On Predicting a Grammar of a Normal Form

Work in progress

Predrag Janičić, Univ of Belgrade

Joint work with Alan Bundy and Alan Smaill

International workshop *Proof, Computation, Complexity*

June 17th - 19th, 2004, Dresden

Problem: Normal Form Prediction

- Let T be a context-free class of expressions, defined by context-free grammar (CFG).
- Let $\mathcal{R}$ be a set of rewrite rules.
- Let L be a set of normal-forms when $\mathcal{R}$ applied to $\mathcal{L}(T)$.
- What is a CFG grammar of L?

Motivation

- Interested in automatic decision procedure (DP) generation.
- Many decision procedures are sequences of normalisations.
- Synthesise using input/output CFGs
- Hence, need to predict output CFG from input CFG.
- Synthesising of DPs understandable to humans.

DP for ground arithmetic

```
...
5 remove neq; Target class: f;
   Rules:[[f, neq(t, t), rm_neq]];
6 remove >; Target class: f;
   Rules:[[f, t>t, rm_gr]];
7 remove -; Target class: t;
   Rules:[[t, -t, rm_minus]];
8 stratify [#, \]; Target class: f;
   Rules:[[f, ~(f#f), st_neg_conj], [f, ~(f\f), st_neg_disj]];
9 thin ~; Target class: f1;
   Rules:[[f1, ~(~f1), thin_neg]];
10 remove ~; Target class: f;
   Rules:[[f, ~false, rm_bottom], [f, ~true, rm_top],
          [f, ~t<t, rm_neg_less], [f, ~t=t, rm_neg_eq]];
...
```

What we already have

- We have several normalizer generators. Each of them can produce specific normalizers (e.g., *remove*).

- Each of the generators takes the input CFG and then looks for rewrite rules that can transform any element of the input class into an element of an output class of a prescribed form.

- Only 5 normalizer generators were sufficient for completely automatic synthesis of a DP for ground arithmetic.

- Now look for more general approach: process automatically all available rewrite rules.

Example CFG Prediction Problems

Input CFG	**Rewrite Rules**	**Normal-Form** CFG
$T \to A \mid f(T) \mid g(T)$	$f(x) \longrightarrow g(x)$	$T' \to A \mid g(T')$
$T \to A \mid f(T) \mid g(T)$	$f(f(x)) \longrightarrow g(x)$	$T' \to A \mid f(A) \mid f(g(T')) \mid g(T')$
$T \to A \mid g(f(T)) \mid g(g(T))$	$f(x) \longrightarrow g(x)$	$T' \to A \mid g(g(T'))$
$T \to a \mid b \mid f(a, T)$	$f(x, y) \longrightarrow g(x, y)$	$T' \to a \mid b \mid g(a, T')$
$T \to A \mid f(T) \mid h(T, T)$	$f(h(x, y)) \longrightarrow h(f(x), f(y))$	$T' \to A \mid f(E) \mid h(T', T')$ $E \to A \mid f(E)$

Sometimes there is no required CFG

Let T is a CFG class defined in the following way:

$$T \;\rightarrow\; f(a, f(c, b)) \mid f(f(a, T), f(c, b))$$

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

$$r_1 : \; f(x, f(y, z)) \longrightarrow f(f(x, y), z)$$
$$r_2 : \; f(f(x, c), b) \longrightarrow f(f(x, b), c).$$

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 (by *pumping lemma*).

Sometimes the set of normal forms is not even recursive

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

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

R is recursive and, also, confluent and terminating.

$ab^m c$ is in normal form *iff* m is not in $\mathcal{C}$.

Since $\mathcal{C}$ is r.e. 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

Sometimes the set of normal forms is recursive

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.

Our procedure

- Complementary CCFG: $A \rightarrow |_{j \in v} D_j \setminus |_{i \in u} C_i$.

- Language of CCFG: $\mathcal{L}^C(A) = \mathcal{L}(A^+) \setminus \mathcal{L}(A^-)$

  where $A^+ \rightarrow |_{j \in v} D_j \quad A^- \rightarrow |_{i \in u} C_i$

- two phases: initialisation Phase (create CCFG for normal form) and Reduction Phase: (reduce CCFG to CFG).

Example for CCFG

Consider the following CCFG:

$$A \;\rightarrow\; a \mid f(A) \setminus f(f(A))$$

The above definition determines the following pair of CFG classes:

$$\begin{aligned} A &\rightarrow a \mid f(A) \\ A^{-} &\rightarrow f(f(A)). \end{aligned}$$

Thus,

$$\begin{aligned} \mathcal{L}(A) &= \left\{ f^i(a) \mid i = 0, 1, 2, \ldots \right\} \\ \mathcal{L}(A^{-}) &= \left\{ f^i(a) \mid i = 2, 3, 4, \ldots \right\} \\ \mathcal{L}^C(A) &= \mathcal{L}(A) \setminus \mathcal{L}(A^{-}) = \left\{ f^i(a) \mid i = 0, 1 \right\} \end{aligned}$$

Initialisation Procedure

Preconditions	T	**Addition to** T
$\exists l \longrightarrow r \in \mathcal{R}.\ \exists \theta.\ l\theta \in \mathcal{L}^*(T)$ $\neg\exists L' \longrightarrow R' \in \mathcal{R}^p.\exists \theta'.\ (L' \longrightarrow R')\theta' \equiv l\theta \longrightarrow r\theta$	$T \rightarrow Ds$	$l\theta \longrightarrow r\theta$
$\exists l \longrightarrow r \in \mathcal{R}.\ \exists \theta.\ l\theta \in \mathcal{L}^*(e)$ $D[r\theta/e] \notin \mathcal{L}^*(T)$	$T \rightarrow D[e] \mid Ds$	stop with failu
$\exists l \longrightarrow r \in \mathcal{R}.\ \exists \theta.\ l\theta \in \mathcal{L}^*(e)$ $D[r\theta/e] \in \mathcal{L}^*(T)$ $\neg\exists L' \longrightarrow R' \in \mathcal{R}^p.\exists \theta'.\ (L' \longrightarrow R')\theta' \equiv D[l\theta/e] \longrightarrow D[r\theta/e]$	$T \rightarrow D[e] \mid Ds$	$D[l\theta/e] \longrightarrow D[$

Initialisation Examples

T	$\mathcal{R}$	$\mathcal{R}^p(T)$
$T \rightarrow a \mid f(T_1)$	$\{g(x) \longrightarrow h(x)\}$	$\{\}$
$T \rightarrow a \mid b \mid f(a, T_1)$	$\{f(x, y) \longrightarrow g(x, y)\}$	$\{f(a, T_1) \longrightarrow g(a, T_1)\}$
$T \rightarrow A \mid f(T_1) \mid h(T_2, T_3)$	$\{f(h(x, y)) \longrightarrow h(f(x), f(y))\}$	$\{f(h(T_1, T_2)) \longrightarrow h(f(T_1), f(T_2$
$T \rightarrow a \mid f(g(T))$	$g(f(x)) \longrightarrow p(x)$	$failure$

Reduction Procedure

Name	**Precondition**	**Input** CCFG	**Output CCFG**
Simpl-unused	redundant symbol T	$T \to \ldots$	deleted T
Simpl-id		$T \to (T \mid Ds) \setminus Cs$	$T \to Ds \setminus Cs$
Simpl-empty		$T \to F_1(\ldots, T, \ldots) \mid \ldots$ $\mid F_n(\ldots, T, \ldots) \setminus \ldots$	deleted T
Simpl-equiv	$E_1 \equiv E_2$		$[E_2 \mapsto E_1]$, deleted E_2
Redun-Pp	$\mathcal{L}(D) \subset \mathcal{L}(Ds)$	$T \to D \mid Ds \setminus \ldots$	$T \to Ds \setminus \ldots$
Redun-Nn	$\mathcal{L}(C) \subset \mathcal{L}(Cs)$	$T \to \ldots \setminus C \mid Cs$	$T \to \ldots \setminus Cs$
Redun-Np	$\mathcal{L}(D) \subset \mathcal{L}(Cs)$	$T \to (D \mid Ds) \setminus Cs$	$T \to Ds \setminus Cs$
Redun-Pn	$\mathcal{L}(C) \cap \mathcal{L}(Ds) = \emptyset$	$T \to Ds \setminus (C \mid Cs)$	$T \to Ds \setminus Cs$

$+$ the $Unfold$ rule (next slide).

Unfold Rule

Input CCFG: $T \to D[S_1, \ldots, S_k] \mid Ds \setminus (D[S'_1, \ldots, S'_k] \mid Cs)$

Preconditions:

- $(D[S'_1, \ldots, S'_k] \mid Cs) \not\Rightarrow^* D[S_1, \ldots, S_k]$
- $S_i \Rightarrow^* S'_i \ \ (i = 1, \ldots, k)$
- $\mathcal{L}(Ds) \cap \mathcal{L}(D[S'_1, \ldots, S'_k]) = \emptyset$

Output CCFG:

$T \to D[E^{(1)}, S_2, \ldots, S_k] \mid \ldots \mid D[S_1, \ldots, S_{k-1}, E^{(k)}] \mid Ds \setminus Cs$

$E^{(i)} \to$ positive rules for $S_i \setminus$ negative rules for $S_i \mid S'_i$

Unfold Example

Input CCFG: $N \rightarrow A|f(N)|h(N,N) \setminus h(f(N),f(N))$

Preconditions:

- $h(f(N),f(N)) \not\Rightarrow^* h(N,N)$
- $N \Rightarrow^* f(N)$.
- $\mathcal{L}(A|f(N)) \cap \mathcal{L}(h(f(N),f(N))) = \emptyset$

Output CCFG:

$$
\begin{array}{rcl}
N & \rightarrow & A|f(N)|h(E,N)|h(N,E) \\
E & \rightarrow & A|f(N)|h(E,N)|h(N,E) \setminus f(N) \\
 & & \text{...and, then (after other rules applied)...} \\
E & \rightarrow & A|h(E,N)|h(N,E)
\end{array}
$$

Properties of the procedure

Soundness: After each step after initialisation phase, the language of the current grammar is equal to the language of normal forms.

Completeness: False and unobtainable. Maybe obtainable in a weaker sense ("...whenever a result exists...")

Termination: The procedure can loop. A care needs to be taken about application of rules. Still to be considered. False for current procedure.

Implementation: Prototype implementation in PROLOG

Restrictions: Only linear rules

Some results

Input CFG	**Rewrite Rules**	**Normal-Form** CFG
$T \to a \mid f(T,T)$	$f(x,a) \longrightarrow a$	$N \to a \mid f(N,E)$ $E \to f(N,E)$
$T \to A \mid f(T,T)$	$f(x,f(y,z)) \longrightarrow f(f(x,y),z)$	$N \to A \mid f(N,A)$
$T \to A \mid T \wedge T \mid T \vee T \mid \neg T$	$\neg(x \wedge y) \longrightarrow \neg x \vee \neg y$ $\neg(x \vee y) \longrightarrow \neg x \wedge \neg y$	$N \to A \mid N \wedge N \mid N \vee N \mid \neg E$ $E \to A \mid \neg E$
$T \to A \mid \neg T$	$\neg(\neg x) \longrightarrow x$	$N := A \mid \neg A$

Conclusions

- Still under construction normal-form prediction procedure.
- Explore completeness/termination trade-off.
- Apply to decision procedure generation.
- Explore and compare to other results.