

Negated String Containment Is Decidable

Vojtěch Havlena 

Brno University of Technology, Czech Republic

Michal Hečko 

Brno University of Technology, Czech Republic

Lukáš Holík 

Aalborg University, Denmark

Brno University of Technology, Czech Republic

Ondřej Lengál 

Brno University of Technology, Czech Republic

Abstract

We provide a positive answer to a long-standing open question of the decidability of the not-contains string predicate. Not-contains is practically relevant, for instance in symbolic execution of string manipulating programs. Particularly, we show that the predicate $\neg \text{Contains}(x_1 \dots x_n, y_1 \dots y_m)$, where $x_1 \dots x_n$ and $y_1 \dots y_m$ are sequences of string variables constrained by regular languages, is decidable. Decidability of a not-contains predicate combined with chain-free word equations and regular membership constraints follows.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Regular languages; Theory of computation $\rightarrow$ Automated reasoning; Theory of computation $\rightarrow$ Logic and verification

Keywords and phrases not-contains, string constraints, word combinatorics, primitive word

Digital Object Identifier 10.4230/LIPIcs.MFCS.2025.56

Related Version *Technical Report*: <http://arxiv.org/abs/2506.22061> [24]

Funding This work was supported by the Czech Ministry of Education, Youth and Sports ERC.CZ project LL1908, the Czech Science Foundation project 25-18318S, and the FIT BUT internal project FIT-S-23-8151. The work of Michal Hečko, a Brno Ph.D. Talent Scholarship  Holder, is funded by the Brno City Municipality.

Acknowledgements We thank the anonymous reviewers for careful reading of the paper and their suggestions that greatly improved its quality.

1 Introduction

String constraints have been recently intensely studied in relation to their applications in analysis of string manipulation in programs, e.g., in the analyses of security of web applications or cloud resource access policies [43]. Apart from a plethora of practical solvers, e.g., CVC5 [26, 27, 7, 28, 42, 38, 41], Z3 [23, 11, 33], OSTRICH [29, 13, 16, 14, 15], Z3-NOODLER [18, 17, 12], TRAU [4, 2, 1] Z3STR/2/3/4/3RE [10, 9, 8], WOORPJE [20], and NFA2SAT [32], the theoretical landscape of string constraints has been intensely studied too. The seminal work of Makanin [35], establishing decidability of word equations, was followed by the work of Plandowski [39] (and later Jež's work on recompression) that placed the problem in PSPACE. A number of relatively recent works study extensions of string constraints with constraints over string lengths, transducer-defined relational constraints, string-integer conversions, extensions of regular properties, replace-all, etc. As the extended string constraints are in general undecidable, these works focus on finding practically relevant decidable fragments such as the straight-line [16, 29, 13, 15, 14] and chain-free [4, 17] fragments, quadratic equations [36], and others (e.g., [5, 21]).



© Vojtěch Havlena, Michal Hečko, Lukáš Holík, and Ondřej Lengál;
licensed under Creative Commons License CC-BY 4.0

50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025).

Editors: Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak; Article No. 56; pp. 56:1–56:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The most essential constraints, from the practical perspective, are considered to be word equations, regular membership constraints, length constraints, and also $\neg\text{Contains}$, as argued, e.g., in [44], and as can also be seen in benchmarks, for instance, in [45, 3]. While the three former types of constraints are intensely studied, $\neg\text{Contains}$ was studied only little. Yet, it is important as well as theoretically interesting: besides the occurrence in existing benchmarks, its importance follows also from its ability to capture other highly practical types of constraints. E.g., the $\text{indexOf}(x, y)$ function should return the position of the first occurrence of y in x . It can be converted to the word equation $x = p.y.s$ after which the returned value equals $|p|$. To ensure that y is indeed the first occurrence in x , there should be no occurrence of y in $p.y'$ where y' is the prefix of y without the last symbol, i.e., $y'.z = y$ for $z \in \Sigma$. This can be expressed as $\neg\text{Contains}(y, p.y')$ (e.g., Z3 solves indexOf in this way [23]).

As mentioned above, the problem is also interesting from the theoretical perspective. Although the positive version, Contains , can be easily encoded using word equations, the negation is difficult. Its precise conversion to word equations would require universal quantification, which is undecidable for word equations in general [22]. The most systematic attempts at solving $\neg\text{Contains}$ have been made in [3, 19]. In [3], the authors extend the flattening underapproximating framework behind the solver TRAU [2, 1] and give a precise solution for $\neg\text{Contains}$ if all involved string variables are constrained by flat languages (a flat language here stands for a finite union of concatenations of iterations of words) and, moreover, if no string variable appears multiple times, thus avoiding most of the difficulty of the problem. Our recent work [19], on top of which we build here, proceeds in a similar direction and removes the restriction of [3] on multiple occurrences of variables, but still requires all languages to be flat, which is a quite severe restriction. Practical heuristics used in solvers generally solve only easy cases and quickly fail on more complex ones, cf. [19], and do not give any guarantees. E.g., CVC5 translates $\neg\text{Contains}$ into a universally quantified disequality [40], which is in turn handled by CVC5's incomplete quantifier instantiation [37].

In this paper, we show decidability of a much more general kind of $\neg\text{Contains}$ than [19, 3], namely of the form $\neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ where $\mathcal{N}$ and $\mathcal{H}$ are string terms (sequences of symbols and variables) and $\Phi_{\mathcal{L}}$ constrains variables by *any regular language*. The constraint is satisfied by an assignment to string variables respecting $\Phi_{\mathcal{L}}$ under which $\mathcal{N}$ is not a factor (i.e., a continuous subword) of $\mathcal{H}$ (i.e., if $\mathcal{N}$ cannot be found in $\mathcal{H}$).

Our solution of the problem leads relatively deep into word combinatorics and automata theory. We rely on the result in [19] giving a decision procedure for a flat-language version of the problem. The work [19] uses an automata-based construction inspired by deciding functional equivalence of streaming string transducers [6]. Using a variation on automata Parikh images, it transforms the problem into an equisatisfiable Presburger arithmetic formula (which is decidable). The general case with variables restricted by arbitrary regular languages, the subject of this paper, is solved by a reduction to this flat-language fragment. The core idea of our proof is that we can always find fresh primitive words in non-flat languages that can be repeated an arbitrary number of times. The result of such a repetition is a word that can share with other variables only subwords of a bounded size, assuming all words assigned to variables are sufficiently long. The reduction technically requires a dive into combinatorics on words and results on primitive words [31, 35, 34, 30], which are closely related to flat languages. Our techniques shares traits with the work of Karhumäki et al. [25], which constructs long primitive words to show that disjunctions of word equations can be encoded into a single equation. First, for variables with non-flat languages occurring on both sides of the constraint, we show that we can replace each of them with a single fresh symbol. This is because non-flat languages allow us to choose a sufficiently complex word for the variable x that can be matched only with the value of x on the other side ($\mathcal{N}$ is the other

side of $\mathcal{H}$ and *vice versa*). For variables with non-flat languages that appear only in $\mathcal{H}$, we show that after enumerating all possible assignments for them up to a certain bound, their languages can be underapproximated by flat languages while preserving satisfiability.

2 Preliminaries

Numbers. We use $\mathbb{N}$ for natural numbers (including zero). For $m, n \in \mathbb{N}$, their *greatest common divisor* is denoted as $\text{gcd}(m, n)$ and their *least common multiple* is denoted as $\text{lcm}(m, n)$.

Words. An *alphabet* Σ is a finite non-empty set of *symbols*. Let Σ be fixed for the rest of the paper. A (finite) word w over Σ is a sequence of symbols $w = a_1 \dots a_n$ from Σ , where n is the *length* of w , denoted as $|w|$. The *empty word* of the length 0 is denoted by ϵ and a concatenation of two words u and v is denoted as $u \circ v$ (or shortly uv). An *iteration* of a word w is defined as $w^0 \triangleq \epsilon$ and $w^{i+1} \triangleq w^i \circ w$ for $i \geq 0$. The set of all words over Σ is denoted as Σ^* . A *primitive word* cannot be written as v^i for any v and $i > 1$, and we will use Greek letters $\alpha, \beta, \gamma, \dots$ from the beginning of the alphabet to denote primitive words. We denote the set of all primitive words Prim . A word u is a *factor* (i.e., a continuous subword) of every word vuv' . Given two words pus and $p'u's'$, we say that the factors u and u' have an overlap of size $k \in \mathbb{N}$ if $|\{ |p| + 1, \dots, |p| + |u| \} \cap \{ |p'| + 1, \dots, |p'| + |u'| \}| = k$. The overlap of u and u' in the words pus and $p'u's'$ contains a *conflict* if there is a position i with $|p| \leq i < |pu|$ and $|p'| \leq i < |p'u'|$ such that the words pus and $p'u's'$ contain a different letter at position i .

Languages. A *language* $\mathcal{L}$ over Σ is a subset of Σ^* . We will sometimes abuse notation and, given a word $w \in \Sigma^*$, use w to also denote the language $\{w\}$. For two languages $\mathcal{L}_1$ and $\mathcal{L}_2$, we use $\mathcal{L}_1 \circ \mathcal{L}_2$ (or just $\mathcal{L}_1 \mathcal{L}_2$) for their concatenation $\{uv \mid u \in \mathcal{L}_1, v \in \mathcal{L}_2\}$. A *bounded iteration* of a language $\mathcal{L}$ is defined as $\mathcal{L}^0 \triangleq \{\epsilon\}$ and $\mathcal{L}^{i+1} \triangleq \mathcal{L}^i \circ \mathcal{L}$ for $i \geq 0$. The (*unbounded*) *iteration* is $\mathcal{L}^* \triangleq \bigcup_{i \geq 0} \mathcal{L}^i$. For a word w we use $\text{Pref}(w)$ ($\text{Suf}(w)$) to denote the set of prefixes (suffixes) of w and $\text{F}(w)$ to denote the set of all factors of w . We lift the definitions to languages as usual. A language $\mathcal{L} \subseteq \Sigma^*$ is *flat* iff it can be expressed as a finite union

$$\mathcal{L} = \bigcup_{i=1}^N w_{i,1} \circ w_{i,2}^* \circ w_{i,3} \circ w_{i,4}^* \circ w_{i,5} \circ \dots \circ w_{i,\ell_i-1}^* \circ w_{i,\ell_i} \quad (1)$$

where every $w_{i,j}$ s.t. $1 \leq i \leq n, 1 \leq j \leq \ell_i$ is a word over Σ , else it is *non-flat*. Flatness of $\mathcal{L}$ can be characterised by the absence of the so-called “butterfly loops”:

► **Fact 1.** A regular language $\mathcal{L} \subseteq \Sigma^*$ is non-flat iff $p\{u, v\}^*s \subseteq \mathcal{L}$ for some $p, s, u, v \in \Sigma^*$ with $u, v \notin w^*$ for any word $w \in \Sigma^*$.

Automata. A (*nondeterministic finite*) *automaton* (NFA) over Σ is a tuple $\mathcal{A} = (Q, \Delta, I, F)$ where Q is a set of *states*, Δ is a set of *transitions* of the form $q \xrightarrow{a} r$ with $q, r \in Q$ and $a \in \Sigma$, $I \subseteq Q$ is the set of *initial states*, and $F \subseteq Q$ is the set of *final states*. A run of $\mathcal{A}$ over a word $w = a_1 \dots a_n$ from state q_0 to state q_n is a sequence of transitions $q_0 \xrightarrow{a_1} q_1, q_1 \xrightarrow{a_2} q_2, \dots, q_{n-1} \xrightarrow{a_n} q_n$ from Δ . The empty sequence is a run with $q_0 = q_n$ over ϵ . We denote by $q_0 \xrightarrow{w}_{\mathcal{A}} q_n$ that $\mathcal{A}$ has such a run, from where we drop the subscript $\mathcal{A}$ if it is clear from the context. The run is *accepting* if $q_0 \in I$ and $q_n \in F$, and the *language* of $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) \triangleq \{w \in \Sigma^* \mid q \xrightarrow{w} r, q \in I, r \in F\}$. Languages accepted by NFAs are called *regular*. $\mathcal{A}$ is a *deterministic finite automaton* (DFA) if $|I| = 1$ and for every symbol $a \in \Sigma$ and every pair of transitions $q_1 \xrightarrow{a} r_1$ and $q_2 \xrightarrow{a} r_2$ in Δ it holds that if $q_1 = q_2$ then $r_1 = r_2$.

The $\neg$ Contains constraint. Let $\mathbb{X}$ be a set of (*string*) *variables*. A *term* is a word $t \in (\mathbb{X} \cup \Sigma)^*$ over variables and symbols. A $\neg$ Contains constraint is a formula $\varphi \triangleq \neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$, where $\mathcal{N}$ and $\mathcal{H}$ (for *N*eedle and *H*aystack; φ holds if we cannot find $\mathcal{N}$ within $\mathcal{H}$) are terms and $\Phi_{\mathcal{L}} \triangleq \bigwedge_{x \in \mathbb{X}} x \in \mathcal{L}(x)$ associates every variable x with a regular language $\mathcal{L}_x$. An *assignment* is a function $\sigma: \mathbb{X} \rightarrow \Sigma^*$, i.e., it assigns strings to variables. We use $\sigma \triangleleft \{x_1 \mapsto w_1, \dots, x_n \mapsto w_n\}$ to denote the assignment obtained from σ by substituting the values of variables $x_1, \dots, x_n$ to $w_1, \dots, w_n$ respectively. We lift σ to terms so that for $a \in \Sigma$, we let $\sigma(a) \triangleq a$, and for terms t, t' , we let $\sigma(t \circ t') \triangleq \sigma(t) \circ \sigma(t')$. We then say that σ satisfies φ , written $\sigma \models \varphi$, if $\sigma(x) \in \mathcal{L}_x$ for every $x \in \mathbb{X}$ and $\sigma(\mathcal{H})$ cannot be written as $u \circ \sigma(\mathcal{N}) \circ v$ for any $u, v \in \Sigma^*$, i.e., $\sigma(\mathcal{N})$ is not a factor of $\sigma(\mathcal{H})$. We call a variable z *two-sided* if it occurs in both $\mathcal{N}$ and $\mathcal{H}$. Moreover, we use $\mathbb{X}_{\text{Flat}}$ to denote the set of variables x occurring in φ s.t. $\mathcal{L}_x$ is a flat language.

Given a term t , a variable $x \in \mathbb{X}$, and a term t_s , we use $t[x/t_s]$ to denote the term obtained by substituting every occurrence of the variable x in t by the term t_s . Moreover, we use $\text{Vars}(t)$ to denote the set of variables with at least one occurrence in the term t .

► **Theorem 2** ([19, Theorem 7.5]). *Satisfiability of the $\neg$ Contains constraint is NP-hard.*

2.1 Normalization

A variable z is *flat* (*non-flat*) if the language $\mathcal{L}_z$ associated with z is flat (*non-flat*), respectively, and *finite* if its corresponding language is finite. Moreover, a variable is called *decomposed* if its language can be represented by a DFA having a single initial, single final state, and containing exactly one nontrivial *maximal strongly connected component* (SCC) and no other SCCs. We say that φ is *normalized* if it contains an occurrence of at least one variable, does not contain any finite variable, and all of its variables are decomposed. Any $\neg$ Contains constraint can be transformed into a disjunction of normalized constraints, as shown by the following lemma.

► **Lemma 3.** *Let $\varphi \triangleq \neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$. Then φ can be transformed to an equisatisfiable disjunction $\bigvee_{1 \leq i \leq n} \neg\text{Contains}(\mathcal{N}_i, \mathcal{H}_i) \wedge \Phi_{\mathcal{L}_i}$ of normalized constraints or the formula true.*

Due to the previous lemma, in the rest of the paper we will focus on solving a single normalized $\neg$ Contains constraint.

In the paper, we will also make use of the following result showing decidability of $\neg$ Contains with only flat variables.

► **Lemma 4** ([19]). *Satisfiability of $\neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ where $\mathcal{L}_x$ is flat for any $x \in \mathbb{X}$ is decidable in NEXPTIME.*

Proof sketch. We can reduce φ into an equisatisfiable Presburger arithmetic formula ψ based on Parikh images of runs of the NFAs for the variables in φ . Decidability of φ follows from decidability of Presburger arithmetic. See [19] for details. ◀

The crucial fact that Lemma 4 depends on is that there is a one-to-one mapping between runs in NFAs of flat languages and their Parikh images; this mapping fundamentally breaks for non-flat languages so one cannot directly extend this technique to the non-flat case.

2.2 Lemmas in Our Toolbox

We introduce fundamental lemmas from the area of combinatorics on words that will be used throughout the rest of the paper. The following lemma will be useful to guarantee the existence of conflicts (i.e., non-matching positions) in sufficiently large overlaps of two words

α^M and β^N for some primitive words $\alpha, \beta \in \Sigma^*$ and large constants $M, N \in \mathbb{N}$. Intuitively, we will control the choice of α and β , and, thus, guarantee that α and β cannot be powers of the same word, essentially applying the contraposition of the following lemma.

► **Lemma 5.** *Let $\alpha \in \Sigma^*$ be a primitive word, and let p and s be two words such that $\alpha = ps$. Then the word sp is primitive.*

Proof. Assume that $\beta^k = sp$ for some $k \geq 2$. Then we have $s = \beta^l u$ and $p = v\beta^m$ for $u \in \text{Pref}(\beta)$, $v \in \text{Suf}(\beta)$ such that $\beta = uv$ and $l + m + 1 = k$. Thus, we have

$$\alpha = v\beta^m\beta^l u = v(uv)^m(uv)^l u = (vu)^{l+m+1} \quad (2)$$

and so the word α is not primitive, a contradiction. ◀

► **Lemma 6** ([31, Proposition 1.2.1 (Fine and Wilf)]). *Let x and y be two words. If the words x^k and y^l , for any $k, l \in \mathbb{N}$ share a common prefix of the length at least $|x| + |y| - \text{gcd}(|x|, |y|)$, then x and y are powers of the same word.*

Using Lemmas 5 and 6, we provide the following corollary that shows existence of conflicts between arbitrary overlaps of repetitions of primitive words of a sufficient size.

► **Corollary 7.** *Let $u = \alpha^M$ and $v = \beta^N$ be two words where $\alpha, \beta \in \text{Prim}$, with $|\alpha| \neq |\beta|$ and $M, N \in \mathbb{N}$. Then any overlap between u and v of the size at least $|\alpha| + |\beta| - \text{gcd}(|\alpha|, |\beta|)$ contains a conflict.*

A natural approach to showing that an assignment σ satisfies φ is to show that $\sigma(\mathcal{H})$ cannot be written as $\sigma(\mathcal{H}) = p \circ \sigma(\mathcal{N}) \circ s$ for any choice of words p and s . Therefore, one would have to consider all prefixes p , infixes u , and corresponding suffixes s with $|u| = |\sigma(\mathcal{N})|$ and show that $\sigma(\mathcal{H}) = pus$ implies $u \neq \sigma(\mathcal{N})$. Note that the choice of the prefix $p \in \text{Pref}(\sigma(\mathcal{H}))$ uniquely determines u and s , and, therefore, we can only refer to different prefixes when showing $\sigma \models \varphi$. The following lemma reduces the number of prefixes we have to consider if we have information about primitive words that are factors of $\sigma(\mathcal{N})$ and $\sigma(\mathcal{H})$.

► **Lemma 8** ([31, Proposition 12.1.3]). *Let $\alpha \in \Sigma^*$ be a primitive word, and let $\alpha^2 = x\alpha y$ for some words $x, y \in \Sigma^*$. Then either $x = \epsilon$ or $y = \epsilon$, but not both.*

We will use the next lemma as a recipe for constructing words $w_z \in \mathcal{L}_z$ for non-flat $\mathcal{L}_z$ such that w_z has as a factor a primitive word that is sufficiently long for our proofs.

► **Lemma 9** ([34]). *Let $x^K = y^L z^M$ such that x, y , and z are string variables and K, L and M are integers such that $K, L, M \geq 2$. Then any solution of the equation has the form $x = \alpha^k$, $y = \alpha^l$, and $z = \alpha^m$ for some word α and numbers $k, l, m \in \mathbb{N}$.*

We provide the following corollary to give insight into how we use Lemma 9 to construct factors that are primitive words of a suitable length.

► **Corollary 10.** *Given two words u and v such that for any word w it holds that $u, v \notin w^*$, we have that any word $\alpha = u^L v^M$ for $L, M \geq 2$ is primitive.*

Proof. By contradiction. Assume that α is not primitive, i.e., $\alpha = t^K = u^L v^M$ for some t and $K, L, M \geq 2$. Applying Lemma 9, we see that $u = w^l$ and $v = w^m$ for some w , which contradicts the assumptions of the corollary. ◀

2.3 Easy Fragments

Before we establish our main result giving the decidability of the hardest fragment of $\neg\text{Contains}$, we first describe what we consider *easy fragments* and how to deal with them. We assume a normalized $\neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ constraint.

1. *The formula is solvable by length abstraction.* This fragment contains formulae that can be solved easily by making the $\mathcal{N}$ eedle longer than the $\mathcal{H}$ aystack. Suppose $\mathcal{N} = t_1 \dots t_m$ and $\mathcal{H} = s_1 \dots s_n$ where every t_i and s_j is either a string variable $x \in \mathbb{X}$ or a symbol $a \in \Sigma$. We can then create a Presburger arithmetic formula φ_{ℓ} over *length variables* $\{x_{\ell} \mid x \in \mathbb{X}\}$ such that $\varphi_{\ell} \triangleq \sum_{1 \leq i \leq m} \ell_i > \sum_{1 \leq j \leq n} \ell_j \wedge \Psi$. In the formula, ℓ_i and ℓ_j are either 1 (if $t_i, s_j \in \Sigma$) or the length variable x_{ℓ} (if $t_i, s_j = x$), and Ψ is a formula constraining the possible values for the length variables (obtained, e.g., using the Parikh images of the variables' languages). If φ_{ℓ} is satisfiable, so is the original $\neg\text{Contains}$.
2. *All variables are flat.* In this case, we can use Lemma 4.

3 Overview

We now move to our main result: deciding a *hard* instance of $\varphi \triangleq \neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$. We can classify normalized $\neg\text{Contains}$ constraints (cf. Section 2.1) that do not fall in the fragments of Section 2.3 based on the occurrences of non-flat variables as follows:

1. constraints where a non-flat variable x occurs both in $\mathcal{N}$ and $\mathcal{H}$ and
2. constraints where all (and at least one) non-flat variables occur only in $\mathcal{H}$.

Note that the above not included cases of **(a)** all variables being flat and **(b)** a non-flat variable being only in $\mathcal{N}$ are covered in Section 2.3. In particular, if there is a variable x that only occurs in $\mathcal{N}$, then $\mathcal{L}_x$ is infinite due to our normalization. Therefore, such a constraint can be solved by making $\mathcal{N}$ longer than $\mathcal{H}$.

We distinguish the classes (1) and (2) above since for (1), the string substituted for some occurrence of x in $\sigma(\mathcal{H})$ may overlap with the string for an occurrence of x in $\sigma(\mathcal{N})$. We deal with the class (1) by substituting two-sided non-flat variables x with fresh symbols. In Section 4, we show that if there is a model σ of the resulting $\neg\text{Contains}$, we can obtain a model σ' of the original constraint φ from σ by assigning $\sigma'(x)$ to a long-enough word that ensures a mismatch for every overlap of $\sigma'(x)$ in $\sigma'(\mathcal{H})$ and $\sigma'(x)$ in $\sigma'(\mathcal{N})$. By doing this, we reduce (1) to either (2) or $\neg\text{Contains}$ over flat variables (potentially with no variables at all).

For deciding the class (2), given in detail in Section 6, we construct an equisatisfiable formula that uses flat underapproximations of languages associated with the remaining (as some might have been removed at step (1)) non-flat variables present in $\mathcal{H}$. Our result is based on the observation that long words in a non-flat language may have a richer structure compared to long words one can construct using flat languages. Therefore, it is unlikely that a flat variable z should have a large conflict-free overlap with a non-flat variable x in an assignment that assigns these two variables sufficiently long words. In particular, we prove that the original language of x can be underapproximated by a flat language while preserving equisatisfiability. After this step, the resulting constraint can be decided using Lemma 4.

4 Removing Two-Sided Non-Flat Variables

In this section, we will show how to transform a normalized constraint $\neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ with an occurrence of a two-sided non-flat variable x into a constraint without occurrences of the variable x . The resulting constraint after removing all two-sided non-flat variables

can then be solved either by reduction to Presburger arithmetic (Lemma 4; if no non-flat variables remain in the constraint) or by the procedure in Section 6 (if there are still non-flat variables left in $\mathcal{H}$). The main result of this section is the following theorem.

► **Theorem 11.** *Let $\varphi \triangleq \neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ be a constraint over the alphabet Σ and let $z \in \text{Vars}(\mathcal{N}) \cap \text{Vars}(\mathcal{H})$ be a non-flat variable. Then the formula $\varphi_{\#} \triangleq \neg\text{Contains}(\mathcal{N}[z/\#], \mathcal{H}[z/\#]) \wedge \Phi_{\mathcal{L}}$ with $\# \notin \Sigma \cup \mathbb{X}$ is equisatisfiable to φ .*

The proof of the theorem is given below. It is based on the observation that assigning long words to two-sided variables necessarily causes some occurrences of the same variable to overlap. Since these variables are non-flat, we can construct long words with a rich internal structure that will guarantee that any sufficiently long overlap necessarily contains a conflict.

Before we give the proof, let us formally introduce the concept of words that do not allow conflict-free overlaps of two occurrences of the same word larger than a certain bound.

► **Definition 12** (ℓ -aligned word). *Let w be a word and $\ell \in \mathbb{N}$. We say that w is ℓ -aligned if for all $p \in \Sigma^*$ such that $1 \leq |p| \leq |w| - \ell$, w is not a prefix of pw .*

Intuitively, w is ℓ -aligned if it cannot overlap with itself on a prefix/suffix of the length larger than or equal to ℓ (except $|w|$). For example, the word $w = abaa$ is 2-aligned since for no non-empty word p of the length at most $|w| - 2 = 2$ it holds that w is a prefix of pw . On the other hand, $w = abaa$ is not 1-aligned since for $p = aba$ of the length 3, it holds that w is a prefix of $pw = abaabaa$.

4.1 Proof of Theorem 11

If φ is satisfiable then so is $\varphi_{\#}$. To see this, take a model $\sigma \models \varphi$ and replace the assignment of z to $\#$, producing σ' . Then, there will be conflicts of $\#$ and some non- $\#$ symbol when checking whether σ' is a model of $\varphi_{\#}$. Alternatively, it might be possible to align $\sigma'(\mathcal{N})$ with $\sigma'(\mathcal{H})$ in a manner such that every $\#$ in $\sigma'(\mathcal{N})$ matches some $\#$ in $\sigma'(\mathcal{H})$. In such a case, if σ' fails to be a model of $\varphi_{\#}$ we reach a contradiction with σ being a model of φ .

For the other direction, assume that $\varphi_{\#}$ is satisfiable, which means there is a model σ' of $\varphi_{\#}$. Next, we will show how to construct a word w_z s.t. $\sigma = \sigma' \cup \{z \mapsto w_z\}$ is a model of φ .

Focusing on variable z , we can write the two sides of the $\neg\text{Contains}$ constraint as $\mathcal{H} = \mathcal{H}_0 z_{\mathcal{H},1} \mathcal{H}_1 \cdots \mathcal{H}_{n-1} z_{\mathcal{H},n} \mathcal{H}_n$ and $\mathcal{N} = \mathcal{N}_0 z_{\mathcal{N},1} \mathcal{N}_1 \cdots \mathcal{N}_{m-1} z_{\mathcal{N},m} \mathcal{N}_m$ where $\mathcal{N}_i, \mathcal{H}_j \in (\mathbb{X}' \cup \Sigma)^*$ for each i and j assuming $\mathbb{X}' = \mathbb{X} \setminus \{z\}$. Moreover, we write the subscript $z_{S,k}$ to distinguish k -th occurrence of z in $S \in \{\mathcal{H}, \mathcal{N}\}$. As $\mathcal{L}_z$ is non-flat, we have that $p\{u, v\}^*s \subseteq \mathcal{L}_z$ for some words p, u, v , and s where u and v are not a power of the same word (Fact 1).

The core of our proof is based on the following observation. Since σ' is a model of $\varphi_{\#}$, the word $\sigma'(\mathcal{N}[z/\#])$ is not a factor of $\sigma'(\mathcal{H}[z/\#])$. Therefore, given any sufficiently long word $w_z \in \mathcal{L}_z$, if the extension $\sigma = \sigma' \cup \{z \mapsto w_z\}$ fails to be a model, then there must be at least one occurrence of the word $\sigma(z)$ in $\sigma(\mathcal{N})$ partially overlapping with an occurrence of the word $\sigma(z)$ in $\sigma(\mathcal{H})$, as shown in the picture below. Thus, if we construct a word $w_z \in \mathcal{L}_z$ that cannot partially overlap with itself, we get σ that is a model of φ .

...	$\sigma(\mathcal{H}_i)$	w_z	$\sigma(\mathcal{H}_{i+1})$	...
...	$\sigma(\mathcal{N}_j)$	w_z	$\sigma(\mathcal{N}_{j+1})$	...

Let $\alpha = u^2 u^k v^2$ and $\beta = u^2 v^l v^2$ be two words where $k = \text{lcm}(|v|, |u|)/|u|$ and $l = \text{lcm}(|v|, |u|)/|v|$. By invoking Corollary 10, we see that both α and β are primitive.

Note that we also have $|\alpha| = |\beta|$ (because $|\alpha| = 2|u| + k|u| + 2|v|$, $|\beta| = 2|u| + l|v| + 2|v|$ and from the definition of l and k we have $k|u| = l|v|$). We now use these two primitive words α and β to construct w_z . Let $\gamma \triangleq \alpha^r \beta^r \circ \alpha^r \beta^r \circ \alpha^{2r} \beta^{2r}$ and let $w_z \in \mathcal{L}_z$ be the word $w_z \triangleq p\gamma s$ where $r \geq 2$ is the smallest number satisfying $r|\alpha| > M + |p| + |s|$ with $M = \max\{|\sigma(\mathcal{H}_i)|, |\sigma(\mathcal{N}_j)| : 1 \leq i \leq n, 1 \leq j \leq m\}$. We set $\sigma = \sigma' \cup \{z \mapsto w_z\}$. Let us now give two lemmas establishing the properties of w_z 's infix γ .

We constructed the infix γ of w_z in a way so that it prevents conflict-free overlaps with itself as shown by the following lemma.

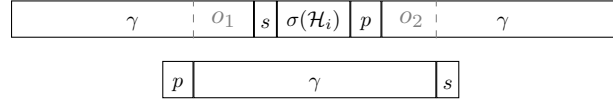
► **Lemma 13.** *The word γ is $(r+1)|\alpha|$ -aligned.*

The full proof of Lemma 13 can be found in [24], but the core of the argument lies in observing that in any overlap of γ with itself of size at least $(r+1)|\alpha|$, there is a factor α^2 having an overlap with α of size $|\alpha|$, or, similarly for β . Therefore, one can apply Lemma 8 and limit overlaps that must be considered.

The following lemma shows that long overlaps between two occurrences of γ are unavoidable when γ has a sufficient length. The a^i in the lemma is used just to position the overlap within $\sigma(\mathcal{H})$ and $\sigma(\mathcal{N})$.

► **Lemma 14.** *For each $0 \leq i \leq |\sigma(\mathcal{H})| - |\sigma(\mathcal{N})|$, every occurrence of γ in $\sigma(\mathcal{H})$ has an overlap with some occurrence of γ in $a^i \circ \sigma(\mathcal{N})$ of size at least $(r+1)|\alpha|$, where a is any symbol in Σ .*

Proof. Let us assume an arbitrary occurrence of γ in $\sigma(\mathcal{H})$. Since each $\mathcal{H}_i$ and $\mathcal{H}_{i+1}$ are separated by z (the same goes for $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$), it suffices to consider only the pessimistic case, which is when an occurrence of γ in $\mathcal{N}$ matches the longest $\sigma(\mathcal{H}_i)$ with p and s on both sides. The situation is schematically depicted below.



In the figure, o_1 and o_2 denote the overlaps on both sides. We show that the size of at least one overlap o_1 and o_2 is greater than $(r+1)|\alpha|$ by expressing the length $|\gamma|$ using its definition and the schematic above:

$$\begin{aligned}
 r(4|\beta| + 4|\alpha|) &= |o_1| + |s| + |\sigma(\mathcal{H}_i)| + |p| + |o_2| && \text{[def. of } \gamma\text{]} \\
 \Rightarrow 7r|\alpha| + r|\alpha| &\leq |o_1| + r|\alpha| + |o_2| && \text{[since } |\alpha| = |\beta| \text{ and def. of } r\text{]} \quad (3) \\
 \Leftrightarrow 7r|\alpha| &\leq |o_1| + |o_2|
 \end{aligned}$$

We have that $|o_1| + |o_2| \geq 7r|\alpha|$, and, thus, at least one of $|o_1|$ and $|o_2|$ is bigger than $3r|\alpha|$. Since $r \geq 2$, we have $3r|\alpha| \geq (r+1)|\alpha|$ and hence γ has an overlap of the required size. ◀

It remains to show that σ is a model of φ . For the sake of contradiction, assume that σ is not a model, meaning that $\sigma(\mathcal{N})$ is a factor of $\sigma(\mathcal{H})$. From Lemmas 13 and 14 we have that each occurrence of γ in $\sigma(\mathcal{N})$ is perfectly aligned with some γ in $\sigma(\mathcal{H})$, which also means that w_z 's are perfectly aligned. Furthermore, we have that w_z 's in $\sigma(\mathcal{N})$ are aligned with consecutive w_z 's in $\sigma(\mathcal{H})$, i.e., any $\sigma(z_{\mathcal{N},i})$ is aligned with some $\sigma(z_{\mathcal{H},i+k})$ for some $0 \leq k \leq n - m$. If this were not the case and we had $\sigma(z_{\mathcal{N},1})$ overlapping with $\sigma(z_{\mathcal{H},1+k})$ while there were some $\sigma(z_{\mathcal{N},i})$ matching with $\sigma(z_{\mathcal{H},i+k+l})$ for $l \geq 1$, there would have to be some $\sigma(\mathcal{N}_j)$ with $1 \leq j < i$ with $|\sigma(\mathcal{N}_j)| > |\sigma(z)|$, which is a contradiction with w_z being longer than any $|\sigma(\mathcal{N}_j)|$ by construction. Hence, for $\sigma'' = \sigma \triangleleft \{z \mapsto \#\}$, $\sigma''(\mathcal{N})$ is also a factor of $\sigma''(\mathcal{H})$, which is a contradiction to σ' being a model of $\varphi_{\#}$. Therefore, Theorem 11 holds.

5 Γ -Expansion and Prefix/Suffix Trees

At this point, we are left with a normalized $\neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ constraint where all variables in $\mathcal{N}$ are flat. If all variables in $\mathcal{H}$ are also flat, we can use Lemma 4 and obtain the result. In the rest of the paper, we will deal with the case when $\mathcal{H}$ contains at least one non-flat variable. Before we give the proof in Section 6, in this section, we introduce two concepts that will be used later: Γ -expansion on non-flat variables and prefix/suffix trees.

5.1 Γ -Expansions on Non-Flat Variables

Intuitively, non-flat languages have words with a rich internal structure compared to flat languages. To illustrate, let x be a flat variable with the language $\mathcal{L}_x = \alpha^*$ for some word α and let z be a non-flat variable with the language $\mathcal{L}_z$. Furthermore, let $w_x \in \mathcal{L}_x$ and $w_z \in \mathcal{L}_z$ be two sufficiently long words. We inspect the case when w_x and w_z share some long common factor u . Since $w_x \in \alpha^*$, we have $u = s\alpha^k p$ for some $s \in \text{Suf}(\alpha)$, $p \in \text{Pref}(\alpha)$, and $k \in \mathbb{N}$. As z is non-flat, the run of $\mathcal{A}_z$ corresponding to the word w_z passes through states at which one can make a choice of which transition to take next. Since u is long, we have to make a lot of “right” choices during the run of $\mathcal{A}_z$ in order to achieve the common factor u , highlighting the difference between the complexity of $\mathcal{L}_x$ and $\mathcal{L}_z$, and suggesting that there is a way to pick w_z to prevent long overlaps with flat variables occurring in $\mathcal{N}$.

Guided by this intuition, we introduce a tool called Γ_z -expansion of a non-flat variable z . Given a prefix $p \in \text{Pref}(\mathcal{L}_z)$ and a suffix $s \in \text{Suf}(\mathcal{L}_z)$, the Γ_z -expansion of (p, s) is the word $\Gamma_z(p, s) = pws \in \mathcal{L}_z$ for a particular w such that only a prefix or a suffix of a bounded length can have long overlaps with (sufficiently long) words that belong to a flat language. This tool will play an important role in our proofs. Loosely speaking, if we start with a model σ and we try to find an alternative model $\sigma' = \sigma \triangleleft \{z \mapsto \Gamma_z(p, s)\}$, then the possible reasons why σ' fails to be a model are narrowed down to the choice of p and s .

In order to define the Γ_z -expansion, we first need some auxiliary definitions. First, as a resulting of our normalization, the map $\text{Base}: \mathbb{X}_{\text{Flat}} \rightarrow 2^{\Sigma^*}$ maps any flat variable x to a singleton containing the primitive word α that forms the basis of $\mathcal{L}_x$, i.e., $\text{Base}(x) \triangleq \{\alpha\}$ such that $\mathcal{L}_x = (\alpha^k)^*$ for some $k \in \mathbb{N}$. We lift the definition of Base to a set X of flat variables as $\text{Base}(X) \triangleq \bigcup_{x \in X} \text{Base}(x)$, and to a string $s \in (\Sigma \cup \mathbb{X}_{\text{Flat}})^*$ as $\text{Base}(s) \triangleq \text{Base}(\text{Vars}(s) \cap \mathbb{X}_{\text{Flat}})$.

Second, given a variable $z \in \mathbb{X}$ with $\mathcal{A}_z = (Q, \Sigma, \Delta, I, F)$, we define the function $\text{con}_z: Q \times Q \rightarrow \Sigma^*$ to give the lexicographically smallest word $\text{con}_z(q, s) \triangleq w$ such that $q \xrightarrow{w}_{\mathcal{A}} s$. Having auxiliary definitions in place, we are ready to define Γ -expansion in the context of the formula $\varphi = \neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ with $\mathcal{N} \in (\mathbb{X}_{\text{Flat}} \cup \Sigma)^*$.

► **Definition 15** (Γ -expansion). *Let $z \in \text{Vars}(\mathcal{H})$ be a decomposed non-flat variable and $\mathcal{A}_z = (Q, \Sigma, \Delta, I, F)$ be a DFA s.t. $\mathcal{L}(\mathcal{A}_z) = \mathcal{L}_z$. Moreover, let $q_{u|v} \in Q$ be a state such that $q_{u|v} \xrightarrow{u} q_{u|v}$ and $q_{u|v} \xrightarrow{v} q_{u|v}$ with $u, v \notin w^*$ for any word w . Furthermore, let $p \in \text{Pref}(\mathcal{L}_z)$ be some prefix and q_p be a state such that $q_0 \xrightarrow{p} q_p$ for some $q_0 \in I$. Similarly, let $s \in \text{Suf}(\mathcal{L}_z)$ be a suffix and q_s be a state such that $q_s \xrightarrow{s} q_f$ for some $q_f \in F$.*

Let $\gamma_z \triangleq u^{2+k}v^2$ for a minimal $k \in \mathbb{N}$ such that $\gamma_z > |\alpha|$ for any $\alpha \in \text{Base}(\mathcal{N})$. Given $K \in \mathbb{N}$, we define the Γ_z^K -expansion of (p, s) to be the word $\Gamma_z^K(p, s) \triangleq p \circ \text{con}(q_p, q_{u|v}) \circ \gamma_z^K \circ \text{con}(q_{u|v}, q_s) \circ s$.

Intuitively, Γ_z -expansion takes a prefix p and finds the shortest word $\text{con}(q_p, q_{u|v})$ that takes the automaton to the state $q_{u|v}$ in which we have the freedom to read the words u and v in any suitable sequence. We loop through $q_{u|v}$ in a specific manner so that the resulting

factor γ_z is primitive thanks to Corollary 10. The situation with the suffix is symmetric. Note that in the following section, we use prefix/suffix variants of the Γ -expansion defined as $\Gamma_{\text{Pref}(z)}^K \triangleq p \circ \text{con}(q_p, q_{u|v}) \circ \gamma_z^K$ and $\Gamma_{\text{Suf}(z)}^K \triangleq \gamma_z^K \circ \text{con}(q_{u|v}, q_s) \circ s$.

The following lemma shows that Γ_z -expansion can be seen almost as introducing a fresh symbol $\#$ for the infix w connecting p and s into a word $pws \in \mathcal{L}_z$. Intuitively, if we have an assignment σ that assigns sufficiently long words to all flat variables, then we can find $K \in \mathbb{N}$ such that any large overlap between $\sigma(\mathcal{N})$ and $\sigma(z) = \Gamma_z^K(p, s)$ contains a conflict. We use M_{Lit} to be the length of the longest literal in φ , M_Q to be the number of states of the largest DFA specifying the language of some variable $x \in \mathbb{X}$, and $M_\alpha \triangleq \max\{|\alpha| : \alpha \in \text{Base}(\mathcal{N}) \cup \{\gamma_z\}\}$.

► **Lemma 16.** *Let $z \in \mathbb{X}$ be a decomposed non-flat variable, $p \in \text{Pref}(\mathcal{L}_z)$, and $s \in \text{Suf}(\mathcal{L}_z)$. Further, let $K \in \mathbb{N}$ be such that $K|\gamma_z| \geq 4M_\alpha + 2M_{\text{Lit}}$ and let σ be an assignment with (i) $|\sigma(x)| \geq 2M_\alpha$ for any flat variable x and (ii) $\sigma(z) = \Gamma_z^K(p, s)$. Every overlap between $\sigma(z)$ and $\sigma(\mathcal{N})$ of the size at least $\max(|p|, |s|) + M_Q + 2M_{\text{Lit}} + 2M_\alpha$ contains a conflict.*

Proof sketch. It suffices to observe that if the overlap of size at least N necessarily contains an overlap between γ_z^K and $\sigma(x) = \alpha^l$ for some flat variable x with $\alpha \in \text{Base}(x)$. The existence of a conflict follows from Corollary 7. The full proof can be found in [24]. ◀

Next, we show that Γ_z -expansion can be used to facilitate modularity in our proofs, allowing us to search for a suitable prefix p and a suitable suffix s separately. Searching for p and s separately requires subtle modifications to φ , resulting in us searching for p and s in the context of the modified formulae φ_{Pref} and φ_{Suf} , respectively. If we find models of φ_{Pref} and φ_{Suf} of a particular form, we compose them into a model of φ .

Let $z \in \text{Vars}(\mathcal{H})$ be a non-flat variable, and let z_{Pref} and z_{Suf} be two fresh variables with their languages restricted to $z_{\text{Pref}} \in \text{Pref}(\mathcal{L}_z)$ and $z_{\text{Suf}} \in \text{Suf}(\mathcal{L}_z)$. Let φ_{Pref} and φ_{Suf} be formulae defined as $\varphi_{\text{Pref}} \triangleq \varphi[z/z_{\text{Pref}} \circ \#]$ and $\varphi_{\text{Suf}} \triangleq \varphi[z/\# \circ z_{\text{Suf}}]$ where $\#$ is a fresh alphabet symbol. Further, let $\sigma^{\text{Pref}} \models \varphi_{\text{Pref}}$ and $\sigma^{\text{Suf}} \models \varphi_{\text{Suf}}$ be two models such that:

1. σ^{Pref} and σ^{Suf} agree on the values of variables different than z_{Pref} and z_{Suf} ,
2. $|\sigma^{\text{Pref}}(x)| > 2M_\alpha \wedge |\sigma^{\text{Suf}}(x)| > 2M_\alpha$ for any flat variable $x \in \mathbb{X}_{\text{Flat}}$,
3. $\sigma^{\text{Pref}}(z_{\text{Pref}}) = p\gamma_z^K$ and $\sigma^{\text{Suf}}(z_{\text{Suf}}) = \gamma_z^L s$ such that $n|\gamma_z| \geq 4M_\alpha + 2M_{\text{Lit}}$ for $n \in \{K, L\}$.

► **Lemma 17.** *The assignment $\sigma^{\text{Pref}} \triangleleft \{z \mapsto p\gamma_z^K s\}$ is a model of φ where $K = \min(K, L)$.*

Proof sketch. The idea behind the proof is that if $\sigma \not\models \varphi$, then $\sigma(z)$ would need to have a large conflict-free overlap with $\sigma(x)$ for some flat variable $x \in \mathbb{X}$. Applying Corollary 7, we reach a contradiction. The full proof of Lemma 17 can be found in [24]. ◀

5.2 Prefix (Suffix) Enumeration through Prefix (Suffix) Trees

Having defined Γ^K -expansion that acts similarly to inserting a fresh symbol $\#$ between a chosen $p \in \text{Pref}(\mathcal{L}_z)$ and a suffix $s \in \text{Suf}(\mathcal{L}_z)$ of a non-flat variable $z \in \text{Vars}(\mathcal{H})$, we can start enumerating prefixes $p \in \text{Pref}(\mathcal{L}_z)$ (or suffixes) up to a certain bound, while searching for a model. We introduce the concept of prefix (suffix) trees that play a major role in our proofs. Below, we give only the definition of a prefix tree; a suffix tree is defined symmetrically.

► **Definition 18 (Choice state).** *Let $\mathcal{A} = (Q, \Delta, I, F)$ be a DFA. We say that a state $q \in Q$ is a choice state if $|\{(q, a, r) \in \Delta : a \in \Sigma, r \in Q\}| > 1$. We write $C(\mathcal{A})$ to denote the set of all choice states of $\mathcal{A}$.*

► **Definition 19** (Prefix tree). Let $z \in \mathbb{X}$ be a variable with its language $\mathcal{L}_z$ given by a DFA $\mathcal{A}_z = (Q_z, \Delta_z, \{q_0\}, F_z)$. We define z 's prefix tree $T_z = (V_z, E_z, r_z, \text{st}_z, \mathcal{W}_z)$ as an (infinite finitely-branching) tree with vertices V_z rooted in $r_z \in V_z$ such that

- $\text{st}_z: V_z \rightarrow Q_z$ is a function that labels non-root vertices of T_z with $\mathcal{A}_z$'s choice states, i.e., $\text{st}_z(v) \in C(\mathcal{A}_z)$ for any $v \neq r_z$ and $\text{st}(r_z) = q_0$,
- $E_z \subseteq V_z \times \Sigma^+ \times V_z$ is a set of labelled edges such that $(v, a_1 \dots a_n, v') \in E_z$ iff there is a run $\text{st}_z(v) \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} \text{st}_z(v')$ in $\mathcal{A}_z$ where for all $0 < i < n$ it holds that $q_i \notin C(\mathcal{A})$.
- $\mathcal{W}_z: V_z \times V_z \rightarrow \Sigma^*$ is a function that maps any two vertices connected by an edge to the label on the edge, i.e., $\mathcal{W}_z(v, v') = w$ iff there exists an edge $(v, w, v') \in E_z$ and is undefined otherwise.

Intuitively, vertices of the tree are labeled by $\mathcal{A}_z$'s choice states $C(\mathcal{A}_z)$, i.e., states in which we can choose between multiple outgoing transitions along different alphabet symbols. Vertices s and s' are connected by an edge in T_z if $\text{st}(s')$ is reachable from $\text{st}(s)$ without passing through any choice state.

A path π in T_z is a sequence of vertices $\pi = s_0 \dots s_n$ where $(s_i, w_i, s_{i+1}) \in E_z$ for any $0 \leq i < n$. We lift the definition of $\mathcal{W}$ to paths as $\mathcal{W}(s_0 \dots s_n) \triangleq \mathcal{W}((s_0, s_1)) \circ \dots \circ \mathcal{W}((s_{n-1}, s_n))$.

► **Definition 20** (Dead-end vertex of a prefix tree). Let $\varphi = \neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ and $T_z = (V, E, v_0, \text{st}, \mathcal{W})$ be the prefix tree for $z \in \mathbb{X}$, and let $\sigma: (\mathbb{X} \setminus \{z\}) \rightarrow \Sigma^*$ be a partial assignment. A vertex $v_n \in V$ is called a dead end in T_z w.r.t. σ if $\sigma' \not\models \varphi[z/z\#]$ where $\sigma' \triangleq \sigma \triangleleft \{z \mapsto \mathcal{W}(v_0 \dots v_n)\}$ for $v_0 \dots v_n$ being the (single) path between v_0 and v_n in T_z .

Intuitively, dead-end vertices (and all vertices that are below them in the prefix tree) are not interesting for obtaining a $\neg\text{Contains}$ model. Consider, e.g., $\varphi \triangleq \neg\text{Contains}(\text{ab}x, xz) \wedge \Phi_{\mathcal{L}}$ with $\mathcal{L}_x = (\text{ab})^+$ and $\mathcal{L}_z = (\text{a}\{\text{b}, \text{c}\}\text{c})^*$. We have $\varphi[z/z\#] = \neg\text{Contains}(\mathcal{N}', \mathcal{H}') = \neg\text{Contains}(\text{ab}x, xz\#)$ and, thus, the vertex $v \in V_z$ corresponding to the prefix abca is a dead end in T_z w.r.t. $\sigma = \{x \mapsto \text{ab}\}$ since $\sigma'(\mathcal{N}') = \text{abab}$ is a factor of $\sigma'(\mathcal{H}') = \text{ababca}\#$.

► **Definition 21** (H -reaching path). Let $\pi = v_0 \dots v_n$ be a path in a prefix tree $T_z = (V, E, v_0, \text{st}, \mathcal{W})$ and $H \in \mathbb{N}$. We say that π is H -reaching if $|\mathcal{W}(v_0 \dots v_n)| \geq H \geq |\mathcal{W}(v_0 \dots v_{n-1})|$.

In our proof, we explore all prefixes of words in a language up to a certain bound H . As we have a prefix tree with edges labelled with words of (possibly) different lengths, stating that we have explored all prefixes of the length precisely H is problematic. Hence, the concept of H -reaching paths is a relaxation allowing paths (prefixes) to slightly vary in length.

6 Underapproximating Non-Flat Variables

In this section, we give the main lemma allowing to underapproximate the language of non-flat variables with a flat language. Throughout this section we use three constants $\lambda_{\text{Flat}}, \lambda_Q, \lambda_{\kappa} \in \mathbb{N}$ with the following semantics:

- The constant λ_Q is the length of prefixes (suffixes) of non-flat variables that we will enumerate in our proofs, searching a model that is shorter w.r.t. some non-flat variable.
- The constant λ_{Flat} is the minimal size of words assigned to flat variables occurring in $\mathcal{N}$.
- λ_{κ} is used as the value of the parameter K in every application of $\Gamma_{\text{Pref}(z)}^K$ or $\Gamma_{\text{Suf}(z)}^K$.

First, let us define parameters of $\neg\text{Contains}(\mathcal{N}, \mathcal{H})$ that we use to define the above bounds. Let M_{Lit} be the length of the longest string literal in $\mathcal{N}$ and $\mathcal{H}$, let M_Q be the largest number of states of a DFA associated with some variable. Furthermore, let M_{α} be the length of the longest word in the set $W_{\alpha}(\mathcal{N}) \cup W_{\gamma}$ where W_{γ} is the set of the primitive words γ_z used to define the Γ_z -expansion for every non-flat variable $z \in \text{Vars}(\mathcal{H})$.

Since λ_{Flat} and λ_κ depend on the value of λ_Q , we start by fixing $\lambda_Q \triangleq 2M_\alpha M_Q + M_{\text{Lit}}$. Intuitively, for any non-flat variable $z \in \mathbb{X}$, we set up λ_Q in a way so that if we consider all prefixes in T_z up to the length $M_\alpha M_Q$, then T_z will contain paths through any state $q \in Q_z$ since $\mathcal{A}_z$ is a single SCC. After extending these paths up to the length λ_Q , we can guarantee that T_z will contain all words read from any state q of the length at least M_α . Considering all possible words of the length $|\beta|$ for some $\beta \in \text{Base}(\mathcal{N})$ readable from a state will be crucial later, as we will show that there can be only a few such words if we fail to find an alternative model $\sigma' \triangleq \sigma \triangleleft \{z \mapsto w_z\}$ s.t. $|\sigma'(z)| < |\sigma(z)|$, assuming the existence of a model σ .

The remaining bounds λ_{Flat} and λ_κ are defined as $\lambda_{\text{Flat}} \triangleq \lambda_Q + 4M_\alpha + M_Q$ and $\lambda_\kappa \triangleq \lambda_{\text{Flat}} + 2M_\alpha + 2M_{\text{Lit}}$. Ignoring some technical details and due to reasons that will be revealed shortly, we need λ_{Flat} to be slightly longer than λ_Q , so that when we later construct $\sigma' \triangleq \sigma \triangleleft \{z \mapsto p\}$ for some particular prefix $|p| \leq \lambda_Q + M_Q$, we can establish some of the string that precedes an occurrence of z in $\sigma_{|\mathbb{X} \setminus \{z\}}(\mathcal{H})$ in the case σ' fails to be a model. Finally, λ_κ is set up so that together with λ_{Flat} they allow Lemma 17 to be applied, where λ_Q and λ_{Flat} play the role of K_0 and N_0 , respectively.

We remark that the exact values of λ_Q , λ_{Flat} , and λ_κ are not important when reading the proof for the first time. It is sufficient to note that $\lambda_Q < \lambda_{\text{Flat}} < \lambda_\kappa$, and that the difference in sizes between these bounds is sufficiently large.

6.1 Overcoming the Infinite by Equivalence with a Finite Index

Our procedure to decide $\varphi = \neg \text{Contains}(\mathcal{N}, \mathcal{H})$ containing non-flat variables in $\mathcal{H}$ originates in enumeration of partial assignments $\eta: \text{Vars}(\mathcal{N}) \rightarrow \Sigma^*$, since it is easy to find suitable values for non-flat variables when $\mathcal{N}$ is a literal due to us fixing values of all variables in $\mathcal{N}$. The problem is that there is an infinite number of such assignments. Our key observation allowing us prove that we can underapproximate non-flat languages using flat ones is that the precise values of flat variables occurring in $\mathcal{N}$ does not matter as long as these variables have assigned sufficiently long words. In general, however, a model $\sigma \models \varphi$ might assign long words only to a subset of flat variables. Therefore, in our decision procedure, we first guess the set X of flat variables that are assigned words shorter than λ_{Flat} . Since there are finitely many such words, we have a finite number of possible choices $\tau: X \rightarrow \Sigma^*$ of values these variables can attain. We enumerate all possible valuations τ , and for every such a valuation τ we produce a new constraint φ_τ in which we replace every short variable $x \in X$ by the word $\tau(x)$. The regular constraints restricting the remaining flat variables are modified to permit only words longer than λ_{Flat} , allowing us to assume in our proofs that flat variables in φ_τ have assigned sufficiently long words.

Let us formalize our observation that the precise length of words assigned to flat variables does not matter as long as they are sufficiently long. Let $\eta: \text{Vars}(\mathcal{N}) \rightarrow \Sigma^*$ be a partial assignment. We define the set $X_{<\lambda}$ as $X_{<\lambda}(\eta) \triangleq \{x \in \text{Vars}(\mathcal{N}) : |\eta(x)| < \lambda\}$. Given a constant $\lambda \in \mathbb{N}$, we say that two partial assignments η and ϑ are λ -equivalent denoted by $\eta \sim_\lambda \vartheta$ iff $\eta \sim_\lambda \vartheta \stackrel{\text{def}}{\iff} \eta|_{X_{<\lambda}(\eta)} = \vartheta|_{X_{<\lambda}(\vartheta)}$.

Clearly, $\sim_\lambda$ has a finite index and if there exists a model σ of φ , then its restriction $\sigma|_{\text{Vars}(\mathcal{N})}$ will fall into one of the equivalence classes induced by $\sim_\lambda$. Setting $\lambda = \lambda_{\text{Flat}}$, we inspect all equivalence classes, checking whether any of them contains a model. Given a representative η of an equivalence class, we replace all variables $x \in \text{Vars}(\mathcal{N})$ with $\eta(x)$ if $|\eta(x)| < \lambda_{\text{Flat}}$, producing a new constraint φ_η . Furthermore, we need to include the fact that the remaining variables in $\mathcal{N}$ have assigned long words. Therefore, the languages of all variables $y \in \text{Vars}(\mathcal{N})$ such that $|\eta(y)| \geq \lambda_{\text{Flat}}$ will have their languages restricted to a new language $\mathcal{L}'_y = \mathcal{L}_y \cap \{|w| \geq \lambda_{\text{Flat}} \mid w \in \Sigma^*\}$ in φ_η . The resulting constraint φ_η is clearly equisatisfiable to φ with models restricted to be $\sim_{\lambda_{\text{Flat}}}$ -equivalent to η .

Some of these instances can be decided without any additional work. In particular, if we have an assignment η such that $X_{<\lambda_{\text{Flat}}}(\eta) = \text{Vars}(\mathcal{N})$, i.e., all variables occurring in $\mathcal{N}$ are short, we fix values of all variables in $\mathcal{N}$, and, thus, the Needle of φ_η is a word. The remaining instances with $X_{<\lambda_{\text{Flat}}}(\tau) \subset \text{Vars}(\mathcal{N})$ contain at least one occurrence of a (long) variable in $\tau(\mathcal{N})$, and, thus, their decidability requires investigation.

6.2 Inspecting the Structure of Non-flat Variables in the Presence of Long Flat Variables

Throughout this section, we fix φ be a $\neg\text{Contains}$ instance resulting from the previous section, i.e., $\varphi \triangleq \varphi_\eta = \neg\text{Contains}(\mathcal{N}, \mathcal{H} \wedge \Phi_{\mathcal{L}, \eta}$ for some equivalence class representative η such that $\text{Vars}(\mathcal{N}) \neq \emptyset$. We start by stating the key theorem for our decidability result.

► **Theorem 22.** *Let z be a (decomposed) non-flat variable present in $\mathcal{H}$. There is a flat language $\mathcal{L}_z^{\text{Flat}} \subset \mathcal{L}_z$ s.t. if there exists a model $\sigma \models \varphi$, then there exists a model $\sigma' \models \varphi[z/z\#]$ s.t. $\sigma' \triangleq \sigma \triangleleft \{z \mapsto w_z\}$ for some word $w_z \in \mathcal{L}_z^{\text{Flat}}$.*

Before presenting quite technical lemmas that allowed us to obtain the result, let us derive some intuition on why the theorem holds. Assume that we have a model σ of φ and we pick some long prefix p and a long suffix s for the variable z , and we glue them together using Γ_z -expansion to produce a word $w \triangleq \Gamma_z^{\lambda_\kappa}(p, s)$ and an altered assignment $\sigma' \triangleq \{z \mapsto w\}$. The core of the theorem lies in analyzing the situation when σ' fails to be a model. By symmetry, we focus on the case when our choice of the prefix p is problematic. We have two possibilities.

- There is a short prefix of p due to which σ' fails to be a model. We address this by systematically exploring the prefix tree of z up to a certain bound, marking the vertices that correspond to such prefixes as dead ends.
- Our choice of p does not cause σ' to immediately fail to be a model, however, by applying Γ_z -expansion we introduce an infix due to which $\sigma' \not\models \varphi$. Since we assume that φ results from a previous section, we know that all flat variables have assigned a long word, i.e., $\sigma'(\mathcal{N})$ contains long factors of the form α^k for some α which forms the basis of a flat variable $x \in \text{Vars}(\mathcal{N})$ and $k \in \mathbb{N}$. Γ_z -expansion glues together a prefix and a suffix using a word γ_z^K where $|\gamma_z| \neq |\alpha|$ for any base α . Therefore, we know that only a limited part of the infix introduced by Γ_z -expansion is problematic, otherwise we would have a long overlap between γ_z^K and some factor α^k of $\sigma'(\mathcal{N})$. Thus, p contains a long factor α^k for some $k \geq 1$ and a primitive α word α that forms the base of a flat variable present in $\mathcal{N}$. We carefully analyze the effect of such a factor on the structure of $\mathcal{A}_z$.

We now provide an overview of lemmas that lead to Theorem 22. Since these lemmas are quite technical, we accompany them with intuition and only sketch their proofs. Full proofs can be found in [24]. To simplify the presentation, we focus primarily on attempting to find a suitable prefix of a non-flat variable, and hence, our results are formulated in the context of a modified formula that contains a fresh alphabet symbol $\#$. Since the situation is symmetric for suffixes, we can use the properties of Γ_z -expansion (Lemma 17) and glue together a suitable prefix and a suitable suffix to produce an altered model.

We start with a technical lemma used frequently in our proofs. The lemma shows that if we know that α is a factor of $\mathcal{H}$, and we know that a part of $\mathcal{H}$ in the proximity of the factor α is incompatible with α , then we can show that a large number of overlaps between $\sigma(\mathcal{N})$ and $\sigma(\mathcal{H})$ must contain a conflict if $\mathcal{N}$ contains a large factor of the form α^N .

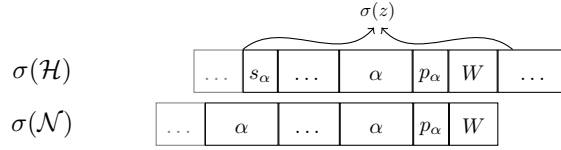
► **Lemma 23.** *Let α and γ be two primitive words, such that $|\alpha| \neq |\gamma|$. Let $\mathcal{H} = t\alpha u\gamma^{\lambda_\kappa}$ and $\mathcal{N} = v\alpha^N w$ where t, u, v and w are (possibly empty) words such that $N|\alpha| > \lambda_{\text{Flat}}$ and $u < \lambda_{\text{Flat}} - 2 \max(|\alpha|, |\gamma|)$. If 1. the prefix p of $\mathcal{H}$ of the size $|p| = |t| + |\alpha| + |w|$ does not contain the word $\mathcal{N}$, 2. $\alpha \notin \text{Pref}(v_0)$ and $v_0 \notin \text{Pref}(\alpha)$, then $\mathcal{N}$ is not a factor of $\mathcal{H}$.*

Proof sketch. Since $\mathcal{N}$ contains a long factor α^N and $\mathcal{H}$ contains at least one α , we can apply Lemma 8 to rule out a lot of overlaps that might be conflict-free. All of the remaining overlaps contain a conflict thanks to Condition 2. The full proof is available in [24]. ◀

► **Lemma 24.** *Let $x \in \mathbb{X}$ be a flat variable with $\text{Base}(x) = \{\alpha\}$, and let $z \in \text{Vars}(\mathcal{H})$ be a (decomposed) non-flat variable. Let $\varphi \triangleq \neg \text{Contains}(\mathcal{N}, \mathcal{H}) = \neg \text{Contains}(\mathcal{N}'x\alpha^M pW, \mathcal{H})$ be formula with $p \neq \alpha$ being a prefix of α and $W = a_0 \dots a_n$ being a non-empty word such that the word pa_0 is not a prefix of α .*

If there exists a model σ with $\sigma(z)$ being of the form $\sigma(z) = s\alpha^k pWV$ for some word V , $k \geq 1$, and a suffix s of α , then $\sigma \triangleleft \{z \mapsto \Gamma_{\text{Pref}(z)}^{\lambda_\kappa}(s\alpha^k pW)\}$ is a model of $\varphi[z/z\#]$.

Intuitively, the rightmost variable in $\mathcal{N}$ is the flat variable x with $\text{Base}(x) = \{\alpha\}$. To the right of x , there is a literal with the prefix $\alpha^M p$ that resembles the flat language $\mathcal{L}_x$. Moreover, $\sigma(z)$ also starts with a prefix $s\alpha^k p$ resembling $\mathcal{L}_x$, followed by W . Thus, the prefix $s\alpha^k pW$ of the word $\sigma(z)$ mimics the suffix of the right-hand side $\sigma(\mathcal{N})$. Hence, if we look solely on the prefix of $\sigma(z)$ and the suffix of $\sigma(\mathcal{N})$, there are no obvious conflicts.



However, $\sigma \models \varphi$, and, therefore, there must be a conflict outside of z when considering the above alignment. The rest of the proof can be found in [24].

Next, we derive a lemma formalizing that we can restrict languages of non-flat variables to flat ones, producing an equisatisfiable instance. Stating a symmetric lemma for suffixes, and applying Lemma 17 would give us the entire proof of Theorem 22.

► **Lemma 25.** *Let x be the rightmost flat variable with $\text{Base}(x) = \{\alpha\}$, and let z be a non-flat variable occurring in $\mathcal{H}$. Let φ be a formula $\varphi \triangleq \neg \text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}} = \neg \text{Contains}(\mathcal{N}'x\alpha^M pW, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ where $\mathcal{N}' \in (\mathbb{X} \cup \Sigma)^*$, $M \geq 0$, $p \neq \alpha$ is a prefix of α , and $W = a_0 \dots a_n$ is a non-empty word such that pa_0 is not a prefix of α . There exists a flat language $\mathcal{L}_z^{\text{Flat}} \subset \mathcal{L}_z$ s.t. if there is a model $\sigma \models \varphi$, then $\sigma \triangleleft \{z \mapsto w_z\} \models \varphi[z/z\#]$ for some word $w_z \in \mathcal{L}_z^{\text{Flat}}$.*

The intuition behind Lemma 25 is the same as the intuition behind Theorem 22. We note that the lemma requires the word W to be non-empty. The case for when $W = \varepsilon$ has a similar, but simpler proof.

Proof sketch. We assume the existence of a model $\sigma \models \varphi$, and we systematically explore the prefix tree T_z of the variable z in a breadth-first fashion up to the bound λ_Q , searching for the word w_z . When inspecting any prefix u , we check whether $\sigma \triangleleft \{z \mapsto u\}$ is a model of $\varphi[z/z\#]$, and if not we mark the last vertex corresponding to u as a dead end and we do not explore it further. At the end of the exploration, we inspect the set $\mathcal{P}_{\geq \lambda_Q}$ of all λ_Q -reaching paths in T_z . If there are no such paths, we know that $|\sigma(z)| < \lambda_Q$, and hence, w_z can be found in the

finite (flat) language $\{w \in \mathcal{L}_z \mid |w| < \lambda_Q\}$. Alternatively, we check for every path π in $\mathcal{P}_{\geq \lambda_Q}$ whether $\sigma \triangleleft \{z \mapsto \Gamma_{\text{Pref}(z)}^{\lambda_\kappa}(u_\pi)\} \models \varphi[z/z\#]$ where u_π is the prefix corresponding to π . Since, the number of possible λ_Q -reaching paths is finite, we have that w_z can be found in the flat language $\{\Gamma_{\text{Pref}(z)}^{\lambda_\kappa}(p_\pi) \mid \pi \in \mathcal{P}_{\geq \lambda_Q}\}$ in the case that $\Gamma_{\text{Pref}(z)}$ -expansion of some u_π leads to a model of $\varphi[z/z\#]$.

Next, it might be that none of the paths in $\mathcal{P}_{\geq \lambda_Q}$ can be $\Gamma_{\text{Pref}(z)}$ -expanded into a model. Let x be the rightmost variable in $\mathcal{N}$ with $\text{Base}(x) = \{\alpha\}$. Recall that none of the paths in $\mathcal{P}_{\geq \lambda_Q}$ contain a dead-end, and that all variables in $\mathcal{N}$ are flat, and, thus, they have assigned long words. Combined with the properties of $\Gamma_{\text{Pref}(z)}$ -expansion, we know the reason why $\sigma' \triangleq \sigma \triangleleft \{z \mapsto \Gamma_{\text{Pref}(z)}^{\lambda_\kappa}(p_\pi)\}$ fails to be a model, i.e., we almost accurately know the position of $\sigma'(\mathcal{N})$ in $\sigma'(\mathcal{H})$. We show that all paths in $\mathcal{P}_{\geq \lambda_Q}$ share the same prefix of the form $s\alpha^M p$ for some large $k \in \mathbb{N}$ and $s \in \text{Suf}(\alpha)$ and $p \in \text{Pref}(\alpha)$. Since z is non-flat, and λ_Q is larger than the number of states of $\mathcal{A}_z$, we have opportunities to diverge from the shared prefix $s\alpha^M p$ in T_z . We show that diverging must immediately lead to a dead-end vertex, and in such a case $\sigma(z)$ has a prefix $s\alpha^M pW$. Hence, we apply Lemma 23 and obtain that $\sigma \triangleleft \{z \mapsto w_{z,M}\}$ is a model of $\varphi[z/z\#]$ where $w_{z,M} = \Gamma_{\text{Pref}(z)}^{\lambda_\kappa}(s\alpha^M pW)$. Note that $w_{z,M}$ depends on an unknown integer M , however, the language containing all $w_{z,M}$ s for every possible choice of k is flat. Alternatively, not-diverging from the path implies that $\sigma(z) \in s\alpha^* p'$ for some $p' \in \text{Pref}(\alpha)$, which is again a flat language. $\blacktriangleleft$

A careful analysis of the proof of Lemma 25 reveals that the lemma, and, therefore, Theorem 22, is not effective in a sense that one cannot directly construct $\mathcal{L}_z^{\text{Flat}}$. However, we can obtain a decision procedure at the cost of a producing larger flat language. We construct T_z and all paths up to the bound λ_Q without having σ available, losing the ability to mark dead-end vertices. In the resulting flat language $\mathcal{L}_z^{\text{Flat}}$ we include all words shorter than λ_Q , and $\Gamma_{\text{Pref}(z)}$ -expansions of all λ_Q -reaching paths. The remaining parts of $\mathcal{L}_z^{\text{Flat}}$ that correspond to the situation when no λ_Q -reaching paths can be $\Gamma_{\text{Pref}(z)}$ -expanded into a model can be computed from $\mathcal{A}_z$ without requiring access to the original model σ . For details, we refer the reader to the full proof of Lemma 25 in [24].

7 Decision Procedure

Finally, we summarize the approach described in previous sections into a decision procedure for $\neg\text{Contains}$. The (nondeterministic) algorithm is shown in Algorithm 1. In the algorithm, for a negated containment φ and a (partial) assignment σ , we use $\sigma(\varphi)$ to denote the $\neg\text{Contains}$ predicate obtained from φ replacing variables whose assignment is defined with the corresponding assignment.

The set $\mathcal{L}_x^{\text{Pref}}(\mathcal{L}_x^{\text{Suf}})$ contains prefixes (suffixes) of words from $\mathcal{L}_x$ that might be used to find an alternative model. The $\text{glue}(\mathcal{L}_x^{\text{Pref}}, \mathcal{L}_x^{\text{Suf}})$ procedure glues together prefixes and suffixes, resulting in a language consisting of entire words (not just prefixes or suffixes) from $\mathcal{L}_x$. The procedure partitions the language $\mathcal{L}_x^{\text{Pref}}$ into $\mathcal{L}_x^{\text{Pref}} = P_w \cup P_{\text{inc}}$ such that P_{inc} consists of words resulting from an application of $\Gamma_{\text{Pref}(z)}$ -expansion. Intuitively, the words in P_w are words from $\mathcal{L}_x$ whereas P_{inc} are only prefixes that need to be completed into full words from $\mathcal{L}_x$ by concatenating suitable suffixes. We decompose $\mathcal{L}_x^{\text{Suf}}$ in the same way into $\mathcal{L}_x^{\text{Suf}} = S_w \cup S_{\text{inc}}$. The procedure then returns $\mathcal{L}'_x = P_w \cup S_w \cup \{p\gamma^{\lambda_\kappa} s \mid (p\gamma^{\lambda_\kappa}, \gamma^{\lambda_\kappa} s) \in P_{\text{inc}} \times S_{\text{inc}}\}$.

► **Theorem 26 (Soundness).** *If Algorithm 1 terminates with an assignment σ , then $\sigma \models \varphi$.*

Proof. Follows from the fact that $\mathcal{L}'_x \subseteq \mathcal{L}_x$ for every non-flat variable x found in $\mathcal{H}'$. $\blacktriangleleft$

Algorithm 1 Decision Procedure for $\neg\text{Contains}$.

Input: $\varphi \triangleq \neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$
Output: Satisfiability of φ

- 1 Normalize φ into $\bigvee_i \varphi_i$ and **Guess** φ_i ; ▷ Section 2.1
- 2 **if** φ_i *is easy* **then return** $\text{solve}(\varphi_i)$; ▷ Section 2.3
- 3 Remove all two-sided $x \in \mathbb{X} \setminus \mathbb{X}_{\text{Flat}}$ from φ_i ; ▷ Theorem 11
- 4 **if** φ_i *is easy* **then return** $\text{solve}(\varphi_i)$; ▷ Section 5
- 5 **Guess** $P \subseteq \mathbb{X}_{\text{Flat}}$; ▷ vars. with assignments longer than λ_{Flat}
- 6 **Guess** assignment σ s.t. $\forall x \in \mathbb{X}_{\text{Flat}} \setminus P: |\sigma(x)| < \lambda_{\text{Flat}}$;
- 7 Let $\varphi' := \sigma(\varphi_i) = \neg\text{Contains}(\mathcal{N}', \mathcal{H}')$;
- 8 **foreach** $x \in (\mathbb{X} \setminus \mathbb{X}_{\text{Flat}}) \cap \text{Vars}(\mathcal{H}')$ **do**
- 9 Constrain prefix/suffix trees T_x/S_x up to bound λ_Q ;
- 10 Constrain prefix language $\mathcal{L}_x^{\text{Pref}}$; ▷ Lemma 25
- 11 Constrain suffix language $\mathcal{L}_x^{\text{Suf}}$; ▷ symmetric to Lemma 25
- 12 Construct flat $\mathcal{L}'_x := \text{glue}(\mathcal{L}_x^{\text{Pref}}, \mathcal{L}_x^{\text{Suf}})$;
- 13 **end**
- 14 **return** $\text{solve}(\varphi' \wedge \{x \mapsto \mathcal{L}'_x \mid x \in \mathbb{X} \setminus \mathbb{X}_{\text{Flat}}\})$; ▷ Lemma 4

► **Theorem 27 (Completeness).** *If φ is satisfiable, then Algorithm 1 terminates with an assignment σ such that $\sigma \models \varphi$. Otherwise Algorithm 1 terminates with the answer UNSAT.*

Proof. Correctness for two-sided non-flat variables follows from Theorem 11. For remaining non-flat variables, correctness follows from Lemma 25. Finally, correctness of the glue procedure follows from Lemma 17. ◀

► **Theorem 28.** *A constraint $\neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ is decidable in EXPSpace.*

Proof sketch. Decidability follows from the analysis of the decision procedure in Algorithm 1 given above. As for EXPSpace membership, first notice that languages of non-flat variables occurring only in $\mathcal{H}$ are replaced with flat languages of polynomial size due to the bounds λ_Q and λ_{κ} . Algorithm 1 then uses Lemma 4, bringing the complexity of the procedure to NEXPTIME. However, obtaining a full model that includes all two-sided non-flat variables brings the procedure to EXPSpace as the length of the word to assigned to a two-sided non-flat variable doubles with each such a variable (cf. Theorem 11). ◀

7.1 Chain-Free Word Equations with $\neg\text{Contains}$

After establishing the decidability of a single $\neg\text{Contains}$ predicate, we immediately obtain decidability of string fragments that permit the so-called *monadic decomposition* [46, 16], i.e., expressing the set of solutions as a finite disjunction of regular membership constraints $\bigwedge_{x \in \mathbb{X}} x \in \mathcal{L}_x$. These include fragments such as the *straight-line* fragment [29] or the more expressive *chain-free* fragment of word equations [4] (note that [4] considers also other predicates). We can therefore easily establish the following theorem.

► **Theorem 29.** *Formula $W \wedge \neg\text{Contains}(\mathcal{N}, \mathcal{H}) \wedge \Phi_{\mathcal{L}}$ where W is a conjunction of chain-free word equations is decidable.*

8 Future Work

This paper shows that chain-free word equations with regular constraints and a single instance of the $\neg\text{Contains}$ predicate are decidable. There are several possible future work directions. First, we wish to investigate the fragment where the number of $\neg\text{Contains}$ constraints is not

limited to a single one. Another direction is examining combinations of $\neg$ Contains with other predicates, such as length constraints or disequalities. The technique for combining these from [19] based on the reduction of the constraints to reasoning over Parikh images of finite automata is not directly applicable here. Also, the resulting complexity of our procedure is EXPSPACE, however, we have hints that the problem might in fact be solvable in NP.

References

- 1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukáš Holík, Ahmed Rezine, and Philipp Rümmer. Flatten and conquer: A framework for efficient analysis of string constraints. In Albert Cohen and Martin T. Vechev, editors, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 602–617. ACM, 2017. doi:10.1145/3062341.3062384.
- 2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukáš Holík, Ahmed Rezine, and Philipp Rümmer. Trau: SMT solver for string constraints. In Nikolaj S. Bjørner and Arie Gurfinkel, editors, *2018 Formal Methods in Computer Aided Design, FMCAD 2018, Austin, TX, USA, October 30 - November 2, 2018*, pages 1–5. IEEE, 2018. doi:10.23919/FMCAD.2018.8602997.
- 3 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukáš Holík, Denghang Hu, Wei-Lun Tsai, Zhillin Wu, and Di-De Yen. Solving not-substring constraint with flat abstraction. In *Programming Languages and Systems*, pages 305–320, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-89051-3_17.
- 4 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bui Phi Diep, Lukáš Holík, and Petr Janků. Chain-free string constraints. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 277–293. Springer, 2019. doi:10.1007/978-3-030-31784-3_16.
- 5 C. Aiswarya, Soumodev Mal, and Prakash Saivasan. On the satisfiability of context-free string constraints with subword-ordering. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '22, New York, NY, USA, 2022*. Association for Computing Machinery. doi:10.1145/3531130.3533329.
- 6 Rajeev Alur and Pavol Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. *SIGPLAN Not.*, 46(1):599–610, January 2011. doi:10.1145/1925844.1926454.
- 7 Clark W. Barrett, Cesare Tinelli, Morgan Deters, Tianyi Liang, Andrew Reynolds, and Nestan Tsiskaridze. Efficient solving of string constraints for security analysis. In William L. Scherlis and David Brumley, editors, *Proceedings of the Symposium and Bootcamp on the Science of Security, Pittsburgh, PA, USA, April 19-21, 2016*, pages 4–6. ACM, 2016. doi:10.1145/2898375.2898393.
- 8 Murphy Berzish, Joel D. Day, Vijay Ganesh, Mitja Kulczynski, Florin Manea, Federico Mora, and Dirk Nowotka. Towards more efficient methods for solving regular-expression heavy string constraints. *Theor. Comput. Sci.*, 943:50–72, 2023. doi:10.1016/j.tcs.2022.12.009.
- 9 Murphy Berzish, Mitja Kulczynski, Federico Mora, Florin Manea, Joel D. Day, Dirk Nowotka, and Vijay Ganesh. An SMT solver for regular expressions and linear arithmetic over string length. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II*, volume 12760 of *Lecture Notes in Computer Science*, pages 289–312. Springer, 2021. doi:10.1007/978-3-030-81688-9_14.
- 10 Berzish, Murphy. *Z3str4: A solver for theories over strings*. PhD thesis, University of Waterloo, 2021. URL: <http://hdl.handle.net/10012/17102>.

- 11 Nikolaj S. Bjørner, Nikolai Tillmann, and Andrei Voronkov. Path feasibility analysis for string-manipulating programs. In Stefan Kowalewski and Anna Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5505 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 2009. doi:10.1007/978-3-642-00768-2_27.
- 12 František Blahoudek, Yu-Fang Chen, David Chocholatý, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Juraj Síč. Word equations in synergy with regular constraints. In Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker, editors, *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings*, volume 14000 of *Lecture Notes in Computer Science*, pages 403–423. Springer, 2023. doi:10.1007/978-3-031-27481-7_23.
- 13 Taolue Chen, Yan Chen, Matthew Hague, Anthony W. Lin, and Zhilin Wu. What is decidable about string constraints with the replaceall function. *Proc. ACM Program. Lang.*, 2(POPL):3:1–3:29, 2018. doi:10.1145/3158091.
- 14 Taolue Chen, Alejandro Flores-Lamas, Matthew Hague, Zhilei Han, Denghang Hu, Shuanglong Kan, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Solving string constraints with regex-dependent functions through transducers with priorities and variables. *Proc. ACM Program. Lang.*, 6(POPL):1–31, 2022. doi:10.1145/3498707.
- 15 Taolue Chen, Matthew Hague, Jinlong He, Denghang Hu, Anthony Widjaja Lin, Philipp Rümmer, and Zhilin Wu. A decision procedure for path feasibility of string manipulating programs with integer data type. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 325–342. Springer, 2020. doi:10.1007/978-3-030-59152-6_18.
- 16 Taolue Chen, Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Decision procedures for path feasibility of string-manipulating programs with complex operations. *Proc. ACM Program. Lang.*, 3(POPL):49:1–49:30, 2019. doi:10.1145/3290362.
- 17 Yu-Fang Chen, David Chocholatý, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Juraj Síč. Solving string constraints with lengths by stabilization. *Proceedings of the ACM on Programming Languages*, 7(OOPSLA2):2112–2141, 2023. doi:10.1145/3622872.
- 18 Yu-Fang Chen, David Chocholatý, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Juraj Síč. Z3-Noodler: An automata-based string solver. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I*, volume 14570 of *Lecture Notes in Computer Science*, pages 24–33. Springer, 2024. doi:10.1007/978-3-031-57246-3_2.
- 19 Yu-Fang Chen, Vojtěch Havlena, Michal Hečko, Lukáš Holík, and Ondřej Lengál. A uniform framework for handling position constraints in string solving. *Proc. ACM Program. Lang.*, 9(PLDI), 2025. doi:10.1145/3729273.
- 20 Joel D. Day, Thorsten Ehlers, Mitja Kulczynski, Florin Manea, Dirk Nowotka, and Danny Bøgsted Poulsen. On solving word equations using SAT. In *RP'19*, volume 11674 of *LNCS*, pages 93–106. Springer, 2019. doi:10.1007/978-3-030-30806-3_8.
- 21 Joel D. Day, Vijay Ganesh, Nathan Grewal, and Florin Manea. On the expressive power of string constraints. *Proc. ACM Program. Lang.*, 7(POPL), January 2023. doi:10.1145/3571203.
- 22 Joel D. Day, Vijay Ganesh, Paul He, Florin Manea, and Dirk Nowotka. The satisfiability of extended word equations: The boundary between decidability and undecidability. *CoRR*, abs/1802.00523, 2018. arXiv:1802.00523.
- 23 Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction*

- and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29–April 6, 2008. *Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. doi:10.1007/978-3-540-78800-3_24.
- 24 Vojtěch Havlena, Michal Hečko, Lukáš Holík, and Ondřej Lengál. Negated string containment is decidable (technical report). *CoRR*, abs/2506.22061, 2025. arXiv:2506.22061.
 - 25 Juhani Karhumäki, Filippo Mignosi, and Wojciech Plandowski. The expressibility of languages and relations by word equations. *J. ACM*, 47(3):483–505, 2000. doi:10.1145/337244.337255.
 - 26 Tianyi Liang, Andrew Reynolds, Cesare Tinelli, Clark W. Barrett, and Morgan Deters. A DPLL(T) theory solver for a theory of strings and regular expressions. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18–22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 646–662. Springer, 2014. doi:10.1007/978-3-319-08867-9_43.
 - 27 Tianyi Liang, Andrew Reynolds, Nestan Tsiskaridze, Cesare Tinelli, Clark W. Barrett, and Morgan Deters. An efficient SMT solver for string constraints. *Formal Methods Syst. Des.*, 48(3):206–234, 2016. doi:10.1007/s10703-016-0247-6.
 - 28 Tianyi Liang, Nestan Tsiskaridze, Andrew Reynolds, Cesare Tinelli, and Clark W. Barrett. A decision procedure for regular membership and length constraints over unbounded strings. In Carsten Lutz and Silvio Ranise, editors, *Frontiers of Combining Systems - 10th International Symposium, FroCoS 2015, Wrocław, Poland, September 21–24, 2015. Proceedings*, volume 9322 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2015. doi:10.1007/978-3-319-24246-0_9.
 - 29 Anthony Widjaja Lin and Pablo Barceló. String solving with word equations and transducers: Towards a logic for analysing mutation XSS. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 123–136. ACM, 2016. doi:10.1145/2837614.2837641.
 - 30 M. Lothaire, editor. *Combinatorics on Words*. Cambridge Mathematical Library. Cambridge University Press, 2 edition, 1997.
 - 31 M. Lothaire. *Algebraic Combinatorics on Words*. Cambridge University Press, 2002.
 - 32 Kevin Lotz, Amit Goel, Bruno Dutertre, Benjamin Kiesl-Reiter, Soonho Kong, Rupak Majumdar, and Dirk Nowotka. Solving string constraints using sat. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification*, pages 187–208, Cham, 2023. Springer Nature Switzerland. doi:10.1007/978-3-031-37703-7_9.
 - 33 Zhengyang Lu, Stefan Siemer, Piyush Jha, Joel D. Day, Florin Manea, and Vijay Ganesh. Layered and staged Monte Carlo tree search for SMT strategy synthesis. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3–9, 2024*, pages 1907–1915. ijcai.org, 2024. URL: <https://www.ijcai.org/proceedings/2024/211>.
 - 34 R. C. Lyndon and M. P. Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Mathematical Journal*, 9(4):289–298, 1962. doi:10.1307/mmj/1028998766.
 - 35 G S Makanin. The problem of solvability of equations in a free semigroup. *Mathematics of the USSR-Sbornik*, 32(2):129, February 1977. doi:10.1070/SM1977v032n02ABEH002376.
 - 36 Jakob Nielsen. Die isomorphismen der allgemeinen, unendlichen gruppe mit zwei erzeugenden. *Mathematische Annalen*, 78(1):385–397, 1917.
 - 37 Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Syntax-guided quantifier instantiation. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021*,

- Proceedings, Part II*, volume 12652 of *Lecture Notes in Computer Science*, pages 145–163. Springer, 2021. doi:10.1007/978-3-030-72013-1_8.
- 38 Andres Nötzli, Andrew Reynolds, Haniel Barbosa, Clark W. Barrett, and Cesare Tinelli. Even faster conflicts and lazier reductions for string solvers. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part II*, volume 13372 of *Lecture Notes in Computer Science*, pages 205–226. Springer, 2022. doi:10.1007/978-3-031-13188-2_11.
 - 39 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 495–500. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814622.
 - 40 Andrew Reynolds, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. High-level abstractions for simplifying extended string constraints in SMT. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*, volume 11562 of *Lecture Notes in Computer Science*, pages 23–42. Springer, 2019. doi:10.1007/978-3-030-25543-5_2.
 - 41 Andrew Reynolds, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. Reductions for strings and regular expressions revisited. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 225–235. IEEE, 2020. doi:10.34727/2020/isbn.978-3-85448-042-6_30.
 - 42 Andrew Reynolds, Maverick Woo, Clark W. Barrett, David Brumley, Tianyi Liang, and Cesare Tinelli. Scaling up DPLL(T) string solvers using context-dependent simplification. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2017. doi:10.1007/978-3-319-63390-9_24.
 - 43 Neha Rungta. A billion SMT queries a day (invited paper). In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I*, volume 13371 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2022. doi:10.1007/978-3-031-13185-1_1.
 - 44 Prateek Saxena, Devdatta Akhawe, Steve Hanna, Feng Mao, Stephen McCamant, and Dawn Song. A symbolic execution framework for JavaScript. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berkeley/Oakland, California, USA*, pages 513–528. IEEE Computer Society, 2010. doi:10.1109/SP.2010.38.
 - 45 Trauc string constraints benchmark collection, 2020. URL: https://github.com/plfm-iis/trauc_benchmarks.
 - 46 Margus Veanes, Nikolaj S. Bjørner, Lev Nachmanson, and Sergey Bereg. Monadic decomposition. *J. ACM*, 64(2):14:1–14:28, 2017. doi:10.1145/3040488.