\wedge (\neg\exists x X_1(x) \wedge X_2(x)) \\
&\wedge (\neg\exists x \underline{D}(x) \wedge \neg X_1(x) \wedge \neg X_2(x)) \\
&\wedge (\forall x \forall y X_1(x) \wedge X_2(y) \rightarrow x < y) \\
&\wedge (\text{form}\,\rho\, X_1) \wedge (\text{form}\,\tau\, X_2) \quad .
\end{aligned}$$

Usually this name conflict would be resolved by renaming one of the two sets X_1. However, this time we do not resolve but study it to understand the scope of a variable. Outside the formula above, the set X_1 from "form $\rho.\tau\, X_1$" is **active**, which means that an occurrence of X_1 in a predicate has to be evaluated under the value this X_1 has. However, this X_1 is **shadowed** by $\exists X_1$ at the beginning of the formula. Therefore, any use of X_1 within the formula or in any subformula constructed by form $\rho\, X_1$ and form $\tau\, X_2$ refers to the freshly introduced X_1, while the old one cannot be accessed.

Of course this new X_1 can also be shadowed by a new introduction of a bound variable X_1 so that we have

$w :=$ 

.

In the case of nested availability constraints, like they occurred in the translation of $a^n b^n$ or matching parentheses, we exploit this shadowing. While other variables may be renamed if necessary, the name of the set $C_\checkmark$ is fixed. Every translation of a constraint $(\dots)_{A \gtrsim k}$ introduces a new variable $C_\checkmark$ so that the formerly active $C_\checkmark$ is shadowed within the domain of the constraint. This makes sure that no ✓ within an inner constraint enforces a check in the outer constraint, since the formula "$\wedge C_\checkmark(x)$" always refers to the active and thus directly surrounding set $C_\checkmark$. Furthermore, we cannot enforce a check in an outer constraint since their sets $C_\checkmark$ are shadowed as well.

4.6 Accepted words

We are now able to translate a given check-closed regular availability expression into an availability formula: We know how to translate each form of a regular availability expression and by that inductively know how to translate the complete expression. Furthermore, we know how to initialise the first call of form with the complete domain $dom(w)$ by $\exists D \forall x D(x)$.

The only missing part is the elimination of the check symbols occurring within the range of a constraint. According to definition 16 on page 19, only those check symbols are removed which are positioned within the range of a constraint. But in order to receive a deterministic acceptance of a word we limited ourselves to check-closed expressions. In those expressions all check symbols are surrounded by a constraint so that we can project any ✓ away in the end by defining the language accepted by an availability formula to be the projection of the accepted words to Σ.

Definition 30 ($\mathcal{L}(\varphi)$)**.** *Let $\Sigma \cup \{\checkmark\}$ be the alphabet the availability sentence φ is built over. We define the language accepted by φ:*

$$\mathcal{L}(\varphi) = \{\pi_\Sigma(w) \mid (\underline{w}) \vDash \varphi\} \quad .$$

4.7 Correctness of the translations

Our first big step towards the main theorem is the proof that the previously introduced translations in form are correct. Therefore we rely on Lemma 3 on page 29 which showed that the *comp* predicate works as an availability constraint if it is invoked on a subword. Since we are so far not concerned with the top-level translation, we use a dummy variable D to start form. This free variable D is then interpreted as $dom(w)$ when we analyse the satisfaction of the formula under w, or it is substituted by another variable when we integrate the corresponding formula into a larger one.

Lemma 4. *Let ρ be a check-closed regular availability expression over an alphabet $\Sigma \cup \{\checkmark\}$. Then*

$$(\underline{w}, dom(w)) \vDash form\,\rho\,D, \textit{ iff } w \textit{ is accepted by } \rho \quad .$$

Proof. We use two structural inductions to prove the lemma. Subexpressions of an expression in check-closed form are usually not check-closed, as we have illustrated after definition 21 on page 22. This makes a generalisation of the claim necessary.

In an expression which is not in check-closed form there is at least one active check symbol. When we translate this check symbol, we introduce a free second-order variable $C_\checkmark$ via "$\wedge C_\checkmark(x)$", but since the check symbol is not surrounded by a constraint, such a set $C_\checkmark$ is never bound. Thus we have to interpret a second free variable $C_\checkmark$ and therefore extend the former satisfaction relation to $(\underline{w}, dom(w), P) \vDash \mathrm{form}\,\rho\,D$, where P is the interpretation of $C_\checkmark$. This relation can be seen as a more general case of the original satisfaction relation, since the interpretation that P implies is ignored if there is no free variable which we can apply it to.

We can formulate our modified claim using this extended satisfaction relation. To prove "$\Rightarrow$" we show that for all sets $P \subseteq dom(w)$ so that $(\underline{w}, dom(w), P) \vDash \mathrm{form}\,\rho\,D$ holds, w is accepted by ρ.

In opposite, when proving "$\Leftarrow$" we show that for a word w which is accepted by ρ, for all different sets $free(w, \rho)$ $(\underline{w}, dom(w), free(w, \rho)) \vDash \mathrm{form}\,\rho\,D$ holds.

["⇒":] Let w be a word, $P \subseteq dom(w)$ a set of positions so that $(\underline{w}, dom(w), P) \vDash \mathrm{form}\,\rho\,D$. We show that w is accepted by ρ.

basis. We have to distinguish four cases:

$\rho = \varnothing$:
: There is no word w and no interpretation P such that $(\underline{w}, dom(w), P) \vDash \mathrm{form}\,\varnothing\,D$ holds since $\exists x\,D(x) \wedge \neg x = x$ is not satisfiable: $\exists x$ is only satisfied if at least one position x exists in the word. However, $\neg x = x$ cannot be true for any position.

$\rho = \varepsilon$:

The only possible choice for a word w so that $(\underline{w}, dom(w), P) \vDash \text{form}\, \varepsilon\, dom(w)$ is $w = \varepsilon$: $\neg\exists x D(x)$ is only true if there exists no x within D's interpretation $dom(w)$. The only word w for that $dom(w) = \varnothing$ holds is the empty word ε.

$\rho = a$:

$\exists x(Q_a(x) \wedge D(x) \wedge \neg\exists y\, D(y) \wedge y \neq x$ states that any word satisfying the formula with its model has to contain a position the letter a is placed at and may contain no other position at all. This is only true for $w = a$.

$\rho = \checkmark$:

The formula $\exists x D(x) \wedge Q_{\checkmark}(x) \wedge \neg\exists y\, D(y) \wedge \neg y = x \wedge C_{\checkmark}(x)$ is similar to the one for a in the first part. This makes clear that the only word which can form a model is $\checkmark$. As a second restriction we have $C_{\checkmark}(x)$ which forces this check symbol to be a position of the set $C_{\checkmark}$. This limits valid interpretations for P to those containing the position in $dom(w)$.

induction hypothesis. Let v_1, v_2 be words and $P_1 \subseteq dom(v_1)$ and $P_2 \subseteq dom(v_2)$ such that $(\underline{v}_1, dom(v_1), P_1) \vDash \text{form}\, \tau\, D$ and $(\underline{v}_2, dom(v_2), P_2) \vDash \text{form}\, \sigma\, D$ hold. We assume that v_1 is accepted by τ and v_2 is accepted by σ.

inductive step. Now we can prove our claim for compound expressions ρ. Let w be a word and $P \subseteq dom(w)$ be an interpretation for which $(\underline{w}, dom(w), P) \vDash \text{form}\, \rho\, D$. We show that w is accepted by ρ.

$\rho = \tau + \sigma$:

We have $\text{form}\, \rho\, D = \text{form}\, (\tau + \sigma)\, D = (\text{form}\, \tau\, D) \vee (\text{form}\, \sigma\, D)$. Since $(\underline{w}, dom(w), P) \vDash \text{form}\, \rho\, D$, it is $(\underline{w}, dom(w), P) \vDash \text{form}\, \tau\, D \vee \text{form}\, \sigma\, D$. This implies that either $(\underline{w}, dom(w), P) \vDash \text{form}\, \tau\, D$ or $(\underline{w}, dom(w), P) \vDash \text{form}\, \sigma\, D$ hold. We can apply the induction hypothesis at least on one the satisfaction relations and receive either w accepted by τ or by σ. This yields that w is accepted by $\tau + \sigma = \rho$.

$\rho = \sigma.\tau$:

We have $\text{form}\, \rho\, D = \text{form}\, (\sigma.\tau)\, D =$

$$\begin{aligned}
&\exists X_1 \exists X_2\, X_1 \subseteq D \wedge X_2 \subseteq D \\
&\wedge (\neg\exists x X_1(x) \wedge X_2(x)) \\
&\wedge (\neg\exists x D(x) \wedge \neg X_1(x) \wedge \neg X_2(x)) \\
&\wedge (\forall x \forall y X_1(x) \wedge X_2(y) \rightarrow x < y) \\
&\wedge (\text{form}\, \sigma\, X_1) \wedge (\text{form}\, \tau\, X_2) \quad .
\end{aligned}$$

A word w which forms a model for the formula has to have two subsets of positions, X_1 and X_2, which are disjoint but together contain the whole domain. Additionally, every position in X_1 is positioned earlier than any position in X_2. This yields two subwords v_1 and v_2 such that v_1's positions

are those of X_1 and v_2's positions are those of X_2 and their concatenation forms w. The formula also states that $(\text{form}\,\sigma\,X_1) \wedge (\text{form}\,\tau\,X_2)$.

We can partition P into two sets P_1 and P_2 such that $P_1 \subseteq dom(v_1)$ and $P_2 \subseteq dom(v_2)$ and $(\underline{v}_1, dom(v_1), P_1)$ is a valid interpretation for $(\text{form}\,\sigma\,X_1)$ and $(\underline{v}_2, dom(v_2), P_2)$ is a valid interpretation for $\text{form}\,\tau\,X_2$. Although the former variable D is now called X_1 or X_2, we can apply the induction hypothesis, especially since a substitution of X_1 and X_2 by D fixes this syntactical problem. Since v_1 is accepted by σ and v_2 is accepted by τ, we know that $w = v_1.v_2$ is accepted by $\rho = \sigma.\tau$.

$\rho = (\tau)^*$:

We have $\text{form}\,\rho\,D = \text{form}\,(\tau^*)\,D =$

$$
\begin{aligned}
&\exists S\, S \subseteq D \\
&\wedge\, (\forall x\, first_D(x) \rightarrow S(x)) \\
&\wedge\, (\forall x \forall y (x < y \wedge S(x) \wedge S(y) \wedge \neg \exists z\, x < z \wedge z < y \wedge S(z)) \rightarrow \\
&\quad (\exists X_1\, X_1 \subseteq D \wedge ((\text{form}\,\tau\,X_1) \\
&\quad\wedge\, X_1(x) \wedge \neg \exists z\, z < x \wedge X_1(z) \\
&\quad\wedge\, \neg \exists z\, D(z) \wedge x < z \wedge z < y \wedge \neg X_1(z) \\
&\quad\wedge\, \neg \exists z\, z \geqslant y \wedge X_1(z))) \\
&\wedge\, (\forall x\, last_S(x) \rightarrow \\
&\quad (\exists X_2\, X_2 \subseteq D \wedge (\text{form}\,\tau\,X_2) \\
&\quad\wedge\, X_2(x) \wedge \neg \exists y\, D(y) \wedge x < y \wedge \neg X_2(y) \wedge \neg \exists z\, z < x \wedge X_2(z))) \quad .
\end{aligned}
$$

A word w so that $(\underline{w}, dom(w), P)$ is a model for this formula contains a set of positions S which the first position in $dom(w)$ belongs to, if such a position exists. Each of the positions in S encodes the beginning of a new subword represented as X_1 or X_2: for two positions x and y which are direct neighbours in S there is a set X_1 which ranges from the earlier position x to the position directly before y. Each of these pairs encodes a subword v_i with $i = 0, \ldots n-1$, if $n-1$ such pairs exist.

If x is the last position in S it is also a position in a subset X_2 such that there is no position after this x until the end of $dom(w)$, which is not in X_2, nor is there any position before x which is part of X_2. Thus X_2 encodes a subword v_n of w which is the last part of w. Now we have subwords v_i $(i = 0 \ldots n-1)$, which are concated and the first one of which starts in the first position of w. The word v_n forms the last subword and immediately follows the last v_i.

In analogy to the concatenation, we can partition P along the subwords so that we can apply the induction hypothesis. Each of the words v_i $(i = 0 \ldots n)$ is accepted by τ and therefore $w = v_0 \ldots v_n$ is accepted by $\rho = (\tau)^*$.

The formula is also satisfied, if no direct neighbours in S or even no positions in S exist at all. The latter implies an empty domain, since otherwise

$(\forall x\, D(x) \wedge (\neg\exists y\, y < x \wedge D(y)) \rightarrow S(x))$ would not be satisfied. This is the case of zero repetitions, which is part of $\mathcal{L}^*$ for any language. The former case implies that S contains at most one position, since otherwise neighbours would exist. This leads either to zero, as discussed above, or to the case of one position in S, which then has to be the first position in w. Now X_2 corresponds to w so that the induction hypothesis guarantees that w is accepted by τ.

$\rho = (\tau)_{A\gtrsim k}$:
We know that there is a P such that $(\underline{w}, dom(w), P)$ satisfies $\operatorname{form}\rho\, D = \operatorname{form}(\tau)_{A\gtrsim k}\, D =$

$$\begin{aligned}
&\exists C_\checkmark\, C_\checkmark \subseteq D \wedge \forall x\, C_\checkmark(x) \\
&\quad \rightarrow (\exists X_0\, X_0 \subseteq D \wedge X_0(x) \wedge \neg\exists y\, y > x \wedge X_0(y) \\
&\qquad \wedge \forall z\, (D(z) \wedge z < x) \rightarrow X_0(z) \\
&\qquad \wedge comp_{A\gtrsim k}(X_0)) \\
&\quad \wedge \operatorname{form}\tau\, D \quad .
\end{aligned}$$

Since $\operatorname{form}\rho\, D$ holds, there is one choice Q for $C_\checkmark$ which the first part of the formula but also $\operatorname{form}\tau\, D$ holds under. For this choice Q we have $(\underline{w}, dom(w), Q) \vDash \operatorname{form}\tau\, D$ and applying the induction hypothesis we see that w is accepted by τ. This shows that Q is a valid choice for $C_\checkmark$ in $\operatorname{form}\tau\, D$. Therefore Q contains one of the sets $free(w, \rho)$ of which at least one must exist since w is accepted by τ.

We show that all words $v_\checkmark$ which begin with the first letter of w and end with an element of this set $free(w, \rho)$ satisfy $|\pi_A(v_\checkmark)| \gtrsim k \cdot |\pi_\Sigma(v_\checkmark)|$ to show that w is accepted by ρ.

Therefore we examine the first part of the formula closer. The word w contains a set of positions $C_\checkmark$ which we interpret by Q. For each $x \in Q$, there is a set X_0, which contains x, but no later position, and also contains all the letters before x, which are still in $dom(w)$, but no earlier one. This yields that X_0 is a subword $v_\checkmark$ ranging from the beginning of w to the position x in Q. The predicate $comp_{A\gtrsim k}(X_0)$ holds for each of these sets X_0, so by Lemma 3 on page 29 $|\pi_A(v_\checkmark)| \gtrsim k \cdot |\pi_\Sigma(v_\checkmark)|$ holds for the corresponding word $v_\checkmark$.

["⇐":] Let ρ be a regular availability expression and w a freely chosen word such that w is accepted by ρ and Q is the checkset $free(w, \rho) \subseteq dom(w)$ for an arbitrary way to accept the word. We show that $(\underline{w}, dom(w), Q) \vDash \operatorname{form}\rho\, D$.

basis. We have to distinguish four cases:

$\rho = \varnothing$:
There is no word w accepted by $\varnothing$, which makes clear that for each word accepted by $\varnothing$ $(\underline{w}, dom(w), Q) \vDash \operatorname{form}\varnothing\, D$ holds, which we wanted to show.

$\rho = \varepsilon$:

The only word w accepted by the expression ε is $w = \varepsilon$. We have to check wether $(\underline{\varepsilon}, dom(\varepsilon), Q) \vDash \mathrm{form}\,\varepsilon\, D$. Since $dom(\varepsilon) = \varnothing$, there is no x at all, so $\neg\exists x \ldots$ is true independent from the formula following and in particular independent of Q.

$\rho = a$**:** The word a is the only word accepted by ρ. Since there is no free check symbol in a, $free(a, a)$ is uniquely defined as $\varnothing$. We have to show that $(\underline{a}, dom(a), \varnothing) \vDash \mathrm{form}\,a\, D$. We have $\mathrm{form}\,a\, D = \exists x(Q_a(x) \wedge D(x)) \wedge \neg\exists y\, D(y) \wedge \neg y = x$. We can evaluate the formula to true because the existential quantifier can choose for x position 0. This position carries an a and there is no position apart from 0.

$\rho = \checkmark$:

The only accepted word w is $\checkmark$ and the check symbol is active, so $free(\checkmark, \checkmark) = \{0\}$. We have $\mathrm{form}\,\checkmark\, D =$

$$\begin{aligned} &\exists x D(x) \wedge Q_{\checkmark}(x) \wedge \neg\exists y\, D(y) \wedge \neg y = x \\ &\quad \wedge C_{\checkmark}(x) \quad . \end{aligned}$$

Again, the existential quantifier can choose 0 as value for x so that the first line is valid. The formula $C_{\checkmark}(x)$ is then also valid since $0 \in \{0\}$.

induction hypothesis Let v_1, v_2 be words such that v_1 is accepted by the regular availability expression τ and v_2 is accepted by σ. We abbreviate $free(v_1, \tau) \subseteq dom(v_1)$ for an arbitrary way to accept v_1 by Q_1 and $free(v_2, \sigma) \subseteq dom(v_2)$ for an arbitrary way to accept v_2 by Q_2. We assume that $(\underline{v_1}, dom(v_1), Q_1) \vDash \mathrm{form}\,\tau\, D$ and $(\underline{v_2}, dom(v_2), Q_2) \vDash \mathrm{form}\,\sigma\, D$.

inductive step We show the claim for compound expressions ρ. Let w a word accepted by ρ and $Q = free(w, \rho)$ the check set of an arbitrary way to accept w in ρ. We show $(\underline{w}, dom(w), Q) \vDash \mathrm{form}\,\rho\, D$.

$\rho = \tau + \sigma$**:** Since w is accepted by ρ with Q, w is accepted by $\tau + \sigma$ with Q and thus it is either accepted by τ with checkset Q or by σ with check set Q. We can apply the induction hypothesis and receive $(\underline{w}, dom(w), Q) \vDash \mathrm{form}\,\tau\, D$ or $(\underline{w}, dom(w), Q) \vDash \mathrm{form}\,\sigma\, D$. So we have $(\underline{w}, dom(w), Q) \vDash (\mathrm{form}\,\tau\, D) \vee (\mathrm{form}\,\sigma\, D)$ which is $(\underline{w}, dom(w), Q) \vDash (\mathrm{form}\,\rho\, D)$.

$\rho = \sigma.\tau$**:** Since w is accepted by ρ, it is accepted by $\sigma.\tau$ which yields that w consists of two words v_1 and v_2 ($w = v_1 v_2$) such that v_1 is accepted by σ and v_2 is accepted by τ. We have $\mathrm{form}\,\rho\, D = \mathrm{form}\,(\sigma.\tau)\, D =$

$$\begin{aligned} &\exists X_1 \exists X_2\, X_1 \subseteq D \wedge X_2 \subseteq D \\ &\wedge (\neg\exists x X_1(x) \wedge X_2(x)) \\ &\wedge (\neg\exists x D(x) \wedge \neg X_1(x) \wedge \neg X_2(x)) \\ &\wedge (\forall x \forall y X_1(x) \wedge X_2(y) \rightarrow x < y) \\ &\wedge (\mathrm{form}\,\sigma\, X_1) \wedge (\mathrm{form}\,\tau\, X_2) \quad . \end{aligned}$$

We choose $dom(v_1)$ as the positions in X_1 and $dom(v_2)$ as the set of positions in X_2. Both sets are in $dom(w)$, disjoint, but together form the whole word and X_1 is completely before X_2.

We can partition Q into two sets Q_1 and Q_2 so that Q_1 contains all positions of Q which are in $dom(v_1)$ and Q_2 contains those of $\mathrm{dom}(v_2)$. Since Q is one way to match the free checks of w onto the active checks in ρ, Q_1 and Q_2 present one way to match the free checks of v_1 onto σ and those of v_2 onto τ. Applying the induction hypothesis we see $(\underline{v_1}, dom(v_1), Q_1) \vDash \mathrm{form}\,\sigma\, D$ and $(\underline{v_2}, dom(v_2), Q_2) \vDash \mathrm{form}\,\sigma\, D$. We substitute D by X_1 in the formula for σ and by X_2 in the formula for τ. With $dom(v_1)$ as interpretation for X_1 and $dom(v_2)$ as interpretation for X_2 we get $(\underline{w}, dom(v_1), dom(v_2), Q) \vDash (\mathrm{form}\,\sigma\, D) \wedge (\mathrm{form}\,\tau\, D)$ since $Q_1 \cup Q_2 = Q$.

Please convince yourself that the formula also holds for ε as v_1 and v_2 if the regular availability expressions σ and τ accept ε as a word: existential quantifiers are only used for the second-order variables for which any "exists" claim can be satisfied by the empty set. Any first-order quantification is either done via $\forall$ or via $\neg\exists x \ldots$ which is true if no x exists.

$\rho = (\tau)^*$**:** A word w accepted by ρ is either ε, a word v_0 accepted by τ or has the form $w = v_0 \ldots v_n$ with each $v_i\,(i = 0 \ldots n)$ accepted by τ. It is $\mathrm{form}\,\rho\, D = \mathrm{form}\,(\tau^*)\, D =$

$$
\begin{aligned}
&\exists S\, S \subseteq D \\
&\wedge\, (\forall x\, first_D(x) \rightarrow S(x)) \\
&\wedge\, (\forall x \forall y (x < y \wedge S(x) \wedge S(y) \wedge \neg\exists z\, x < z \wedge z < y \wedge S(z)) \rightarrow \\
&\quad (\exists X_1\, X_1 \subseteq D \wedge (\mathrm{form}\,\tau\, X_1) \\
&\quad \wedge\, X_1(x) \wedge \neg\exists z\, z < x \wedge X_1(z) \\
&\quad \wedge\, \neg\exists z\, D(z) \wedge x < z \wedge z < y \wedge \neg X_1(z) \\
&\quad \wedge\, \neg\exists z\, z \geqslant y \wedge X_1(z))) \\
&\wedge\, (\forall x\, last_S(x) \rightarrow \\
&\quad (\exists X_2\, X_2 \subseteq D \wedge (\mathrm{form}\,\tau\, X_2) \\
&\quad \wedge\, X_2(x) \wedge \neg\exists y\, D(y) \wedge x < y \wedge \neg X_2(y) \wedge \neg\exists z\, z < x \wedge X_2(z)))
\end{aligned}
$$

We give an interpretation for S, X_1, and X_2 for the different cases of $w = \varepsilon$, $w = v_0$, and $w = v_0 \ldots v_n$ to illustrate the effect of the existential quantifiers.

In the case of zero subwords, ε, we choose S as the empty set, which is the only subset of $dom(\varepsilon) = \varnothing$. Each subformula begins with a universally quantified first-order variable, which allows us to evaluate these subformulas to true.

In the case of only one subword v_0 we can choose S as the set only containing the first position. This makes $(x < y \wedge S(x) \wedge S(y))$ unsatisfiable, so that the implication is trivial. Furthermore, the first position is a position of S, after which no further position in S occurs. We can choose X_2 as all

positions in $dom(w) = dom(v_0)$ which satisfies the last line of the formula. It remains to prove that $(\underline{v}_0, dom(v_0), Q) \vDash \operatorname{form}\tau\, X_2$. Since Q is one choice for $free(w,\tau) = free(v_0,\tau)$ we can apply the induction hypothesis and receive that $(\underline{v}_0, dom(v_0), Q) \vDash \operatorname{form}\tau\, D$ which is transformed into the desired form by substituting D by X_2.

In the case of several subwords we choose S as all positions which are the beginning of a subword v_i for $(0 \leqslant i \leqslant n)$. Again, S is a subset of D interpreted as $dom(w)$ the first position of $dom(w)$ also occurs in. For every two direct neighbours x and y in S, we choose the area between them – beginning in x and ending before y – as X_1. This fulfills $X_1(x)$ as well as the inclusion of every position until y and the exclusion of any other position. To see that $\operatorname{form}\tau\, X_1$ holds for each $dom(v_i)$ with $i = 0 \ldots n-1$ as X_1, we extend the argument we used in the concatenation and partition Q into Q_0 to Q_n where each of the sets corresponds to one choice for the free checks in v_0 to v_n. We can apply the induction hypothesis and see $(\underline{v}_i, dom(v_i), Q_i) \vDash \operatorname{form}\tau\, D$ for $i = 0 \ldots n$ and the desired claim if we substitute D by X_1 for $i = 0 \ldots n-1$.

We choose $dom(v_n) \subseteq dom(w)$ as X_2 and receive that $\operatorname{form}\tau\, X_2$ holds by substituting D by X_2 in $(\underline{v}_n, dom(v_n), Q_n) \vDash \operatorname{form}\tau\, D$ The last element of S is in X_2, since it is the beginning of v_n, also every element after this last element in S until the end of $dom(w)$ is in X_2, since v_n is the last subword and since v_n starts with the last position of S, no earlier position is part of X_2.

$\rho = (\tau)_{A \gtrsim k}$:

The expression ρ does not contain active checks, so $Q = \varnothing$. Since w is accepted by $\rho = (\tau)_{A \gtrsim k}$, w is accepted by τ with the checkset $free(w,\tau)$ so that the check of the availability constraint $A \gtrsim k$ is performed successfully on every prefix v of w ending in a position of $free(w,\tau)$. We call this checkset Q_0 and show that it is a sufficient choice for $C_{\checkmark}$. With this interpretation,

$$
\begin{aligned}
&\exists C_{\checkmark}\, C_{\checkmark} \subseteq D \wedge \forall x\, C_{\checkmark}(x) \\
&\quad \rightarrow (\exists X_0\, X_0 \subseteq D \wedge X_0(x) \wedge \neg\exists y\, y > x \wedge X_0(y) \\
&\qquad \wedge \forall z\, (D(z) \wedge z < x) \rightarrow X_0(z) \\
&\qquad \wedge comp_{A \gtrsim k}(X_0)) \\
&\quad \wedge \operatorname{form}\tau\, D
\end{aligned}
$$

expresses that for each position x in Q_0 there is a subset X_0. For each of these positions x we choose the domain of the subword v ranging from the beginning of $dom(w)$ to the position x as interpretation for the corresponding X_0. This is a valid choice since x but no later position than x are contained as well as any position in $dom(w)$ which is placed earlier than x. Since the availability constraint holds for each of these words v, the *comp* predicate is valid on X_0 interpreted as $dom(v)$ by Lemma 3 on page 29. Since w is accepted by τ with $free(w,\tau) = Q_0$, we know by the induction hypothesis

that $(\underline{w}, dom(w), Q_0) \vDash \mathrm{form}\,\tau\, D$. This shows that the last line of the formula is also valid for the choice of Q_0 for $C_\checkmark$.

We have shown that w is accepted by an arbitrary availability expression ρ iff there is a set $P \subseteq dom(w)$ so that $(\underline{w}, dom(w), P) \vDash \mathrm{from}\,\rho\, D$. If ρ is in check-closed form the interpretation P can be dropped, since there is only one freely occurring variable, D, which is interpreted by $dom(w)$.

□

4.8 Main theorem

In the previous section, we have shown that for a check-closed regular availability expression ρ and a word w $(\underline{w}, dom(w)) \vDash \mathrm{form}\,\rho\, D$ holds iff w is accepted by ρ. Our next step towards the main theorem is to transform the formula that form constructs into a sentence. Therefore we have to remove the free variable D while we fix its value to be $dom(w)$.

Lemma 5. *Let ρ be a check-closed expression and w a word over $\Sigma \cup \{\checkmark\}$. We have*

$$(\underline{w}) \vDash \exists D\, \forall x D(x) \wedge form\,\rho\, D \text{ iff } w \text{ is accepted by } \rho \quad .$$

Proof. By Lemma 4 on page 32, we know that $(\underline{w}, dom(w)) \vDash \mathrm{form}\,\rho\, D$ holds iff w is accepted by ρ.

We prove that $(\underline{w}, dom(w)) \vDash \mathrm{form}\,\rho\, D$ iff $(\underline{w}) \vDash \exists D\, \forall x D(x) \wedge \mathrm{form}\,\rho\, D$.

"⇒": Since $(\underline{w}, dom(w)) \vDash \mathrm{form}\,\rho\, D$ holds, we know that $dom(w)$ is a valid choice for D to satisfy $\mathrm{form}\,\rho\, D$. This choice also satisfies $\exists D\, \forall x D(x)$, since it states that each position of w is a part of the set of all positions in w.

"⇐": It is $(\underline{w}) \vDash \exists D\, \forall x D(x) \wedge \mathrm{form}\,\rho\, D$. We assume that a valid choice for D is a strict subset of $dom(w)$. But then there is a position x, so that $x \notin D$. This makes the formula $\forall x D(x)$ false so that the whole formula becomes false. Hence, a strict subset cannot be a valid choice and thus the whole set $dom(w)$ is the valid choice. But then $\mathrm{form}\,\rho\, D$ holds if D is interpreted as $dom(w)$, which is what we had to show.

□

Lemma 5 introduces the construction which we will use to prove the following theorem.

Theorem 4 ($\mathcal{L}(\rho) = \mathcal{L}(\varphi)$)**.** *For any check-closed availability expression ρ there is an availability sentence φ so that $\mathcal{L}(\varphi) = \mathcal{L}(\rho)$.*

Proof. Out of an arbitrary check-closed expression ρ we can construct a formula $\varphi = \exists D\, \forall x D(x) \wedge \mathrm{form}\,\rho\, D$, so that a word forms a model for φ iff it is accepted by ρ, as Lemma 5 shows. This means that any word accepted by ρ is also accepted by φ and vice versa. For each word w which is accepted by the check-closed expression ρ, only

$\pi_\Sigma(w)$ is in the language $\mathcal{L}(\rho)$, as illustrated in Fact 1 on page 22. Thus $\mathcal{L}(\rho) = \{\pi_\Sigma(w) \mid w \text{ is accepted by } \rho\}$ in analogy to $\mathcal{L}(\varphi) = \{\pi_\Sigma(w) \,|\, (\underline{w}) \vDash \varphi\,\} = \{\pi_\Sigma(w) \mid w \text{ is accepted by } \rho\}$ where the last equality holds due to Lemma 5 on the preceding page. Since the definition of the accepted languages is equal, the languages are equal. □

5 Transformation into Σ_1^1 with additional *comp* predicate

[KR03] introduces WS1S^{card} as an extension of MSO logic with cardinalities. These cardinality constraints are predicates of the form $|X_1| + \ldots |X_r| < |Y_1| + \ldots |Y_s|$ over second-order sets X_i $(i = 1 \ldots r)$ and Y_j $(j = 1 \ldots s)$.

We can translate each *comp* predicate into a WS1S^{card} formula. In

$$\begin{aligned} comp_{A \gtrsim k}(X) \text{ iff } & \mathcal{Z}(X) \wedge \exists Y\, Y \subseteq X \wedge [\forall x(X(x) \wedge Q_\Sigma(x)) \rightarrow Y(x)] \wedge Q_\Sigma(Y) \\ & \wedge \exists Z\, Z \subseteq X \wedge [\forall x(X(x) \wedge Q_A(x)) \rightarrow Z(x)] \wedge Q_A(Z) \\ & \wedge |Z| \gtrsim k \cdot |Y| \end{aligned}$$

the first two lines are already expressible in MSO. Therefore we only have to encode $|Z| \gtrsim k \cdot |Y|$ in WS1S^{card} to transform our formula into WS1S^{card}. As we saw in chapter 3 on page 18 k is a fraction $k = \frac{l}{m}$ with $l, m \in \mathbb{N}$. Thus, $|Z| \gtrsim k \cdot |Y|$ can also be written as $m \cdot |Z| \gtrsim l \cdot |Y|$. Since m and l are constant, we can encode $m \cdot |Z|$ and $l \cdot |Y|$ by additions

$$\underbrace{|Z| \cdots + |Z|}_{m} \gtrsim \underbrace{|Y| \cdots + |Y|}_{l} \quad .$$

The constraint $m \cdot |Z| > l \cdot |Y|$ is then expressed via swapping the left and right side of the unequality so that we have $|Y| \cdots + |Y| < |Z| \cdots + |Z|$. The second case $m \cdot |Z| \geqslant l \cdot |Y|$ can be encoded as $\neg(|Z| + \cdots + |Z| < |Y| + \cdots + |Y|)$. Hence we are able to encode our formulas in WS1S^{card} but unfortunately WS1S^{card} is undecidable [KR03, page 682, item (i)]. However, the fragment [FO$\exists^*_{MSO}$FO] with arbitrary cardinality constraints ranging over the second-order variables quantified in the $\exists^*_{MSO}$ block is decidable [KR03, page 690, Theorem 16].

Therefore, we transform our formulas into Σ_1^1 with additional *comp* predicate. Please note that this does not directly imply decidability since the translation of each *comp* predicate into WS1S^{card} introduces two new second-order variables Y and Z. For the resulting class of formulas so far neither decidability nor undecidability results have been proven.

5.1 Eliminating second-order quantifiers encoding subwords

We use a trick, which has also been used by [EF95], to eliminate unnecessary MSO-quantifiers. By looking at our formulas, we can see that we usually use set variables to encode subwords. Set variables are able to express arbitrary combinations of positions, not just a sequence.

In fact, a subword is already characterised by its first and last position. Instead of invoking a new second-order variable $\underline{D}$ to define the domain of a subformula, we now use the two borders f and l.

This changes the encoding of subdomains presented in section 4.2 on page 26 to

$$\begin{array}{llll} \exists x & \varphi & \rightsquigarrow & \exists x \quad (f \leqslant x \wedge x \leqslant l) \quad \wedge\varphi \\ \forall x & \varphi & \rightsquigarrow & \forall x \quad (f \leqslant x \wedge x \leqslant l) \quad \rightarrow \varphi \\ \exists X & \varphi & \rightsquigarrow & \exists X \quad (\forall x X(x) \rightarrow (f \leqslant x \wedge x \leqslant l)) \quad \wedge\varphi \\ \forall X & \varphi & \rightsquigarrow & \forall X \quad (\forall x X(x) \rightarrow (f \leqslant x \wedge x \leqslant l)) \quad \rightarrow \varphi \end{array} \quad .$$

Before this change the case of an empty domain was implicitly treated via $\underline{D} = \varnothing$. Now we need to do the case distinction by hand for every translation in form except for the empty set and the choice $(+)$. Some expression can immediately decide wether the empty domain forms a sufficient model.

The case of an empty domain is represented by $\operatorname{form}\rho\,\varnothing$. Let τ and σ be arbitrary raes. The translations are

$$\begin{aligned} \operatorname{form}\varnothing\,\varnothing &= \exists x \neg(x = x) \\ \operatorname{form}\varepsilon\,\varnothing &= \forall x\, x = x \\ \operatorname{form}a\,\varnothing &= \exists x \neg(x = x) \\ \operatorname{form}\checkmark\,\varnothing &= \exists x \neg(x = x) \\ \operatorname{form}(\sigma+\tau)\,\varnothing &= (\operatorname{form}\sigma\,\varnothing) \vee (\operatorname{form}\tau\,\varnothing) \\ \operatorname{form}(\sigma.\tau)\,\varnothing &= (\operatorname{form}\sigma\,\varnothing) \wedge (\operatorname{form}\tau\,\varnothing) \\ \operatorname{form}(\tau)^*\,\varnothing &= \forall x\, x = x \\ \operatorname{form}(\tau)_{A\gtrsim k}\,\varnothing &= (\operatorname{form}\tau\,\varnothing) \end{aligned}$$

Let ρ be a check-closed rae. The top-level translation is done via

$$\begin{aligned} &[\neg\exists x\, x = x \wedge \operatorname{form}\rho\,\varnothing] \\ &\vee\, [\exists f(\forall x\, f \leqslant x) \wedge \exists l(\forall x\, x \leqslant l) \wedge \operatorname{form}\rho\,(f,l)] \end{aligned} \quad .$$

The translations in the cases where letters occur can be constructed using the changed encoding of subdomains. The following formulas already exploit several easier ways to express the facts by the help of the borders. For example, ε cannot be satisfied since there is at least one position f.

$$\begin{aligned} \operatorname{form}\varnothing\,(f,l) &= \exists x \neg(x = x) \\ \operatorname{form}\varepsilon\,(f,l) &= \exists x\, \neg(x = x) \\ \operatorname{form}a\,(f,l) &= f = l \wedge Q_a(f) \\ \operatorname{form}\checkmark\,(f,l) &= f = l \wedge Q_\checkmark(f) \\ \operatorname{form}(\sigma+\tau)\,(f,l) &= (\operatorname{form}\sigma\,(f,l)) \vee (\operatorname{form}\tau\,(f,l)) \\ \operatorname{form}(\sigma.\tau)\,(f,l) &= [\exists l_1\, \exists f_1\, f \leqslant f_1 \wedge (l_1 < f_1 \wedge \neg\exists w\, l_1 < w \wedge w < f_1) \wedge f_1 \leqslant l \\ &\qquad \wedge (\operatorname{form}\sigma\,(f,l_1)\,) \wedge (\operatorname{form}\tau\,({}_1f,l)\,)] \\ &\quad \vee\, [((\operatorname{form}\sigma\,(f,l)\,) \wedge (\operatorname{form}\tau\,\varnothing\,) \\ &\quad \vee\, [((\operatorname{form}\sigma\,\varnothing\,) \wedge (\operatorname{form}\tau\,(f,l)\,) \end{aligned}$$

The concatenation requires several case distinctions depending on wether ε is accepted by σ, τ or by both.

$$
\begin{aligned}
\operatorname{form}(\tau)^*(f,l) = &\exists S\,[\forall x\, S(x) \rightarrow (f \leqslant x \wedge x \leqslant l)] \\
&\wedge S(f) \\
&\wedge (\forall x \forall y (x < y \wedge S(x) \wedge S(y) \wedge \neg \exists z\, x < z \wedge z < y \wedge S(z)) \rightarrow \\
&\quad (\exists l_x\, x \leqslant l_x \wedge l_x < y \wedge (\neg \exists w\, l_x < w \wedge w < y) \wedge (\operatorname{form}\tau(x, l_x))) \\
&\wedge (\forall x\, last_S(x) \rightarrow \\
&\quad (\operatorname{form}\tau(x,l)) \\
\operatorname{form}(\tau)_{A\gtrsim k}(f,l) = &\exists C_{\checkmark}\,[\forall x\, C_{\checkmark}(x) \rightarrow (f \leqslant x \wedge x \leqslant l)] \\
&\wedge \forall x\, C_{\checkmark}(x) \\
&\quad \rightarrow (\exists X_0 \wedge X_0(x) \wedge \neg \exists y\, y > x \wedge X_0(y) \\
&\qquad \wedge X_0(f) \wedge \neg \exists y\, y < f \wedge X_0(y) \\
&\qquad \wedge \forall z\, (f \leqslant z \wedge z < x) \rightarrow X_0(z) \\
&\qquad \wedge comp_{A\gtrsim k}(X_0)) \\
&\wedge \operatorname{form}\tau(f,l)
\end{aligned}
$$

This leaves us with a set S for each Kleene operator and the sets $C_{\checkmark}$ and X_0 for every constraint.

5.2 Transforming *comp* and availability extensions

We can eliminate $C_{\checkmark}$ and X_0 completely by moving the responsibility to invoke the *comp* predicate from the constraint towards the check symbol. But therefore the predicate *comp* must be changed so that it works on FO variables and not on sets.

We define

$$
\begin{aligned}
comp_{A\gtrsim k}(f,l) \operatorname{iff} &\exists Y\,[\forall x\,(f \leqslant x \wedge x \leqslant l \wedge Q_\Sigma(x)) \rightarrow Y(x)] \wedge Q_\Sigma(Y) \\
&\wedge \exists Z\, Z \subseteq Y \wedge [\forall x (Y(x) \wedge Q_A(x)) \rightarrow Z(x)] \wedge Q_A(Z) \\
&\wedge |Z| \gtrsim k \cdot |Y| \quad ,
\end{aligned}
$$

where the formerly used $\mathcal{Z}$ predicate is no longer necessary, since by definition we use a connected set: the sequence from f to l. Furthermore the replacement of X by Y when defining Z does not have an effect. The difference between Y and X is that check positions are already projected away in Y while they still occur in X. Since Z implies a projection to $A \subseteq \Sigma$ check positions are anyway projected away. Therefore, we can exploit Lemma 5 on page 39 and see the following property.

Observation 2. *Let Σ be an alphabet, w a word over $\Sigma \cup \{\checkmark\}$ and f, l two first-order variables marking the left and right border of $dom(w)$. Then $comp_{A\gtrsim k}(f,l)$ holds iff $|\pi_A(w)| \gtrsim k \cdot |\pi_\Sigma(w)|$ holds.*

Instead of using a set X_0 when translating a constraint, we can now again use two borders. The first one is given as the first position in the subword the constraint is checked on. When we translate the constraint, we fix this position under the name $f_\checkmark$. In analogy to $C_\checkmark$, we fix $F_\checkmark$ onto ist name and exploit the shadowing this introduces. Later, when translating a check symbol, we use $f_\checkmark$ to define the left border for the application of the *comp* predicate, while the position where ✓ is placed is used as the right border.

For an arbitrary expression τ we have

$$\operatorname{form}(\tau)_{A \gtrsim k}(f,l) = \exists f_\checkmark \{ f = f_\checkmark \wedge \operatorname{form} \tau (f,l) \} \quad .$$

For technical reasons, we have to introduce a third parameter in form to pass the information which check constraint is checked down to the check symbols. The translation of a check constraint replaces the former used constraint by itself.

$$\operatorname{form}(\tau)_{A \gtrsim k}(f,l)(B \gtrsim l) = \exists f_\checkmark \{ f = f_\checkmark \wedge \operatorname{form} \tau (f,l)(A \gtrsim k) \} \quad .$$

To indicate that there is no surrounding constraint, we initialise the new parameter with $\varnothing$ on the top-level. The translation of a check symbol if the parameter is still not initialised is simply

$$\operatorname{form} \checkmark (f,l) \varnothing = (f = l) \wedge Q_\checkmark(f) \quad .$$

We need not check any constraint, since there is no constraint surrounding this check symbol.

However, in a check-closed expression we will never translate a check symbol while the parameter is still empty. Here we translate

$$\begin{aligned} \operatorname{form} \checkmark (f,l)(A \gtrsim k) = & (f = l) \wedge Q_\checkmark(l) \\ & \wedge comp_{A \gtrsim k}(f_\checkmark, l) \quad . \end{aligned}$$

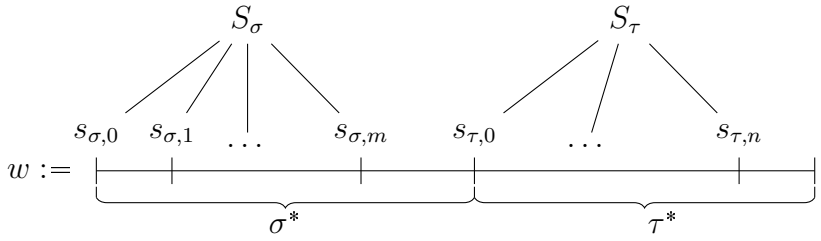
In all other expressions the new parameter is simply passed down and has no impact on the translation. By now, we can translate a regular availability expression into a formula in MSO with only existential quantifiers where only one variable S occurs for each Kleene star or Kleene plus, occuring in the expression.

5.3 Eliminating multiple sets of star positions

A Σ^1_1 formula consists of one block of existential quantifiers of MSO variables and a body where only FO quantification occurs. So far, our formula contains several wildly spread existentially quantified MSO variables which have been introduced by the Kleene operators on the different levels.

We show by induction that we can lift the property of being a Σ_1^1 formula onto the next level. A sketch of the proof for the Kleene plus was given in [EF95, page 112]. It exploits the disjointness of the scope of Kleene operators on different subexpressions.

We illustrate the use of this property in the following example. Consider two expressions, σ^* and τ^*, which can be joined to form the expression $\rho = (\sigma^*).(\tau^*)$:



We can join the sets S_σ and S_τ so that they form a new set S which ranges over the union of the intervals. We make sure that the new set behaves like both sets behaved before by limiting the subformulas to the old sets via the borders of the subdomains. For example, what we claimed for the last element in S_σ before, we now claim for the last element in S, that is within (σ^*)'s domain.

Theorem 5. *Every check-closed regular availability expression is expressible via a Σ_1^1 formula with additional comp predicates for FO.*

Proof. In the previous sections we introduced techniques to replace second-order variables encoding sequences by FO variables which encode the borders of the sequence and saw in Fact 2 on page 43 that *comp* can also be adapted to FO without changing its meaning. A proof in analogy to Lemma 4 on page 32 can be done by a canonical structural induction. This time $f_\checkmark$ is a free variable which needs an interpretation. However, the elaborate investigation of the different sets $free(w, \rho)$ is no more necessary, since $f_\checkmark$ always encodes the first position in the surrounding constraint.

Also, an argumentation in analogy to Lemma 5 on page 39 can be given for the adapted top-level translation.

We show by induction that the remaining second-order quantifiers can be brought to the front so that the resulting formula is in Σ_1^1 form.

basis :

Since none of the non-recursive regular availability expressions contains an MSO variable, they are all in the Σ_1^1 form.

induction hypothesis :

Let σ and τ be regular availability expressions for which we assume that they can be transformed into the Σ_1^1 formulas $\exists S_{\sigma,1} \ldots \exists S_{\sigma,n} \chi_1$ and $\exists S_{\tau,1} \ldots \exists S_{\tau,m} \chi_2$, with $m, n \in \mathbb{N}$ and χ_1 and χ_2 formulas,where no MSO quantification occurs.

inductive step :

$\sigma \vee \tau$:

We assume that $m \leqslant n$, which is no restriction since it can always be reached via swapping of σ and τ, using the commutativity of $+$.

We transform $(\exists S_{\sigma,1} \dots \exists S_{\sigma,n} \chi_1) \vee (\exists S_{\tau,1} \dots \exists S_{\tau,m} \chi_2)$ into $\exists S_{\sigma,1} \dots \exists S_{\sigma,n} \chi_1 \vee \chi_2'$, where χ_2' is χ_2, only that it adresses the variables $S_{\sigma,1}$ to $S_{\sigma,m}$ instead of $S_{\tau,1}$ to $S_{\tau,m}$. If there are further variables $S_{\sigma,m+1}$ to $S_{\sigma,n}$, they are not used in χ_2', just as they were not used in χ_2.

$\sigma.\tau$:

This translation uses the fact that the different sets in the translation of σ and τ can be melted as long as their evaluation is restricted to the corresponding part of the new set. This fact was illustrated in 5.3 on the previous page and has its reason in the disjointness of the domains for the expression for σ and for τ. Depending on wether $m \leqslant n$ or $n < m$, we use the larger number, which we now call k and construct a formula $\exists S_1 \dots \exists S_k \varphi \chi_1'[x_1, y_1] \wedge \chi_2'[x_2, y_2]$. The formula φ is a FO formula to determine the borders x_1, x_2, y_1 and y_2 for the two subdomains for the word matching σ and the word matchingτ. This reflects the translation of the concatenation for two existing subwords in the previous section. the formulas χ_1' and χ_2' are the same as χ_1 and χ_2, but address S_1 to S_k and are limited to the borders that φ implies. The several cases of the concatenation where one of the words has an empty domain is not studied closer, since the resulting formula for the empty case is always FO, since the Kleene star is immediately translated into a tautology and cannot introduce new sets S_i. The various choices ($\vee$) introduced can be handled in the same way than those introduced by a choice in a rae ($+$).

τ^* :

Following our translation, we introduce a new variable S in the front, then there are φ_1 and φ_2, FO formulas, which characterise the borders for the intermediate subwords and the last subword satisfying τ. We transform $\exists S \varphi_1 (\exists S_1 \dots \exists S_m \chi_2) \varphi_2 (\exists S_1 \dots \exists S_m \chi_2)$ into $\exists S \exists S_1 \dots \exists S_m \varphi_1 \chi_2 \varphi_2 \chi_2$ in a generalisation of the concatenation, again using the disjointness of the range of the sets S_1 to S_m with their equally named successors within the formula.

$(\tau)_{A \gtrsim k}$:

The formula

$$\operatorname{form}(\tau)_{A \gtrsim k}(f, l) = \exists f_\checkmark \{ f = f_\checkmark \wedge \operatorname{form} \tau (f, l) \}$$

is by induction hypothesis equivalent to a formula

$$\operatorname{form}(\tau)_{A \gtrsim k}(f, l) = \exists f_\checkmark \{ f = f_\checkmark \wedge \exists S_1 \dots \exists S_n \varphi \} \quad ,$$

where φ is a Σ_1^1 body. We can transform this formula into

$$\operatorname{form}(\tau)_{A \gtrsim k}(f, l) = \exists S_1 \dots \exists S_n \exists f_\checkmark \{ f = f_\checkmark \wedge \varphi \} \quad ,$$

since all the sets and $f_\checkmark$ are existentially quantified and therefore the ordering is not important.

□

6 Conclusion

We have seen that the acceptance of a word in a rae is a non-deterministic process which can produce different words in the language of the rae out of one accepted word. However, for check-closed expressions the word in the language is uniquely defined for each accepted word.

As main result we established a language equality between the languages of check-closed raes and availability formulas constructed with the help of form. An extension onto arbitrary raes would be easily possible by a slight modification of the top-level-translation and a change in the definition of the language accepted by an availability formula. However, further studies in the non-deterministic setting are not of our concern now. Still, they might be of interest to understand the non-deterministic acceptance of the availability automata and the differences in the expressions which lead to deterministic automata and those for which the corresponding automaton is inherently non-deterministic.

Currently, our main concern is the question whether the emptiness problem for one given rae is decidable. The transformation of the formulas which characterise raes into Σ_1^1 formulas with assigned *comp* predicates marks one important step towards the answer of the question.

For the use of *comp* as a FO predicate we have seen that we can easily encode our formulas in W1S1^{card} introduced by [KR03], but unfortunately the resulting formula falls into a class neither decidability nor undecidability results have been proven for. It is an interesting task to evaluate this class in extension of the notions and proofs proposed. However, even the sets introduced by the translation of the *comp* predicate into WS1S^{card} are fixed onto one meaning by the FO parameters for *comp*. Therefore a direct encoding of the *comp* predicate seems of interest, too. Inspired by [SSMH04] we will try to encode the cardinality constraints represented by the *comp* predicates on FO in Presburger formulas to solve the problem.

Another interesting question is the search for a decidable fragment of the availability logic or an encoding of fragments into linear temporal logic (LTL) [Var08] in order to do model checking.

References

[DG07] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1-2):69 – 86, 2007. Automata, Languages and Programming.

[DKV09] Manfred Droste, Werner Kuich, and Heiko Vogler, editors. *Handbook of weighted automata*. Monographs in Theoretical Computer Science, an EATCS Series. Springer, 2009.

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Perspectives in Mathematical logic. Springer, 1995.

[HMO10] J. Hoenicke, R. Meyer, and E.-R. Olderog. Kleene, rabin, and scott are available. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory (CONCUR)*, volume 6269 of *Lecture Notes in Computer Science*, pages 462–477. Springer, 2010.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley series in computer science. Addison-Wesley Publishing Company, 1979.

[KR03] Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In *Proceedings of the 30th international conference on Automata, languages and programming*, ICALP'03, pages 681–696, Berlin, Heidelberg, 2003. Springer.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science, an EATCS Series. Springer, 2004.

[Sch08] Uwe Schöning. *Theoretische Informatik – kurz gefasst*. HochschulTaschenbuch. Spektrum, Akademischer Verlag, 2008.

[SSMH04] Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In *Proceedings ICALP*, LNCS, pages 1136–1149. Springer, 2004.

[Tho97] Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of formal languages: Beyond words*, volume 3, pages 389–455. Springer, 1997.

[Tri82] Kishor S. Trivedi. *Probability and statistics with reliability, queuing, and computer science applications*. Prentice Hall, 1982.

[Var08] Moshe Y. Vardi. From church and prior to psl. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*, pages 150 – 171. Springer, 2008.