

Context-Free Subphrase Grammars

A Grammar Formalism for Embedded Domain-Specific Languages

Björn Lötters¹[0000-0003-1403-2129]

Technische Hochschule Mittelhessen (University of Applied Sciences),
Wiesenstr. 14, 35390 Giessen, Hesse, Germany

`bjoern.loetters@thm.de`

[Student Research Paper \(Full Paper\)](#)

Abstract. *Embedded Domain-Specific Languages* (EDSLs) have gained significant popularity as a safe and interoperable tool for abstraction in functional programming languages. Despite numerous approaches to their development, none of these seem to fully exploit the capabilities offered by *Context-Free Grammars* (CFGs) in an equally declarative way. We identify the lack of modularity inherent to phrase-structure grammars such as CFGs as the main reason for this. In this paper, we introduce *Context-Free Subphrase Grammars* (CFSGs), a formalism based on an alternative interpretation of operator precedence that applies naturally to arbitrary production rules. The result is a lightweight and modular grammar formalism that adheres to the pure phrase-oriented emphasis of CFGs and aligns well with the composable and modular design of functional programming languages. We provide evidence for the completeness and soundness of CFSGs with respect to CFGs, affirming their ability to describe arbitrary context-free syntax for EDSLs.

Keywords: Embedded Domain-Specific Languages · Modular Grammar Formalism · Context-Free Grammars · Operator Precedence

1 Introduction

Amidst the ever-growing number of software applications, *Domain-Specific Languages* (DSLs) become increasingly important. In the words of Hudak “a domain-specific language is the ultimate abstraction” [13]. It allows us to express solutions in a natural way, using the abstractions of the problem domain rather than the implementation domain. Especially DSLs embedded into more general host languages like Haskell [17] can take advantage of this. These so-called *Embedded Domain-Specific Languages* (EDSLs) inherit the features of the host language [12], providing a safe and interoperable way to hide implementation details.

To make this embedding work on a syntactic level, a wide spectrum of techniques has already been proposed, ranging from purely declarative to more procedural approaches. Haskell, for example, allows us to declare custom *infix* operators, which is extensively used by library authors to model their problem

domains¹. Agda [19] and other theorem provers go a step further and support the declaration of custom *mixfix* or *distfix* operators. In contrast to this approach, languages such as Racket [10] and Lean [18] impressively show how we can embed DSLs using parser extensions and syntax transformers [6] [24].

Yet, despite of this wide spectrum, what all these approaches seem to have in common is a trade-off between their *declarativity* and *expressiveness*. While we can model EDSLs in Haskell and Agda in a purely declarative way, the underlying grammar formalism is only able to describe a proper subset of *Context-Free Languages* (CFLs) [1] [2]. On the other hand, meta-programming techniques that enable parser extensions and syntax transformations offer great power, but impose constraints on the spatial or temporal arrangement of the program², which effectively impacts the degree of declarativity.

In this paper we propose *Context-Free Subphrase Grammars* (CFSGs): a novel grammar formalism that further relaxes this tension. Just like Chomsky’s *Context-Free Grammars* (CFGs) [5], our formalism is declarative yet expressive enough to generate the full class of CFLs. But in contrast to them, without adopting their lack of *modularity* – a drawback that is primarily discussed in the field of grammar engineering and computational linguistics [26] [14]. However, unlike prior solutions to this problem, our approach is based on a new rule scheme we call *subphrase rules*. In this way, our grammar formalism adheres to the pure phrase-oriented emphasis of CFGs and aligns well with the modular design of functional programming languages. This makes it a suitable foundation for a lightweight syntax definition formalism that allows the rapid development of EDSLs in a step-by-step fashion. In summary, we therefore contribute:

- An alternative, language-oriented interpretation of operator precedence using the notion of *subphrases* in section 2.
- A proper formalization of *Context-Free Subphrase Grammars* (CFSGs) in section 3.
- Evidence about the meta-theoretic properties of CFSGs in section 4 and in particular their completeness and soundness w.r.t. CFGs

As there is a large amount of different approaches to composable and modular syntax definitions, we will highlight and compare the most relevant work to ours in section 5. We finally complete our paper with a brief excerpt of future work in section 6 and a conclusion in section 7.

2 The Notion of Subphrases

As Kats et al. put it, “Grammars as proposed by Chomsky provide an intuitive, natural means to describe languages” [15]. So why not use *Context-Free*

¹ An excerpt of Haskell libraries that use EDSLs can be found in the official wiki: http://wiki.haskell.org/Embedded_domain_specific_language

² Racket, for example, enforces a separation of macro expansion and execution time [6], while Lean relies on the order of its commands such that language extensions can only affect the rest of the program [24].

Grammars (CFGs) as an embedded tool in programming languages to describe EDSLs? While this might sound like a promising idea at first, CFGs quickly turn out to lack essential properties we know and love in software engineering. Among them are composability, reusability, scalability, and modularity: properties that are well-supported by functional programming languages. Surprisingly, Aasa’s precedence grammars [1], despite being a proper subclass of CFGs, seem to be better suited for this purpose, since they are widely used as the foundation for mixfix operators in dependently typed languages such as Agda [19], Coq [22], Lean [18] and Idris [4]. But how is that and how can we scale the idea of precedence grammars to arbitrary CFGs? In this section, we try to answer this question in that we develop a new rule scheme called *subphrase rules* that eventually assigns operator precedence a proper semantics in CFGs.

2.1 The Lack of Modularity

To begin with, let us first observe why CFGs lack modularity. For this, it is helpful to compare the two grammars in figure 1 and 2. The former one uses the well-known operator-precedence climbing method to unambiguously describe only those expressions that respect the precedence between $+$ and $*$. The latter one sacrifices this unambiguity in favor of a simpler grammar that is easier to understand for less experienced readers.

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid x \end{aligned}$$

Fig. 1: An unambiguous CFG whose productions are tightly coupled

$$\begin{aligned} E &\rightarrow E + E \\ E &\rightarrow E * E \\ E &\rightarrow (E) \mid x \end{aligned}$$

Fig. 2: An ambiguous CFG whose productions are loosely coupled

Now, imagine adding a new right-associative operator $\hat{}$ for the exponentiation to both grammars. Following the scheme of the operator-precedence climbing method, we must first add the two production rules $P \rightarrow F \hat{} P \mid F$ to the former grammar. Since P is not yet reachable from the start non-terminal symbol E , we must incorporate it at the correct position in the precedence hierarchy. When exponentiation should bind more tightly than multiplication, this means that we must replace any occurrence of F in the production rules of T with P . Clearly, such an operation contradicts the idea of a reusable, scalable, composable and modular design in software engineering. This is due to the cyclic dependency between the different phrases of the language – the archenemy of a modularity.

In contrast to this, we can easily extend the latter grammar by adding a production rule $E \rightarrow E \hat{} E$. While this is as elegant as defining functions in a functional language, we have given up on unambiguity here. For example, we neither considered the associativity nor the precedence of $\hat{}$, which may lead to wrong interpretations. Nevertheless, we also gain something from this: As there is only one non-terminal symbol in the whole grammar, it only indicates the position where other phrases may occur and might as well be seen as a wildcard for

any phrase. Thanks to this strong generalization, the cyclic dependency between the phrases is resolved and the archenemy of modularity is defeated.

Not least because of this, Aasa’s precedence grammars are well-suited for the modular design of functional programming languages. They equip an ambiguous CFGs such as the one in figure 2 with a precedence relation. In this way, the ambiguities are resolved by filtering out unwanted parse trees. Yet, they also come with another drawback: They restrict CFGs to a proper subclass.

2.2 The Notion of Subphrases

The aforementioned drawbacks with precedence grammars and CFGs eventually leads us the central question of our research: Can we somehow apply the notion of precedence to arbitrary CFGs to make them as modular as precedence grammars?

As soon as we play with this idea, we must quickly realize: The semantics of precedence does not scale naturally to arbitrary production rules. This circumstance was already acknowledged by Aasa insofar “there is no adequate definition of what it means for a production in a grammar to have higher precedence than another production” [1]. Aasa later defines precedence for mixfix operators (originally called distfix operators) in terms of those *parse trees* that respect the intended precedence. A more general approach is taken by the *Syntax Definition Formalism* (SDF) [11] [21], a formalism for the development of context-free grammars to which we will come back in section 5. It offers a notion of *priority* between production rules. The semantics of this priority is then given in terms of *tree filters*.

But grammars do not *only* describe trees. Grammars such as CFGs describe languages by means of phrases. Our goal is therefore to find an adequate semantics for precedence on the level of these phrases instead of trees. As a starting point, let us consider the language $\mathcal{L}(E)$ in figure 3.

$$\begin{aligned}\mathcal{L}(E) &= \{ a + b \mid a \in \mathcal{L}(E), b \in \mathcal{L}(T) \} \cup \mathcal{L}(T) \\ \mathcal{L}(T) &= \{ a * b \mid a \in \mathcal{L}(T), b \in \mathcal{L}(F) \} \cup \mathcal{L}(F) \\ \mathcal{L}(F) &= \{ (a) \mid a \in \mathcal{L}(E) \} \cup \{ x \}\end{aligned}$$

Fig. 3: The language $\mathcal{L}(E)$ described by the grammar in figure 1

The recursive definition of this language using set theory reveals an essential property of operator precedence: It implies a containment hierarchy between the different phrases of our grammar. In our example this is $\mathcal{L}(F) \subseteq \mathcal{L}(T) \subseteq \mathcal{L}(E)$, which perfectly reflects our original operator precedence again. This observation leads us to a notion of *subphrases*: Instead of defining a separate precedence relation in addition to the actual grammar, we state the inclusive relationship between phrases as part of the grammar. This yields a new rule scheme besides production rules which we call *subphrase rules*. We use the notation $T \sqsubseteq E$ for these rules and say that *T is a subphrase of E*.

An interesting consequence of this idea is that the relation described by these subphrase rules induces a lattice, where the greatest phrase (i.e., the *start sym-*

bol) equals the top \top of the lattice (i.e., E in our case). As a natural consequence, we can then introduce a bottom \perp to obtain a notion of an *end symbol* that represents the smallest phrase or, in terms of operator precedence, the “greatest precedence level” (i.e., F in our case). Visualizing this bounded lattice then shows great similarity with the directed precedence graph proposed by Danielsson et al. [7] to parse mixfix operators. To see this, consider the slightly more realistic example in subsection 3.4. Here, we limit ourselves to the grammar in figure 4 that uses subphrase rules to describe the language $\mathcal{L}(E)$.

$$\begin{aligned} E &\rightarrow E + E_{\downarrow} \\ T &\subseteq E \\ T &\rightarrow T * T_{\downarrow} \\ F &\subseteq T \\ F &\rightarrow (\top) \mid x \end{aligned}$$

Fig. 4: The grammar from figure 1 using subphrase rules

When we consider the production rules of this grammar, we can observe that the cyclic dependency between them is resolved. This is due to *non-terminal expressions* such as E_{\downarrow} or \top which enable us to describe (sub-) phrases according to the subphrase rules. In this grammar, \top describes the same language as E and E_{\downarrow} describes the same language as T , which is the greatest subphrase of E . Now, if we wish to add a new syntax for exponentiation to our grammar, we do not have to alter previous rules. It is sufficient to add a production rule $P \rightarrow P_{\downarrow} \wedge P$ as well as the two subphrase rules $P \subseteq T$ and $F \subseteq P$, which puts T , P and F in the intended order. In this way, we achieved our goal of a modular grammar formalism. And even more than that, since we only describe subphrase relations that apply to phrases in general, this approach naturally scales to arbitrary CFGs.

3 Subphrase Grammars

Within this section we properly formalize *Context-Free Subphrase Grammars* (CFSGs). For this purpose, we first define an abstract syntax for our grammar formalism in subsection 3.1. In contrast to CFGs, not every instance of CFSGs is well-formed, which is why we give a proper definition of well-formedness in subsection 3.2. We then define the semantics of our grammar formalism in terms of the derivation relation in subsection 3.3. A full example in subsection 3.4 finally concludes this section.

3.1 Abstract Syntax

The abstract syntax of our grammar formalism is given by the productions in figure 5. It is based on the following three generator sets:

- A finite set of terminal symbols \mathbb{T}
- A finite set of variable symbols \mathbb{V} with $\mathbb{V} \cap \mathbb{T} = \emptyset$

- A finite set of domain symbols \mathbb{D} with $\mathbb{D} \cap \mathbb{T} = \emptyset$

As there may be contexts where we consider multiple grammars at once, we use $G_{\mathbb{T}}$ to disambiguate the set of terminal symbols \mathbb{T} of the grammar G from others (likewise for \mathbb{V} , \mathbb{D} , \mathbb{S} and \mathbb{E}).

t	terminal symbol	\mathbb{T}
A, B, C	variable symbol	\mathbb{V}
$\mathbb{A}, \mathbb{B}, \mathbb{C}$	domain symbol	\mathbb{D}
$X, Y, Z ::=$	non-terminal symbol	\mathbb{S}
$A^{\mathbb{A}}$	variable	
$\top^{\mathbb{A}}$	top	
$\perp^{\mathbb{A}}$	bottom	
$E ::=$	non-terminal expression	\mathbb{E}
X	non-terminal symbol	
X_{\downarrow}	subphrase	
$E_1 \sqcup E_2$	join	
$E_1 \sqcap E_2$	meet	
$S ::=$	symbol expression	
t	terminal	
E	non-terminal	
$R ::=$	rule	
$X \rightarrow S_1 \dots S_n$	production	
$X \sqsubseteq Y$	subphrase	
$G ::=$	grammar	
\emptyset	empty	
$R; G$	rule	

Fig. 5: The abstract syntax of our formalism.

In contrast to CFGs, there is no generator set for non-terminal symbols. This is because non-terminal symbols are represented as elements of the set $\mathbb{S} = (\mathbb{V} \cup \{ \top, \perp \}) \times \mathbb{D}$. The reason for this is an important observation: When all non-terminal symbols of a grammar are related by the subphrase relation, we cannot describe the full class of CFLs. This is because, we cannot describe deeper nested structures without them being a subphrase of the start symbol as well. To solve this restriction, there is a subphrase relation per domain symbol $\mathbb{A} \in \mathbb{D}$. Note, how we do not choose the term *domain* at random. Just like there may be multiple DSLs embedded in a programming language, we allow the definition of multiple domains. A variable symbol may then occur in various domains to identify different non-terminal symbols, leading to an analogy between domains and namespaces. As a consequence, we can identify domains by the modules of a host language in which our formalism is embedded.

Besides this there is another noteworthy difference compared to CFGs. We distinguish between *non-terminal symbols*, which stand for named phrases and may be associated with production rules, and *non-terminal expressions*, which describe “anonymous phrases” and may not be associated with production rules by themselves. As we will later see, non-terminal expressions describe exactly the elements of the lattice that is induced by the subphrase rules.

3.2 Well-Formedness

As we already pointed out before, not any instance of our grammar formalism is well-formed. The reason for this is the introduction of multiple domains. Since every domain is associated with a subphrase relation, we must be careful not to mix them. We achieve this by enforcing that the domain symbols of any two non-terminal symbols must match, when they are used in or put into relation. We formalize this notion of well-formedness using the following judgements:

- We write $\vdash G$ to state that G is a well-formed grammar.
- We write $\vdash X : \mathbb{A}$ to state that $X \in \mathbb{S}$ is a well-formed non-terminal symbol of domain $\mathbb{A} \in \mathbb{D}$.
- We write $\vdash E : \mathbb{A}$ to state that $E \in \mathbb{E}$ is a well-formed non-terminal expression of domain $\mathbb{A} \in \mathbb{D}$.

TERMINAL $\frac{}{\vdash \top : \mathbb{A}}$	TOP $\frac{}{\vdash \top^{\mathbb{A}} : \mathbb{A}}$	BOTTOM $\frac{}{\vdash \perp^{\mathbb{A}} : \mathbb{A}}$	VARIABLE $\frac{}{\vdash A^{\mathbb{A}} : \mathbb{A}}$
SUBPHRASE $\frac{\vdash X : \mathbb{A}}{\vdash X_{\downarrow} : \mathbb{A}}$	JOIN $\frac{\vdash E_1 : \mathbb{A} \quad \vdash E_2 : \mathbb{A}}{\vdash E_1 \sqcup E_2 : \mathbb{A}}$	MEET $\frac{\vdash E_1 : \mathbb{A} \quad \vdash E_2 : \mathbb{A}}{\vdash E_1 \sqcap E_2 : \mathbb{A}}$	
SUBPHRASE RULE $\frac{\vdash X : \mathbb{A} \quad \vdash Y : \mathbb{A} \quad \vdash G}{\vdash X \sqsubseteq Y; G}$	PRODUCTION RULE $\frac{\vdash S_1 : \mathbb{B}_1 \quad \cdots \quad \vdash S_n : \mathbb{B}_n \quad \vdash G}{\vdash X \rightarrow S_1 \dots S_n; G}$		
	EMPTY GRAMMAR $\frac{}{\vdash \emptyset}$		

Fig. 6: Well-formedness for our Formalism

3.3 Semantics

In this subsection we define the semantics of CFSGs. That is, we give a proper definition of the language that is described by a grammar. As our notion of subphrases plays a vital role here, we also provide a definition for the actual subphrase relation as well as the lattice induced in this way. However, due to

lack of space, we leave out most of the foundations on lattices and order theory. For details, please refer to standard literature such as [3] or [8].

If not specified otherwise, we assume $\mathbb{A} \in G_{\mathbb{D}}$ to be an arbitrary domain of some well-formed grammar $G = R_1; \dots; R_n; \emptyset$ for the rest of this subsection.

Definition 1 (Domain Partition of Non-Terminal Symbols). *We call a subset $\mathbb{S}_{\mathbb{A}} = \{ X \in \mathbb{S} \mid X = A^{\mathbb{A}} \vee X = \top^{\mathbb{A}} \vee X = \perp^{\mathbb{A}} \}$ of \mathbb{S} the **set of non-terminal symbols of domain \mathbb{A}** .*

Definition 2 (Subphrase Preorder). *Let $\mathcal{S}_{\mathbb{A}}^* : \mathbb{S}_{\mathbb{A}} \times \mathbb{S}_{\mathbb{A}}$ denote the reflexive and transitive closure of the following relation:*

$$\begin{aligned} & \{ (X, Y) \in \mathbb{S}_{\mathbb{A}} \times \mathbb{S}_{\mathbb{A}} \mid \exists i \in \mathbb{N} : 1 \leq i \leq n \wedge R_i = X \sqsubseteq Y \} \\ \cup & \{ (X, \top^{\mathbb{A}}) \mid X \in \mathbb{S}_{\mathbb{A}} \} \quad \cup \quad \{ (\perp^{\mathbb{A}}, X) \mid X \in \mathbb{S}_{\mathbb{A}} \} \end{aligned}$$

Definition 2 is the basis for our actual subphrase relation and captures all subphrase rules of G , supplemented by the top element $\top^{\mathbb{A}}$ and bottom element $\perp^{\mathbb{A}}$. As we can see at this point, even if there is no subphrase rule in G , any non-terminal symbol is related to $\top^{\mathbb{A}}$ and $\perp^{\mathbb{A}}$.

Definition 3 (Equivalence of Non-Terminal Symbols). *Let $\mathcal{S}_{\mathbb{A}}^{\sim} = \{ (X, Y) \mid (X, Y) \in \mathcal{S}_{\mathbb{A}}^* \wedge (Y, X) \in \mathcal{S}_{\mathbb{A}}^* \}$ be an equivalence relation. We denote the quotient set $\mathbb{S}_{\mathbb{A}} / \mathcal{S}_{\mathbb{A}}^{\sim}$ as $\mathbb{S}_{\mathbb{A}}^{\sim}$ and write $[X] = \{ Y \in \mathbb{S}_{\mathbb{A}} \mid (X, Y) \in \mathcal{S}_{\mathbb{A}}^{\sim} \}$ for the equivalence class of the non-terminal symbol $X \in \mathbb{S}_{\mathbb{A}}$.*

Definition 4 (Subphrase Relation). *Let the partial-order $\mathcal{S}_{\mathbb{A}}^{\sqsubseteq} = \{ ([X], [Y]) \mid (X, Y) \in \mathcal{S}_{\mathbb{A}}^* \}$ be the subphrase relation of domain \mathbb{A} .*

With definition 4 we can finally define what we call the *subphrase lattice*. As it constitutes a lower set lattice, we already know that it is a special kind of a completely distributive lattice that is sometimes called stone [25] or superalgebraic lattice [9] in the literature.

Definition 5 (Subphrase Lattice). *Let $\downarrow_{\mathbb{A}} Q = \{ y \in \mathbb{S}_{\mathbb{A}}^{\sim} \mid \exists x \in Q : (y, x) \in \mathcal{S}_{\mathbb{A}}^{\sqsubseteq} \}$ denote the smallest lower set containing Q . Moreover, let $\downarrow_{\mathbb{A}} x = \{ y \in \mathbb{S}_{\mathbb{A}}^{\sim} \mid (y, x) \in \mathcal{S}_{\mathbb{A}}^{\sqsubseteq} \}$ denote the principal ideal with x as its principal element. We call the set of all lower sets ordered by set inclusion $\mathcal{O}(\mathbb{S}_{\mathbb{A}}^{\sim})$ the subphrase lattice.*

Since some of the elements of the subphrase lattice are of special interest to us (e.g., the principal ideals), we use non-terminal expressions to describe them. With the following definition 6 we provide a proper denotation for these non-terminal expressions.

Definition 6 (Denotation of Non-Terminal Expressions). *Let $\llbracket E \rrbracket : \mathcal{O}(\mathbb{S}_{\mathbb{A}}^{\sim})$ be the denotation of $E \in \mathbb{E}$. It is defined inductively over the abstract syntax as:*

$$\begin{aligned} \llbracket X \rrbracket &= \downarrow_{\mathbb{A}} [X] \\ \llbracket X_{\downarrow} \rrbracket &= (\llbracket X \rrbracket \setminus \{ [X] \}) \cup \llbracket \perp^{\mathbb{A}} \rrbracket \\ \llbracket E_1 \sqcup E_2 \rrbracket &= \llbracket E_1 \rrbracket \cup \llbracket E_2 \rrbracket \\ \llbracket E_1 \sqcap E_2 \rrbracket &= \llbracket E_1 \rrbracket \cap \llbracket E_2 \rrbracket \end{aligned}$$

Definition 7 (Covering Relation). Let $x, y \in \mathcal{O}(\mathbb{S}_{\mathbb{A}}^{\sim})$ be two elements of the subphrase lattice. We say x **covers** y and write $x \succ y$ iff $y \subsetneq x$ and $y \subseteq z \subsetneq x \implies y = z$.

What follows is an interesting proposition about the expression X_{\downarrow} . We use this expression to denote the greatest proper subphrase of X in a relative manner (recall our example in subsection 2.2). Because of proposition 1 we know that X_{\downarrow} is always covered by X as long as $X \neq \perp^{\mathbb{A}}$. That is, intuitively, we can use X_{\downarrow} to “climb” down the subphrase lattice until we reach the bottom.

Proposition 1. $\forall X \in \mathbb{S}_{\mathbb{A}} : X \neq \perp^{\mathbb{A}} \implies \llbracket X \rrbracket \succ \llbracket X_{\downarrow} \rrbracket$

Proof. Assume $X \in \mathbb{S}_{\mathbb{A}}$ and $X \neq \perp^{\mathbb{A}}$. We can prove our goal when $\nexists x \in \mathcal{O}(\mathbb{S}_{\mathbb{A}}^{\sim}) : \llbracket X_{\downarrow} \rrbracket \subset x \subset \llbracket X \rrbracket$ holds. Hence, assume there would be such an element $x \in \mathcal{O}(\mathbb{S}_{\mathbb{A}}^{\sim})$. Since $\llbracket X_{\downarrow} \rrbracket$ removes exactly the principal element $\llbracket X \rrbracket$ from the principal ideal $\downarrow_{\mathbb{A}} \llbracket X \rrbracket$ we know that $|\llbracket X_{\downarrow} \rrbracket| = |\llbracket X \rrbracket| - 1$. However, there cannot be such an element x with $\llbracket X_{\downarrow} \rrbracket \subset x \subset \llbracket X \rrbracket$ as this would require $|\llbracket X_{\downarrow} \rrbracket| \leq |\llbracket X \rrbracket| - 2$. \square

At this point we finally define our derivation relation. Since there are two rule schemes in our formalism, we split its definition in two subrelations. Note at this point, how the subrelation for rule application resembles the derivation relation for CFGs.

Definition 8 (Rule Application). Let $\cdot \Rightarrow_R \cdot : (\mathbb{T} \cup \mathbb{E})^* \times (\mathbb{T} \cup \mathbb{E})^*$ denote the relation we obtain by

$$\alpha X \beta \xrightarrow[\exists i \in [1, n] : R_i = X \rightarrow S_1 \dots S_m]{} \alpha S_1 \dots S_m \beta$$

Definition 9 (Subphrase Substitution). Let $\cdot \Rightarrow_S \cdot : (\mathbb{T} \cup \mathbb{E})^* \times (\mathbb{T} \cup \mathbb{E})^*$ denote the relation we obtain by

$$\alpha E_1 \beta \xrightarrow[\llbracket E_2 \rrbracket \subseteq \llbracket E_1 \rrbracket]{} \alpha E_2 \beta$$

Definition 10 (Derivation Relation). Let $\gamma, \delta \in (\mathbb{T} \cup \mathbb{E})^*$ be two strings. We say γ (directly) yields δ or δ is (directly) derived from γ and write $\gamma \Rightarrow \delta$ iff $\gamma \Rightarrow_R \delta \vee \gamma \Rightarrow_S \delta$ holds. We use $\cdot \Rightarrow^* \cdot : (\mathbb{T} \cup \mathbb{E})^* \times (\mathbb{T} \cup \mathbb{E})^*$ to denote the reflexive and transitive closure and call this the derivation relation.

Definition 11 (Domain Language). Let $\gamma \in (\mathbb{T} \cup \mathbb{E})^*$ be a string. We write $\mathcal{L}_G(\gamma) = \{ \omega \in \mathbb{T}^* \mid \gamma \Rightarrow^* \omega \}$ to denote the language described by γ in G . Moreover, we write $\mathcal{L}_{\mathbb{A}}(G) = \mathcal{L}_G(\tau^{\mathbb{A}})$ and call this the domain language of \mathbb{A} .

Now that we know what language is being described by a CFSG, we can also introduce a notion of equivalence for grammars. As our formalism describes not only one but one language per domain, there is no specific start non-terminal symbol. For this reason, we have different notions of equivalence here. Here, definition 12 introduces a notion of equivalence under a specific domain. Depending on the context, this might be too weak. In these cases, we quantify over a set of domains to strengthen the respective statement.

Definition 12 (Equivalence of Grammars). Let G_1 and G_2 be two well-formed grammars and $\mathbb{A} \in G_{1\mathbb{D}} \cap G_{2\mathbb{D}}$ a domain of both grammars. We say G_1 is equivalent to G_2 under domain \mathbb{A} and write $G_1 \equiv_{\mathbb{A}} G_2$ if and only if $\mathcal{L}_{\mathbb{A}}(G_1) = \mathcal{L}_{\mathbb{A}}(G_2)$ holds.

Proposition 2 (Congruence of Non-Terminal Symbols). $\forall X, Y \in \mathbb{S}_{\mathbb{A}} : (X, Y) \in \mathcal{S}_{\mathbb{A}}^{\sim} \implies \mathcal{L}_G(X) = \mathcal{L}_G(Y)$

Proof. This trivially follows from $\llbracket X \rrbracket = \downarrow_{\mathbb{A}} [X] = \downarrow_{\mathbb{A}} [Y] = \llbracket Y \rrbracket$ and $\mathcal{L}_G(Y) \subseteq \mathcal{L}_G(X) \wedge \mathcal{L}_G(X) \subseteq \mathcal{L}_G(Y)$. \square

As we can describe arbitrary elements of the subphrase lattice using non-terminal expressions, we might ask ourselves at this point how these elements relate to ordinary non-terminal symbols and which languages they describe. The following definition gives a satisfying answer to this question.

Definition 13 (Sublanguage Lattice). Let $\mathcal{F}_{\mathbb{A}} : \mathcal{O}(\mathbb{S}_{\mathbb{A}}^{\sim}) \rightarrow \mathbb{L}_{\mathbb{A}}$ be a surjective map with $\mathcal{F}_{\mathbb{A}}(x) = \bigcup_{[X] \in x} \mathcal{L}_G(X)$ and $\mathbb{L}_{\mathbb{A}} = \{ \mathcal{F}_{\mathbb{A}}(x) \mid x \in \mathcal{O}(\mathbb{S}_{\mathbb{A}}^{\sim}) \}$. We call the set $\mathbb{L}_{\mathbb{A}}$ ordered by set inclusion the sublanguage lattice of domain \mathbb{A} .

Due to lack of space, we do not provide full evidence for $\mathbb{L}_{\mathbb{A}}$ being a lattice here. The idea, however, is to show that $\mathcal{F}_{\mathbb{A}}$ is an order- and join-preserving map (cf. [3] and [8]). This leaves $\mathbb{L}_{\mathbb{A}}$ as a finite join-semilattice and hence as a finite lattice. However, while this lattice is interesting on its own, even more interesting is the following lemma 1 about $\mathcal{F}_{\mathbb{A}}$. Because of this, we know that the language of *any* non-terminal expression $E \in \mathbb{E}$ is just the union of $\mathcal{L}(X)$ for all of its subphrases $X \in \mathbb{S}_{\mathbb{A}}$.

Lemma 1 (Congruency of $\mathcal{F}_{\mathbb{A}}$ and \mathcal{L}_G). $\forall E \in \mathbb{E} : \mathcal{F}_{\mathbb{A}}(E) = \mathcal{L}_G(E)$

Proof. Assume $E \in \mathbb{E}$. We can see that $\mathcal{F}_{\mathbb{A}}(E) = \mathcal{L}_G(E)$ holds by the following equations. \square

$$\begin{aligned} \mathcal{F}_{\mathbb{A}}(\llbracket E \rrbracket) &= \bigcup \{ \mathcal{L}(X) \mid X \in \mathbb{S}_{\mathbb{A}} \wedge \llbracket X \rrbracket \subseteq \llbracket E \rrbracket \} \\ &= \{ \omega \in \mathbb{T}^* \mid \llbracket X \rrbracket \subseteq \llbracket E \rrbracket \wedge X \Rightarrow^* \omega \} \\ &= \{ \omega \in \mathbb{T}^* \mid E \Rightarrow_S X \wedge X \Rightarrow^* \omega \} = \mathcal{L}(E) \end{aligned}$$

3.4 Example

We conclude this section with a full example here. For this reason, consider the CFSG in 7. It describes a common language of arithmetic and logical expressions that all reside in the same domain *Exp*. Note, how this is a design decision that is up to the grammar engineer: We might as well introduce a domain *Arith* and another domain *Logic* to logically separate these sublanguages, for example. The subphrase lattice that corresponds to this grammar is partially presented below the grammar. As we can see here, the subphrase lattice and its corresponding sublanguage lattice may show the same structure. However, this does not always

have to be the case, because all elements of the subphrase lattice might describe the same language in the extreme.

$$\begin{array}{ll}
Add^{Exp} \rightarrow Add^{Exp} + Add^{Exp} \downarrow ; & Imp^{Exp} \rightarrow Imp^{Exp} \downarrow \Rightarrow Imp^{Exp} ; \\
Sub^{Exp} \rightarrow Sub^{Exp} - Sub^{Exp} \downarrow ; & \\
Sub^{Exp} \sqsubseteq Add^{Exp} ; & Or^{Exp} \rightarrow Or^{Exp} \vee Or^{Exp} \downarrow ; \\
Add^{Exp} \sqsubseteq Sub^{Exp} ; & Or^{Exp} \sqsubseteq Imp^{Exp} ; \\
\\
Mul^{Exp} \rightarrow Mul^{Exp} * Mul^{Exp} \downarrow ; & And^{Exp} \rightarrow And^{Exp} \wedge And^{Exp} \downarrow ; \\
Div^{Exp} \rightarrow Div^{Exp} \div Div^{Exp} \downarrow ; & And^{Exp} \sqsubseteq Or^{Exp} ; \\
Mul^{Exp} \sqsubseteq Add^{Exp} ; & \\
Div^{Exp} \sqsubseteq Mul^{Exp} ; & \perp^{Exp} \rightarrow (\top^{Exp}) ; \\
Mul^{Exp} \sqsubseteq Div^{Exp} ; & \perp^{Exp} \rightarrow x ; \quad \emptyset
\end{array}$$

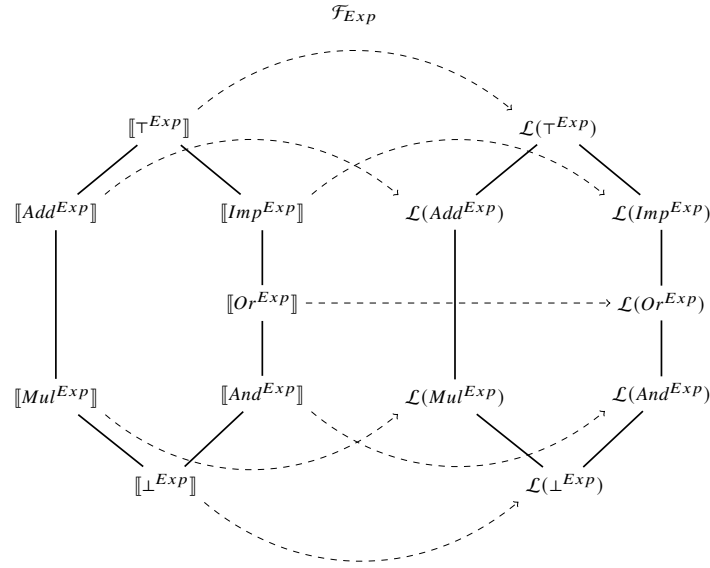


Fig. 7: An exemplary CFSGs with its subphrase and sublanguage lattice

4 Meta Theory

This section is about the meta-theoretic properties of our formalism. In particular, we are interested in how our formalism compares to CFGs. That is, whether it is sound (theorem 2) and complete (theorem 1) with respect to CFGs. As it turns out, this is indeed the case. This leaves CFSGs as an alternative grammar formalism to describe the full class of CFLs.

Before we dive into the two main theorems 1 and 2 here, we need to give a proper definition of CFGs in subsection 4.1 first. On top of that, we observe in subsection 4.2 that any grammar of our formalism has a normal form that directly corresponds to some CFG. The last two subsections 4.3 and 4.4 then present the two main theorems stating the actual completeness and soundness property.

4.1 Context-Free Grammars

We refer to a *Context-Free Grammar* (CFG) as a tuple $G = (\Sigma, N, P, S)$ where Σ is the finite set of terminal symbol, N is the finite set of non-terminal symbols with $\Sigma \cap N = \emptyset$, $P \subseteq N \times (\Sigma \cup N)^*$ is the finite set of production rules and $S \in N$ is the start non-terminal symbol. The derivation relation is then defined as the reflexive and transitive closure of

$$\alpha X \beta \xrightarrow[\exists X \rightarrow \gamma \in P]{=} \alpha \gamma \beta$$

where $\alpha, \beta \in (\Sigma \cup N)^*$. We write $\gamma \Rightarrow^* \delta$ to state that γ yields δ (or δ is derived from γ). To disambiguate the use of this derivation relation with the one of our formalism, we may also write $\gamma \xRightarrow[G]{*} \delta$ when γ yields δ with respect to G . We generally say that $\mathcal{L}_G(\alpha) = \{ \omega \in \Sigma^* \mid \alpha \Rightarrow^* \omega \}$ is the language described by $\alpha \in (\Sigma \cup N)^*$ with respect to G . Specifically, the language $\mathcal{L}(G) = \mathcal{L}_G(S)$ is the language described by the grammar G .

4.2 Context-Free Grammar Normal Form

In this subsection we will examine a special form of a CFSG: the *CFG Normal Form* (CFGNF). Any CFSG that exhibits the CFGNF can be straightforwardly transformed into a CFG (hence the name). It is defined as follows.

Definition 14 (CFG Normal Form). *Let $G = R_1; \dots; R_n; \emptyset$ be a well-formed grammar where $R_i = A_i^{\mathbb{A}_i} \rightarrow S_{1_i} \dots S_{m_i}$ with $A_i \in \mathbb{V}$ and $\mathbb{A}_i \in \mathbb{D}$ for $1 \leq i \leq n$. We say G is in CFG Normal Form (CFGNF) if $\forall i, j \in [1, n] : A_i \neq A_j \Rightarrow \mathbb{A}_i \neq \mathbb{A}_j$ holds.*

A first and very important consequence of the CFGNF is the *complete determination of the domain language* stated in lemma 2. It captures the essence why we can transform any grammar that exhibits the CFGNF into a CFG.

Lemma 2 (Complete Determination of the Domain Language). *Let G be a grammar in CFGNF. Then for any domain $\mathbb{A} \in \mathbb{D}$ of G there is a non-terminal symbol $A^{\mathbb{A}} \in \mathbb{S}_{\mathbb{A}}$ such that for all derivations $\gamma \Rightarrow^* \delta$ with $\gamma, \delta \in (\mathbb{T} \cup \mathbb{E})^*$*

$$\begin{aligned} \gamma \Rightarrow^* \delta &= \gamma \Rightarrow_S^* \alpha A^{\mathbb{A}} \beta \Rightarrow_R \alpha S_1 \dots S_n \beta \Rightarrow^* \delta \\ \vee \quad \gamma \Rightarrow^* \delta &= \gamma \Rightarrow_S^* \delta \end{aligned}$$

is true. That is, the domain language of \mathbb{A} is completely determined by $A^{\mathbb{A}}$ in that any derivation $\gamma \Rightarrow^* \delta$ can only produce terminal symbols by applications of $A^{\mathbb{A}}$'s production rules. We write $\bar{\mathbb{A}}$ to denote this non-terminal symbol $A^{\mathbb{A}}$ of domain \mathbb{A} .

Proof. Let G be a grammar in CFGNF and $\mathbb{A} \in \mathbb{D}$ be a domain of G . Because of the CFGNF we know that there is at most one non-terminal symbol $A^{\mathbb{A}}$ associated with production rules in \mathbb{A} . Otherwise there would be a non-terminal symbol $B^{\mathbb{A}}$ where $B \neq A \Rightarrow \bar{\mathbb{A}} \neq \bar{\mathbb{A}}$ violates the CFGNF.

Now, let $\gamma \Rightarrow^* \delta$ be a derivation with $\gamma, \delta \in (\mathbb{T} \cup \mathbb{E})^*$. We can show by induction over the derivation $\gamma \Rightarrow^* \delta$ that it equals either $\gamma \Rightarrow_S^* \alpha A^{\mathbb{A}} \beta \Rightarrow_R \alpha S_1 \dots S_n \beta \Rightarrow^* \delta$ or $\gamma \Rightarrow_S^* \delta$:

1. $\gamma \Rightarrow^* \gamma$ – By reflexivity of \Rightarrow_S^* .
2. $\gamma \Rightarrow \phi \Rightarrow^* \delta$ – By case analysis on $\gamma \Rightarrow \phi$.
 - (a) $\gamma \Rightarrow_S \phi$ – Thanks to the induction hypothesis we know that

$$\begin{aligned} \phi \Rightarrow^* \delta &= \phi \Rightarrow_S^* \alpha A^{\mathbb{A}} \beta \Rightarrow_R \alpha S_1 \dots S_n \beta \Rightarrow^* \delta \\ \vee \quad \phi \Rightarrow^* \delta &= \phi \Rightarrow_S^* \delta \end{aligned}$$

already holds. By transitivity of \Rightarrow_S^* we can now conclude that

$$\begin{aligned} \gamma \Rightarrow^* \delta &= \gamma \Rightarrow_S \phi \Rightarrow_S^* \alpha A^{\mathbb{A}} \beta \Rightarrow_R \alpha S_1 \dots S_n \beta \Rightarrow^* \delta \\ \vee \quad \gamma \Rightarrow^* \delta &= \gamma \Rightarrow_S \phi \Rightarrow_S^* \delta \end{aligned}$$

holds as well.

- (b) $\gamma \Rightarrow_R \alpha S_1 \dots S_n \beta$ – Because we apply a production rule here, we know that $\gamma = \alpha A^{\mathbb{A}} \beta$ and hence $\gamma \Rightarrow_S^* \alpha A^{\mathbb{A}} \beta$ is true. Consequently,

$$\gamma \Rightarrow^* \delta = \gamma \Rightarrow_S^* \alpha A^{\mathbb{A}} \beta \Rightarrow_R \alpha S_1 \dots S_n \beta \Rightarrow^* \delta$$

holds. \square

The intriguing thing about lemma 2 is its quantification over all derivations. As a consequence, any derivation $\gamma \Rightarrow^* \omega$ of a word $\omega \in \mathbb{T}^*$ must contain n applications of $A^{\mathbb{A}}$'s production rules. Moreover, any application of some production rule of $A^{\mathbb{A}}$ is prefixed by a possibly empty sequence of subphrase derivations:

$$\begin{aligned} \gamma &\Rightarrow_S^* \alpha_1 A^{\mathbb{A}} \beta_1 \Rightarrow_R \alpha_1 S_1 \dots S_{m_1} \beta_1 \\ &\Rightarrow_S^* \alpha_2 A^{\mathbb{A}} \beta_2 \Rightarrow_R \alpha_2 S_1 \dots S_{m_2} \beta_2 \\ &\vdots \\ &\Rightarrow_S^* \alpha_n A^{\mathbb{A}} \beta_n \Rightarrow_R \omega \end{aligned}$$

Since this observation is an essential part of the soundness and completeness of our formalism, it also shows the importance of the CFGNF. As it turns out, any CFSG can be equivalently transformed into the CFGNF. However, the following preliminary work is required before we can actually substantiate this with lemma 9.

Definition 15 (Substitution of Non-Terminal Expressions). *Let G be a grammar and $E_1, E_2 \in G_{\mathbb{E}}$ be two non-terminal expressions. We define substitution of a non-terminal expression E_1 with a non-terminal expression E_2 as the consistent replacement of all occurrences of E_1 in G with E_2 and write $[E_1 \mapsto E_2]G$ to denote the grammar we yield after applying this substitution. Analogously, we write $[E_1 \mapsto E_2]\alpha$ to denote the consistent replacement of any occurrence of E_1 with E_2 in the string $\alpha \in (G_{\mathbb{T}} \cup G_{\mathbb{E}})^*$.*

It is important to note here, that this kind of substitution does not preserve the well-formedness of a grammar nor does it always produce a syntactically valid grammar at all. Yet, the above definition of substitution is a central part of the grammar transformations that come next.

What follows is a grammar transformation we call *elimination*. It allows us to remove all occurrences of so-called *proper non-terminal expression* in a language-preserving way.

Definition 16 (Proper Non-Terminal Expression). *We say $E \in \mathbb{E}$ is a proper non-terminal expression iff $E \notin \mathbb{S}$.*

Definition 17 (Elimination of Proper Non-Terminal Expressions). *Let $G = R_1; \dots; R_n; \emptyset$ be a well-formed grammar and $E \in G_{\mathbb{E}} \setminus G_{\mathbb{S}}$ be a proper non-terminal expression. Moreover, let $A^{\mathbb{A}}$ be a non-terminal symbol where $A \notin G_{\mathbb{V}} \cup G_{\mathbb{T}}$ is a fresh variable symbol and $\mathbb{A} \notin G_{\mathbb{D}} \cup G_{\mathbb{T}}$ is a fresh domain symbol. We define the elimination of E in G as*

$$[E \mapsto A^{\mathbb{A}}]R'_1; \dots; R'_k; R'_{k+1}; \dots; R'_m; \emptyset$$

where the two conditions

$$\{R'_1, \dots, R'_k\} = \{R_1, \dots, R_n\} \quad (1)$$

$$\{R'_{k+1}, \dots, R'_m\} = \{A^{\mathbb{A}} \rightarrow X \mid X \in G_{\mathbb{S}} \wedge \llbracket X \rrbracket \subseteq \llbracket E \rrbracket\} \quad (2)$$

hold and write $\{E \mapsto A^{\mathbb{A}}\}G$ to denote this grammar.

Lemma 3 (Elimination Preserves Well-Formedness). *The elimination $\{E \mapsto A^{\mathbb{A}}\}G$ produces a well-formed grammar.*

Proof. Since G is already well-formed and no proper non-terminal expressions nor subphrase rules are introduced with the elimination $\{E \mapsto A^{\mathbb{A}}\}G$, it must describe a well-formed grammar as well. \square

Lemma 4 (Elimination is Language-Preserving). $\forall \mathbb{B} \in G_{\mathbb{D}} :$
 $G \equiv_{\mathbb{B}} \{E \mapsto A^{\mathbb{A}}\}G.$

Proof. To show that $\forall \mathbb{B} \in G_{\mathbb{D}} : G \equiv_{\mathbb{B}} G'$ with $G' = \{E \mapsto A^{\mathbb{A}}\}G$ holds, it is sufficient to prove $\gamma \xRightarrow[G]{R} \alpha E \beta \xRightarrow[G]{S} \delta \iff \gamma \xRightarrow[G']{R} \alpha A^{\mathbb{A}} \beta \xRightarrow[G']{R} \delta$ because the substitution of E with $A^{\mathbb{A}}$ only affects occurrences on right-hand sides of

production rules and proper non-terminal expressions can themselves not be associated with production rules. By the consistent replacement of E with $A^{\mathbb{A}}$ we already know that $\gamma \xRightarrow[G]{R} \alpha E \beta \iff \gamma \xRightarrow[G']{R} \alpha A^{\mathbb{A}} \beta$ holds. Furthermore, because there is a production rule $A^{\mathbb{A}} \rightarrow X$ in G' if and only if $\llbracket X \rrbracket \subseteq \llbracket E \rrbracket$ for each $X \in G_{\mathbb{S}}$ we know that $\alpha E \beta \xRightarrow[G]{S} \delta \iff \alpha A^{\mathbb{A}} \beta \xRightarrow[G']{R} \delta$ holds as well. \square

Definition 18 (Expression-Free Normal Form). *Let G be a well-formed grammar. We say G is in Expression-Free Normal Form (EFNF) if there is no occurrence of a proper non-terminal expression in G .*

Lemma 5 (Existence of the EFNF). *Let G be a well-formed grammar. Then there is a grammar G' in EFNF such that $\forall \mathbb{A} \in G_{\mathbb{D}} : G \equiv_{\mathbb{A}} G'$ holds.*

Proof. This follows straightforwardly from definition 17 and lemma 4. \square

Lemma 6 (Expression-Free Derivation). *Let G be a grammar in EFNF and $E_1, E_2 \in G_{\mathbb{E}} \setminus G_{\mathbb{S}}$ be two proper non-terminal symbols. Then it holds for any domain $\mathbb{A} \in G_{\mathbb{D}}$ and any word $\omega \in \mathcal{L}_{\mathbb{A}}(G)$ that there are two non-terminal symbols $X, Y \in G_{\mathbb{S}}$ such that*

$$\top^{\mathbb{A}} \Rightarrow^* \alpha E_1 \beta \Rightarrow_S \alpha E_2 \beta \Rightarrow^* \omega \implies \top^{\mathbb{A}} \Rightarrow^* \alpha X \beta \Rightarrow_S \alpha Y \beta \Rightarrow^* \omega$$

is true.

Proof. Assume that $\top^{\mathbb{A}} \Rightarrow^* \alpha E_1 \beta \Rightarrow_S \alpha E_2 \beta \Rightarrow^* \omega$ is a derivation of ω . Since proper non-terminal expressions are never associated with production rules, there must be a non-terminal symbol $Y \in \mathbb{S}$ such that

$$\top^{\mathbb{A}} \Rightarrow^* \alpha E_1 \beta \Rightarrow_S \alpha E_2 \beta \Rightarrow_S^* \alpha Y \beta \Rightarrow^* \omega$$

is a derivation of ω . Furthermore, because G is in EFNF there is no derivation $\gamma \Rightarrow_R \alpha E_1 \beta$ and hence there must be a non-terminal symbol $X \in \mathbb{S}$ such that

$$\top^{\mathbb{A}} \Rightarrow^* \alpha X \beta \Rightarrow_S^* \alpha E_1 \beta \Rightarrow_S \alpha E_2 \beta \Rightarrow_S \alpha Y \beta \Rightarrow^* \omega$$

is a derivation of ω . \square

Definition 19 (Instantiation of Non-Terminal Symbols). *Let $G = R_1; \dots; R_n; \emptyset$ be a grammar in EFNF and $X \in G_{\mathbb{S}}$ a non-terminal symbol of G . Moreover, let $A^{\mathbb{A}}$ be a non-terminal symbol where $A \notin G_{\mathbb{V}} \cup G_{\mathbb{T}}$ is a fresh variable symbol and $\mathbb{A} \notin G_{\mathbb{D}} \cup G_{\mathbb{T}}$ is a fresh domain symbol. We define the instantiation of X as the transformation that yields the grammar*

$$[X \mapsto A^{\mathbb{A}}]R'_1; \dots; R'_k; R'_{k+1}; \dots; R'_j; R'_{j+1}; \dots; R'_m; \emptyset$$

where the three conditions

$$\{ R'_{l+1}, \dots, R'_m \} = \{ R_i \mid i \in [1, n] \wedge \nexists Y \in G_{\mathbb{S}} : R_i = X \sqsubseteq Y \vee R_i = Y \sqsubseteq X \} \quad (1)$$

$$\{ R'_{k+1}, \dots, R'_l \} = \{ A^{\mathbb{A}} \rightarrow Y \mid Y \in G_{\mathbb{S}} \wedge \llbracket Y \rrbracket \subseteq \llbracket X \rrbracket \} \quad (2)$$

$$\{ R'_1, \dots, R'_k \} = \{ Y \rightarrow A^{\mathbb{A}} \mid Y \in G_{\mathbb{S}} \wedge \llbracket X \rrbracket \subseteq \llbracket Y \rrbracket \} \quad (3)$$

hold. We overload the notation $\{ X \mapsto A^{\mathbb{A}} \}G$ to denote this grammar.

Lemma 7 (Instantiation Preserves Well-Formedness). *The instantiation $\{X \mapsto A^A\}G$ produces a well-formed grammar.*

Proof. Since G is already well-formed, we only have to consider those rules of G that are affected by the substitution of X with A^A . Because of condition (1) we remove all ill-formed subphrase rules and since there are no occurrences of proper non-terminal expressions the instantiation $\{X \mapsto A^A\}G$ produces a well-formed grammar. \square

Lemma 8 (Instantiation is Language-Preserving). $\forall \mathbb{B} \in G_{\mathbb{D}} :$
 $G \equiv_{\mathbb{B}} \{X \mapsto A^A\}G.$

Proof. Our goal is to show that $G \equiv_{\mathbb{C}} G'$ holds for any domain $\mathbb{C} \in G_{\mathbb{D}}$ of G with $G' = \{X \mapsto A^A\}G$. Given a domain $\mathbb{C} \in G_{\mathbb{D}}$, this is true when $\mathcal{L}_{\mathbb{C}}(G') \subseteq \mathcal{L}_{\mathbb{C}}(G)$ and $\mathcal{L}_{\mathbb{C}}(G) \subseteq \mathcal{L}_{\mathbb{C}}(G')$ hold.

Subgoal $\mathcal{L}_{\mathbb{C}}(G') \subseteq \mathcal{L}_{\mathbb{C}}(G)$. To substantiate this we can actually even prove that

$$\gamma' \xRightarrow[G']{\quad} \delta' \quad \implies \quad \gamma \xRightarrow[G]{\quad} \delta$$

is true for any two strings $\gamma', \delta' \in G'_{\mathbb{T}} \cup G'_{\mathbb{B}}$ with $\gamma = [A^A \mapsto X]\gamma'$ and $\delta = [A^A \mapsto X]\delta'$. Hence, assume that $\gamma' \xRightarrow[G']{\quad} \delta'$ holds. We have to show that $\gamma \xRightarrow[G]{\quad} \delta$ holds under this assumption. By case analysis on $\gamma' \xRightarrow[G']{\quad} \delta'$ we obtain two cases:

1. $\alpha' E'_1 \beta' \xRightarrow[G']{S} \alpha' E'_2 \beta' \implies$ Since any subphrase rule included in G' is also included in G (consider condition (1) to see this), any subphrase derivation in G' is also possible in G .
2. $\alpha' Y' \beta' \xRightarrow[G']{R} \alpha' S'_1 \dots S'_p \beta' \implies$ Here, we face three cases that could have caused this rule application:
 - (a) $Y' \rightarrow S'_1 \dots S'_p \in \{R'_{l+1}, \dots, R'_m\} \implies$ There must be a production rule $Y \rightarrow S_1 \dots S_p$ in G with $Y = [A^A \mapsto X]Y'$ and $S_1 \dots S_p = [A^A \mapsto X]S'_1 \dots S'_p$ (consider condition (1) to see this) and consequently, there must also be a derivation $\alpha Y \beta \xRightarrow[G]{R} \alpha S_1 \dots S_p \beta$ where $\alpha = [A^A \mapsto X]\alpha'$ and $\beta = [A^A \mapsto X]\beta'$.
 - (b) $Y' \rightarrow S'_1 \dots S'_p \in \{R'_{k+1}, \dots, R'_l\} \implies$ In this case it holds by definition that $Y' = A^A$ and $S'_1 \dots S'_p = [X \mapsto A^A]Z$ for some $Z \in G_{\mathbb{S}}$ with $\llbracket Z \rrbracket \subseteq \llbracket X \rrbracket$ (consider condition (2) to see this). Because of $\llbracket Z \rrbracket \subseteq \llbracket X \rrbracket$, we know that there is a subphrase derivation $\alpha X \beta \xRightarrow[G]{S} \alpha Z \beta$ with $\alpha = [A^A \mapsto X]\alpha'$ and $\beta = [A^A \mapsto X]\beta'$.
 - (c) $Y' \rightarrow S'_1 \dots S'_p \in \{R'_1, \dots, R'_k\} \implies$ In this case it holds by definition that $Y' = [X \mapsto A^A]Z$ for some $Z \in G_{\mathbb{S}}$ with $\llbracket X \rrbracket \subseteq \llbracket Z \rrbracket$ and $S'_1 \dots S'_p = A^A$ (consider condition (3) to see this). Because of $\llbracket X \rrbracket \subseteq \llbracket Z \rrbracket$, we know that there is a subphrase derivation $\alpha Z \beta \xRightarrow[G]{S} \alpha X \beta$ with $\alpha = [A^A \mapsto X]\alpha'$ and $\beta = [A^A \mapsto X]\beta'$.

Subgoal $\mathcal{L}_{\mathbb{C}}(G) \subseteq \mathcal{L}_{\mathbb{C}}(G')$. We can show that there are two strings $\gamma', \delta' \in (G'_{\mathbb{T}} \cup G'_{\mathbb{E}})^*$ such that

$$\tau^{\mathbb{C}} \xRightarrow[G]{*} \gamma \xRightarrow[G]{*} \delta \xRightarrow[G]{*} \omega \quad \implies \quad \tau^{\mathbb{C}} \xRightarrow[G']{*} \gamma' \xRightarrow[G']{*} \delta' \xRightarrow[G']{*} \omega$$

is true for any word $\omega \in G_{\mathbb{T}}^*$ and $\gamma, \delta \in (G_{\mathbb{T}} \cup G_{\mathbb{E}})^*$. Hence, assume that $\tau^{\mathbb{C}} \xRightarrow[G]{*} \gamma \xRightarrow[G]{*} \delta \xRightarrow[G]{*} \omega$ is a derivation of the word $\omega \in G_{\mathbb{T}}^*$ for any $\gamma, \delta \in (G_{\mathbb{T}} \cup G_{\mathbb{E}})^*$. By case analysis on $\gamma \xRightarrow[G]{*} \delta$ we obtain two cases:

1. $\alpha E_1 \beta \xRightarrow[G]{*} \alpha E_2 \beta \implies$ Because of lemma 6 we already know that there are two non-terminal symbols $Y, Z \in G_{\mathbb{S}}$ such that $\tau^{\mathbb{C}} \xRightarrow[G]{*} \alpha Y \beta \xRightarrow[G]{*} \alpha Z \beta \xRightarrow[G]{*} \omega$ holds. Hence, we also know that $\llbracket Z \rrbracket \subseteq \llbracket Y \rrbracket$ is true in G . As the subphrase relation of G' is a subset of the subphrase relation of G (consider condition (1) to see this), we face two cases here with $Z' = [X \mapsto A^{\mathbb{A}}]Z$ and $Y' = [X \mapsto A^{\mathbb{A}}]Y$:
 - (a) $\llbracket Z' \rrbracket \subseteq \llbracket Y' \rrbracket$ in $G' \implies$ This case is trivial, since there is a $\gamma' = [X \mapsto A^{\mathbb{A}}]\alpha Y \beta$ and $\delta' = [X \mapsto A^{\mathbb{A}}]\alpha Z \beta$ such that $\tau^{\mathbb{C}} \xRightarrow[G']{*} \gamma' \xRightarrow[G']{*} \delta' \xRightarrow[G']{*} \omega$ holds.
 - (b) $\llbracket Z' \rrbracket \not\subseteq \llbracket Y' \rrbracket$ in $G' \implies$ This is the case when the only source for the relation between Z' and Y' is a subphrase rule $U \subseteq V$ of G where either $U = X$ or $V = X$. Since such a subphrase rule is removed within G' , there is no possible subphrase derivation $Y' \xRightarrow[G']{*} Z'$ in G' . However, as $\llbracket Z \rrbracket \subseteq \llbracket X \rrbracket \subseteq \llbracket Y \rrbracket$ must have held in G in this case, there is a production rule $A^{\mathbb{A}} \rightarrow Z'$ and a production rule $Y' \rightarrow A^{\mathbb{A}}$ in G' which implies a $\gamma' = [X \mapsto A^{\mathbb{A}}]\alpha Y \beta$ and $\delta' = [X \mapsto A^{\mathbb{A}}]\alpha Z \beta$ such that $\tau^{\mathbb{C}} \xRightarrow[G']{*} \gamma' \xRightarrow[G']{*} A^{\mathbb{A}} \xRightarrow[G']{*} \delta' \xRightarrow[G']{*} \omega$.
2. $\alpha Y \beta \xRightarrow[G]{*} \alpha S_1 \dots S_p \beta \implies$ By definition of G' there must be a production rule $Y' \rightarrow S'_1 \dots S'_p$ with $Y' = [X \mapsto A^{\mathbb{A}}]Y$ and $S'_1 \dots S'_p = [X \mapsto A^{\mathbb{A}}]S_1 \dots S_p$ (consider condition (1) to see this). Hence, there is a $\gamma' = [X \mapsto A^{\mathbb{A}}]\alpha Y \beta$ and $\delta' = [X \mapsto A^{\mathbb{A}}]\alpha S_1 \dots S_p \beta$ such that $\tau^{\mathbb{C}} \xRightarrow[G']{*} \gamma' \xRightarrow[G']{*} \delta' \xRightarrow[G']{*} \omega$ holds.

□

Lemma 9 (Existence of CFGNF). *Let G be a well-formed grammar. Then there is another grammar G' that exhibits the CFGNF such that $\forall \mathbb{A} \in G_{\mathbb{D}} : G \equiv_{\mathbb{A}} G'$.*

Proof. Let G be a well-formed grammar. Because of lemma 5 we know there must be a grammar G'' in EFNF such that $\forall \mathbb{A} \in G_{\mathbb{D}} : G \equiv_{\mathbb{A}} G''$ is true. And thanks to lemma 8 we can instantiate all non-terminal symbols $X \in G''_{\mathbb{S}}$ to obtain the grammar $G' = R_1; \dots; R_n; \emptyset$ where $\forall \mathbb{A} \in G_{\mathbb{D}} : G \equiv_{\mathbb{A}} G'$ holds. Because of this, no subphrase rule can be left in G' and due to the substitution with fresh non-terminal symbols $A_i^{\mathbb{A}_i}$ where $A_i \in G'_{\mathbb{V}}$ it holds that $\forall i \in [1, n] : R_i = A_i^{\mathbb{A}_i} \rightarrow S_1 \dots S_m$ and $\forall i, j \in [1, n] : i \neq j \Rightarrow A_i \neq A_j \wedge \mathbb{A}_i \neq \mathbb{A}_j$ are true. Hence, G' is indeed in CFGNF. □

4.3 Completeness

The first main theorem (theorem 1) about our grammar formalism is that of completeness with respect to CFGs. Within this subsection we will therefore see that our formalism is indeed powerful enough to describe the full class of CFLs. Before we prove theorem 1, though, we first define a function f_{\rightarrow} that translates any CFG into a well-formed instance of our formalism.

Definition 20 (The Function f_{\rightarrow}). Let $G = (\Sigma, N, P, S)$ be a CFG. We define $f_{\rightarrow}(G) = R_1; \dots; R_{|P|}; \emptyset$ as a grammar such that $\mathbb{T} = \Sigma$, $\mathbb{V} = \mathbb{D} = N$ and $\{R_1, \dots, R_{|P|}\} = \{f_{\uparrow}(X) \rightarrow f_{\uparrow}(S_1) \dots f_{\uparrow}(S_n) \mid X \rightarrow S_1 \dots S_n \in P\}$ hold. Here, the function f_{\uparrow} is defined as the bijection:

$$\begin{aligned} f_{\uparrow}(t) &= t & t \in \Sigma \\ f_{\uparrow}(X) &= X^X & X \in N \end{aligned}$$

Lemma 10 (The Grammar $f_{\rightarrow}(G)$ is Well-Formed). Let G be a CFG. The grammar we obtain by $f_{\rightarrow}(G)$ is well-formed.

Proof. We do not generate subphrase rules nor proper non-terminal expressions. Hence, $f_{\rightarrow}(G)$ must be well-formed. \square

Lemma 11 (The Grammar $f_{\rightarrow}(G)$ is in CFGNF). Let G be a CFG. The grammar we obtain by $f_{\rightarrow}(G)$ is in CFGNF.

Proof. By definition of f_{\rightarrow} and f_{\uparrow} the conditions for the CFGNF trivially hold. \square

Theorem 1 (Completeness). Let G be a CFG. Then there is a well-formed grammar G' such that $\exists \mathbb{A} \in G'_{\mathbb{D}} : \mathcal{L}(G) = \mathcal{L}_{\mathbb{A}}(G')$.

Proof. Let $G = (\Sigma, N, P, S)$ be a CFG. Choose $G' = f_{\rightarrow}(G)$ and $\mathbb{A} = S$. Because of lemma 11 and 2, we know there must be a non-terminal symbol $\tilde{X} \in G'_{\mathbb{S}_X}$ that determines the language $\mathcal{L}_X(G')$ for any domain $X \in G'_{\mathbb{D}}$. As this can only be the non-terminal symbol $X^X \in G'_{\mathbb{S}_X}$ that originated from the non-terminal symbol $X \in N$ of G , it must hold that $\mathcal{L}_G(X) = \mathcal{L}_{G'}(X^X)$. Moreover, since $\llbracket S^S \rrbracket \subseteq \llbracket \mathbb{T}^S \rrbracket$ always holds, we can conclude $\mathcal{L}(G) = \mathcal{L}_S(G') = \mathcal{L}(\mathbb{T}^S)$. \square

4.4 Soundness

The second main theorem (theorem 2) about our grammar formalism is that of soundness with respect to CFGs. Within this section we will therefore see that our formalism cannot describe anything that is not a CFL.

Definition 21 (The Function f_{\leftarrow}). Let $G = R_1; \dots; R_n; \emptyset$ a grammar in CFGNF and $\mathbb{A} \in G_{\mathbb{D}}$ be a domain. We define $f_{\leftarrow}(G) = (\Sigma, N, P, \mathbb{A})$ as a CFG

such that $\Sigma = \mathbb{T}$, $N = \mathbb{D} \cup \{ \perp \}$ and $P = \{ f_l(X) \rightarrow f_l(S_1) \dots f_l(S_m) \mid i \in [1, n] \wedge R_i = X \rightarrow S_1 \dots S_m \}$ hold. The function f_l is defined as follows:

$$\begin{aligned} f_l(t) &= t & t \in \mathbb{T} \\ f_l(E) &= \mathbb{B} & \exists \mathbb{B} \in \mathbb{D} : [\tilde{\mathbb{B}}] \in \llbracket E \rrbracket \\ f_l(E) &= \perp & \text{otherwise} \end{aligned}$$

Theorem 2 (Soundness). *Let G be a well-formed CFSG and $\mathbb{A} \in G_{\mathbb{D}}$ be a domain. Then there is a CFG G' such that $\mathcal{L}_{\mathbb{A}}(G) = \mathcal{L}(G')$.*

Proof. Let G be a well-formed CFSG and $\mathbb{A} \in G_{\mathbb{D}}$ be a domain. Because of lemma 9 we know that there must be an equivalent grammar G'' in CFGNF and a CFG $G' = f_{\leftarrow}(G'')$. Hence, all we need to show here is that f_{\leftarrow} preserves the language. Because of lemma 2 we know there must be a non-terminal symbol $\tilde{\mathbb{B}} \in G''_{\mathbb{S}_{\mathbb{B}}}$ that completely determines the language $\mathcal{L}_{\mathbb{B}}(G'')$ for any domain $\mathbb{B} \in G''_{\mathbb{D}}$. Consequently, it holds for any non-terminal expression $E \in \mathbb{E}$ with $[\tilde{\mathbb{B}}] \in \llbracket E \rrbracket$ that $\mathcal{L}_{G''}(E) = \mathcal{L}_{G''}(\tilde{\mathbb{B}})$. We map all these non-terminal expressions to the non-terminal symbol $\mathbb{B} \in N$ of G' . Since there are no other non-terminal symbols associated with production rules, any other non-terminal expression $E \in \mathbb{E}$ with $[\tilde{\mathbb{B}}] \notin \llbracket E \rrbracket$ describes the empty language and is mapped to the special symbol \perp that is never associated with production rules. At this point we can convince ourselves that $\mathcal{L}_{G'}(\mathbb{B}) = \mathcal{L}_{G''}(\tau^{\mathbb{B}})$ holds and therefore $\mathcal{L}_{\mathbb{A}}(G) = \mathcal{L}(G')$ must be true as well. \square

5 Related Work

In this section, we briefly summarize the work of others and relate it to ours. As there are many different approaches to modular syntax definitions, we restrict us to those that seem to have the highest relevance compared to our work.

- *Syntax Definition Formalism* – The *Syntax Definition Formalism* (SDF) [11] [21] is a modular formalism for the development of CFGs. In contrast to our formalism, the approach of the SDF is a broader one that goes beyond our lightweight focus: It emphasizes the development of formal languages in an encompassing way, providing fine-grained control over the grammar using various disambiguation techniques (including priority rules). In this way, large CFGs and even lexical syntax can be defined as unambiguously as possible.
- *Grammatical Framework* – The *Grammatical Framework* (GF) [20] is very similar to the SDF but takes an even broader approach. It is a dependently typed programming language that allows the development of multilingual grammars and natural languages. Even more than with the SDF, its scope is far beyond our goal of a lightweight formalism.
- *Modular Grammar Specification* – In [14] A. Johnstone et al. describe a modularization technique for context-free grammars that allows their composition by the suppression of undesired production rules. This is related to

the problem we identified in subsection 2.1. With our formalism we sought for a solution to this problem, that neither requires the grammar engineer to repeat nor delete existing production rules.

6 Future Work

Although our paper lays a first grammatical foundation, some important aspects must be examined before CFSGs can be used in a real-world language:

- *Static and Dynamic Semantics* – Our formalism is able to describe the full class of CFLs. Yet, for a real-world scenario it must be possible to assign static and dynamic semantics to these languages. First experiments already indicated that Knuth’s attribute grammars for CFGs [16] naturally extend to our CFSGs.
- *Strong Equivalence* – Known parsing techniques for arbitrary CFGs (e.g., Tomita’s GLR technique [23]) can be safely applied to our formalism, when we can show that there is a transformation from CFSGs to CFGs that preserves the structure of the derivation trees (i.e., when both grammars are strongly equivalent).
- *Low Priority Prefix and Postfix Operators* – With our notion of subphrases it is not possible to model low priority prefix and postfix operators. Of course, these can still be modeled using ordinary production rules, but modularity might suffer from this. Therefore, it is an open question whether there is another kind of non-terminal expression that enables this sort of operators.

7 Conclusion

In this paper, we identify a tension between the declarativity and expressiveness of existing techniques for the development of *Embedded Domain-Specific Languages* (EDSLs) in functional programming languages and propose *Context-Free Subphrase Grammars* (CFSGs) as a solution to relax this tension (section 3). The fundamental observation of our work is an alternative interpretation of operator precedence using the notion of subphrases (section 2). In contrast to other approaches, this interpretation scales to arbitrary production rules in a natural way and adheres to the lightweight, pure phrase-oriented emphasis of CFGs. By providing evidence of the completeness and soundness with respect to CFGs (section 4), we eventually show that CFSGs constitute a modular and declarative yet expressive grammar formalism for the development of EDSLs. With our work, we hope to push research in a direction that enables arbitrary context-free syntax extensions in a lightweight and purely declarative way.

References

1. Aasa, A.: Precedences in specifications and implementations of programming languages. *Theoretical Computer Science* **142**(1), 3–26 (1995).

- [https://doi.org/https://doi.org/10.1016/0304-3975\(95\)90680-J](https://doi.org/https://doi.org/10.1016/0304-3975(95)90680-J), <https://www.sciencedirect.com/science/article/pii/S030439759590680J>, selected Papers of the Symposium on Programming Language Implementation and Logic Programming
2. Aho, A.V., Sethi, R., Ullman, J.D.: *Compilers, Principles, Techniques, and Tools*. Addison-Wesley (1986)
 3. Birkhoff, G.: *Lattice Theory*. Colloquium publications, American Mathematical Society (1948), <https://books.google.de/books?id=o4bu3ex9BdkC>
 4. Brady, E.: Idris, a general-purpose dependently typed programming language: Design and implementation. *Journal of Functional Programming* **23**(5), 552–593 (2013). <https://doi.org/10.1017/S095679681300018X>
 5. Chomsky, N.: On certain formal properties of grammars. *Information and Control* **2**(2), 137–167 (1959). [https://doi.org/https://doi.org/10.1016/S0019-9958\(59\)90362-6](https://doi.org/https://doi.org/10.1016/S0019-9958(59)90362-6), <https://www.sciencedirect.com/science/article/pii/S0019995859903626>
 6. Culpepper, R., Felleisen, M., Flatt, M., Krishnamurthi, S.: From Macros to DSLs: The Evolution of Racket. In: Lerner, B.S., Bodík, R., Krishnamurthi, S. (eds.) *3rd Summit on Advances in Programming Languages (SNAPL 2019)*. Leibniz International Proceedings in Informatics (LIPIcs), vol. 136, pp. 5:1–5:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2019). <https://doi.org/10.4230/LIPIcs.SNAPL.2019.5>, <https://drops.dagstuhl.de/opus/volltexte/2019/10548>
 7. Danielsson, N., Norell, U.: Parsing mixfix operators. vol. 5836, pp. 80–99 (09 2008). https://doi.org/10.1007/978-3-642-24452-0_5
 8. Davey, B.A., Priestley, H.A.: *Introduction to Lattices and Order*. Cambridge University Press, 2 edn. (2002). <https://doi.org/10.1017/CBO9780511809088>
 9. Ern , M., Gehrke, M., Pultr, A.: Complete congruences on topologies and down-set lattices. *Applied Categorical Structures* **15**, 163–184 (04 2007). <https://doi.org/10.1007/s10485-006-9054-3>
 10. Felleisen, M., Findler, R., Flatt, M., Krishnamurthi, S., Barzilay, E., McCarthy, J., Tobin-Hochstadt, S.: The racket manifesto. In: Ball, T., Bodik, R., Lerner, B., Morrisett, G., Krishnamurthi, S. (eds.) *1st Summit on Advances in Programming Languages, SNAPL 2015*. pp. 113–128. Leibniz International Proceedings in Informatics, LIPIcs, Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing (May 2015). <https://doi.org/10.4230/LIPIcs.SNAPL.2015.113>
 11. Heering, J., Hendriks, P.R.H., Klint, P., Rekers, J.: The syntax definition formalism sdf—reference manual—. *SIGPLAN Not.* **24**(11), 43–75 (nov 1989). <https://doi.org/10.1145/71605.71607>, <https://doi.org/10.1145/71605.71607>
 12. Hudak, P.: Modular domain specific languages and tools. In: *Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203)*. pp. 134–142 (1998). <https://doi.org/10.1109/ICSR.1998.685738>
 13. Hudak, P.: Building domain-specific embedded languages. *ACM Comput. Surv.* **28**(4es), 196–es (dec 1996). <https://doi.org/10.1145/242224.242477>, <https://doi.org/10.1145/242224.242477>
 14. Johnstone, A., Scott, E., van den Brand, M.: Modular grammar specification. *Science of Computer Programming* **87**, 23–43 (2014). <https://doi.org/https://doi.org/10.1016/j.scico.2013.09.012>, <https://www.sciencedirect.com/science/article/pii/S0167642313002414>
 15. Kats, L.C.L., Visser, E., Wachsmuth, G.: Pure and declarative syntax definition: Paradise lost and regained. In: *Proceedings of Onward! 2010*. ACM (2010)

16. Knuth, D.E.: Semantics of context-free languages. *Math. Syst. Theory* **2**(2), 127–145 (1968), <http://dblp.uni-trier.de/db/journals/mst/mst2.html#Knuth68>, correction: *Mathematical Systems Theory* 5(1): 95-96 (1971)
17. Marlow, S., et al.: Haskell 2010 language report. Available online <http://www.haskell.org/>(May 2011) (2010)
18. Moura, L.d., Ullrich, S.: The lean 4 theorem prover and programming language. In: Platzer, A., Sutcliffe, G. (eds.) *Automated Deduction – CADE 28*. pp. 625–635. Springer International Publishing, Cham (2021)
19. Norell, U.: Towards a practical programming language based on dependent type theory (2007)
20. Ranta, A.: Grammatical framework. *Journal of Functional Programming* **14**(2), 145–189 (2004). <https://doi.org/10.1017/S0956796803004738>
21. de Souza Amorim, L.E., Visser, E.: Multi-purpose syntax definition with sdf3. In: *Software Engineering and Formal Methods: 18th International Conference, SEFM 2020, Amsterdam, The Netherlands, September 14–18, 2020, Proceedings*. p. 1–23. Springer-Verlag, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-58768-0_1, https://doi.org/10.1007/978-3-030-58768-0_1
22. The Coq Development Team: Coq, <https://coq.inria.fr>
23. Tomita, M.: An efficient context-free parsing algorithm for natural languages. In: *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 2*. pp. 756–764. IJCAI’85, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1985)
24. Ullrich, S., de Moura, L.: Beyond notations: Hygienic macro expansion for theorem proving languages. *Log. Methods Comput. Sci.* **18**(2) (2022). [https://doi.org/10.46298/lmcs-18\(2:1\)2022](https://doi.org/10.46298/lmcs-18(2:1)2022), [https://doi.org/10.46298/lmcs-18\(2:1\)2022](https://doi.org/10.46298/lmcs-18(2:1)2022)
25. Weaver, N.: *Lipschitz Algebras*. WORLD SCIENTIFIC (1999). <https://doi.org/10.1142/4100>, <https://www.worldscientific.com/doi/abs/10.1142/4100>
26. Wintner, S.: Modular context-free grammars. *Grammars* **5**(1), 41–63 (2002). <https://doi.org/10.1023/A:1014216630973>, <https://doi.org/10.1023/A:1014216630973>