Bachelor's Thesis Computing Science

Feature-based Randomness Constraints in Control Improvisation

Stefan Boneschanscher
s1011532

August 16, 2022

First supervisor/assessor:
dr. Sebastian Junges

Second assessor:
dr. Jurriaan Rot

Radboud University

Abstract

We introduce a feature-based randomness constraint to the control improvisation problem. The goal of control improvisation is to find an improviser that creates improvisations according to some specifications. The feature constraint limits the chance with which a word with specific features can occur. We give a definition for our expanded version of control improvisation, show that a general solution for the expanded definition requires exponential time, identify some cases in which a polynomial time solution can be found, and finally, provide an algorithm that can solve certain instances of the expanded problem in polynomial time.

Contents

Chapter 1

Introduction

Can a computer improvise? For some people this is a philosophical question. Can we call the output a computer gives after receiving a set of instructions an improvisation? We will leave that up to the reader to decide. Instead, this paper focuses on the computational challenge of getting a computer to improvise.

In *Control Improvisation* [5], Fremont et al. tried to capture this challenge as a formal problem: given a specification language and a set of constraints-, create an improviser that creates words in said language while adhering to this set of constraints. Inspired by *Machine Musicianship* [8] (in which Rowe researched computer generation of variations on a reference melody), Fremont et al. identified three key ideas of improvisation: an improvisation should adhere to a set of rules (e.g. musical conventions), it should bear resemblance to the reference it is based on (e.g. the reference melody) while not being an exact copy, and it should be random enough to avoid repetition when creating multiple improvisations.

It is difficult for a problem to capture the complete reality of this world, but control improvisation is a great attempt at capturing the idea of improvisation as a computational problem. However, while two variations on the same melody may both adhere to the constraints and be equally alike to the reference melody, we may still prefer one over the other. For example, one melody may be very difficult to play on an instrument while the other is easier.

Creating an improviser that limits the probability of generating difficult variations is not a possibility that is captured in the definition by Fremont et al. In this paper, we expand the definition of CI to include a *feature constraint (FC)*. The constraint evaluates the features of a possible improvisation and may lower the probability of this improvisation to occur. This allows us, going back to the example of music, to lower the likelihood of improvisations with a large number of difficult notes.

One of the features offered by CI as defined by Fremont et al. is its

efficiency. In their 2017 paper they show that a polynomial-time general algorithm exists to create improvisers for certain types of specifications. This algorithm is able to run in polynomial time as only a limited number of operations on the specifications are needed. These operations include counting the number of words in the specification and random sampling from the specification.

We show that we can solve CI with certain types of feature constraints using only a limited amount of these operations. For these cases we provide a polynomial-time algorithm. For other types of feature constraints we need to access each word individually. In that case the previously used operations may need to be applied an exponential number of times. We provide a general exponential-time algorithm that finds an improviser for CI problems with any feature constraint and show that a faster than exponential-time algorithm cannot exist for these cases.

As an understanding of the original CI definition helps to understand the problem presented in this paper, we will first discuss the definition by Fremont et al. and provide some examples. Then we will incorporate FC into that definition and provide the aforementioned algorithms. We will conclude with a discussion of the implications of our findings and possible options for future work.

The paper is structured as follows: First, we provide basic notation and discuss earlier work to familiarize the reader with CI in Ch. 2. In Ch. 3 Sec. 3.1-3.3, we continue by expanding the definition of CI to include a FC, then show CI-FC requires exponential time to guarantee a solution and suggest two possible algorithms for solving CI-FC. Sec. 3.4 provides two examples where CI-FC can be solved in polynomial time, lists conditions where a polynomial-time solution can be guaranteed and provides an algorithm for solving CI-FC instances that satisfy these conditions. Ch. 4 discusses related work. We finally draw a conclusion and discuss possible future work in Ch. 5.

In short, the contributions of this paper are:

- An expanded definition of control improvisation including a feature-based randomness constraint (CI-FC).
- A proof showing that finding a solution for a CI-FC instance while treating the FC as a black-box at least requires exponential time.
- Two variations of an exponential-time algorithm for solving CI-FC.
- A list of conditions under which CI-FC can be solved in polynomial time.
- A polynomial-time algorithm for solving CI-FC instances that satisfy these conditions.

Chapter 2

Preliminaries

In this chapter we will list notation that is frequently used in the paper and explain what the control improvisation problem is. Most of the definitions and proofs related to the original problem are taken from *Control Improvisation* [5] where the original problem is defined. We limit ourselves to the parts of CI that are relevant for this paper and provide new examples as well as a more comprehensive explanation of the proofs.

2.1 Notation:

The notation in Table 2.1 is used throughout the paper. These definitions are taken from or based on the definitions in *Discrete Mathematics and Its Applications* by Rosen [7].

$A \cup B$	$= \{x \mid x \in A \vee x \in B\}$, the *union* of two sets.
$A \cap B$	$= \{x \mid x \in A \wedge x \in B\}$, the *intersection* of two sets.
$A \setminus B$	$= \{x \mid x \in A \wedge x \notin B\}$, the *difference* of two sets.
$S_1 \subseteq S$	A set S_1 is said to be a *subset* of S if every element of S_1 is also an element of S
$\lvert S \rvert$	the size of a set S.
$\mathbb{Z}$	$\{\dots, -2, -1, 0, 1, 2, \dots\}$, the set of *integers*.
$\mathbb{Q}$	$= \{p/q \mid p \in \mathbb{Z}, q \in \mathbb{Z}, \text{ and } q \neq 0\}$, the set of all *rational numbers*,
$\mathbb{R}$	the set of *real numbers*
Σ	an alphabet.
$\Sigma^{\leq n}$	$\cup_{0 \leq i \leq n} \Sigma^i$
$\lvert w \rvert$	= the length of a word $w \in \Sigma^*$.
DFA	Deterministic Finite Automata

Table 2.1: Frequently used notation.

Because languages can be represented in many different forms, with varying complexities for each form, we must limit the scope of the problem to specific representations. Fremont et al. provide the following definition for a language *specification* which we will use:

Definition 1. A *specification* $\mathcal{X}$ is a finite representation of a language $\mathcal{L}(\mathcal{X})$ over a finite alphabet Σ. For example, $\mathcal{X}$ could be a finite automata or a context-free grammar.

Example 1. Over the alphabet $\Sigma = \{a, b\}$ we can create the specification $\mathcal{X}$ shown in Fig. 2.1. Where $\mathcal{X}$ is a DFA that accepts the language $\mathcal{L}(\mathcal{X}) = (a + b)^* a$:

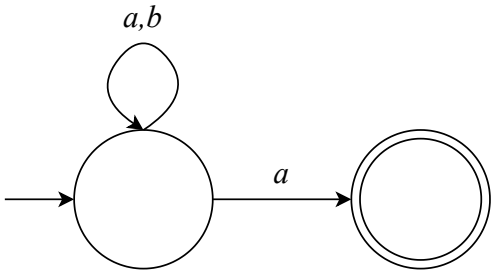

Figure 2.1: DFA: $\mathcal{X}$

2.2 Control improvisation

The control improvisation problem tries to capture the challenge of creating an improviser that can generate words that adhere to the three key concepts of improvisation mentioned by Fremont et al.:

- A set of rules that each improvisation should adhere to, for example musical conventions or the syntax of the input for a system. These rules are described in the *hard specification*, that describes the language that each improvisation should be in.
- A reference which a part of the improvisations should resemble. This reference is given in the *soft specification*, that describes the language that most improvisations should be in, except for an allowed margin of error ϵ.
- Something that ensures the improvisations are random. This is guaranteed by two probability constraints: a minimum probability λ, and a maximum probability ρ. The probability of any improvisation i being generated by the improviser should be between these limits: $\lambda \leq \Pr[i] \leq \rho$.

Fremont et al. use the hard and soft specification to define two sets of words. First, a set of words that are in the language of the hard specification, which they call *improvisations*. Second, a subset of those improvisations that are also in the language of the soft specification, which they call *admissible improvisations*. A specification which is finite may still represent an infinite number of words. To ensure the number of improvisations is finite a length restriction is imposed on the words added to the set of improvisations. This length bound is given by the numbers $m, n \in \mathbb{R}$ where m is the minimal length and n the maximal length of an improvisation. The improvisations and admissible improvisations are formally defined by Fremont et al. as:

Definition 2. Fix a *hard specification* $\mathcal{H}$, a *soft specification* $\mathcal{S}$, and length bounds $m, n \in \mathbb{N}$. An *improvisation* is any word $w \in \mathcal{L}(\mathcal{H})$ such that $m \leq |w| \leq n$, and we write I for the finite set of all improvisations. An improvisation $w \in I$ is *admissible* if $w \in \mathcal{L}(\mathcal{S})$, and we write A for the finite set of all admissible improvisations.

Example 2. We want to create improvisations based on the word *abba*. As alphabet we use $\Sigma = \{a, b\}$. As hard specification we pick the DFA $\mathcal{H}$ that accepts words containing exactly two b's. As our soft specification we pick the DFA $\mathcal{S}$ that accepts words with two consecutive b's. We pick the length constraints $m = n = 4$. Now our set of improvisations are all words $w \in \mathcal{L}(\mathcal{H})$, with $|w| = 4$. This gives the set $I = \{aabb, abab, baab, abba, baba, bbaa\}$. Now to find the set of admissible improvisations we take $A = I \cap \mathcal{L}(\mathcal{S}) = \{aabb, abba, bbaa\}$.

We now have these sets of improvisations, but not a method for sampling from these sets. This brings us to the second and third concept. We want at least $1 - \epsilon$ of all samples to come from A, and we want the probabilities of a sample occurring to be between the bounds λ and ρ. A distribution that achieves these goals for a CI instance is called an *improvising distribution*. The formal definition of this provided by Fremont et al. is:

Definition 3. Given $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, \epsilon, \lambda, \rho)$ with $\mathcal{H}, \mathcal{S}, m$ and n as in Definition 2, $\epsilon \in [0, 1] \cap \mathbb{Q}$ an error probability, and $\lambda, \rho \in [0, 1] \cap \mathbb{Q}$ probability bounds, a distribution $D : \Sigma^* \to [0, 1]$ is an *improvising distribution* if it satisfies the following requirements:

Hard constraint:	$\Pr[w \in I \mid w \leftarrow D] = 1$
Soft constraint:	$\Pr[w \in A \mid w \leftarrow D] \geq 1 - \epsilon$
Randomness:	$\forall w \in I,\ \lambda \leq D(w) \leq \rho$

If such an improvising distribution exists, we say $\mathcal{C}$ is *feasible*. An *improviser* for a feasible $\mathcal{C}$ is an expected finite-time probabilistic algorithm whose output distribution is an improvising distribution.

Example 3. We want to create an improvising distribution for our improvisations from *Example 2*. We pick $\lambda = 1/10$, $\rho = 1/5$ and $\epsilon = 1/4$. We find

that when we assign the maximal probability $\rho = 0.2$ to each word in A we can still not satisfy the soft constraint because $3 \cdot 1/5 \not\geq 1 - 1/4$. To remedy this problem, we decide to increase the maximum to $\rho = 1/4$. We assign the probability $1/4$ to all words in A to satisfy the soft constraint. Now a distribution that satisfies the randomness constraint cannot exist, because even when assigning the remaining words in I the minimal probability λ, the sum of the probabilities is greater than one. We can lower the minimum $\lambda = 3/40$. This allows for a distribution where the words in $I \setminus A$ are assigned a probability of $1/12$.

Now that we have defined what an improviser is we can finally define the complete problem. We will use the definition provided by Fremont et al.:

Definition 4. Given $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, \epsilon, \lambda, \rho)$, the *control improvisation (CI)* problem is to decide whether $\mathcal{C}$ is feasible, and if so to generate an improviser for $\mathcal{C}$. The *size* $|\mathcal{C}|$ of a CI instance is the total size of the bit representation of its parameters, treating m and n as being represented in unary[1] and all other numerical parameters in binary.

The definition of what a specification can be is quite broad. Solving a CI instance where the specifications are given as a non deterministic finite automaton (NFA) or context free grammar (CFG) may be more complex than one where the specifications are given as a DFA. To make a clear distinction between instances with different specifications, Fremont et al. define different classes of CI instances:

Definition 5. If $\mathcal{A}$ and $\mathcal{B}$ are classes of specifications, $CI(\mathcal{A}, \mathcal{B})$ is the class of CI instances $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, \epsilon, \lambda, \rho)$ where $\mathcal{H} \in \mathcal{A}$ and $\mathcal{S} \in \mathcal{B}$. When discussing decision problems, we use the same notation for the feasibility problem associated with the class (i.e. given $\mathcal{C} \in CI(\mathcal{A}, \mathcal{B})$, decide whether it is feasible).

In our research we will limit ourselves to CI instances of the class $CI(DFA, DFA)$. This is the class where both the hard specification $\mathcal{H}$ and the soft specification $\mathcal{S}$ are represented as DFA.

2.3 Solving control improvisation

To solve a CI instance we must first check feasibility. If the instance is feasible, we must construct an improviser. Fremont et al. found that the feasibility of a CI instance depends entirely on the ratio between the number of improvisations $|I|$, admissible improvisations $|A|$, and λ, ρ and ϵ. They captured this idea in the following theorem:

[1] The unary representation of a number is a string of 1's with length equal to the number. For example $5 = 11111$. By choosing this representation, the size of the input will grow linearly with an increase of the length bounds.

Theorem 1. *For any $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, \epsilon, \lambda, \rho)$, the following are equivalent:*

1. *$\mathcal{C}$ is feasible.*
2. *The following inequalities hold* [2]*:*
   - *(a)* $1/\rho \leq |I| \leq 1/\lambda$
   - *(b)* $1-\epsilon/\rho \leq |A|$
   - *(c)* $|I| - |A| \leq \epsilon/\lambda$

The intuition behind each of these inequalities is a follows:

(a) ρ must be greater than $1/|I|$, because when ρ is smaller, the sum of all probabilities can never equal 1. Following the same logic, λ must never be smaller than $1/|I|$, because then the sum of all probabilities would always exceed 1.

(b) ρ must be greater than $1-\epsilon/|A|$, because when ρ is smaller, the sum of the $\Pr[w]$ for all $w \in A$ could never exceed $1-\epsilon$ and thus the allowed error rate of the soft constraint would be exceeded.

(c) Using the same logic as in (b), λ should never be greater than $\epsilon/|I|-|A|$, because even if each word in $I \setminus A$ would be assigned the minimal probability, its sum would exceed ϵ and thus exceed the allowed error rate again.

The proof of this theorem is relevant, because we will apply a similar strategy in one of our own proofs in the next chapter. Furthermore, the second part of the proof offers a generic procedure for solving CI. For these reasons we will show an annotated version of the abridged proof.

Proof.
(1. $\rightarrow$ 2.) : We show that if $\mathcal{C}$ is feasible, the inequalities must hold. If $\mathcal{C}$ is feasible there exists an improvising distribution D for $\mathcal{C}$. Because D is an improvising distribution, it satisfies the hard constraint which we can write as:

$$\Pr[w \in I | w \leftarrow D] = \sum_{w \in I} D(w) = 1$$

Because D satisfies the randomness constraint, we know that:

$$\forall w \in I,\ D(w) \leq \rho, \text{ so: } 1 = \sum_{w \in I} D(w) \leq \sum_{w \in I} \rho$$

[2] If $\lambda = 0$, we treat division by zero as yielding ∞, so that both the inequalities involving λ are trivially satisfied. This makes sense, as $\lambda = 0$ means there is no lower bound on the probabilities of improvisations.

Further, we have:

$$\sum_{w \in I} \rho = \rho \cdot |I|$$

So we can conclude:

$$1 \leq \rho \cdot |I| \text{ and } 1/\rho \leq |I|$$

Applying the same strategy for λ:

$$\Pr[w \in I | w \leftarrow D] = \sum_{w \in I} D(w) = 1$$

$$\forall w \in I,\ D(w) \geq \lambda, \text{ so: } 1 = \sum_{w \in I} D(w) \geq \sum_{w \in I} \lambda$$

$$\sum_{w \in I} \lambda = \lambda \cdot |I|$$

$$1 \geq \lambda \cdot |I| \text{ and } 1/\lambda \geq |I|$$

Because $A \subseteq I$, the randomness constraint also holds for A. Because of this, we can apply the same strategy while now using the soft constraint:

$$\Pr[w \in A | w \leftarrow D] = \sum_{w \in A} D(w) \geq 1 - \epsilon$$

$$\forall w \in A,\ D(w) \leq \rho, \text{ so: } 1 - \epsilon \leq \sum_{w \in A} D(w) \leq \sum_{w \in A} \rho$$

$$\sum_{w \in A} \rho = \rho \cdot |A|$$

$$1 - \epsilon \leq \rho \cdot |A| \text{ and } |A| \geq (1 - \epsilon)/\rho$$

For the final inequality we use both the hard constraint and the soft constraint:

$$\Pr[w \in I | w \leftarrow D] - \Pr[w \in A | w \leftarrow D] = \Pr[w \in I \backslash A | w \leftarrow D] \leq 1 - (1 - \epsilon) = \epsilon$$

$$\Pr[w \in I \setminus A | w \leftarrow D] = \sum_{w \in I \backslash A} D(w)$$

$$\forall w \in I \setminus A,\ D(w) \geq \lambda, \text{ so: } \sum_{w \in I \backslash A} D(w) \geq \sum_{w \in I \backslash A} \lambda$$

$$\sum_{w \in I \backslash A} \lambda = \lambda \cdot |I \setminus A|$$

$$\epsilon \geq \lambda \cdot |I \setminus A| \text{ and } |I| - |A| \leq \epsilon/\rho$$

(2. $\rightarrow$ 1.): We show that, if the inequalities hold, we can construct an improviser. First, we find the lowest error margin possible and call this ϵ_{opt}:

$$\epsilon_{opt} = \max(1 - \rho|A|, \lambda|I \setminus A|)$$

This is either $1 - \rho|A|$, the probability left after assigning all words in A the maximum probability, or $\lambda|I \setminus A|$, the probability needed to assign the minimum probability to all words not in A. Using inequality $2a$ we know $|I \setminus A| \leq |I| \leq 1/\lambda$ which means $0 \leq \epsilon_{opt} \leq 1$.

We use ϵ_{opt} to define the distribution D on I, which picks uniformly from A with probability $1 - \epsilon_{opt}$ and otherwise picks uniformly from $I \setminus A$. This distribution is well defined, because if $A = \emptyset$ then $\epsilon_{opt} = 1$, and if $I \setminus A = \emptyset$ then $\rho \cdot |A| = \rho \cdot |I| \geq 1$ and $\lambda \cdot |I \setminus A| = 0$ so $\epsilon_{opt} = 0$. From the definition of D it follows that D satisfies the hard constraint.

If $\epsilon_{opt} = 1 - \rho \cdot |A|$ we can rewrite this using inequality $2b$:

$$\epsilon_{opt} = 1 - \rho \cdot |A| \leq 1 - \rho((1 - \epsilon)/\rho) = \epsilon$$

Otherwise $\epsilon_{opt} = \lambda|I \setminus A|$, in which case we rewrite using inequality $2c$:

$$\epsilon_{opt} = \lambda|I \setminus A| \leq \lambda(\epsilon/\lambda) = \epsilon$$

In either case $\epsilon_{opt} \leq \epsilon$, and from our definition of D we know $\Pr[w \in A | w \leftarrow D]$ $= 1 - \epsilon_{opt} \geq 1 - \epsilon$, so D satisfies the soft constraint.
For any word $w \in A$, we can conclude:

$$D(w) = (1 - \epsilon_{opt})/|A| \leq (1 - (1 - \rho \cdot |A|))/|A| = \rho$$

If $\epsilon_{opt} = 1 - \rho \cdot |A|$, then $D(w) = \rho \geq \lambda$;
otherwise $\epsilon_{opt} = \lambda|I \setminus A|$, so:

$$D(w) = (1-\lambda \cdot |I \setminus A|)/|A| = (1-\lambda|I|+\lambda|A|)/|A| \geq (1-1+\lambda \cdot |A|)/|A| = \lambda$$

Thus $D(w) \geq \lambda$.

With a similar strategy for any word $w \in I \setminus A$ we can conclude:

$$D(w) = \epsilon_{opt}/|I \setminus A| \geq \lambda \cdot |I \setminus A|/|I \setminus A| = \lambda$$

If $\epsilon_{opt} = 1 - \rho \cdot |A|$, then:

$$D(w) = (1-\rho \cdot |A|)/|I \setminus A| = (1-\rho \cdot |A|)/(|I|-|A|) \leq (1-\rho \cdot |A|)/((1/\rho)-|A|) = \rho$$

otherwise $\epsilon_{opt} = \lambda \cdot |I \setminus A|$, so:

$$D(w) = (\lambda \cdot |I \setminus A|)/|I \setminus A| = \lambda \leq \rho$$

Therefore, for any $w \in I$ we always have $\lambda \leq D(w) \leq \rho$, thus D satisfies the randomness requirement.

This shows that D is an improvising distribution. Since it has finite support and rational probabilities, there is an expected finite-time probabilistic algorithm sampling from it, and this algorithm is an improviser for $\mathcal{C}$. □

The proof of **Theorem 1** provides us not only with inequality equations to check feasibility, it also provides us with a set of steps that we can follow to create an improviser. These steps can be completed using only the following operations on specifications as defined by Fremont et al.:

Intersection:	Given two specifications $\mathcal{X}$ and $\mathcal{Y}$, compute a specification $\mathcal{Z}$ such that $\mathcal{L}(\mathcal{Z}) = \mathcal{L}(\mathcal{X}) \cap \mathcal{L}(\mathcal{Y})$.
Difference:	Given two specifications $\mathcal{X}$ and $\mathcal{Y}$, compute a specification $\mathcal{Z}$ such that $\mathcal{L}(\mathcal{Z}) = \mathcal{L}(\mathcal{X}) \setminus \mathcal{L}(\mathcal{Y})$.
Length Restriction:	Given a specification $\mathcal{X}$ and $m, n \in \mathbb{N}$ in unary, compute a specification $\mathcal{Y}$ such that $\mathcal{L}(\mathcal{Y}) = \{\mathcal{L}(\mathcal{X}) \mid m \leq \lvert w \rvert \leq n\}$.
Counting:	Given a specification $\mathcal{X}$ and a bound $n \in \mathbb{N}$ in unary on the length of strings in $\mathcal{L}(\mathcal{X})$ (which is therefore finite), compute $\lvert\mathcal{L}(\mathcal{X})\rvert$.
Uniform sampling:	Given a specification $\mathcal{X}$ and a bound $n \in \mathbb{N}$ in unary on the length of strings in $\mathcal{L}(\mathcal{X})$, sample uniformly at random from $\mathcal{L}(\mathcal{X})$.

How can we measure the quality of this procedure or any other procedure for solving CI? Fremont et al. identify three key aspects for this measurement:

- correctness, a scheme that solves a CI instance should provide an improviser for that instance if it is feasible, and if it is not feasible it should not;
- scheme efficiency, the runtime of the scheme itself;
- improviser efficiency, the runtime of the improviser created by the scheme.

It may be possible to create a scheme with a very low runtime that leaves the largest part of the work to the improviser. This would result in an improviser with a very high runtime. Because of this possibility it is important to consider both the runtime of the scheme and the runtime of the improviser when talking about the runtime of the scheme. Using these three aspects Fremont et al. define the term *polynomial-time improvisation scheme* for a correct scheme where the runtime of both the scheme and the improviser is polynomial.

Definition 6. A *polynomial-time improvisation scheme* for a class $\mathcal{P}$ of CI instances is an algorithm S with the following properties:

Correctness:	For any $\mathcal{C} \in \mathcal{P}$, if $\mathcal{C}$ is feasible then $\mathcal{S}(\mathcal{C})$ is an improviser for $\mathcal{C}$, and otherwise $\mathcal{S}(\mathcal{C}) = \perp$.
Scheme efficiency:	There is a polynomial $p : \mathbb{R} \to \mathbb{R}$ such that the runtime of S on any $\mathcal{C} \in \mathcal{P}$ is at most $(p\|\mathcal{C}\|)$.
Improviser efficiency:	There is a polynomial $q : \mathbb{R} \to \mathbb{R}$ such that for every $\mathcal{C} \in \mathcal{P}$, if $G = S(\mathcal{C}) \neq \perp$ then G has an expected runtime of at most $(q\|\mathcal{C}\|)$.

The general procedure described above uses only five operations on specifications. If these five operations can be done in polynomial time, then this procedure is a polynomial-time improvisation scheme. These operations when applied to either two DFA's or a DFA and a context free grammar can be computed in polynomial time as Fremont et al. [5] have shown. This means that the procedure is a polynomial-time improvisation scheme for CI instances of the type $CI(\mathcal{DFA}, \mathcal{DFA})$.

Chapter 3

Control Improvisation with Feature Constraints

3.1 Expanding control improvisation

In this section we will expand the control improvisation problem that was defined in the previous chapter. In the original problem we saw the possibility to use the soft specification to describe "desired" words which should occur at least some of the time. We could also describe required behaviour using the hard specification. These two specifications are the only control offered on the frequency of different words produced by the improviser. To allow for more control over the features of the generated words we will add the *feature constraint*. This constraint limits the probability with which a specific word occurs based on its features.

For example, imagine a surveillance robot that patrols an area. This robot can accelerate, brake, and steer left and right. Now imagine using CI to improvise the behaviour of the robot. While we do want to allow the robot to brake at any time, we might also want to discourage improvisations which contain many brakes. Frequent braking and accelerating requires more energy and would cause the robot to make progress more slowly. In this case we may want to use a feature constraint that limits the occurrence of improvisations that contain many brakes.

To apply the feature constraint we need a *feature constraint function* that evaluates a word and returns a probability that we can then apply as an upper bound on the likelihood of that word occurring. We define this function as follows:

Definition 7. A feature constraint function is a function: $fc : \Sigma^* \to \mathbb{Q} \cap [0, 1]$. The function takes a word $w \in \mathcal{L}(\mathcal{H})$ and produces a probability $fc(w) \in [0, 1] \cap \mathbb{Q}$.

Example 4. The feature constraint function for our robot could look like

this: $fc(w) = 1/(\#_b(w)+1)$. Here w is the word, i.e. the instructions for the robot and $\#_b(w)$ is a function that counts how frequently 'brake' (abbreviated to 'b') occurs in w. Another possible constraint function could limit the probability for words with more than five brakes: $fc(w) = \text{ if } (\#_b(w) > 5) : 0.05, \text{ else} : 0.1$.

We now expand the definition of CI using our feature constraint function. The expanded version adds a new parameter: fc, the feature constraint function, and a new constraint: $\forall w \in I,\ D(w) \leq fc(w)$, the actual feature constraint that applies fc on a probability distribution D. This results in the following expanded definition of an improvising distribution, which includes the feature constraint:

Definition 8. Given $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, fc, \epsilon, \lambda, \rho)$ with $\mathcal{H}, \mathcal{S}, m$ and n as in Definition 2, *fc as in Definition 7*, $\epsilon \in [0,1] \cap \mathbb{Q}$ an error probability, and $\lambda, \rho \in [0,1] \cap \mathbb{Q}$ probability bounds, a distribution $D : \Sigma^* \to [0,1]$ is an improvising distribution if it satisfies the following requirements:

Hard constraint:	$\Pr[w \in I \mid w \leftarrow D] = 1$
Soft constraint:	$\Pr[w \in A \mid w \leftarrow D] \geq 1 - \epsilon$
Randomness:	$\forall w \in I,\ \lambda \leq D(w) \leq \rho$
Feature constraint:	$\forall w \in I,\ D(w) \leq fc(w)$

An improviser for $\mathcal{C}$ is a finite-time probabilistic algorithm with an output distribution that is such an improvising distribution. The *control improvisation problem with feature constraints* (*CI-FC*) is to decide whether $\mathcal{C}$ is feasible, and if so to generate an improviser for $\mathcal{C}$.

3.2 Solving control improvisation with a feature constraint

The method created for the original CI problem cannot solve CI-FC. In Example 5 we show some of the problems that arise when attempting to apply this method. In the rest of Sec. 3.2 we will show that we need a new method to check feasibility and conclude that, if we treat the feature constraint function as a black box, checking feasibility may take exponential time.

Example 5. Just like we might not want our robot to brake too many times we may want to avoid the frequent use of expensive or difficult actions in other improvisations. Let's say we create melodies for a jazz trumpeter based on a reference melody with a particularly high and difficult note. Frequent usage of this note may make the melodies difficult and tiring to play. We will use the example of the jazz trumpeter to apply our new expanded CI definition. As a reference melody we have the melody *"a, b, c, a"*. A DFA

that creates variations on this melody is given in Fig. 3.1.

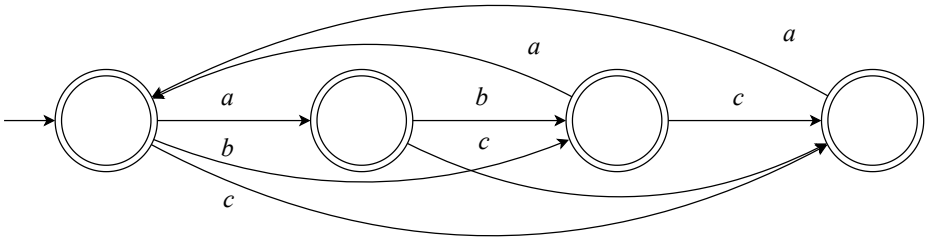

Figure 3.1: DFA $\mathcal{H}$

We will use the DFA from Fig. 3.1 '$\mathcal{H}$' as our hard specification. If we use the length bounds: $m = n = 4$ we get a total of twenty improvisations, such as: *abca, abac, cabc, bcaa*. To increase similarity to the reference melody we will use the DFA $\mathcal{S}$ in Fig. 3.2. that accepts all words in $\mathcal{L}(\mathcal{H})$ containing the sequence: *"abc"*, or *"bca"*. This results in six admissible improvisations: *abca, babc, cabc, bcaa, bcab, bcac*.

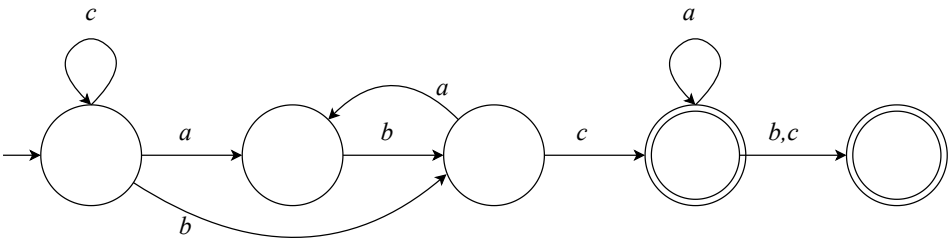

Figure 3.2: DFA $\mathcal{S}$

For our jazz improvisations we don't really mind if some variations appear more frequently and others do not appear at all. We pick our $\lambda = 0$ and $\rho = 0.15$ to reflect this idea. We want at least half of our improvisations to follow the reference melody closely, so we pick $\epsilon = 0.5$. Lastly we need to define our feature constraint function: $fc_c(w) = 1/(2^{\#_c(w)+2})$. This gives us the problem: $\mathcal{C}_\mathcal{J} = (\mathcal{H}, \mathcal{S}, 4, 4, fc_c, 0.5, 0, 0.15)$.

To solve $\mathcal{C}_\mathcal{J}$ we must first check feasibility. We know that three inequalities existed for the original problem that when satisfied guaranteed feasibility. Applying these yields:

1. $1/\rho \leq |I| \leq 1/\lambda \Rightarrow 5 \leq 20 \leq \infty$
2. $(1-\epsilon)/\rho \leq |A| \Rightarrow 0.5/0.1 \leq 6$
3. $|I| - |A| \leq \epsilon/\lambda \Rightarrow 14 \leq \infty$

The fact that these equations are satisfied does however not provide a guarantee that the feature constraint can be satisfied at the same time. For example, when we apply the FC-function to a word $w \in I$ we may find that $fc(w) < \lambda$. In which case either the randomness or the feature constraint cannot be satisfied.

In the example above we see that we need to do more than simply checking the inequalities found by Fremont et al. to check feasibility. One reason for this is the need to inspect each $w \in I$ to guarantee that $\forall w \in I, fc(w) \geq \lambda$. This is however not the only limitation. Even for a CI instance with $\lambda = 0$ we must consider the hard constraint $\Pr[w \in I | w \leftarrow D] = 1$. To check whether a CI instance can satisfy the hard constraint and the feature constraint at the same time we must find: $\sum_{w \in I} fc(w)$. When fc is treated as a black box we cannot predict the outcome of $fc(w)$. So we must apply the function to all $w \in I$ until $\sum_{w \in I} fc(w) \geq 1$. This leads us to the following lemma.

Lemma 1. Checking feasibility of a CI-FC instance while treating FC as a black box requires inspecting all $w \in I$.

The size of I depends on $\mathcal{H}$ and the length bounds m, n. While the length bounds guarantee that $|I|$ is finite, it may still be exponential with regards to the size of $\mathcal{H}$, m and n. An example of such a case is shown in Example 6.

Example 6. Consider $\mathcal{H}$, the DFA in Fig. 3.3 which produces all words in the language $(a + b)^*$, and length restrictions $m = 0$ and $n = x$. Construct I by applying the length restrictions to $\mathcal{H}$. Looking at the size of I, we now see it is exponential: $|I| = 2^{x+1} - 1$.

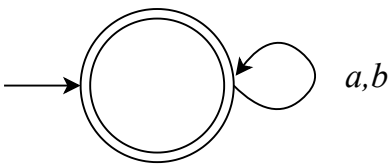

Figure 3.3: DFA $\mathcal{H}$

Lemma 2. For a CI-FC instance $\mathcal{C}$ the count of all improvisations I may be exponential with regards to the size of $\mathcal{C}$.

From Lemma 1 and Lemma 2 we conclude that:

Lemma 3. In general checking feasibility of a CI-FC instance $\mathcal{C}$ requires exponential time with regards to $|\mathcal{C}|$.

If checking feasibility takes exponential time, then finding an improvising distribution must also take at least exponential time. Assuming a faster than exponential-time algorithm to find such a distribution exists, we could apply this algorithm to a CI problem to check its feasibility. If the algorithm succeeds, we can conclude it is feasible, and if it fails to terminate within the expected runtime, we can conclude the problem is not feasible. This would give us a faster than exponential-time algorithm for checking feasibility. As we have shown, this cannot be the case, so a faster than exponential-time algorithm for finding an improvising distribution cannot exist.

In conclusion: when we treat the FC-function as a black box we must inspect each word in I to apply the function. The number of words in I may be exponential with respect to the size of the CI-FC instance. This means CI-FC instances exist which require at least exponential time to solve. This leads us to formulate the following theorem:

Theorem 2. *Given a general algorithm for CI-FC, there is an exponential function $e : \mathbb{R} \to \mathbb{R}$ such that the runtime of this algorithm for any CI-FC instance $\mathcal{C}$ is at least $(e|\mathcal{C}|)$.*

3.3 Solving general FC-CI instances

While we have proven that a faster than exponential-time scheme does not exist, a general algorithm with an exponential runtime does exist. The complexity of this algorithm is linear in the number of words in the language of the hard constraint $|I|$. This number can be exponential in the size of representation of the language which means worst case the general algorithm is exponential.

Lemma 4. An exponential-time algorithm solving CI-FC exists.

We present two variants of an exponential-time algorithm for FC-CI. The first algorithm creates an improviser by storing the improvising distribution in a large table from which it can sample efficiently. The second algorithm stores a compressed version of the improvising distribution from which sampling requires a long time.

3.3.1 Complete table algorithm

The idea behind this algorithm is: if we need to compute $fc(w)$ for each word to check feasibility, then we can simply store these values in a large table to create an improvising distribution. The improviser then uses this table to generate each word with its assigned probability. This allows the improviser to quickly generate words, but requires storing a possibly very large table.

The algorithm consists of the following steps. First, complete all of the steps of the old algorithm needed to check for feasibility. Find $|I|$ and $|A|$, check whether the three inequalities hold, if they hold continue, else return not feasible. To continue, compute $\epsilon_{max} = 1 - (|I| \cdot \lambda)$ this is the remaining probability (rem) if we would assign the λ to all words in the improvising distribution.

Continue by building a table of size $|I|$. Generate each word in $w \in A$, check if $fc(w) < \lambda$ if so terminate and return not feasible, else assign it a probability and store the pair w, p in the table. While the rem is not zero assign $p = \min(fc(w), \rho, \lambda + rem)$ and set $rem = rem - p + \lambda$ to account for the extra probability assigned. When $rem = 0$ always assign $p = \lambda$.

If there is a remainder after each word in A has been generated perform an additional feasibility check. If $|A| \cdot \lambda + (\epsilon_{max} - rem) < \epsilon$ the problem is not feasible. If the problem may still be feasible continue by generating each word $w \in (I \setminus A)$ and applying the same steps to build the table.

If there is a remainder after each word in I has been generated the problem is not feasible. If the remaining probability is 0 the constructed table is an improvising distribution. An improviser is any algorithm that samples randomly from this distribution.

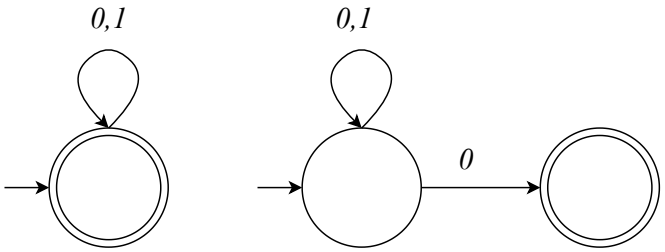

Figure 3.4: left: $\mathcal{H}$, right: $\mathcal{S}$

Example 7. Take the CI-FS problem $\mathcal{C} = (\mathcal{H}, \mathcal{S}, 1, 3, fc_{\#1}, 0.55, 1/25, 1/8)$. With $\mathcal{H}$ and $\mathcal{S}$ given in Figure 3.4. $\mathcal{L}(\mathcal{H})$ contains all binary strings of all lengths and $\mathcal{L}(\mathcal{S})$ contains all binary strings ending with a 0.
We apply length restriction and counting to find $|I|$ and $|A|$. Now we check and see all inequalities hold:

1. $1/\rho \leq |I| \leq 1/\lambda \Rightarrow 10 \leq 14 \leq 25$
2. $(1 - \epsilon)/\rho \leq |A| \Rightarrow 3.6 \leq 7$
3. $|I| - |A| \leq \epsilon/\lambda \Rightarrow 7 \leq 11.25$

We compute $\epsilon_{max} = 1 - 14 \cdot \lambda = 11/25$. Now we construct the table, first generating the words in A:
All words in A are assigned the maximal probability without the remainder

$w \in A$	p	remainder	$w \in (I \setminus A)$	p	remainder
0	$1/8$	$213/600$	1	$1/15$	$41/600$
00	$1/8$	$162/600$	01	$1/15$	$25/600$
10	$1/15$	$146/600$	11	$1/20$	$19/600$
000	$1/8$	$95/600$	001	$1/15$	$3/600$
010	$1/15$	$79/600$	011	$1/25 + 3/600$	0
100	$1/15$	$63/600$	101	$1/25$	0
110	$1/20$	$57/600$	111	$1/25$	0

Table 3.1: An exponential algorithm exemplified.

dropping to 0. We check for feasibility: $7 \cdot 24/600 + (264/600 - 57/600) = 5/8 \not\leq \epsilon$ so the problem may still be feasible. We continue building the table by generating all words in $I \setminus A$. Note that the word 011 is assigned $\lambda + 3/600$ as the remainder is too small to assign $fc(011) = 1/20$. Because the remainder has reached 0 while creating the table and $\nexists w \in I, fc_{\#1}(w) < \lambda$ we can conclude this is an improvising distribution for $\mathcal{C}$.

3.3.2 Compressed table algorithm

This algorithm is similar to the algorithm in Sect. 3.3.1. It runs through the same procedure, but does not build the complete table. Instead, it creates a compressed version of the table. This is possible because the table created by the first algorithm always follow the following structure: a set of words assigned the maximal probability $\min(fc(w), \rho,)$, followed by one word that is assigned the probability $\lambda + rem$, the remaining words which are assigned the probability λ.

By storing the input of the problem and the *(word, probability)* pair that is assigned $\lambda + rem$ we can find the probability of all words in the compressed table by simply generating all words up to that point and seeing whether the word comes before or after the word that is assigned $\lambda + rem$.

Sampling from this compressed improvising distribution can simply be done by generating a random number in $[0, 1]$, generating all words in the original order, summing their probabilities until the random number is reached or exceeded, the last word generated at that point is the sample.

This method for sampling is very inefficient and may take exponential time. It is likely sampling can be done more efficiently by storing more information such as the number of words assigned λ. However, no matter how efficiently the sampling is done or how little information is stored the algorithm will always require exponential time to check for all $w \in I$ that $fc(w) \geq \lambda$.

Example 8. If we look back at Table 3.1 in Example 7. We can see the words 0 up to 001 are all assigned $\min(fc(w), \rho,)$, then 001 is assigned $\lambda + rem$ and finally 101 and 111 are both assigned λ. So the compressed table could look like:

$w \in (I \setminus A)$	p
$0, \ldots$	$\min(fc(w), \rho,)$
011	$\lambda + 1/200$
$101, \ldots$	λ

3.4 Solving CI-FC in polynomial time

In Sec. 3.2 we have shown that a general algorithm for CI-FC has an exponential runtime. If we no longer treat the FC-function as a black box we may find groups of functions for which we can solve CI-FC in polynomial time. We will look at a few examples where a solution can be found in polynomial time. We will then go over the properties needed for a polynomial-time solution to exist and describe a polynomial-time improvisation scheme for CI instances that adhere to these properties.

3.4.1 Constant FC-functions

The first group of functions we will discuss is the constant FC-functions. These are functions that only have two constant values as results. For example, if the word contains at least five a's it returns 0.1, otherwise it returns ρ.

We will begin by showing an example solution for a CI-FC instance with a constant FC-function and then discuss the implications of the applied solution.

Example 9. Consider a simplified model of our surveillance robot that should not brake too many times. As alphabet we use $\Sigma = \{a, b\}$. As hard specification we pick a DFA $\mathcal{H}$ such that $\mathcal{L}(\mathcal{H}) = \Sigma^*$ and as soft specification we pick the DFA S such that $\mathcal{L}(\mathcal{S}) = \{a + ba\}^*$. Using $n = m = 3$, $\lambda = 0.1$, $\rho = 0.2$, $\epsilon = 0.5$ and $fc(w) = [\#_b(w) \geq 2]0.1, [\#_b(w) < 2]1$.

We construct $\mathcal{FC}$ a DFA that describes the 'if-condition' in our FC-function by accepting all words containing two or more b's.

Then we create the specification of all improvisations by applying length restriction to $\mathcal{H}$ which gives us $\mathcal{I}$.

By computing $\mathcal{I} \cap \mathcal{FC} = \mathcal{I}_{0.1}$ we find the specification of all words in I that are restricted by the feature constraint.

$\mathcal{I} \setminus \mathcal{FC} = \mathcal{I}_\rho$ gives us all words in I that are not limited by the feature constraint.

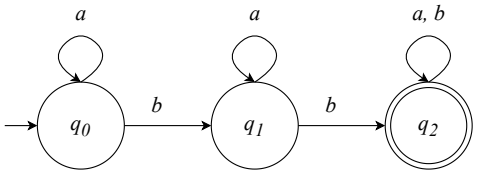

Figure 3.5: $\mathcal{FC}$

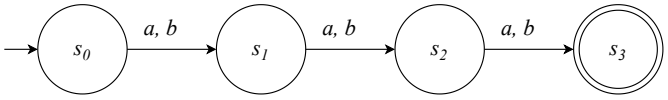

Figure 3.6: $\mathcal{I}$

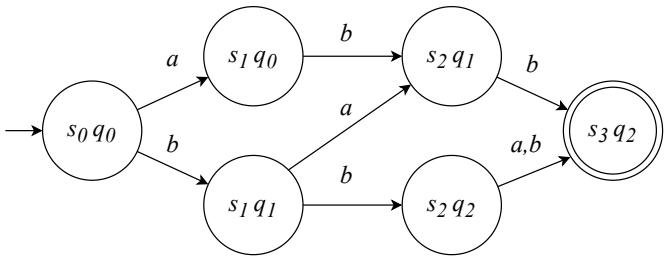

Figure 3.7: $\mathcal{I}_{0.1}$

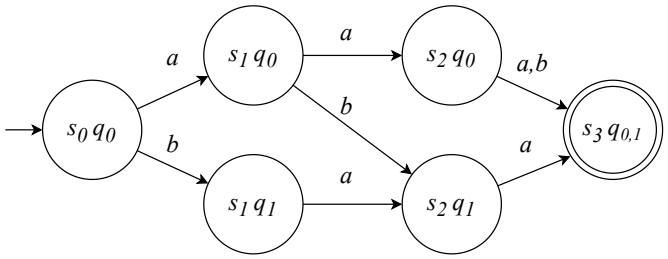

Figure 3.8: $\mathcal{I}_\rho$

Now we can construct the following specifications: $\mathcal{I}_{0.1} \cap \mathcal{S} = \mathcal{A}_{0.1}$, $\mathcal{I}_\rho \cap \mathcal{S} = \mathcal{A}_\rho$, $\mathcal{I}_{0.1} \setminus \mathcal{S} = (\mathcal{I} \setminus \mathcal{A})_{0.1}$, $\mathcal{I}_\rho \setminus \mathcal{S} = (\mathcal{I} \setminus \mathcal{A})_\rho$.

To each of these specifications we can apply counting to check feasibility. If the problem is feasible we can apply uniform sampling to create an improvising distribution. Applying counting gives: $|\mathcal{A}_\rho| = 3$, $|\mathcal{A}_{0.1}| = 0$, $|(\mathcal{I} \setminus \mathcal{A})_\rho| = 1$ and $|(\mathcal{I} \setminus \mathcal{A})_{0.1}| = 4$.

Because $|(\mathcal{I} \setminus \mathcal{A})_{0.1}|$ is not zero we must first check $\lambda \leq 0.1$ to see if the upper limit imposed by the feature constraint is not smaller than the minimum probability. To check the soft constraint we can compute $\epsilon \leq |\mathcal{A}_\rho| \cdot \rho + |\mathcal{A}_{0.1}| \cdot 0.1 = 0.6$ and $|\mathcal{I}| - (|\mathcal{A}_\rho| + |\mathcal{A}_{0.1}|) \leq \epsilon/\lambda$, $8 - 3 \leq 0.5/0.1$.

Note that for the second function we use the original inequality as it only depends on the lower bound. To check the hard constraint we can still use this part of the inequality $|\mathcal{I}| \leq 1/\lambda$ which holds: $8 \leq 1/0.1 = 10$. The other part involves ρ so we must include the feature constraint: $1 \leq |\mathcal{I}_{0.1}| \cdot 0.1 + |\mathcal{I}_{\rho}| \cdot \rho = 4 \cdot 0.1 + 4 \cdot 0.2 = 1.2$. This means we can conclude the problem is feasible.

Because the problem is feasible we can create an improvising distribution. We do this by rolling a weighted four sided die that chooses from the specifications and applying uniform sampling on the chosen specification. In this case the die chooses $(\mathcal{I} \setminus \mathcal{A})_{\rho}$ with a probability of $\lambda \cdot 1 = 0.1$, $\mathcal{I} \setminus \mathcal{A})_{0.1}$ with $p = \lambda \cdot 4 = 0.4$, $\mathcal{A}_{0.1}$ with $p = 0$ and $\mathcal{A}_{\rho}$ with $p = \epsilon = 0.5$. This distribution is a solution for this CI problem.

In Example 9 we can see that we do not need to apply the feature constraint function to individual words. The steps taken in this solution actually closely resemble the polynomial-time improvisation scheme by Fremont et al. The major difference is the addition of three new specifications and the adaptation of the feasibility functions to accommodate for the extra specifications. The operations used in this example still only take polynomial time on DFA's. Because of this, the solution itself also only uses polynomial time.

3.4.2 Piecewise constant FC-functions

There is a second group of functions that is similar to the constant FC-functions. These are the functions with multiple constant outputs, but where the number of different outputs is still limited. For example, functions that can be written as algebraic expressions over one or more letter counting function such as: $fc(w) = 1/\#_a(w)$ or $fc(w) = 1/1+\#_a(w) \cdot 1/1+\#_b(w)$. This type of function allows for a larger number of possible outcomes. In Example 10 we will show what happens to the number of specifications needed when we increase the number of possible outcomes of our feature constraint function.

Example 10. Recall the jazz trumpeter problem from Sec. 3.2: $\mathcal{C}_{\mathcal{J}} = (\mathcal{H}, \mathcal{S}, 4, 4, fc_c, 0.5, 0, 0.15)$ with $fc_c(w) = 1/(2^{\#_c(w)+2})$. We will solve it using the same strategy as in Example 9. We construct a series of DFA's: $\mathcal{FC}_0$, $\mathcal{FC}_1$, $\mathcal{FC}_2$, $\mathcal{FC}_3$ and $\mathcal{FC}_4$, where each DFA accepts all words in Σ^* with exactly $0, 1, 2, 3, 4$ c's. We apply length restriction to $\mathcal{H}$ which gives us $\mathcal{I}$ and use $\mathcal{I}$ to compute $\mathcal{A} = \mathcal{I} \cap \mathcal{S}$. We create the set SI containing five specifications: $\mathcal{I}_0$, $\mathcal{I}_1$, $\mathcal{I}_2$, $\mathcal{I}_3$, $\mathcal{I}_4$, where each specification $\mathcal{I}_i = \mathcal{I} \cap \mathcal{FC}_i$.
We use these specifications to create the set $SA = \{\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4\}$, where $\mathcal{A}_i = \mathcal{I}_i \cap \mathcal{A}$, and the set $SIA = \{(\mathcal{I} \setminus \mathcal{A})_0, (\mathcal{I} \setminus \mathcal{A})_1, (\mathcal{I} \setminus \mathcal{A})_2, (\mathcal{I} \setminus \mathcal{A})_3, (\mathcal{I} \setminus \mathcal{A})_4\}$, where $(\mathcal{I} \setminus \mathcal{A})_i = \mathcal{I}_i \setminus \mathcal{A}$.
To check feasibility we must check if the following things are true:

1. For all $\mathcal{I}_i \in SI$ if $1/(2^{i+2}) < \lambda$, then $\mathcal{L}(\mathcal{I}_\rangle) = \emptyset$. Because $\lambda = 0$ we know this holds.

2. $|\mathcal{I}| \leq 1/\lambda$ and $\sum_{\mathcal{I}_i \in SI} |I_i| \cdot \min(\rho, 1/(2^{i+2})) \geq 1$. Which both hold: $20 \leq \infty$ and $4 \cdot \rho + 11 \cdot 1/8 + 5 \cdot 1/16 + 0 + 0 = 23/80 \geq 1$.

3. $\sum_{A_i \in SA} |A_i| \cdot \min(\rho, 1/(2^{i+2})) \geq 1 - \epsilon$ which is satisfied: $4 \cdot 1/8 + 2 \cdot 1/16 = 5/8 \geq 1 - 0.5$.

4. $|\mathcal{I}| - |\mathcal{A}| \leq \epsilon/\lambda$ which is also satisfied: $20 - 6 = 14 \leq \infty$.

This means an improvising distribution exists. To define an improvising distribution for $\mathcal{C}_\mathcal{J}$ we attach a probability to each specification. Because we know that $\lambda = 0$, we can simply assign the maximal probability to each specification in SA: $p[\mathcal{A}_i] = \min(\rho, 1/(2^{i+2})) \cdot |\mathcal{A}_i|$. Then we compute $1 - \sum_{A_i \in SA} p[\mathcal{A}_i] = 3/8$ To satisfy the hard constraint we must distribute this 'remaining' probability over the specifications in SIA. We assign $p[(\mathcal{I} \setminus \mathcal{A})_2] = 3 \cdot 1/16$, $p[(\mathcal{I} \setminus \mathcal{A})_1] = 3 \cdot 1/16$ and all other specification in SIA a probability of 0. We then pick one of these specifications at random using these probabilities and apply random sampling to the chosen specification. This is an improviser for $\mathcal{C}_\mathcal{J}$.

In this example we we needed to create a total of fifteen extra specifications, three for each output of the FC-function. This is in line with Example 9, where we had a FC-function with two outputs and needed to create six additional specifications. We formalize this observation in the following lemma:

Lemma 5. Given a CI-FC instance $\mathcal{C}$ with a FC-function fc, we can solve $\mathcal{C}$ by constructing $3 \cdot |fc|$ additional specifications, where $|fc|$ is the number of elements in the codomain of fc. As this allows the construction of specifications intersecting with $\mathcal{I}$, $\mathcal{A}$ and $\mathcal{I} \setminus \mathcal{A}$ for each output of fc.

3.4.3 General polynomial-time solution for (piecewise) constant FC-functions

In our previous two examples we have observed that solutions for a CI-FC instance can be found in polynomial time in some instances. In this section we formalize the conditions under which we know a polynomial-time solution can be found and provide a polynomial-time improvisation scheme for CI-FC instances that meet these conditions. From our examples we can identify three key elements:

1. The number of possible solutions of the FC-function $|fc|$. We know that the number of extra specification created is equal to twice the number of outputs of the FC-function. If we imagine a CI-FC instance

$\mathcal{C}$ with an injective FC-function we know that we must construct more specification than words in I. As we know that $|I|$ may be exponential with regards to $|\mathcal{C}|$ we must conclude that this solution strategy may exponential time for $\mathcal{C}$. This problem may persist for any fc where $|fc|$ is linear with regard to I.

When $|fc|$ is bound by a constant a polynomial-time solution exists. An example of this is the set of (piecewise) constant FC-functions as they have a limited number of outputs. As we saw in Example 9 where $|fc|$ was limited to a constant and Example 10 where $|fc|$ was limited by the length bound which is constant.

2. A polynomial-time method for finding the specifications that group words with the same fc value. In our examples it was easy to find DFA's that allowed us to create specifications where each word in the specification would have an identical output from fc. It is important that this can be done in polynomial time. An injective fc might again require exponential time to find the specifications.

3. The size of the specifications that group words with the same fc value must not be to large. This condition is important because operations such as the union between two specifications and sampling from the final specifications are dependent on the size of the specifications. To guarantee a polynomial-time solution for a CI-FC instance $\mathcal{C}$ the size of the created specifications must be polynomial with regards to $|\mathcal{C}|$.

Theorem 3. *Given a CI-FC instance $\mathcal{C}$ with a FC-function fc. If there exist a set SFC of DFA's with $|SFC| = |fc|$ that satisfies the properties (1)-(4), then a polynomial-time improvisation scheme for $\mathcal{C}$ exists.*

1. *For each DFA $\mathcal{FC} \in SFC$, $\forall v, w \in \mathcal{L}(\mathcal{FC}), fc(v) = fc(w)$.*

2. *There is a polynomial $p : \mathbb{R} \to \mathbb{R}$ such that $|SFC| \leq p(|\mathcal{C}|)$.*

3. *Given $\mathcal{C}$, each $\mathcal{FC} \in SFC$ can be constructed in polynomial time with regard to $|\mathcal{C}|$.*

4. *For each $\mathcal{FC} \in SFC$, there is a polynomial $q : \mathbb{R} \to \mathbb{R}$ such that $|\mathcal{FC}| \leq q(|\mathcal{C}|)$.*

Before we prove this theorem we will first construct the following lemma's to support our final proof.

Lemma 6. Given a CI-FC instance $\mathcal{C}$ that satisfies the conditions in Theorem 3, if $\mathcal{C}$ is feasible then the following functions, with w an arbitrary element from $\mathcal{I}_i$, are satisfied:

1. $\forall \mathcal{I}_i \in SI,\ fc(w) < \lambda \to \mathcal{L}(\mathcal{I}_\rangle) = \emptyset$.

2. $|\mathcal{I}| \leq 1/\lambda$ and $\sum_{\mathcal{I}_i \in SI} |I_i| \cdot \min(\rho, fc(w)) \geq 1$.
3. $\sum_{A_i \in SA} |A_i| \cdot \min(\rho, fc(w)) \geq 1 - \epsilon$.
4. $|\mathcal{I}| - |\mathcal{A}| \leq \epsilon/\lambda$.

Proof. Given a CI-FC instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, fc, \epsilon, \lambda, \rho)$, with the set SFC as defined in Theorem 3. We begin by constructing some specifications that we will use in this proof:
Apply length restriction to $\mathcal{H}$ which gives $\mathcal{I}$ and use it to compute $\mathcal{A} = \mathcal{I} \cap \mathcal{S}$.
Create the set $SI = \{\mathcal{I}_i \leftarrow \mathcal{I} \cap \mathcal{FC}, \mathcal{FC} \in SFC, i \in \mathbb{N}\}$,
the set $SA = \{\mathcal{A}_i \leftarrow \mathcal{I}_i \cap \mathcal{A}, \mathcal{I}_i \in SI\}$
and the set $SIA = \{(\mathcal{I} \setminus \mathcal{A})_i \leftarrow \mathcal{I}_i \setminus \mathcal{A}, \mathcal{I}_i \in SI\}$

1. If $\mathcal{C}$ is feasible then the feature constraint holds: $\forall w \in I, D(w) \leq fc(w)$. We know that $\forall \mathcal{I}_i \in SI, \mathcal{I}_i \subset \mathcal{I}$. So $\forall \mathcal{I}_i \in SI,\ fc(w) < \lambda \rightarrow \mathcal{L}(\mathcal{I}_\rangle) = \emptyset$.
2. If $\mathcal{C}$ is feasible there exists an improvising distribution D for $\mathcal{C}$.

$$\Pr[w \in \mathcal{L}(\mathcal{I}) | w \leftarrow D] = \sum_{w \in \mathcal{L}(\mathcal{I})} D(w) = 1$$

$$\forall w \in \mathcal{L}(\mathcal{I}), D(w) \leq fc(w) \wedge\ \forall w \in \mathcal{L}(\mathcal{I}), D(w) \leq \lambda \leftrightarrow$$

$$\forall w \in \mathcal{L}(\mathcal{I}), D(w) \leq \min(fc(w), \lambda)$$

$$\sum_{w \in \mathcal{L}(\mathcal{I})} \min(fc(w), \lambda) \geq \sum_{w \in \mathcal{L}(\mathcal{I})} D(w) = 1$$

$$\sum_{w \in \mathcal{L}(\mathcal{I})} \min(fc(w), \lambda) = \sum_{\mathcal{I}_i \in SI} \sum_{w \in \mathcal{L}(\mathcal{I})_i} \min(fc(w), \lambda)$$

$$\sum_{\mathcal{I}_i \in SI} \sum_{w \in \mathcal{L}(\mathcal{I})_i} \min(fc(w), \lambda) = \sum_{\mathcal{I}_i \in SI} |I_i| \cdot \min(\rho, fc(w))$$

This shows the first half of the equation:

$$\sum_{\mathcal{I}_i \in SI} |I_i| \cdot \min(\rho, fc(w)) \geq 1$$

$$\Pr[w \in \mathcal{L}(\mathcal{I}) | w \leftarrow D] = \sum_{w \in \mathcal{L}(\mathcal{I})} D(w) = 1$$

$$\forall w \in \mathcal{L}(\mathcal{I}),\ D(w) \geq \lambda, \text{ so: } 1 = \sum_{w \in \mathcal{L}(\mathcal{I})} D(w) \geq \sum_{w \in \mathcal{L}(\mathcal{I})} \lambda$$

$$\sum_{w \in \mathcal{L}(\mathcal{I})} \lambda = \lambda \cdot |\mathcal{I}|$$

This shows the second half of the equation.

$$1 \geq \lambda \cdot |\mathcal{I}| \text{ and } 1/\lambda \geq |\mathcal{I}|$$

3. We know the soft constraint is satisfied:

$$\Pr[w \in A | w \leftarrow D] = \sum_{w \in \mathcal{L}(\mathcal{A})} D(w) \geq 1 - \epsilon$$

$\mathcal{L}(\mathcal{A}) \subset \mathcal{L}(\mathcal{I})$ from this we know the following as we have shown it for $w \in \mathcal{L}(\mathcal{I})$:

$$\sum_{w \in \mathcal{L}(\mathcal{I})} \min(fc(w), \lambda) \geq \sum_{w \in \mathcal{L}(\mathcal{A})} D(w)$$

So we can conclude: $\sum_{A_i \in SA} |A_i| \cdot \min(\rho, fc(w)) \geq 1 - \epsilon$

4. For this inequality we use both the hard and soft constraint:

$$\Pr[w \in \mathcal{L}(\mathcal{I}) | w \leftarrow D] - \Pr[w \in \mathcal{L}(\mathcal{A}) | w \leftarrow D] =$$

$$\Pr[w \in \mathcal{L}(\mathcal{I} \setminus \mathcal{A}) | w \leftarrow D] \leq 1 - (1 - \epsilon) = \epsilon$$

$$\Pr[w \in \mathcal{L}(\mathcal{I} \setminus \mathcal{A}) | w \leftarrow D] = \sum_{w \in \mathcal{L}(\mathcal{I} \setminus \mathcal{A})} D(w)$$

$$\forall w \in \mathcal{L}(\mathcal{I} \setminus \mathcal{A}),\ D(w) \geq \lambda, \text{ so: } \sum_{w \in \mathcal{L}(\mathcal{I} \setminus \mathcal{A})} D(w) \geq \sum_{w \in \mathcal{L}(\mathcal{I} \setminus \mathcal{A})} \lambda$$

$$\sum_{w \in \mathcal{L}(\mathcal{I} \setminus \mathcal{A})} \lambda = \lambda \cdot |\mathcal{I} \setminus \mathcal{A}|$$

$$\epsilon \geq \lambda \cdot |\mathcal{I} \setminus \mathcal{A}| \text{ and } |\mathcal{I}| - |\mathcal{A}| \leq \epsilon / \rho$$

□

Lemma 7. Given a CI-FC instance $\mathcal{C}$ if the following inequalities, with w an arbitrary element from $\mathcal{I}_i$, are satisfied we can construct an improviser for $\mathcal{C}$:

1. $\forall \mathcal{I}_i \in SI,\ fc(w) < \lambda \rightarrow \mathcal{L}(\mathcal{I}_i) = \emptyset$.
2. $|\mathcal{I}| \leq 1/\lambda$ and $\sum_{\mathcal{I}_i \in SI} |I_i| \cdot \min(\rho, fc(w)) \geq 1$.
3. $\sum_{A_i \in SA} |A_i| \cdot \min(\rho, fc(w)) \geq 1 - \epsilon$.

4. $|\mathcal{I}| - |\mathcal{A}| \leq \epsilon/\lambda$.

Proof. Given a CI-FC instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, fc, \epsilon, \lambda, \rho)$. Find the series of DFA's SFC. Apply length restriction to $\mathcal{H}$ which gives $\mathcal{I}$ and use it to compute $\mathcal{A} = \mathcal{I} \cap \mathcal{S}$. Create the set $SI = \{\mathcal{I}_i \leftarrow \mathcal{I} \cap \mathcal{FC}, \mathcal{FC} \in SFC, i \in \mathbb{N}\}$, the set $SA = \{\mathcal{A}_i \leftarrow \mathcal{I}_i \cap \mathcal{A}, \mathcal{I}_i \in SI\}$
and the set $SIA = \{(\mathcal{I} \setminus \mathcal{A})_i \leftarrow \mathcal{I}_i \setminus \mathcal{A}, \mathcal{I}_i \in SI\}$
Find $\epsilon_{opt} = \max(\lambda \cdot (|\mathcal{I}| - |\mathcal{A}|), 1 - \sum_{\mathcal{A}_i \in SA} |\mathcal{A}_i| \cdot \min(\rho, fc(w))$ where w is an arbitrary element in $\mathcal{A}_i$.
If $\epsilon_{opt} = 1 - \sum_{\mathcal{A}_i \in SA} |\mathcal{A}_i| \cdot \min(\rho, fc(w)$, then assign $\forall \mathcal{A}_i \in SA, \Pr[\mathcal{A}_i] = |\mathcal{A}_i| \cdot \min(\rho, fc(w)$.
Let $rem = \epsilon_{opt} - \lambda \cdot (|\mathcal{I}| - |\mathcal{A}|)$.
Now iterate over all $(\mathcal{I} \setminus \mathcal{A})_i \in SAI$ and set $\Pr[(\mathcal{I} \setminus \mathcal{A})_i] = \min(|\mathcal{I} \setminus \mathcal{A}_i| \cdot \lambda + rem, |\mathcal{I} \setminus \mathcal{A}_i| \cdot fc(w))$, where w is an arbitrary element in $(\mathcal{I} \setminus \mathcal{A})_i$, and let $rem = rem - (\Pr[(\mathcal{I} \setminus \mathcal{A})_i] - |\mathcal{I} \setminus \mathcal{A}_i| \cdot \lambda)$.
If $\epsilon_{opt} = \lambda \cdot (|\mathcal{I}| - |\mathcal{A}|)$, then assign $\forall (\mathcal{I} \setminus \mathcal{A})_i \in SAI, p[(\mathcal{I} \setminus \mathcal{A})_i] = \lambda \cdot |(\mathcal{I} \setminus \mathcal{A})_i|$.
Let $rem = \epsilon_{opt} - \lambda \cdot |\mathcal{A}|$.
Now iterate over all $\mathcal{A}_i \in SA$ and set $\Pr[\mathcal{A}_i] = \min(|\mathcal{A}_i| \cdot \lambda + rem, |\mathcal{A}_i| \cdot fc(w))$, where w is an arbitrary element in $(\mathcal{A})_i$, and let $rem = rem - (\Pr[\mathcal{A}_i] - |\mathcal{A}_i| \cdot \lambda)$.
Note that in either case if $rem = 0$ then $\Pr[\mathcal{X}] = |\mathcal{X}| \cdot \lambda$ as we know that either $|\mathcal{X}| = 0$, or $fc(w) \geq \lambda$ for any $w \in \mathcal{L}(\mathcal{X})$.
Now use the assigned probabilities to create a weighted die that randomly selects one of the specifications from $SA \cup SIA$. Rolling this die and then applying random sampling to the selected specification is an improviser for $\mathcal{C}$. □

By combining the feasibility check in Lemma 6 and the construction of the improviser in Lemma 7 we get a complete improvisation scheme for CI-FC. A version of this improvisation scheme is described in Algorithm 1 which is used to proof Theorem 3.

Algorithm 1. Given a CI-FC instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, fc, \epsilon, \lambda, \rho)$.
Find the series of DFA's SFC. Apply length restriction to $\mathcal{H}$ which gives $\mathcal{I}$ and use it to compute $\mathcal{A} = \mathcal{I} \cap \mathcal{S}$.
Create the set $SI = \{\mathcal{I}_i \leftarrow \mathcal{I} \cap \mathcal{FC}, \mathcal{FC} \in SFC, i \in \mathbb{N}\}$,
the set $SA = \{\mathcal{A}_i \leftarrow \mathcal{I}_i \cap \mathcal{A}, \mathcal{I}_i \in SI\}$
and the set $SIA = \{(\mathcal{I} \setminus \mathcal{A})_i \leftarrow \mathcal{I}_i \setminus \mathcal{A}, \mathcal{I}_i \in SI\}$
Check whether all conditions are satisfied:

1. $\forall \mathcal{I}_i \in SI,\ fc(w) < \lambda \rightarrow \mathcal{L}(\mathcal{I}_i) = \emptyset$. With w a random sample from $\mathcal{I}_i$.
2. $|\mathcal{I}| \leq 1/\lambda$ and $\sum_{\mathcal{I}_i \in SI} |I_i| \cdot \min(\rho, fc(w)) \geq 1$. With w a random sample from $\mathcal{I}_i$.
3. $\sum_{A_i \in SA} |A_i| \cdot \min(\rho, fc(w)) \geq 1 - \epsilon$. With w a random sample from $\mathcal{I}_i$.

4. $|\mathcal{I}| - |\mathcal{A}| \leq \epsilon/\lambda$.

If any function is not satisfied the problem is not feasible, otherwise continue.
Find $\epsilon_{opt} = \max(\lambda \cdot (|\mathcal{I}| - |\mathcal{A}|), 1 - \sum_{\mathcal{A}_i \in SA} |\mathcal{A}_i| \cdot \min(\rho, fc(w))$ where w is a random sample from $\mathcal{A}_i$.
If $\epsilon_{opt} = 1 - \sum_{\mathcal{A}_i \in SA} |\mathcal{A}_i| \cdot \min(\rho, fc(w)$, then assign $\forall \mathcal{A}_i \in SA, \Pr[\mathcal{A}_i] = |\mathcal{A}_i| \cdot \min(\rho, fc(w)$.
Let $rem = \epsilon_{opt} - \lambda \cdot (|\mathcal{I}| - |\mathcal{A}|)$.
Now iterate over all $(\mathcal{I} \setminus \mathcal{A})_i \in SAI$ and set $\Pr[(\mathcal{I} \setminus \mathcal{A})_i] = \min(|\mathcal{I} \setminus \mathcal{A}_i| \cdot \lambda + rem, |\mathcal{I} \setminus \mathcal{A}_i| \cdot fc(w))$, where w is a random sample from $(\mathcal{I} \setminus \mathcal{A})_i$, and let $rem = rem - (\Pr[(\mathcal{I} \setminus \mathcal{A})_i] - |\mathcal{I} \setminus \mathcal{A}_i| \cdot \lambda)$.
If $\epsilon_{opt} = \lambda \cdot (|\mathcal{I}| - |\mathcal{A}|)$, then assign $\forall (\mathcal{I} \setminus \mathcal{A})_i \in SAI, p[(\mathcal{I} \setminus \mathcal{A})_i] = \lambda \cdot |(\mathcal{I} \setminus \mathcal{A})_i|$.
Let $rem = \epsilon_{opt} - \lambda \cdot |\mathcal{A}|$.
Now iterate over all $\mathcal{A}_i \in SA$ and set $\Pr[\mathcal{A}_i] = \min(|\mathcal{A}_i| \cdot \lambda + rem, |\mathcal{A}_i| \cdot fc(w))$, where w is a random sample from $(\mathcal{A})_i$, and let $rem = rem - (\Pr[\mathcal{A}_i] - |\mathcal{A}_i| \cdot \lambda)$.
Use the assigned probabilities to create a weighted die that randomly selects one of the specifications from $SA \cup SIA$. Rolling this die and then applying random sampling to the selected specification is an improviser for $\mathcal{C}$.

While Algorithm 1 is intended for cases where the size of SFC is small, this scheme can be applied regardless of the size of SFC. In this case the performance of this scheme may be worse than that of the exponential schemes mentioned in Sec 3.3.

To prove our claim in Theorem 3 we will show that Algorithm 1 is a polynomial-time improvisation scheme. Recall Definition 6, we need to guarantee correctness, scheme efficiency and improviser efficiency. Further it is important to recall the operations introduced in Chapter 2: intersection, difference, length restriction, counting and uniform sampling. These operations when applied to DFA's can be done in polynomial time [5].

Proof. (Theorem 3) We break the proof into the following three parts:

- Correctness, Lemma 6 shows that if the problem is feasible, the functions are satisfied and Lemma 7 shows that when the functions are satisfied an improviser exists.
- Scheme efficiency, we can break down the improvisation scheme into multiple parts to analyse it's efficiency:
    1. The construction of SFC: From the assumptions in Theorem 3 we know that given $\mathcal{C}$, SFC can be found in polynomial time.
    2. Finding SI, SA, SAI: From the assumptions in Theorem 3 we know there is a polynomial $p : \mathbb{R} \to \mathbb{R}$ such that $|SFC| \leq p(|\mathcal{C}|)$ and $\forall \mathcal{FC} \in SFC$, there is a polynomial $p : \mathbb{R} \to \mathbb{R}$ such that $\mathcal{FC} \leq p(|\mathcal{C}|)$.

We apply length restriction to $\mathcal{H}$ to construct $\mathcal{I}$ and construct $\mathcal{A} = \mathcal{S} \cap \mathcal{I}$.
We create SI where $|SI| = |SFC|$ because of its definition. For the creation of this set we must compute the intersection of $\mathcal{I}$ and all $\mathcal{FC} \in SFC$. This is a total of $|SFC|$ intersections.
We create SA where $|SA| = |SI|$ because of its definition. For the creation of this set we must compute the intersection of $\mathcal{A}$ and all $\mathcal{I}_i \in SI$. This is a total of $|SI| = |SFC|$ intersections.
We create SIA where $|SIA| = |SI|$ because of its definition. For the creation of this set we must compute the intersection of $\mathcal{A}$ and all $\mathcal{I}_i \in SI$. This is a total of $|SI| = |SFC|$ intersections.
We know the intersection operation and length restriction operation can both be done in polynomial time. We know that the number of operations in each instance is polynomial with regards to $\mathcal{C}$. We know that the $\mathcal{FC} \in SFC$ are not exponentially larger than $|\mathcal{C}|$. So this step can be completed in polynomial time.

3. Checking feasibility: the functions for checking feasibility apply counting on all specifications in SI and SA. From the previous step we know that $|SI|$ and $|SA|$ are both at most polynomial in size with regards to $\mathcal{C}$, and we know that counting can be done in polynomial time. Lastly the functions use uniform sampling which can also be done in polynomial time.
4. Finding ϵ_{opt}: again we apply counting and uniform sampling to all specifications in SA, as wel as counting on $\mathcal{I}$ and $\mathcal{A}$.
5. Assigning probabilities: we to assign probabilities we iterate over both the specifications in SA and SAI, applying counting and uniform sampling to each specification in SA and SAI.

We can see that each part can be completed in polynomial time, as there is a fixed number of parts the complete scheme can also be completed in polynomial time.

– Improviser efficiency, the improviser has two components, making a weighted random choice between the specifications in $SA \cup SIA$ and sampling from the chosen specification. We know that $|SA \cup SIA| \leq 2 \cdot |SFC|$ and that $|SFC|$ is polynomial with respect to $\mathcal{C}$. We also know that applying uniform sampling to an element of $SA \cup SIA$ can be done in polynomial time. So we can conclude the improviser can sample in polynomial time.

□

The key takeaway from this proof is that the number of additional specifications is completely dependent on the size of $|SFC|$. This observation is

in line with Lemma 5. This means that the performance of the improvisation scheme we introduce in Sec. 3.4.3 is also dependent on $|SFC|$.

3.5 Between the polynomial and the exponential cases

In Sec. 3.3 we have shown that the general CI-FC problem can only be solved in exponential time, we discussed cases in which we can guarantee a polynomial-time solution in Sec. 3.4. This does however leave the question whether there are more instances for which a faster then exponential-time solution exists. This may be the case and in Sec 3.5 we will go over a possible approach for solving some of these instances.

The assumption under which we were able to find polynomial solutions in Sec. 3.4 was that the FC-function was a piecewise constant function. Using this assumption we were able to group different words with an identical output of the FC-function in specifications. If the FC-function is no longer (piecewise) constant grouping words with identical output may result in an exponential number of groups. In some cases it may be possible to group words with a similar rather than identical output by turning the continuos function into a piecewise constant function.

Example 11. Imagine a FC-function fc over the alphabet $\Sigma = \{a, b, c\}$ which for some ordering of the words in $(a+b+c)^*$ along the x-axis produces the linear graph in Fig. 3.9. We can approximate fc with a function that maps the first half of the words to their shared minimum 0, and the second half of the words to their shared minimum 0.5. We can then check whether the problem with the simplified function is feasible, if so then the problem with the original fc is feasible, if not then we can continue refining using the same method.

Splitting the FC-function may find that a problem is feasible in polynomial time. To show that the problem is not feasible may require the simplified function to be refined to the point that it is the original function again. There are two ways the refinement scheme can terminate early with an accurate not feasible result. For this it is important to remember the highest value of each function section that was made constant.

1. The first is by tracking the possible 'gain' from refining a section of the function. After we check feasibility for a simplified function we know how many words each 'section' of the function has. We can use this to compute how much more probability we could possibly assign after refining the function.
   For example we may notice that ten words are in the $[0, 0.5]$ section and another 2 words in the $[0.5, 1]$ section. By refining the $[0, 0.5]$

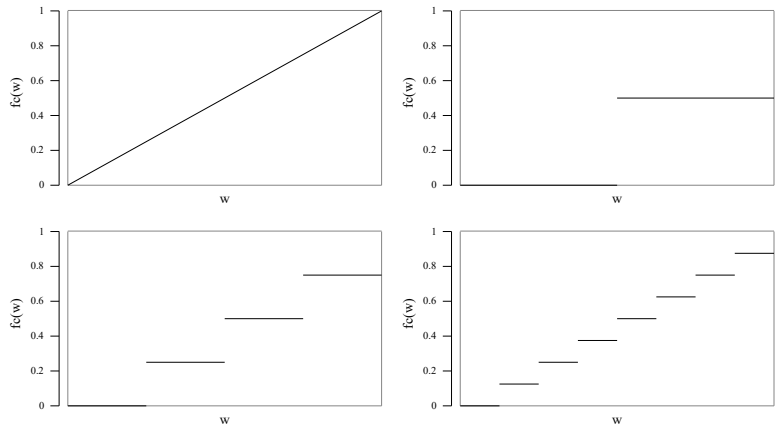

Figure 3.9: approximating a linear FC-function

section we can 'gain' at most a $10 \cdot 0.5 = 5$ increase to the sum of our maximum allowed probability as we currently assume that all ten words may only be assigned a probability of zero.
When the current maximal probability plus the maximal gain is not enough to satisfy the hard or soft constraint we can say the problem is not feasible.

2. The second is by finding a section of the function $[x, y]$ where $y < \lambda$. If after checking feasibility we find that this section contains a word we know the problem is not feasible, because the maximum probability a word in this section could ever be assigned is y.
For example we have a CI-FC instance with $\lambda = 0.2$, after approximating the function we find the section $[0.1, 0.15]$ which contains a word that is also in $\mathcal{H}$. No matter how much more we refine our approximate function this word can never be assigned a probability greater than 0.15 because of the feature constraint, thus not satisfying the randomness constraint.

This notion of possible 'gain' from a section may also be used to optimise the refinement strategy. By first refining sections with the most gain a solution may be found more quickly.

This method of solving CI-FC by approximating the FC-function is only one possible approach to solving the group of CI-FC instances for which we have not given a polynomial-time solution. This group of CI-FC instances may be an interesting direction for future research.

Chapter 4

Related Work

The concept of control improvisation was first introduced in *Control Improvisation with Application to Music* [3] by Donze et al. in 2013. In 2017 Fremont et al. provided a more generalised version of control improvisation in *Control Improvisation* [5], of which an earlier version was released in 2014 [4].

Donze et al. (and later Fremont et al.) drew their inspiration for a general control improvisation problem from earlier works such as: *Machine Musicianship* [8] which discusses the algorithmic generation of music, *black-box fuzz testing* [9] and randomised variants of the supervisory control problem [2].

Based on the 2014 version of Control Improvisation, Valle et al. looked at a practical application of CI in *Specification Mining for Machine improvisation* [10]. In this paper, Valle et al. combine the concept of CI with specification mining to apply it to the improvisation of blues songs. At the same time Akkaya et al. considered the possible practical implementation of CI for the purpose of mimicking the use of lighting appliances in a residential unit in *Control Improvisation with Probabilistic Temporal Specifications* [1]. This paper also introduces *multi-constraint control improvisation (MCI)* where multiple soft constraints are allowed. An exponential-time improvisation scheme for MCI was created by Fremont et al. [5] as well as proof that checking feasibility with multiple soft constraints is #P-hard.

It may seem that a feature-based randomness constraint is similar to adding another soft constraint. While CI-FC may be used for some applications where MCI could also be used, MCI focusses on adding more constraints to the entire distribution, while CI-FC focusses on the features of individual elements in the distribution. This means that, while an MCI instance with many soft constraints may result in the same distribution as a CI-FC instance with a feature constraint, it may be necessary to have one soft constraint per improvisation as MCI can only provide guarantees on groups of words, not individual words. MCI can however also be applied in

cases where CI-FC is not powerful enough, for example Fremont et al. have shown that MCI can handle slightly more complex soft constraints than CI, and by extension CI-FC, allows [5].

After the 2017 Control Improvisation paper an extended version of CI was created in *Reactive Control Improvisation* [6]. In this paper Fremont and Seshia created a version of CI named Reactive Control Improvisation (RCI) that can function in an uncertain environment. Instead of an improviser that creates whole words, the improviser in RCI will create a word by picking symbols alternating with an adversarial environment. In 2021 an alternative to RCI was offered by Vazquez-Chanlatte et al. in *Entropy Guided Control Improvisation*[11], namely Entropic Reactive Control Improvisation (ERCI). Instead of an adversarial environment, ERCI admits an arbitrary combination of nondeterministic and probabilistic uncertainty in the environment. These less strict assumptions about the environment allow a broader application of reactive control improvisation using ERCI.

Chapter 5

Conclusions

After an introduction to the topic of CI, we have expanded the definition of CI to include a feature constraint. The newly defined CI-FC can be used to assign different maximal likelihoods to individual improvisations based on their features.

After analysing CI-FC we found that in general a polynomial-time improvisation scheme for CI-FC cannot exist. When the feature constraint function is treated as black box it is necessary to inspect all possible improvisations. As the number of possible improvisations may be exponential, this can require exponential time. We present two versions of an exponential-time improvisation scheme. The first version provides a very large but fast improviser which stores the entire improvising distribution as a table. The second version gives an improviser that stores minimal information, but requires exponential time to find improvisations. It may be the case that a more efficient method for (partial) compression of the table exists, but that is left open for future research.

Furthermore, we identify two groups of FC-functions for which polynomial-time solutions can be found. After identifying the conditions that allow a faster solution we formalize them and formulate a polynomial-time improvisation scheme for CI-FC instances that satisfy these conditions. The key factor in deciding the tractability of a CI-FC instance appears to be the number of possible outputs of the FC-function. We propose an optimization strategy that may be applied to find solutions in cases where the FC-function has a large number of possible outputs.

A direction for future work that may be interesting is the expansion to multiple feature constraints, or a feature constraint limiting the minimal probability of a word occurring. Another possible direction is to look at different classes of specifications for CI-FC, or considering the possibility of FC-functions that can be described by a group of context free grammars. Finally, it may be interesting to implement the feature constraint in both reactive control improvisation and entropy guided control improvisation.

Bibliography

[1] Ilge Akkaya, Daniel J Fremont, Rafael Valle, Alexandre Donzé, Edward A Lee, and Sanjit A Seshia. Control improvisation with probabilistic temporal specifications. In *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 187–198. IEEE, 2016.

[2] Christos G Cassandras and Stéphane Lafortune. *Introduction to discrete event systems*. Springer, 2008.

[3] Alexandre Donzé, Sophie Libkind, Sanjit A Seshia, and David Wessel. Control improvisation with application to music. Technical report, CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, 2013.

[4] Daniel J. Fremont, Alexandre Donzé, Sanjit A. Seshia, and David Wessel. Control improvisation. *ArXiv e-prints*, November 2014.

[5] Daniel J. Fremont, Alexandre Donzé, and Sanjit A. Seshia. Control improvisation, 2017.

[6] Daniel J Fremont and Sanjit A Seshia. Reactive control improvisation. In *International conference on computer aided verification*, pages 307–326. Springer, 2018.

[7] Kenneth H Rosen and Kamala Krithivasan. *Discrete mathematics and its applications: with combinatorics and graph theory*. Tata McGraw-Hill Education, 2012.

[8] Robert Rowe. *Machine musicianship*. MIT press, 2004.

[9] Michael Sutton, Adam Greene, and Pedram Amini. *Fuzzing: brute force vulnerability discovery*. Pearson Education, 2007.

[10] Rafael Valle, Alexandre Donzé, Daniel J Fremont, Ilge Akkaya, Sanjit A Seshia, Adrian Freed, and David Wessel. Specification mining for machine improvisation with formal specifications. *Computers in Entertainment (CIE)*, 14(3):1–20, 2016.

[11] Marcell Vazquez-Chanlatte, Sebastian Junges, Daniel J Fremont, and Sanjit Seshia. Entropy-guided control improvisation. *arXiv preprint arXiv:2103.05672*, 2021.