On Predicting the Grammar of a Normal-Form

On Predicting the Grammar of a Normal-Form

Alan Bundy[1], Predrag Janičić[2], and Alan Smaill[1]

[1] School of Informatics, University of Edinburgh,
Appleton Tower, Crichton St, Edinburgh, EH8 9LE, UK
{A.Bundy,A.Smaill}@ed.ac.uk
[2] Faculty of Mathematics, University of Belgrade,
Studentski trg 16,11000 Belgrade, Yugoslavia.
janicic@matf.bg.ac.yu

Abstract. We introduce the problem of predicting the formal grammar of the normal-form that is generated from a class of expressions by exhaustive application of a set of rewrite rules. We describe and implement a sound but incomplete procedure for solving this problem and report on its theoretical and experimental properties.

1 Introduction

Consider the following problem. Let T be a context-free class of expressions, defined, for instance, by Backus Naur Form (BNF) [3]. Let $\mathcal{R}$ be a set of rewrite rules. Let N be the class of normal-forms of expressions in T under the exhaustive application of rewrite rules from $\mathcal{R}$. What is the grammar of N?

In this paper we investigate this problem, in the context of term rewriting, apparently for the first time, as we have been unable to uncover any prior work on it.[1]

We define *Complementary Backus Naur Form* (CBNF): an extension of BNF grammars, in order to describe the normal-form. We then define a normal-form prediction procedure as a two phase process. In the first phase, we incrementally construct a CBNF describing the normal-form by exhaustively applying two initialisation rules. In the second phase, this CBNF is collapsed to a BNF by applying a set of transformation rules. We explore this prediction procedure both theoretically and experimentally.

Why is this normal-form prediction problem interesting? It first came to our attention as part of a project on the automatic synthesis of decision procedures [4]. Suppose that we have a class of formulae, T, for which we would like to generate a decision procedure and that this formula class is described by a BNF grammar. The class of decision procedures we were interested in could be synthesised from a collection of rewrite rule sets, where each rewrite rule set was applied exhaustively to the current formula until it was in normal-form. One normal-form would be followed by another until a trivial normal-form is

[1] What related work we could find is discussed in §7. We would be grateful for pointers to any other relevant work we have overlooked.

reached consisting only of the propositional constants: true and false. To track the progress of this synthesis process, we used a BNF for each normal-form. So the question naturally arises: given the BNF of the input formula class and a set of rewrite rules, can we predict the BNF of the normal-form it produces?

In our work, to date, on the semi-automatic and automatic synthesis of decision procedures [1, 4], we have used several normalizer generators. Our generators can produce normalizers for specific groups of normalisations (namely those rewrite rules characterised as remove, stratify, absorb, left-assoc and thin in [1]). Each of these generators takes the input BNF and then looks for these specific form of rewrite rules, which can transform any element of the input class into an element of an output class of a prescribed form. If such rewrite rules are found, then the corresponding normalizer is defined as their exhaustive application. Only five normalizer generators were sufficient for completely automatic synthesis of a decision procedure for ground arithmetic, while we had to build four additional arithmetic-specific normalizer generators for synthesising a decision procedures for linear arithmetic.

Note that in this approach we have several normalizer generators and each of them knows how to select appropriate rewrite rules out of the set of all available rewrite rules. So, building a normalizer needs: the input BNF, the normalizer type (which is used for constructing the output BNF) and the set of rewrite rules out of which a subset of appropriate rules has to selected. Now we want to extend that work in the following direction: we want to build a general system which, for the given input BNF and the given set of rewrite rules computes the output BNF in a generic manner. So, one normalisation stage would now have the following parameters: the input BNF and the set of rewrite rules. The result is the output BNF (which can be reached from the input BNF by exhaustive application of the given rewrite rules).

2 Some Examples

To illustrate the normal-form prediction problem, Table 1 gives some simple examples. In example 1 we see that any expression of the form $f(T)$ will be replaced with one of the form $g(T)$. In this simple example the left-hand side of the rewrite rule exactly subsumed one disjunct, $f(T)$, and did not overlap with any other disjunct.

Life is seldom so simple. Sometimes the input BNF needs to be manipulated first to get it into this form. We see the need for this in example 2. The problem here is that the rewrite rule will not completely replace all terms of the form $f(T)$: residual terms of the form of $f(A)$ and $f(g(T))$ will remain. To separate the various forms of $f(T)$ we need to unfold it using the definition of T to the equivalent form:

$$T ::= A \mid f(A) \mid f(f(T)) \mid f(g(T)) \mid g(T)$$

Then $f(f(T))$ can be replaced by $g(T)$, giving a BNF characterising the normal form as in Table 1. Example 3 shows that sometimes only a proper subterm of

No.	Input BNF	Rewrite Rules	Normal-Form BNF
1.	$T ::= A \mid f(T) \mid g(T)$	$f(x) \Rightarrow g(x)$	$N ::= A \mid g(N)$
2.	$T ::= A \mid f(T) \mid g(T)$	$f(f(x)) \Rightarrow g(x)$	$N ::= A \mid f(A) \mid f(g(N)) \mid g(N)$
3.	$T ::= A \mid g(f(T)) \mid g(g(T))$	$f(x) \Rightarrow g(x)$	$N ::= A \mid g(g(N))$
4.	$T ::= a \mid b \mid f(a, T)$	$f(x, y) \Rightarrow g(x, y)$	$N ::= a \mid b \mid g(a, N)$
5.	$T ::= A \mid f(T) \mid h(T, T)$	$f(h(x, y)) \Rightarrow h(f(x), f(y))$	$N ::= A \mid f(E) \mid h(N, N)$ $E ::= A \mid f(E)$

Table 1. Some Example Normal-Form Prediction Problems. *The last column shows the* BNF *for the normal-form generated by applying rewrite rules from the third column to expressions from the second column. A is another grammatical class, which is assumed to be defined elsewhere.*

a disjunct is replaced. Example 4 shows that the disjuncts that are added by the rewrite rules are not determined solely by the right hand sides of the rewrite rules. These right hand sides may need to be instantiated to reflect the different ways in which the left hand sides are instantiated during rewriting. Example 5 shows that sometimes we need two BNF clauses to express a normal form. The creation of additional clauses is done by an unfolding process that is more powerful than that required for example 2.

3 Formal Notation

We will adopt the convention that grammar class variables, sometimes sub-scripted, are represented by capital letters, domain constants with letters a, b, c, domain functions with letters f, g, h and domain variables by x_i for $i \geq 1$, although for readability we will often use x as an abbreviation for x_1, y for x_2, etc. Subscripts are needed on grammar classes because our prediction procedure uses matching and unification and we will need to keep track of which bits of a expression must be identical and which bits can be different, e.g., $f(T_1, T_1, T_2)$ defines a class of expressions in which the first two arguments of f must be identical, but the third can differ from them. We use sub-scripted xs for domain variables to simplify their conversion to sub-scripted grammar class variables, as explained below. If $E[e]$ is an expression with a distinguished subexpression e.

We will sometimes write the body of a BNF as $\Big|_{i \in v} D_i$, where i varies over some set of values, v. For instance, we will let $T ::= \Big|_{i \in [n]} D_i$ be the initial class of expressions, where $[n] = \{i \mid 1 \leq i \leq n\}$. We will refer to the D_i as *disjuncts*. We will exploit the commutativity and associativity of $\mid$ to rearrange the disjuncts to suit our purposes. In particular, we will often promote a particular disjunct to be the first, e.g., $D_j \mid \Big|_{i \in v \setminus \{j\}} D_i$. Our prediction procedure will use matching and unification. To aid this, grammar class variables are standardised apart with the use of subscripts.

To describe the grammar classes of normal-forms we will require *complementary* BNFs (CBNFs), an extension of BNFs. We can define CBNF as follows.

Definition 1 (Complementary BNF) *A* complementary BNF *rule consists of two disjunctions separated by the set difference operator*[2]*, i.e.,*

$$N ::= \bigm| _{j \in v} D_j \setminus \bigm| _{i \in u} C_i$$

The idea is that N consists of expressions of the form D_j, for some j, provided these are *not* of the form C_i, for any i. We will refer to the D_j disjuncts as the *positive part* and the C_i disjuncts as the *negative part*. The expression N to the left of the ::= will be called the *head* of the equation.

We will use the capital letter D, possibly sub-scripted, to range over disjuncts in the positive part and, similarly, C to range over disjuncts in the negative part. We will use Ds to range over a disjunction in the positive part and Cs for a disjunction in the negative part.

The idea behind CBNFs is that exhaustive application of rewrite rules, $l \Rightarrow r$, from $\mathcal{R}$ will add expressions containing r but remove expressions containing l. This is reflected in the CBNF by adding new disjuncts adapted from r to the positive part and new expressions adapted from l to the negative part.

If E is a grammar class variable then $\mathcal{L}(E)$ is the set of (ground and non-ground) expressions that it defines. So $e \in \mathcal{L}(E)$, means that expression e is a member of the class E. Note that $\mathcal{L}(E_1) \subset \mathcal{L}(E_2)$ means that class E_1 is a subclass of class E_2.

Definition 2 (Language of CBNF) *Each* CBNF *rule corresponds to a pair of standard* BNF *rules; for example,* $N ::= \bigm|_{j \in v} D_j \setminus \bigm|_{i \in u} C_i$ *gives two rules,* $N^+ ::= \bigm|_{j \in v} D_j$ *and* $N^- ::= \bigm|_{i \in u} C_i$*. The set of* BNF *rules, perhaps with additional rules, forms a standard* BNF*. The language associated with a given grammar symbol,* $\mathcal{L}(N^+)$ *or* $\mathcal{L}(N^-)$*, is then defined in the standard way, for example via the class of derivations, as in [3], or via an inductive definition, as in [2]. We associate a language with the* CBNF *rule as follows:*

$$\mathcal{L}^C(N) = \mathcal{L}(N^+) \setminus \mathcal{L}(N^-)$$

We also associate languages with more complex expressions as follows:

$$\mathcal{L}(\bigm| _{i \in v} D_i) = \bigcup_{i \in v} \mathcal{L}(D_i)$$

$$\text{If } n \geq 0 \text{ then } \mathcal{L}(f(E_1, \ldots, E_n)) = \{f(e_1, \ldots, e_n) \mid \forall i \in [n].\ e_i \in \mathcal{L}(E_i)\}$$

For a CBNF $N ::= Ds \setminus Cs$ we also write $\mathcal{L}(Ds \setminus Cs)$ for $\mathcal{L}^C(N)$.

Our prediction procedure will manipulate grammars for the class of intermediate expressions and for the normal-form class generated by rewriting. To show our procedure sound we will want to show that grammars output by our procedure describe the actual intermediate expressions and normal-forms that are generated. Below we give definitions of these classes.

[2] Note that we have overloaded the set difference operator, since it is also used in the standard way as a binary function on sets.

Let $x \Rightarrow_{\mathcal{R}} y$ mean x rewrites to y by an application of a rule from $\mathcal{R}$ and $\Rightarrow^*_{\mathcal{R}}$ is the reflexive, transitive closure of $\Rightarrow_{\mathcal{R}}$.

Definition 3 (Intermediate Class) *The* intermediate class of formulae, $int(T, \mathcal{R})$, *generated during application of the rewrite rule set* $\mathcal{R}$ *to formulae defined by the* BNF T *is defined by:*

$$int(T, \mathcal{R}) = \{e \mid \exists t \in \mathcal{L}(T).\ t \Rightarrow^*_{\mathcal{R}} e\}$$

Definition 4 (Normal-Form) *The* normal-form, $norm(T, \mathcal{R})$, *generated by exhaustive application of the rewrite rule set* $\mathcal{R}$ *to formulae defined by the* BNF T *is defined by:*

$$norm(T, \mathcal{R}) = \{n \mid \exists t \in \mathcal{L}(T).\ t \Rightarrow^*_{\mathcal{R}} n \wedge \neg\exists n'.\ n \Rightarrow_{\mathcal{R}} n'\}$$

4 A BNF Prediction Procedure

The prediction procedure has two phases:

1. In the first phase the CBNF for the normal-form class is constructed by incremental addition of disjuncts to both the positive and negative part. This phase consists of the exhaustive application of two *initialisation rules*.
2. In the second phase the CBNF for the normal-form is transformed into one or more BNFs. This phase consists of the exhaustive application of the *complementary rules*.

4.1 Phase 1: Initialisation of the CBNF

The effect of rewriting $\mathcal{L}(T)$ is to add expressions that contain instances of r, where $l \Rightarrow r \in \mathcal{R}$, and to remove expressions that contain instances of l. Note that, these expressions will not necessarily contain all instances of r or l. Which instances are added or removed depends on how the ls can be matched to redexes in I. To discover which instances to add and remove we will incrementally construct a set of instantiated and embedded rewrite rules: $\mathcal{R}^p$. Initially, $\mathcal{R}^p = \emptyset$. Using the current values of I, we will determine which instances of each rule $l \Rightarrow r \in \mathcal{R}$ should be added to $\mathcal{R}^p$. Determining this is the job of the initialisation rules, which we give below. If these rules no longer apply, then this phase terminates with success.

Definition 5 (Grammar Classes) *Before, during and after this phase,* I *and* N *are defined in terms of* $\mathcal{R}^p$ *and* T*, as follows.*

$$I ::= \bigm|_{i\in[n]} D_i\{I/T\} \mid \bigm|_{L\Rightarrow R\in\mathcal{R}^p} R$$

$$N ::= \bigm|_{i\in[n]} D_i\{N/T\} \mid \bigm|_{L\Rightarrow R\in\mathcal{R}^p} R\{N/I\} \setminus \bigm|_{L\Rightarrow R\in\mathcal{R}^p} L$$

where $\bigm|_{i\in[n]} D_i$ *is the body of the definition of* T*.*

where the substitutions of N for I and I for T *etc* preserve subscripts, e.g., $f(x_1, x_1, x_2)\{I/T\} \equiv f(I_1, I_1, I_2)$ and $f(I_1, I_1, I_2)\{N/I\} \equiv f(N_1, N_1, N_2)$.

To illustrate this, consider example 5 in Table 1. Suppose that $\mathcal{R}^p = \{f(h(I_1, I_2)) \Rightarrow h(f(I_1), f(I_2))\}$ then I and N are defined by:

$$I ::= A \mid f(I_1) \mid h(I_2, I_3) \mid h(f(I_4), f(I_5))$$
$$N ::= A \mid f(N_1) \mid h(N_2, N_3) \mid h(f(N_4), f(N_5)) \setminus f(h(I_1, I_2))$$

Note that the negative part of a CBNF is defined using the intermediate class rather than the normal-form class, i.e., we will write

$$N ::= (A \mid f(N_1) \mid g(N_2)) \setminus (f(f(I_1)))$$

rather than

$$N ::= (A \mid f(N_1) \mid g(N_2)) \setminus (f(f(N_3)))$$

for N in example 2 from Table 1.

We do this to deal with the following problem. Clearly $f(f(x))$, for instance, is not in $\mathcal{L}(N)$. Now consider whether $f(f(f(f(x))))$ is in $\mathcal{L}(N)$. It shouldn't be, since this term will be rewritten to $g(x)$ by the rewrite rule. However, there is a problem. Since $f(f(x))$ is *not* in $\mathcal{L}(N)$ then $f(f(f(f(x))))$ is not of the form $f(f(N_3))$. However, $f(f(x))$ *is* in $\mathcal{L}(I)$, so $f(f(f(f(x))))$ is excluded by the negative part of the CBNF, as required.

We now describe the initialization rules that incrementally construct $\mathcal{R}^p$: a set of instantiated and embedded rewrite rules. Each time we add an element to $\mathcal{R}^p$, the definition of I changes. This may allow initialization to apply again, so we must repeatedly apply the initialization process to exhaustion. It will be convenient to use a substitution $\xi = \{I/x\}$ to convert domain variables, e.g., x_1 to grammar variables, e.g., I_1.

Init1: Suppose $\exists l \Rightarrow r \in \mathcal{R}$ such that $\exists \theta.\ l\xi\theta \in \mathcal{L}(I)$. We will add $l\xi\theta \Rightarrow r\xi\theta$ to $\mathcal{R}^p$, provided it is not subsumed by an existing member of $\mathcal{R}^p$, i.e., provided $\neg\exists L' \Rightarrow R' \in \mathcal{R}^p.\exists\theta'.\ (L' \Rightarrow R')\theta' \equiv l\xi\theta \Rightarrow r\xi\theta$.

Init2: Let $I ::= D[e] \mid Ds$, where e is a proper subexpression of D (i.e., $e \neq D$) and e is not a class variable[3]. If there exist $e' \in \mathcal{L}(e)$ and $\exists l \Rightarrow r \in \mathcal{R}$ such that $\exists\theta.\ l\xi\theta \equiv e'$, then stop with failure.

The above Init rules are obviously incomplete as they cannot handle proper subexpressions of BNFs entries which can be rewritten by the given rewrite rules. We have implemented a version of $Init2$ which *does* handle proper subexpressions by creating additional grammar classes. Unfortunately, we have not yet been able to prove the soundness of this rule[4], so have omitted it from this paper. The initialization rules are summarised in Table 2 and illustrated by the examples in Table 3.

[3] Note that the case when e *is* a class variable is dealt with when the rule is applied to its definition.

[4] Although, our intuition is that it *is* sound.

Preconditions	I	**Addition to** $\mathcal{R}^p$
$\exists l \Rightarrow r \in \mathcal{R}.\ \exists\theta.\ l\xi\theta \in \mathcal{L}(I)$ $\neg\exists L' \Rightarrow R' \in \mathcal{R}^p.\exists\theta'.\ (L' \Rightarrow R')\theta' \equiv l\xi\theta \Rightarrow r\xi\theta$	$I ::= Ds$	$l\xi\theta \Rightarrow r\xi\theta$
$e' \in \mathcal{L}(e)$ $\exists l \Rightarrow r \in \mathcal{R}.\exists\theta.\ l\xi\theta \equiv e'$	$I ::= D[e] \mid Ds$	stop with failure

Table 2. Summary of the Initialisation Rule. *The* $Init1$ *rule updates the current values of* $\mathcal{R}^p$ *by adding an instantiated, abstracted rewrite rule. The application of the initialisation rules terminates with success when they can no longer be applied. It terminates with failure if* $Init2$ *can be applied*

No.	I	$\mathcal{R}$	$\mathcal{R}^p$
1.	$I ::= a \mid b \mid f(a, I_1)$	$\{f(x,y) \Rightarrow g(x,y)\}$	$\{f(a, I_1) \Rightarrow g(a, I_1)\}$
2.	$I ::= a \mid g(f(I_1))$	$\{f(x) \Rightarrow h(x)\}$	/
3.	$I ::= a \mid f(I_1)$	$\{g(x) \Rightarrow h(x)\}$	$\{\}$
4.	$I ::= A \mid f(I_1) \mid h(I_2, I_3)$	$\{f(h(x,y)) \Rightarrow h(f(x), f(y))\}$	$\{f(h(I_1, I_2)) \Rightarrow h(f(I_1), f(I_2))\}$
5.	$I ::= a \mid f(I_1)$	$\{f(x) \Rightarrow g(x), g(x) \Rightarrow h(x)\}$	$\{f(I_1) \Rightarrow g(I_1)\}$
	$I ::= a \mid f(I_1) \mid g(I_2)$	$\{f(x) \Rightarrow g(x), g(x) \Rightarrow h(x)\}$	$\{f(I_1) \Rightarrow g(I_1), g(I_1) \Rightarrow h(I_1)\}$

Table 3. Some Examples of Initialisation. *Each row shows the result of one application of the* $Init1$ *rule. The first column gives the example number, the second column shows the initial state of the intermediate class,* I*, the third column gives the set of rewrite rules,* $\mathcal{R}$*, and the fourth column gives the instantiated, abstracted rewrite rules,* $\mathcal{R}^p$*. For example 2 the initialisation fails. Example 4 is example 5 from Table 1 and provides the input to the example in Table 5. Example 5 shows the need for repeated application of the* $Init1$ *rule.*

4.2 Phase 2: Transforming CBNFs to BNFs

The CBNF describing the normal-form is reduced to a BNF by application of the complementary rules. A discussion of each complementary rule is followed by a summary in Table 4.

Trivial: We simplify the positive part of a CBNF by omitting disjuncts consisting solely of its head symbol. Thus $B ::= (A \mid B \mid C \mid D) \setminus \ldots$ is simplified to $B ::= (A \mid C \mid D) \setminus \ldots$

Simplification: Suppose one member of a disjunction contains a redundant disjunct, i.e., the disjunction is of the form $D \mid Ds$, where $\mathcal{L}(D) \subset \mathcal{L}(Ds)$. Then D can be deleted, i.e., $D \mid Ds$ is replaced with Ds. This transformation rule can be applied to either the positive or the negative part of a CBNF. We will call this rule *Simp*. The following example illustrates the application of *Simp*. Suppose N were defined by:

$$N ::= (A \mid f(A) \mid f(g(N_1)) \mid g(g(N_2)) \mid g(N_3)) \setminus \ldots$$

Since $\mathcal{L}(g(g(N_2))) \subset \mathcal{L}(A|f(A)|f(g(N_1))|g(N_3))$, in particular, $\mathcal{L}(g(g(N_2))) \subset \mathcal{L}(g(N_3))$, then *Simp* can delete $g(g(N_2))$ to give:

$$N ::= (A \mid f(A) \mid f(g(N_1)) \mid g(N_3)) \setminus \ldots$$

Subsumption: Suppose the positive part of a CBNF for N contains a disjunct, D, which is subsumed by the negative part of the CBNF, Cs, i.e., $\mathcal{L}(D) \subset \mathcal{L}(Cs)$, in a CBNF $(D \,|\, Ds) \setminus Cs$, then D can be deleted, i.e., the CBNF can be replaced by $Ds \setminus Cs$. We will call this rule *Subsume*. The following example illustrates the application of *Subsume*. Suppose N is:

$$N ::= (A \mid f(A) \mid f(f(N_1)) \mid f(g(N_2)) \mid g(N_3)) \setminus f(f(I_1))$$

Since $\mathcal{L}(f(f(N_1))) \subset \mathcal{L}(f(f(I_1)))$ then *Subsume* can delete $f(f(N_1))$ to give:

$$N ::= (A \mid f(A) \mid f(g(N_2)) \mid g(N_3)) \setminus f(f(I_1))$$

Redundancy: Suppose the negative part of a CBNF for N contains a disjunct, C, which has no intersection with the positive part of the CBNF, Ds, i.e., $\mathcal{L}(C) \cap \mathcal{L}(Ds) = \emptyset$, in a CBNF $Ds \setminus (C \,|\, Cs)$, then C can be deleted because it is trying to remove things that do not occur, i.e., the CBNF can be replaced by $Ds \setminus Cs$. We will call this rule *Redun*. The following example illustrates the application of *Redun*. Suppose N were:

$$N ::= (A \mid g(N_3)) \setminus f(I_1)$$

Since $\mathcal{L}(f(I_1)) \cap \mathcal{L}(A \mid g(N_3)) = \emptyset$ then *Redun* can delete $f(I_1)$ to give:

$$N ::= A \mid g(N_3)$$

Unfolding: Suppose that the positive part of some CBNF, $N ::= D \,|\, Ds \setminus C \,|\, Cs$, contains a disjunct, D, that is not totally subsumed by the negative part, Cs, i.e., $\mathcal{L}(D) \not\subset \mathcal{L}(Cs)$, but that some instance of it, $D\phi$, is subsumed by C, i.e., $\mathcal{L}(D\phi) \subset \mathcal{L}(C)$, but D is the only positive disjunct that overlaps with C, i.e., $\mathcal{L}(Ds) \cap \mathcal{L}(C) = \emptyset$. This case falls between those dealt with by the *Subsume* and *Redun* rules; we can neither delete D nor some part of Cs because some instances of D are subsumed by Cs but not others. We need to separate out the different instances of D: those that are totally subsumed by Cs and those that are disjoint, so that *Subsume* can work on the latter and then *Redun* can prune those parts of Cs that have served their purpose. The $Unfold$ rule performs this separation.
Suppose $\phi = \{d_1/N_1, \ldots, d_k/N_k\}$ and write D as $D(N_1, \ldots, N_k)$. We replace the original CBNF with two CBNFs.

$$N ::= D(E_1, \ldots, E_k) \mid Ds \setminus Cs$$
$$D(E_1, \ldots, E_k) ::= \bigm| _{\langle i_1, \ldots, i_k \rangle \in v \times \ldots \times v} D(D_{i_1}, \ldots, D_{i_k}) \setminus C \mid Cs$$

where E is a new grammatical class and $\bigm|_{i \in v} D_i = D(E_1, \ldots, E_k) \mid Ds$. Note that Ds contains occurrences of N rather than E and that even $D(E_1, \ldots, E_k)$ may contain some occurrences of N if these N are not instantiated by ϕ. Note that the second CBNF is context-sensitive. The role of the *Cancel* rule

is to try to simplify such context-sensitive CBNFs to context-free ones. The following example illustrates the application of $Unfold$. Suppose N were:

$$N ::= (A \mid f(N_1) \mid g(N_2)) \setminus f(f(I_1))$$

Note that $\mathcal{L}(f(N_1)) \not\subset \mathcal{L}(f(f(I_1)))$ but $f(N_1)\{f(x)/N_1\} \in \mathcal{L}(f(f(I_1)))$. So $Unfold$ applies and will separate $f(N_1)$ to give:

$$\begin{aligned} N &::= A \mid f(E) \mid g(N_2) \\ f(E) &::= (f(A) \mid f(f(E_1)) \mid f(g(N_1))) \setminus f(f(I_3)) \end{aligned}$$

to which the $Subsum$ rule applies.

Note that the positive part of the new CBNF has 3 disjuncts because v has size 3 and $k = 1$. If D had been $D(E_1, E_2)$ then there would be 9 disjuncts. Note that the precondition $\mathcal{L}(Ds) \cap \mathcal{L}(C) = \emptyset$ is a cause of incompleteness. If a negative disjunct overlaps with more than one positive disjunct then $Unfold$ cannot be soundly applied.

Cancellation: The $Unfold$ rule introduces context-sensitive CBNFs, whereas we want only context-free CBNFs. The following rule sometimes allows us to convert context-sensitive CBNFs into context-free CBNFs.

If each disjunct in the LHS and RHS has the same dominant k-ary function then k new equations can be formed, each one from the i^{th} arguments. Formally, for a given rule:

$$f(\boldsymbol{E}) ::= \bigm|_{i \in [n]} f(\boldsymbol{d^i})$$

where $\boldsymbol{E}$ is the vector $E_1, \ldots, E_k$, *etc*, suppose there are, for each $j \in [k]$, expressions $d_j^1, \ldots, d_j^{n_j}$ such that by reordering we can get

$$f(\boldsymbol{E}) ::= \bigm|_{\langle i_1,\ldots,i_n\rangle \in [n_1]\times\cdots\times[n_k]} f(d_1^{i_1}, \ldots, d_k^{i_k})$$

Then we can replace this with the the k rules $E_j ::= \bigm|_{i \in [n_i]} d_j^i$ for $j \in [k]$.

We call this the $Cancel$ rule. It is used only in conjunction with the $Unfold$ rule to try to simplify the context-sensitive BNFs it produces into context-free ones. We can illustrate the $Cancel$ rule with the following example.

$$f(E) ::= (f(A) \mid f(g(E_1))) \setminus f(f(I_1))$$

By $Redun$, we obtain

$$f(E) ::= f(A) \mid f(g(E_1))$$

Since every disjunct is dominated by $f(\ldots)$ these can all be cancelled to give:

$$E ::= (A \mid g(E_1))$$

A worked example of the prediction procedure is given in Table 5.

Name	**Precondition**	**Input** CBNF	**Output CBNF**
$Trivial$		$N ::= (N \mid Ds) \setminus Cs$	$N ::= Ds \setminus Cs$
$Simp$	$\mathcal{L}(D) \subset \mathcal{L}(Ds)$	$D \mid Ds$	Ds
$Subsum$	$\mathcal{L}(D) \subset \mathcal{L}(Cs)$	$(D \mid Ds) \setminus Cs$	$Ds \setminus Cs$
$Redun$	$\mathcal{L}(C) \cap \mathcal{L}(Ds) = \emptyset$	$Ds \setminus (C \mid Cs)$	$Ds \setminus Cs$
$Unfold$	$\mathcal{L}(D) \not\subset \mathcal{L}(Cs)$ $\wedge \mathcal{L}(D\phi) \subset \mathcal{L}(C)$ $\wedge \mathcal{L}(Ds) \cap \mathcal{L}(C) = \emptyset$	$N ::= (D(N_1, \ldots, N_k) \mid Ds)$ $\setminus (C \mid Cs)$	$N ::= (D(E_1, \ldots, E_k) \mid Ds) \setminus Cs$ $D(E_1, \ldots, E_k) ::=$ $\Big\vert_{\langle i_1, \ldots, i_k \rangle \in v \times \ldots \times v} D(D_{i_1}, \ldots, D_{i_k})$ $\setminus C \mid Cs$
$Cancel$	See above	$f(\boldsymbol{E}) ::= \Big\vert_{i \in [n]} f(\boldsymbol{d^i})$	$E_1 ::= \Big\vert_{i \in [n]} d_1^i$ $\vdots$ $E_k ::= \Big\vert_{i \in [n]} d_k^i$

Table 4. Summary of Complementary Rules. *Each transformation rule replaces the input expression with the output expression, provided the precondition is met. Rules are fired non-deterministically. In rule* $Unfold$, $\phi = \{d_1/N_1, \ldots, d_k/N_k\}$, E *is a new grammatical class and* $\Big\vert_{i \in v} D_i = D(E_1, \ldots, E_k) \mid Ds$. Note that the procedure must be applied recursively to both clauses in the CBNF produced by the $Unfold$ rule. *The application of these transformation rules terminates with success if all the* CBNF*s are transformed into context-free* BNF*s. Otherwise, it terminates with failure if no more rules are applicable.*

5 Theoretical Analysis

We now discuss issues of soundness, completeness and termination of the prediction procedure described in §4.

5.1 Soundness

By *soundness* of the prediction procedure we mean that any output BNF for N, produced by the prediction procedure when applied to an input BNF for T and a set of rewrite rules $\mathcal{R}$, precisely describes the normal-form that would be produced by exhaustively applying the rewrite rules from $\mathcal{R}$ to formulae of the input BNF T. In this section, we prove the soundness of the prediction procedure described in §4. To formalise the theorem, we need one more definition.

Definition 6 (Predicted Normal-Form) *Suppose that the prediction procedure defined in §4 is applied to* T *and* $\mathcal{R}$ *and terminates with success, outputting a* BNF *for the predicted formal form* N *with zero or more auxiliary clauses. Let* $pred(T, \mathcal{R}) = \mathcal{L}(N)$.

Soundness can now be formalised as $pred(T, \mathcal{R}) = norm(T, \mathcal{R})$, whenever the prediction procedure terminates with success.

Lemma 1 (Soundness of Phase 1). *If the application of the initialisation rules terminates with success then* $\mathcal{L}^C(N) = norm(T, \mathcal{R})$.

Rule	N
$Simp$	$N ::= (A \mid f(N_1) \mid h(N_2, N_3)) \setminus f(h(I_1, I_2))$
$Unfold$	$N ::= (A \mid f(E_1) \mid h(N_2, N_3))$ $f(E) ::= (f(A) \mid f(f(E_1)) \mid f(h(N_2, N_3))) \setminus (f(h(I_1, I_2)))$
$Subsume$	$N ::= A \mid f(N_1) \mid h(N_2, N_3)$ $E ::= (f(A) \mid f(f(E_1))) \setminus f(h(I_1, I_2))$,
$Redun$	$N ::= A \mid f(N_1) \mid h(N_2, N_3)$ $E ::= f(A) \mid f(f(E_1))$
$Cancel$	$N ::= A \mid f(N_1) \mid h(N_2, N_3)$, $E ::= A \mid f(E_1)$

Table 5. A Worked Example. *This is example 5 from Table 1 with input* BNF $T ::= A \mid f(T) \mid h(T, T)$ *and rewrite rule* $f(h(x, y)) \Rightarrow h(f(x), f(y))$. *Since* $f(h(x, y)) \in I$, *initialisation rule* $Init1$ *fires, the rewrite rule* $f(h(I_1, I_2)) \Rightarrow h(f(I_1), f(I_2))$ *is added to* $\mathcal{R}^p$ *and the initialisation process terminates with success. We pick up the story from that point. The entry in each left hand column specifies the rule applied to derive its row. The right hand column gives the current state of* N. *The prediction procedure terminates in the last row as all clauses in the* CBNF *have been reduced to* BNF *clauses.*

Proof. If the initialisation rules terminate then every rewrite rule that is applicable to T is an instance of a member of $\mathcal{R}^p$. Therefore, $\mathcal{L}(I) = int(T, \mathcal{R})$ and $\mathcal{L}^C(N) = norm(T, \mathcal{R})$ (see discussion in §4.1).

Lemma 2 (Complementary Rules are Equivalence Preserving). *For each of the complementary rules of §4.2, provided the preconditions are true, then the language defined by the output* CBNF *is equal to the language defined by the input* CBNF.

Proof. We consider each rule in turn.

Trivial : Given a BNF rule $N ::= N \mid \bigmid_{j \in v} D_j$, by definition $\mathcal{L}(N)$ is the least set such that $\mathcal{L}(N) = \mathcal{L}(N) \cup \bigcup_{j \in v} \mathcal{L}(D_j)$; thus $\mathcal{L}(N) = \bigcup_{j \in v} \mathcal{L}(D_j)$, justifying the simplification.

Simp: Assume $\mathcal{L}(D) \subset \mathcal{L}(Ds)$ then: $\mathcal{L}(D \mid Ds) = \mathcal{L}(D) \cup \mathcal{L}(Ds) = \mathcal{L}(Ds)$.

Subsum: Assume $\mathcal{L}(D) \subset \mathcal{L}(Cs)$ then: $\mathcal{L}((D \mid Ds) \setminus Cs) = (\mathcal{L}(D) \cup \mathcal{L}(Ds)) \setminus \mathcal{L}(Cs) = \mathcal{L}(Ds) \setminus \mathcal{L}(Cs) = \mathcal{L}(Ds \setminus Cs)$.

Redun: Assume $\mathcal{L}(C) \cap \mathcal{L}(Ds) = \emptyset$ then: $\mathcal{L}(Ds \setminus (C \mid Cs)) = \mathcal{L}(Ds) \setminus (\mathcal{L}(C) \cup \mathcal{L}(Cs)) = \mathcal{L}(Ds) \setminus \mathcal{L}(Cs) = \mathcal{L}(Ds \setminus Cs)$.

Unfold: The preconditions for this rule are purely heuristic; except for $\mathcal{L}(Ds) \cap \mathcal{L}(C) = \emptyset$.

$$
\begin{aligned}
& e \in \mathcal{L}((D(N_1, \ldots, N_k) \mid Ds) \setminus (C \mid Cs)) \\
& \iff e \in (\mathcal{L}(D(N_1, \ldots, N_k)) \cup \mathcal{L}(Ds)) \setminus (\mathcal{L}(C) \cup \mathcal{L}(Cs)) \\
& \iff e \in (\mathcal{L}(D(E_1, \ldots, E_k)) \cup \mathcal{L}(Ds)) \setminus \mathcal{L}(Cs) \\
& \iff e \in \mathcal{L}(D(E_1, \ldots, E_k) \mid Ds) \setminus Cs)
\end{aligned}
$$

where

$$
D(E_1, \ldots, E_k) ::= \bigmid_{\langle i_1, \ldots, i_k \rangle \in v \times \ldots \times v} D(D_{i_1}, \ldots, D_{i_k}) \setminus (C \mid Cs)
$$

Note that C can be dropped from the negative part of the CBNF for N because its role is totally dealt with in the CBNF for $D(E_1, \ldots, E_k)$. $D(N_1, \ldots, N_k)$ can be replaced by $D(E_1, \ldots, E_k)$ because the definition of $D(E_1, \ldots, E_k)$ ensures they will generate the same language.

Cancel: We note that this rule only applies to context-sensitive CBNFs and that such CBNFs are only introduced, along with a new class E by the $Unfold$ rule. So we prove this case only in the context that the only instances of the class E occur with dominant symbol f.

$$
\begin{aligned}
& f(e_1, \ldots, e_k) \in \mathcal{L}(f(E_1, \ldots, E_k)) \\
& \iff f(e_1, \ldots, e_k) \in \mathcal{L}\Big(\bigm| _{\langle i_1, \ldots, i_n \rangle \in [n_1] \times \cdots \times [n_k]} f(d_1^{i_1}, \ldots, d_k^{i_k}) \Big) \\
& \iff f(e_1, \ldots, e_k) \in \bigcup_{\langle i_1, \ldots, i_n \rangle \in [n_1] \times \cdots \times [n_k]} \{ f(e_1^{i_1}, \ldots, e_k^{i_k}) \mid \forall j \in [k].e_j^{i_j} \in \mathcal{L}(d_j^{i_j}) \} \\
& \iff \forall j \in [k].e_j \in \mathcal{L}\Big(\bigm| _{i \in [n_i]} d_j^i \Big) \\
& \iff \forall j \in [k].e_j \in \mathcal{L}(E_j)
\end{aligned}
$$

Theorem 1 (Soundness of the Prediction Procedure). *If the prediction procedure, applied to T and $\mathcal{R}$, terminates with success, then $pred(T, \mathcal{R}) = norm(T, \mathcal{R})$.*

Proof. By lemma 1, $\mathcal{L}^C(N) = norm(T, \mathcal{R})$. By induction using lemma 2, the output of the application of the complementary rules is equal to its input. Hence, on termination, $pred(T, \mathcal{R}) = \mathcal{L}(N) = norm(T, \mathcal{R})$.

5.2 Completeness

What kind of completeness might we expect from our prediction procedure? There are sets of rewrite rules and context-free sets of formulae whose normal-form is not context-free, and hence not describable with a BNF. For instance, for input BNF $T ::= f(a, f(c, b)) | f(f(a, T), f(c, b))$ and the rewrite rules $f(x, f(y, z)) \Rightarrow f(f(x, y), z)$ and $f(f(x, c), b) \Rightarrow f(f(x, b), c)$, the set of normal forms is a set of expressions in left-associative normal form with n arguments a, n arguments b and n arguments c, in that order ($n \geq 1$). However, this language is not context-free, which can be proved by the *pumping lemma* for context-free languages [3].

Therefore, a prediction procedure that always terminates with success (i.e., with a BNF describing the set of normal forms) is not attainable. Instead, we can define *completeness* to mean that if a BNF describing the normal-form exists, then the prediction procedure outputs an equivalent BNF.

The prediction procedure as described in §4 is incomplete. One cause of incompleteness is the $Unfold$ rule, as suggested in §4.2. Phase 2 cannot transform the following CBNF:

$$
\begin{aligned}
I &::= A \mid f(f(I, I), I) \\
N &::= A \mid f(f(N, N), N) \setminus f(I, f(I, I))
\end{aligned}
\tag{1}
$$

into an equivalent BNF, although there is one (e.g., $N ::= A | f(f(N, A), A)$). This is so because *Unfold* requires a negative disjunct to subsume $f(f(N, N), N)\varphi$ for some φ. Moreover, for the corresponding original problem:

$$T ::= A \mid f(f(T, T), T)$$

with a given rewrite rule $f(x, f(y, z)) \Rightarrow f(f(x, y), z)$, even phase 1 cannot instantiate the given rewrite rule, so fails to prepare the input for phase 2 — which should be the BNF and CBNF given in (1) above.

5.3 Termination

We have explored various measures to show termination. It is easy to find suitable measures that cover all the rules except $Init1$ and $Unfold$. But neither have we been able to identify a non-terminating example. So the question of termination is currently open.

6 Implementation

We have made a prototype implementation of the prediction procedure in Prolog.[5] Prolog is well suited to prototype implementation of procedures described as sets of transformation rules and procedures dealing with parsing, so the program is short and transparent. Some of the preconditions used in the described algorithm are computationally infeasible as stated, e.g., $\mathcal{L}(D) \subset \mathcal{L}(Ds)$, so were replaced by checking of finite derivations within appropriate grammars.

The complementary rules are tried in the following fixed ordering *Trivial*, *Simpl*, *Subsum*. *Redun*, *Cancel*, and *Unfold*.

Some of the more interesting evaluation examples are given in Table 6. We also successfully applied the procedure to the development examples Table 1. Notice that some of the generated normal-forms are not in the simplest form. For instance, in example 3 in Table 6, derivations that involve the class E cannot produce expressions without non-terminal symbols, so the generated normal form could be replaced by the equivalent BNF with just $N ::= a$. We will try to address this problem, at least to some extent in our future work (however, this task has its limitations since, for instance, it is not decidable whether a context-free grammar produces a non-empty language). In example 4 the given rewrite rule is non-terminating; however, taking into account that the set of normal forms is the set of expression that cannot be further rewritten, we see that the normal form is $N ::= a$, as given by our prediction procedure. Examples 7, 8 and 9 illustrate normalisations needed in several decision procedures (as they can be combined into an algorithm for producing disjunctive normal form).

We didn't try to make the implementation maximally efficient, but rather to make it flexible and easy to understand and modify. Indeed, the described procedure and the corresponding implementation are supposed to work in problems

[5] We evaluated the implementation in SWI Prolog and on PC 466Mz. The source code is available upon request from the second author.

such as synthesis of decision procedures, thus — off-line, so the time used is not critical. Yet, the made implementation works fast and all the examples given in Table 6 were solved in 0.8 seconds of CPU time in total (which shows that the proposed prediction procedure is feasible, at least for simple problems).

No.	**Input** BNF	**Rewrite Rules**	**Normal-Form** BNF
1.	$T ::= A \mid f(T)$	$f(x) \Rightarrow g(x)$ $g(x) \Rightarrow h(x)$	$N ::= A \mid h(T)$
2.	$T ::= a \mid f(T,T)$	$f(x,a) \Rightarrow a$	$N ::= a \mid f(N,E)$ $E ::= f(N,E)$
3.	$T ::= a \mid f(T)$	$f(x) \Rightarrow f(f(x))$	$N ::= a$
4.	$T ::= A \mid f(g(T))$	$g(f(x)) \Rightarrow p(x)$	$N ::= A \mid f(E)$ $E ::= g(A) \mid p(A)$
5.	$T ::= A \mid f(T,T)$	$f(x,f(y,z)) \Rightarrow f(f(x,y),z)$	$N ::= A \mid f(N,A)$
6.	$T ::= A \mid T \wedge T \mid T \vee T$	$x \wedge (y \vee z) \Rightarrow (x \wedge y) \vee (x \wedge z)$ $(y \vee z) \wedge x \Rightarrow (y \wedge x) \vee (z \wedge x)$	$N ::= A \mid E' \wedge E \mid N \vee N$ $E ::= A \mid E'' \wedge E$ $E' ::= A \mid E' \wedge E$ $E'' ::= A \mid E'' \wedge E$
7.	$T ::= A \mid T \wedge T \mid T \vee T \mid \neg T$	$\neg(x \wedge y) \Rightarrow \neg x \vee \neg y$ $\neg(x \vee y) \Rightarrow \neg x \wedge \neg y$	$N ::= A \mid T \wedge T \mid T \vee T \mid \neg E$ $E ::= A \mid \neg E$
8.	$T ::= A \mid \neg T$	$\neg(\neg x) \Rightarrow x$	$N := A \mid \neg A$

Table 6. Some Results from the Evaluation. *The examples above illustrate some of the normal-forms produced by our Prolog implementation of the prediction procedure. Phase 1 currently fails for example 4, so the* CBNF *was provided by hand for phase 2. The procedure described in this paper fails on examples 6 and 7. However, they are included here because we have a version of the implementation (with stonger version of underlying Unfold), extending the account in this paper, which succeeds on these examples, but whose theory has not yet been fully developed.*

7 Conclusion

We have motivated, described, implemented and evaluated a procedure for predicting the normal-form produced when a set of rewrite rules is applied exhaustively to a class of expressions. This procedure has been shown to be sound but incomplete. The evaluation shows that it can effectively predict the normal-forms of a wide range of examples.

To the best of our knowledge, we are the first people to identify and solve this problem, so there is no directly related work with which to compare our work. However, some of the complementary grammar transformations, especially unfolding, are reminiscent of logic program transformations that preserve the declarative semantics of such programs (see e.g. [7]). We have already adapted techniques from that area and there could be further opportunities. There is also some relevant work in the context of a form of string rewriting [6].

Our normal-form prediction problem is orthogonal to Knuth-Bendix completion [5]. Given a set of expressions E and a set of rewrite rules $\mathcal{R}$, the Knuth-Bendix completion procedure produces a confluent and terminating set of rewrite rules set $\mathcal{R}'$, such that both $\mathcal{R}$ and $\mathcal{R}'$ produce the same set of normal-forms when applied to E. So, whereas the focus of Knuth-Bendix completion is the transformation of the rewrite rule set, our focus is the characterisation of the set of normal-forms.

In further work we will tackle the following tasks:

- Continue to develop and evaluate our program on an increasingly harder and wider range of examples. This may reveal the need for further transformation rules or the extension of existing ones, e.g., an extension to the $Unfold$ rule that would enable it to deal with the counter-example in §5.2. In particular, we have limited our attention to context-free input grammars, but have been forced to consider context-sensitive grammars during intermediate processing. We will try to extend our work to deal with context-sensitive input grammars.
- Explore further theoretical properties of our procedure, especially completeness and termination. This exploration will proceed concurrently with the search for counter-examples and modifications to our program.
- Apply our program to the synthesis of decision procedures, an application we described in §1.

A longer version of this paper, with more details of the proofs, implementation, *etc*, can be found at `http://homepages.inf.ed.ac.uk/bundy/drafts/drafts.html`.

References

1. A. Bundy. The Use of Proof Plans for Normalization. In R. S. Boyer, editor, *Essays in Honor of Woody Bledsoe*, pages 149–166. Kluwer, 1991. Also available from Edinburgh as DAI Research Paper No. 513.
2. S. Feferman. Finitary inductively presented logics. In *Logic Colloquium '88*, pages 191–220, Amsterdam, 1989. North-Holland.
3. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 1979.
4. P. Janičić and A. Bundy. Automatic synthesis of decision procedures: a case study of ground and linear arithmetic. 2003. Technical Report forthcoming, School of Informatics, University of Edinburg. Submitted to Journal of Automated Reasoning.
5. D. E. Knuth and P. B. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational problems in abstract algebra*, pages 263–297. Pergamon Press, 1970.
6. J. Michaelis. Transforming linear context-free rewriting systems into minimalist grammars. In P. de Groote, G. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics*, volume 2099, pages 228–244, Berlin, 2001. Springer.
7. A. Pettorossi and M. Proletti. Transformation of logic programs. In D. M. Gabbay, C. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence: Logic Programming*, volume 5, pages 607–787. Oxford University Press, 1998.

A Discussion: Phase 1

The Initialisation phase given in §4.1 is obviously incomplete as it cannot handle proper subexpressions of BNFs entries which can be rewritten by the given rewrite rules. Basically there are two options for handling such cases: by staying within the original class or by adding a new BNF class.

Consider the following BNF:

$$I ::= a \mid f(g(I))$$

and the given rewrite rule $g(f(x)) \Rightarrow p(x)$. The expression $g(I)$ can derive $g(f(g(I)))$ and by the given rewrite rules this expression can be rewritten to $p(g(I))$. Thus, according to the general idea for handling rewrite rules, we are interested in this instance of the rule: $f(g(f(g(x)))) \Rightarrow f(p(g(x)))$, and so we add $f(p(g(I))$ as a positive disjunct to the class I. However, $g(I)$ (which is a subexpression of $f(p(g(I))))$ can derive $g(f(g(I)))$, and the given rewrite rule is applicable to $f(p(g(f(g(I)))))$. Therefore, we have to add the corresponding instantiated rewrite rule $f(p(g(I))) \Rightarrow f(p(p(g(I))))$, i.e., we have to add $f(p(p(g(I))))$ as a positive disjunct to the class I. Obviously, this process loops and the class I has the form:

$$I ::= a \mid f(g(I)) \mid f(p(g(I))) \mid f(p(p(g(I)))) \mid f(p(p(p(g(I))))) \mid \ldots$$

The explanation for this non-termination is permanent extending of the definition and the language of I. In same cases (like the one above) extending the definition of I is unavoidlable. However, in some cases it is not — namely, for each instantiated rewrite rule $L \Rightarrow R$, we can check whether R is already in the language of I (while, of course, we always have to add L as a negative disjunct to the definition of N) and *only if it is not*, then the procedure can terminate with failure. This would prevent non-termination in many cases, while would still allow handling proper subexpressions of words from $\mathcal{L}(I)$ and widen the realm of the procedure. This change would require slight changes in the current phase 1 as the classes I and N wouldn't be updated unconditionally with respect to $\mathcal{R}^p$, but with respect to the above conditions. Notice that this variant of the initialisation won't keep us from BNF languages which are changed by an exhaustive application of a rewrite rule set — because $Init1$ allow this possibility (the possibility of extending the original language of I). This variant of the initialisation seems as a most promising alternative to the variant given in §4.1. To recapitulate, it would change the rule $Init2$ by the following version:

Init2: Let $I ::= D[e] \mid Ds$, where e is a proper subexpression of D (i.e., $e \neq D$) and e is not a class variable[6]. If there exist $e' \in \mathcal{L}(e)$ and $\exists l \Rightarrow r \in \mathcal{R}$ such that $\exists \theta.\ l\xi\theta \equiv e'$, then let $L \Rightarrow R$ is $D[l\xi\theta/e] \Rightarrow D[r\xi\theta/e]$. If $R \notin \mathcal{L}(I)$, then stop with failure. Otherwise, if $L \Rightarrow R$ is not subsumed by an existing member of $\mathcal{R}^p$, (i.e., provided $\neg\exists(L' \Rightarrow R') \in \mathcal{R}^p.\exists\theta'.\ (L' \Rightarrow R')\theta' \equiv L \Rightarrow R$), then add L as a negative disjunct to the class N.

[6] Note that the case when e *is* a class variable is dealt with when the rule is applied to its definition.

In the proposed version both $Init1$ and $Init2$ are sound, but maybe even terminating, which is a subject of our current exploration.

Another option for handling non-termination of the phase 1 and dealing with proper subexpression of words from $\mathcal{L}(I)$ is to introduce a new BNF class which corresponds the subexpression of a BNF entry for I. In our example, new class T' would be:

$$\begin{aligned} I &::= a \mid f(T') \\ N &::= a \mid f(g(N)) \\ T' &::= g(I) \end{aligned}$$

and after applying initialisation rule, we would have:

$$\begin{aligned} I &::= a \mid f(N') \\ N &::= a \mid f(N) \\ I' &::= g(I) \mid p(I') \\ N' &::= g(I) \mid p(N') \setminus g(f(I')) \end{aligned}$$

This is sound transformation, however, such output of the phase 1 complicates the phase 2 (because the definitions of N' and I' involve both I and I') and in addition it does not guarantee neither termination nor completeness. That is why we decided to use simple, rather weak variant for Init. It keeps things clear and flexible, while the phase 1 can also be improved regardless of the rest of the procedure. Also, our experiments show that this variant of Init rule can still cover a large number of examples (not the one given above).

B Not All Sets of Normal Forms are Context-Free

We will prove that for a context-free language of terms, the language of normal-forms of terms obtained under the exhaustive application of given rewrite rules is not necessarily context-free. We will prove this by constructing a counter-example and by using the *pumping lemma* [3]:

Theorem 2 (Pumping lemma). *Let L be a context-free language. Then, for every $x \in L$ for which $|x| > 2^n$, we have $x = r_1 q_1 r q_2 r_2$ where*

1. $|q_1 r q_2| \leq 2^n$;
2. $q_1 q_2 \neq 0$;
3. *for all* $i \geq 0$, $r_1 q_1^{[i]} r q_2^{[i]} r_2 \in L$.

Proposition 1 *Let T is a* BNF *class defined in the following way:*

$$T ::= f(a, f(c, b)) \mid f(f(a, T), f(c, b))$$

Let $\mathcal{R}$ is a set consisting of the following rewrite rules:

$$\begin{aligned} r_1 \; & f(x, f(y, z)) \Rightarrow f(f(x, y), z) \\ r_2 \; & f(f(x, c), b) \Rightarrow f(f(x, b), c). \end{aligned}$$

Let $\mathcal{S}$ be the set of normal forms of $\mathcal{L}(T)$ under the exhaustive application of the set $\mathcal{R}$. L is not context-free.

Proof. The set of rewrite rules $\mathcal{R}$ is obviously terminating. It is easy to show that $\mathcal{R}$ is locally confluent, so it is also confluent. Thus, if $\mathcal{R}$ rewrites $t \in \mathcal{L}(T)$ to a normal form $\hat{t}$, then $\hat{t}$ would be also obtained if first the rule r_1 is exhaustively applied, giving t', and then the rule r_2 is exhaustively applied to t'.

If $t \in \mathcal{L}(T)$ then t was generated by applying the second BNF rule m times ($m \geq 0$) and then the first BNF rule once. It can be easily proved that t has $3m+2$ occurrences of the symbol f and $m+1$ occurrences of each of symbols a, b and c. Also, all occurrences of the symbols a precede all occurrences of symbols b and c.

The exhaustive application of r_1 transforms t into t', while t' is in left-associative normal form (while all occurrences of the symbols a still precede all occurrences of the symbols b and c). So, the term t' is of the form:

$$\underbrace{f(f(f(\ldots(f}_{3m+2}\underbrace{(a,a),a),\ldots,a)}_{m+1},\underbrace{c),b),c),b),\ldots,c),b)}_{2m+2}$$

The exhaustive application of r_2 "moves" all occurrences of the symbol b leftwards, "through" occurrences of the symbol c and the term $\hat{t}$ is of the form:

$$\underbrace{f(f(f(\ldots(f}_{3m+2}\underbrace{(a,a),a),\ldots,a)}_{m+1},\underbrace{b),b),\ldots),b)}_{m+1},\underbrace{c),c)\ldots,c)}_{m+1}$$

Let us denote the above term by $e(m)$ ($m \geq 0$). In $e(m)$ all occurrences of the symbols a still precede all occurrences of the symbol b and all occurrences of the symbol b precede occurrences of the symbol c.

The language $\mathcal{S}$ is the set $\{e(m) \mid m \geq 0\}$ Let us prove that it is not context-free.[7]. Let us suppose the opposite, i.e., let us suppose that $\mathcal{S}$ *is* context-free. If it is context-free, then for $\mathcal{S}$ the statement of pumping lemma holds. Choose m so large that $|e(m)| > 2^n$ (for instance, let $m > 2^n$). Then

$$e(m) = r_1 q_1 r q_2 r_2$$

and

$$r_1 q_1^{[i]} r q_2^{[i]} r_2 \in \mathcal{S} \text{ for each } i \geq 0$$

The language $\mathcal{S}$ consists of terms $e(m)$ each of which has the same number of occurrences of the symbols a, b and c. It holds $q_1 q_2 \neq 0$. If $q_1 q_2$ does not contain any of the symbols a, b and c, then $r_1 q_1^{[0]} r q_2^{[0]} r_2 = r_1 r r_2$ wouldn't be well-formed term and would not belong to $\mathcal{S}$. On the other hand, each element of $\mathcal{S}$ has the same number of occurrences of the symbols a, b and c, therefore $q_1 q_2$ must contain the same number of the symbols a, b and c. So, q_1 or q_2 must contain at least two out of these symbols. Say q_1 has occurrences of the symbol a and the symbol b. Then, in $q_1^{[2]}$ (within $r_1 q_1^{[i]} r q_2^{[i]} r_2 \in \mathcal{S}$) there are occurrences

[7] Analogous example for string-rewriting system is discussed in [3, p128]

of the symbol b that precede some occurrences of the symbol a. However, this is impossible, as in each of terms $e(m)$ all occurrences of the symbols a precede all occurrences of the symbols b. Other cases are similar.

The above propositions leads us to the following one:

Proposition 2 *Let T be a context-free class of terms and $\mathcal{R}$ a set of rewrite rules, then the language of normal-forms of terms in $\mathcal{L}(T)$ under the exhaustive application of rewrite rules from $\mathcal{R}$ is not necessarily context-free.*

Thus, not always are languages of normal-forms definable by BNFs.

C Is the Normal Form Set Always At Least Recursive?

Proposition 3 *For a given a recursive set $\mathcal{S}$ of strings and confluent and terminating (finite or infinite but recursive) set of rewrite rules $\mathcal{R}$, the set of normal forms for elements from $\mathcal{S}$ with respect to $\mathcal{R}$ is not necessarily recursive.*

Proof. Let $\mathcal{C}$ be a recursively enumerable set that is not recursive. Then there exists an injective recursive function f such that $Ran(f) = \mathcal{C}$. Let us consider the set $\mathcal{S}$ of strings $\{ab^m c \mid m \in \mathbf{N}\}$ and the following infinite string rewriting system:

$$R = \{ab^{f(n)}c \longrightarrow b^n \mid n \in \mathbf{N}\} .$$

Obviously, R is recursive and, also, confluent and terminating.

We are interested in describing the set of normal forms of strings from $\mathcal{S}$ with respect to $\mathcal{R}$. For $m \in \mathbf{N}$ it holds:

$ab^m c$ is irreducible (i.e., is in normal form) *if and only if* m is not in $\mathcal{C}$.

Since $\mathcal{C}$ is recursively enumerable but not recursive, it follows that "m is not in $\mathcal{C}$" is not decidable and thus the set of normal forms with respect to $\mathcal{R}$ is non-recursive (and even not recursively enumerable).[8]

The statement analogous to the above one can be proved for sets of terms and term rewriting rules.

Proposition 4 *For a given a recursive set $\mathcal{S}$ and finite, confluent and terminating, set of rewrite rules $\mathcal{R}$, if the set of normal forms $\mathcal{S}'$ for elements from $\mathcal{S}$ with respect to $\mathcal{R}$ is subset of $\mathcal{S}$, then $\mathcal{S}'$ is recursive.*

Proof. $x \in \mathcal{S}'$ if and only if $x \in \mathcal{S}$ and there is no subexpression of x which is unifiable with left hand side of some rule from $\mathcal{R}$.

[8] We are grateful to Andrea Sattler-Klein for this example.

D And When We Finally Have the Resulting Language

Having defined normal forms in one or another way, at each stage (of our process of the making a sequence of normalisations) we'll might need to have an algorithm to test whether an expression (i.e., formula) x belongs to that class. It seems that the problem of testing whether a certain expression belong to the current language (i.e., set of normal forms) can be handled (provided that language is described via some finite device, e.g., formal grammar, or as a complement of some grammar's languages). However, what we might really need is to test whether that language/grammar is of some specific form. Namely, our goal (in our process of generating the sequence of the normal forms) is usually some specific BNF or some specific language. Ideally — BNF with only two entries — $\{\top, \bot\}$ (if we are synthesising a decision procedure for some theory). However, it is undecidable to know whether two formal grammars generate the same language, so it is very likely that the problem $L(G') \setminus L(G) = \{\top, \bot\}?$ is undecidable, too. This suggests that we need not only the algorithm which can make a characterisation of the resulting language, but that that characterisation has to be very simple and suitable for the above questions (at least, in most of the cases). In the process of automatic synthesis of subsequent normal form one can hope that he can reach a trivial BNF as a final result, although there are no guarantees of that kind.

E Example with stronger $Unfold$

Negative rules C_s are being added to the new class E.

$$
\begin{aligned}
I &::= x:a \mid and(x:I, y:I) \mid or(x:I, y:I) \mid \backslash \\
N &::= x:a \mid and(x:N, y:N) \mid or(x:N, y:N) \mid \backslash and(x:I, or(y:I, z:I)) \mid and(or(y:I, z:I), x:I) \\
&\quad \text{unfold applied: deleted} enable(N, and(x:N, y:N)) \\
I &::= x:a \mid and(x:I, y:I) \mid or(x:I, y:I) \mid \backslash \\
N &::= x:a \mid and(x:N, y:E) \mid or(x:N, y:N) \mid \backslash and(or(y:I, z:I), x:I) \\
and(x:N, y:E) &::= and(x:N, x:a) \mid and(x:N, and(x:N, y:E)) \mid and(x:N, or(x:N, y:N)) \mid \backslash and(x:I, or(y:I, z:I)) \\
&\quad \text{subsume applied: deleted} enable(and(x:N, y:E), and(x:N, or(x:N, y:N)))[by: and(x:I, or(y:I, z:I))] \\
I &::= x:a \mid and(x:I, y:I) \mid or(x:I, y:I) \\
N &::= x:a \mid and(x:N, y:E) \mid or(x:N, y:N) \mid \backslash and(or(y:I, z:I), x:I) \\
and(x:N, y:E) &::= and(x:N, x:a) \mid and(x:N, and(x:N, y:E)) \mid \backslash and(x:I, or(y:I, z:I)) \\
&\quad \text{redun\_r applied: deleted} disable(and(x:N, y:E), and(x:I, or(y:I, z:I))) \\
I &::= x:a \mid and(x:I, y:I) \mid or(x:I, y:I) \\
N &::= x:a \mid and(x:N, y:E) \mid or(x:N, y:N) \mid \backslash and(or(y:I, z:I), x:I) \\
and(x:N, y:E) &::= and(x:N, x:a) \mid and(x:N, and(x:N, y:E)) \\
&\quad \text{cancel applied to:} enable(and(x:N, y:E), and(x:N, x:a)) \\
&\quad \text{cancel applied to:} enable(and(x:N, y:E), and(x:N, and(x:N, y:E))) \\
I &::= x:a \mid and(x:I, y:I) \mid or(x:I, y:I) \\
N &::= x:a \mid x:N \mid and(x:N, y:E) \mid or(x:N, y:N) \mid \backslash and(or(y:I, z:I), x:I) \\
E &::= x:a \mid and(x:N, y:E) \mid \backslash and(or(y:I, z:I), x:I)
\end{aligned}
$$

trivial applied: deleted$enable(N, x : N)$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : N, y : E) \mid or(x : N, y : N) \mid \backslash and(or(y : I, z : I), x : I)$

$E ::= x : a \mid and(x : N, y : E) \mid \backslash and(or(y : I, z : I), x : I)$

unfold applied: deleted$enable(N, and(x : N, y : E))$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : E1, y : E) \mid or(x : N, y : N)$

$E ::= x : a \mid and(x : N, y : E) \mid \backslash and(or(y : I, z : I), x : I)$

$and(x : E1, y : E) ::= and(x : a, y : E) \mid and(and(x : E1, y : E), y : E) \mid and(or(x : N, y : N), y : E) \mid \backslash and(or(y : I, z : I), x : I)$

subsume applied: deleted$enable(and(x : E1, y : E), and(or(x : N, y : N), y : E))[by : and(or(y : I, z : I), x : I)]$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : E1, y : E) \mid or(x : N, y : N)$

$E ::= x : a \mid and(x : N, y : E) \mid \backslash and(or(y : I, z : I), x : I)$

$and(x : E1, y : E) ::= and(x : a, y : E) \mid and(and(x : E1, y : E), y : E) \mid \backslash and(or(y : I, z : I), x : I)$

redun_r applied: deleted$disable(and(x : E1, y : E), and(or(y : I, z : I), x : I))$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : E1, y : E) \mid or(x : N, y : N)$

$E ::= x : a \mid and(x : N, y : E) \mid \backslash and(or(y : I, z : I), x : I)$

$and(x : E1, y : E) ::= and(x : a, y : E) \mid and(and(x : E1, y : E), y : E)$

cancel applied to:$enable(and(x : E1, y : E), and(x : a, y : E))$

cancel applied to:$enable(and(x : E1, y : E), and(and(x : E1, y : E), y : E))$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : E1, y : E) \mid or(x : N, y : N)$

$E ::= x : a \mid y : E \mid and(x : N, y : E) \mid \backslash and(or(y : I, z : I), x : I)$

$E1 ::= x : a \mid and(x : E1, y : E)$

trivial applied: deleted$enable(E, y : E)$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : E1, y : E) \mid or(x : N, y : N)$

$E ::= x : a \mid and(x : N, y : E) \mid \backslash and(or(y : I, z : I), x : I)$

$E1 ::= x : a \mid and(x : E1, y : E)$

unfold applied: deleted$enable(E, and(x : N, y : E))$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : E1, y : E) \mid or(x : N, y : N)$

$E ::= x : a \mid and(x : E2, y : E)$

$E1 ::= x : a \mid and(x : E1, y : E)$

$and(x : E2, y : E) ::= and(x : a, y : E) \mid and(and(x : E2, y : E), y : E) \mid \backslash and(or(y : I, z : I), x : I)$

redun_r applied: deleted$disable(and(x : E2, y : E), and(or(y : I, z : I), x : I))$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : E1, y : E) \mid or(x : N, y : N)$

$E ::= x : a \mid and(x : E2, y : E)$

$E1 ::= x : a \mid and(x : E1, y : E)$

$and(x : E2, y : E) ::= and(x : a, y : E) \mid and(and(x : E2, y : E), y : E)$

cancel applied to:$enable(and(x : E2, y : E), and(x : a, y : E))$

cancel applied to:$enable(and(x : E2, y : E), and(and(x : E2, y : E), y : E))$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : E1, y : E) \mid or(x : N, y : N)$

$E ::= x : a \mid y : E \mid and(x : E2, y : E)$

$E1 ::= x : a \mid and(x : E1, y : E)$

$E2 ::= x : a \mid and(x : E2, y : E)$

trivial applied: deleted$enable(E, y : E)$

$I ::= x : a \mid and(x : I, y : I) \mid or(x : I, y : I)$

$N ::= x : a \mid and(x : E1, y : E) \mid or(x : N, y : N)$

$E ::= x : a \mid and(x : E2, y : E)$

$E1 ::= x : a \mid and(x : E1, y : E)$

$E2 ::= x : a \mid and(x : E2, y : E)$

F Example with strogner $Unfold$

We allow several C to be used in *Unfold*.

$$
\begin{array}{rcl}
I & ::= & x:a \mid and(x:I, y:I) \mid neg(x:I) \mid or(x:I, y:I) \\
N & ::= & x:a \mid and(x:N, y:N) \mid neg(x:N) \mid or(x:N, y:N) \mid \backslash neg(and(x:I, y:I)) \mid neg(or(x:I, y:I)) \\
 & & \text{unfold applied: deleted} enable(N, neg(x:N)) \\
I & :: & x:a \mid and(x:I, y:I) \mid neg(x:I) \mid or(x:I, y:I) \\
N & :: & x:a \mid and(x:N, y:N) \mid neg(x:E) \mid or(x:N, y:N) \\
neg(x:E) & :: & neg(x:a) \mid neg(and(x:N, y:N)) \mid neg(neg(x:E)) \mid neg(or(x:N, y:N)) \mid \backslash neg(and(x:I, y:I)) \mid neg(or(x:I, y:I \\
 & & \text{subsume applied: deleted} enable(neg(x:E), neg(and(x:N, y:N)))[by : neg(and(x:I, y:I))] \\
I & :: & x:a \mid and(x:I, y:I) \mid neg(x:I) \mid or(x:I, y:I) \\
N & :: & x:a \mid and(x:N, y:N) \mid neg(x:E) \mid or(x:N, y:N) \\
neg(x:E) & :: & neg(x:a) \mid neg(neg(x:E)) \mid neg(or(x:N, y:N)) \mid \backslash neg(and(x:I, y:I)) \mid neg(or(x:I, y:I)) \\
 & & \text{subsume applied: deleted} enable(neg(x:E), neg(or(x:N, y:N)))[by : neg(or(x:I, y:I))] \\
I & :: & x:a \mid and(x:I, y:I) \mid neg(x:I) \mid or(x:I, y:I) \\
N & :: & x:a \mid and(x:N, y:N) \mid neg(x:E) \mid or(x:N, y:N) \\
neg(x:E) & :: & neg(x:a) \mid neg(neg(x:E)) \mid \backslash neg(and(x:I, y:I)) \mid neg(or(x:I, y:I)) \\
 & & \text{redun\_r applied: deleted} disable(neg(x:E), neg(and(x:I, y:I))) \\
I & :: & x:a \mid and(x:I, y:I) \mid neg(x:I) \mid or(x:I, y:I) \\
N & :: & x:a \mid and(x:N, y:N) \mid neg(x:E) \mid or(x:N, y:N) \\
neg(x:E) & :: & neg(x:a) \mid neg(neg(x:E)) \mid \backslash neg(or(x:I, y:I)) \\
 & & \text{redun\_r applied: deleted} disable(neg(x:E), neg(or(x:I, y:I))) \\
I & :: & x:a \mid and(x:I, y:I) \mid neg(x:I) \mid or(x:I, y:I) \\
N & :: & x:a \mid and(x:N, y:N) \mid neg(x:E) \mid or(x:N, y:N) \\
neg(x:E) & :: & neg(x:a) \mid neg(neg(x:E)) \\
 & & \text{cancel applied to:} enable(neg(x:E), neg(x:a)) \\
 & & \text{cancel applied to:} enable(neg(x:E), neg(neg(x:E))) \\
I & :: & x:a \mid and(x:I, y:I) \mid neg(x:I) \mid or(x:I, y:I) \\
N & :: & x:a \mid and(x:N, y:N) \mid neg(x:E) \mid or(x:N, y:N) \\
E & :: & x:a \mid neg(x:E)
\end{array}
$$