Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 887–890

ISBN 978-83-60810-22-4

Checking the Conformance of Grammar Refinements with respect to Initial Context-Free Grammars

Bryan Temprado-Battad, Antonio Sarasa-Cabezuelo, José-Luis Sierra
Facultad de Informática. Universidad Complutense de Madrid. 28040 Madrid, Spain
Email: {bryan, asarasa, jlsierra}@fdi.ucm.es

***Abstract*—According to this paper, to refine an *initial* context-free grammar supposes to devise an equivalent grammar that preserves the main syntactic structures of the initial one while making explicit other structural characteristics (e.g., associativity and priority of the operators in an expression language). Although, generally speaking, checking the equivalence of two context-free grammars is an undecidable problem, in the scenario of grammar refinement it is possible to exploit the relationships between the initial grammar and the grammar refinement to run a heuristic conformance test. These relationships must be made explicit by associating *core* non-terminal symbols in the initial grammar with *core* non-terminal symbols in the grammar refinement. Once it is made, it is possible to base the heuristic test on searching regular expressions involving both terminal and *core* non-terminal symbols that describe each *core* non-terminal symbol, and on checking the equivalence of carefully chosen pairs of such regular expressions. The paper describes the method and illustrates it with an example.**

I. Introduction

FROM a language engineering perspective, the necessity to ensure the correctness of the successive refinements of a context-free grammar becomes apparent. Unfortunately, in the last term checking this correctness supposes to check the weak equivalence of two arbitrary context-free grammars[1], a classical undecidable problem [3][2]. As a consequence, it is not possible to devise a *complete* checking algorithm, which can work in all the cases. However, since we are not facing arbitrary context-free grammars, but grammars related by a refinement relation, we still can propose some heuristic methods working reasonably well in many practical situations. This paper describes one of such methods.

II. The Refinement Work-Flow

In order to systematize the process of context-free grammars refinement we propose the work-flow of Fig. 1.

The work-flow begins with the *providing the initial grammar* activity. The goal of this activity is to get an initial or base grammar to refine.

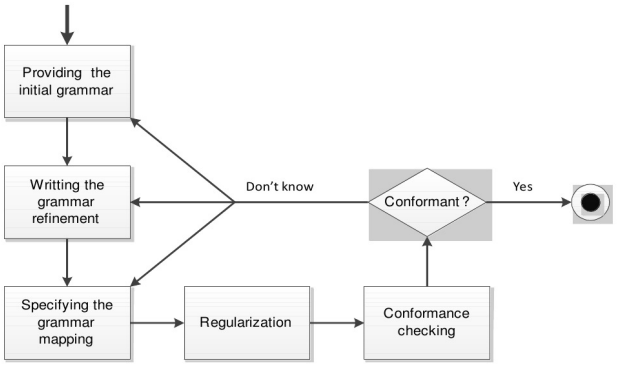

Fig 1. Refinement work-flow

Next activity is *writing the grammar refinement*. This activity is oriented to refine the initial grammar by reflecting structural features and properties not present in such an initial grammar (e.g., associativity and precedence of operators). In this way, the initial grammar should introduce the *core* syntactic constructs of the language, in form of *core* non-terminal symbols, while the refinement should address the structural refinement of these constructs, without changing the generated language.

Once the grammar refinement has been written, next activity is the *specifying the grammar mapping* activity. The goal of this activity is to make explicit a *grammar mapping* identifying which non-terminal symbols in the refinement correspond with each relevant (core) non-terminal symbol in the initial grammar.

Next step is *regularization*. This activity, which is carried out automatically, actually is the first part of the conformance checking method. Its goal is to associate, with each core non-terminal symbol, a *definitional regular expression.* Definitional regular expressions must involve both terminal and core non-terminal symbols. In addition, each sentence in the language described by the definitional regular expression ε for a core non-terminal A must be derivable from A (i.e., $\alpha \in L(\varepsilon) \Rightarrow A \rightarrow^* \alpha$). Finally, ε must completely characterize

[1] i.e., to check whether the two grammars generate the same language.

[2] Although structural equivalence of context-free grammars (i.e., to check whether two context-free grammars produces structurally equivalent parse trees) is decidable [4], grammar refinements usually do not preserve the structure of the parse trees. Thus, checking this restricted form of structural equivalence is not sufficient in this context.

one of the intermediary languages derived from *A*, in the following sense: $A \rightarrow^* w \Rightarrow \exists\, \alpha \in L(\varepsilon)\ (\alpha \rightarrow^* w)$[3].

Finally, the last step is given by the *conformance checking* activity. This activity tries to actually check the weak equivalence on the basis of the definitional regular expressions associated with each core non-terminal in both grammars, as well as on the basis of the grammar mapping. Therefore, it constitutes the second part of the conformance checking method.

It is worthwhile to notice that, being the checking method necessarily incomplete, during its execution it will be possible to get one of two possible answers: (i) *the grammars are actually equivalent* and (ii) *it has not been possible to prove whether the grammars are equivalent.* While in the first case, the method will ensure the correctness of the performed refinement, in the second case the answer is not conclusive: indeed, the grammars could be actually equivalent in a way not envisioned by the method, and therefore we can't conclude the non-equivalence of the grammar. In this case, it could be possible to re-factor the refinement, to modify simultaneously both the refinement as the initial grammar, and even to rethinking the grammar mapping, as the iterative nature of the work-flow makes apparent. Next sections go inside each activity of this work-flow.

III. Providing the Initial Grammar, Writing the Grammar Refinement and Specifying the Grammar Mapping

The first activity to do in the refinement work-flow is to provide a suitable *initial grammar*. As said before, this grammar formally characterizes the syntax of the computer language addressed (i.e., it is able to generate exactly the sentences of the language). However, it does not necessarily do it in a way which is convenient to undertake a systematic implementation of such a language. Therefore, it could be necessary to modify this initial grammar to yield a *grammar refinement*.

(a)

```
Sents ::= Sents ; Sent | Sent
Sent ::= id := Exp
Exp ::= Exp + Exp | Exp * Exp | id | (Exp)
```

(b)

```
SS ::= SS ; S | S
S ::= id := E
E ::= E + T | T
T := F * T | F
F ::= id | (E)
```

Fig 2. (a) An initial context-free grammar, (b) a refinement of (a)

In order to illustrate these aspects, let us consider the initial grammar shown in Fig. 2a. This grammar characterizes a simple language of assignment instructions, in which arithmetic expressions can be assigned to identifiers. However, it does not take care of characterizing precedence and associativity of operators. As a collateral consequence, it exhibits ambiguity. In order to solve these shortcomings, it is possible to *refine* the initial grammar, getting an equivalent grammar characterizing the mentioned priority and associativity of operators. Fig. 2b outlines a possible refinement taking these features into account.

In addition to provide the initial grammar and to specify the grammar refinement, in order to get the benefits of our checking approach, grammar writers must make the structural relationships between initial grammars and grammar refinements explicit. It is done by specifying a *grammar mapping*. On one hand, such a mapping supposes to recognize a set of representative syntactic structures in the grammar refinement that result of refining structures in the initial grammar. It is done by identifying a set of *core* non-terminal symbols in the grammar refinement. In particular, the initial symbol must be one of these core symbols. On the other hand, this mapping makes it possible to associate to each core non-terminal symbol in the grammar refinement a distinct non-terminal symbol in the initial grammar. Following a similar convention, these symbols in the initial grammar will be called *core* non-terminal symbols of the initial grammar. Non-terminal symbols that are not core symbols we will be called *auxiliary* non-terminal symbols.

Concerning our example, the establishment of a grammar mapping is straightforward. Indeed, in the refinement of Fig. 2b we can identify three core symbols (`SS`, `S` and `E`), which are mapped respectively to `Sents`, `Sent` and `Exp` in the initial grammar of Figure 2a.

IV. Regularization

As said before, the goal of the *regularization* activity is to associate with each core non-terminal a definitional regular expression. In addition, this expression must only comprise terminal symbols, and core non-terminal symbols. For doing so, the algorithm of Fig. 3 is used.

This algorithm successively visits each non-terminal symbol *A*, determining and refining a definitional regular expression ε_A for *A*. It starts by determining an arbitrary order for the non-terminal symbols. While the final regular expressions will depend on this order, the results will be equivalent for any order chosen. Then, it visits each non-terminal symbol *A*. For this purpose, it begins by establishing a first value for the definitional regular expression ε_A as $\oplus\{\alpha \mid A \rightarrow \alpha \in P\}$, with *P* the set of syntax rules. Here, by $\oplus\Gamma$ we denote $\varnothing$ when $\Gamma=\varnothing$, α when $\Gamma=\{\alpha\}$ and $\alpha_0 \mid \ldots \mid \alpha_k$ when $\Gamma=\{\alpha_0, \ldots,\alpha_k\}$ ($k \geq 1$). Therefore ε_A is initially set to the disjunction of the RHSs of the rules for *A*. Then it substitutes the expression ε_B for each already visited *auxiliary* non-terminal *B* in ε_A (it is denoted by $\varepsilon_A[B/\varepsilon_B]$). Notice that core non-terminals in ε_A are preserved, since this process only attempts to eliminate auxiliary non-terminals. Next, it simplifies ε_A to replace several forms of immediate recursion by iteration. For this purpose, a well-known result borrowed from the theory of language equations is used [2], which makes it possible to derive

$$A = (\beta^* \delta \gamma^* \alpha)^* \beta^* \delta \gamma^* \quad \textbf{(1)}$$

from

[3]Here, as usual, α denotes a sentential form –a string of terminal and non-terminal symbols-, and *w* a sentence –a string of terminal symbols.

$$A = A\alpha A \mid \beta A \mid A\gamma \mid \delta \quad \textbf{(2)}$$

Input: (i) A context-free grammar G ≡(S_N, S_T ,S,P) -S_N is the set of non terminal symbols, S_T the set of terminal symbols, S the initial symbol, and P the set of syntax rules; (ii) The set of *core* non-terminals *K*
Output: A set of equations, with an equation of the form *A*=ε for each core non-terminal *A*, with ε a definitional regular expression
Method:

```
let S_N = {A_0, A_1, ..., A_n} in
 for i = 0 to n
  ε_i := ⊕{α | A → α ∈ P }
  for j = 0 to i-1
    if A_j ∉ K then
      ε_i := ε_i [A_j / ε_j]
    end if
  end for
  ε_i := simplify(ε_i, normalize(ε_i))
  if A_i ∉ K then
   for j = 0 to i-1
     ε_j := ε_j [A_i / ε_i]
   end for
  end if
 end for
end let
return { A_i = ε_i | A_i ∈ K }
```

Fig 3. Regularization method.

(a)
Ordering: `Sents, Sent, Exp`
Regularization:

```
Sents = ( Sent; )* Sent
Sent  = id := Exp
Exp = ((id | (Exp) ) (+|*))*(id | (Exp) )
```

(b)
Ordering: `F,T,E,S,SS`
Regularization:

```
SS = S (; S)*
S = id := E
E = ((id | (E)) *)* (id | (E) )
   (+ ((id | (E)) *)* (id | (E)) )*
```

Fig 4. (a) Regularization of grammar in Fig. 2a; (b) Regularization of grammar in Fig. 2b.

This result is taken as as a basic transformation pattern to eliminate several forms of immediate recursion during the regularization stage. In order to make it possible to apply this pattern, we need to start by *normalize* definitional regular expressions ε_A for non-terminal symbols *A* in order to yield regular expressions of the form $A\alpha A \mid \beta A \mid A\gamma \mid \delta$ equivalent to ε_A. Such a normalization basically works by applying the identities

$$(\varepsilon_0|\varepsilon_1)\,\varepsilon_2 = \varepsilon_0\,\varepsilon_2 | \,\varepsilon_1\,\varepsilon_2 \quad \textbf{(3)}$$

$$\varepsilon_2\,(\varepsilon_0|\varepsilon_1) = \varepsilon_2\,\varepsilon_0 | \,\varepsilon_2\,\varepsilon_1 \quad \textbf{(4)}$$

to the begin and the end of the expression in order to push up the left and right recursive positions of *A*. Due to space constrains, these normalization details will be omitted here.

In order to exemplify regularization, Fig. 4 shows the results of the regularization activity when carried out on grammars of Fig. 2a and Fig. 2b.

V. Conformance Checking

The last activity to be considered is *conformance checking* itself, which is carried out by the method of Fig. 5.

The first step involved in the activity is to *align* the core non-terminal symbols in the two regularizations, in such a way those regularizations use the same names for those symbols. It is indicated by the *align* operation (details omitted). In addition to aligning the names of the core non-terminal, this operation is supposed to replace the auxiliary non-terminals that could remain in the definitional expressions by _ (_ is assumed to not be allowed as a grammar symbol). Once aligned both regularizations, the next step actually addresses the checking process. For this purpose, it maintains two sets: (i) Γ, which contains the core symbols whose conformance must be checked, and (ii) *V*, which contains the core symbols whose conformance has been already undertook (*visited* non-terminals). Then, the checking process proceeds until Γ becomes empty.

Input: (i)The regularization of the initial grammar R_l; (ii) The regularization of the refinement grammar R_r ; (iii) The terminal alphabet Σ_T ; (iv) The initial symbol *S* of the initial grammar; (v) The grammar mapping Θ from the refinement to the initial grammar
Output: "yes" if it the conformance can be proved, "don't know" otherwise
Method:

```
(R_l, R_r) := align(R_l, R_r, Θ, Σ_T)
Γ := {S}; V := ∅
while Γ ≠ ∅
  pick A from Γ
  Γ := Γ - {A}; V := V ∪ {A}
  pick A=ε^l_A from R_l,
  pick A=ε^r_A from R_r
 if _ ∈ ε^l_A ∨ _ ∈ ε^r_A then
    return "don't know"
  fi
  if ε^l_A ~ ε^r_A then
    Γ := Γ ∪ { B | B ∈ ε^l_A ∧ B∉ V ∪ Σ_T }
  else
     return "don't know"
  end if
end while
return "yes"
```

Fig 5. Conformance checking method

In each iteration, a core symbol *A* in Γ is chosen, it is recorded as visited, and the definitional expressions for *A* in the initial grammar (ε^I_A) and in the grammar refinement (ε^r_A) are considered (remember the aligned regularizations shares the names for the core non-terminals). If ε^I_A or ε^r_A contain a _ symbol, it means regularization failed to produce definitional expressions comprising only terminal and core non-terminal symbols. Thus, the checking process ends with a non-conclusive response. Otherwise, ε^I_A and ε^r_A are checked for equivalence (i.e., it is studied whether $\varepsilon^I_A \sim \varepsilon^r_A$ holds). If the test fails (i.e., ε^I_A and ε^r_A are not indeed equivalent), the overall process finishes with a non-conclusive answer. Otherwise, the method tries to ensure that, if $\alpha \in L(\varepsilon^I_A)$ (and, thus, $\alpha \in L(\varepsilon^r_A)$), then it is possible to derive exactly the same sentences from α in the initial grammar than in the grammar refinement. Indeed, if it holds for any core non-terminal in α, it will hold for the overall sentential form. For this purpose, the method schedules the checking of those core non-terminals in α not yet visited. Thus, if the set Γ is finally emptied, it is possible to ensure that all the *proof obligations* concern-

ing the core non-terminal symbols scheduled by the method have been satisfied, and, therefore, the equivalence has been effectively proven. In this way, it is possible to finish with a conclusive and positive answer.

Alignment of core non-terminal names	
Aligned regularization for the initial grammar	

`Sents = ( Sent; )* Sent`
`Sent = id := Exp`
`Exp = ((id | (Exp) ) (+|*))*(id | (Exp) )`	

Aligned regularization for the grammar refinement	

`Sents = Sent (; Sent)*`
`Sent = id := Exp`
`Exp = ((id | (Exp) ) *)* (id | (Exp) )`
`(+ ((id | (Exp) ) *)* (id | (Exp) ) )*`	

Conformance checking	
Init	**Γ** = {Sents} **V =** ∅
It. 1	**Symbol to check:** Sents **Γ** = ∅ **V =** {Sents} **¿** `( Sent; )* Sent ~ Sent (; Sent)*` **?: YES** **Γ** = {Sent}
It. 2	**Symbol to check:** Sent **Γ** = ∅ **V =** {Sents,Sent} **¿** `id := Exp ~ id := Exp` **? : YES** **Γ** = {Exp}
It. 3	**Symbol to check:** Exp **Γ** = ∅ **V =** {Sents,Sent,Exp} **¿** `((id | (Exp) ) (+|*))* (id | (Exp) ) ~` `((id | (Exp) ) *)* (id | (Exp) )` `(+ ((id | (Exp) ) *)* (id | (Exp) ) )*` **? : YES**
End	Success (YES answer) !

Fig 6. Checking the conformance of grammar in Fig. 2b with respect to grammar in Fig. 2a

Finally, notice that the conformance checking method relies on checking the equivalence of regular expressions. For this purpose, it is possible to use one of the well-known approaches reported in the literature (see, for instance, [1]), which, in last term, rely on converting regular expressions to equivalent finite automata, and to check equivalence between automata.

Fig. 6 details the application of the conformance checking method to the grammar refinement of Fig. 2b and the initial grammar of Fig. 2a.

VI. Conclusions and Future Work

Grammar refinement is a usual activity in language engineering. Initial context-free grammars are refined to impose finer structures on the initial syntactic categories, in order to make explicit important features of the target language (e.g., operator associativity and precedence). Grammars are also refined to get equivalents satisfying the constraints imposed by particular development tools (e.g., parser generators). Therefore, checking the correctness of refinements with respect to initial grammars should be a must in any systematic language engineering process. Although the unconstrained problem is undecidable, this paper has shown how it is still possible to provide some useful automatic assistance. For this purpose, it has proposed an interactive approach, focused on checking equivalence of definitional regular expressions for core non-terminals

We are currently improving the efficiency of the different algorithms involved in our proposal. We are also investigating the inclusion of new regularization patterns and strategies. In addition, we want to check the approach with several grammars for non-trivial domain-specific languages. As future work, we want to use this approach to check the conformance of processing-oriented grammars with respect to XML schemas in order to assist the language-oriented processing of XML documents [5].

Acknowledgment

Thanks are due to the project grants TIN2010-21288-C02-01.

References

[1] Aho A.V, Ullman J.D. The Theory of Parsing, Translation and Compiling. Vol. I - Parsing. Prentice-Hall. 1972

[2] Andrei, S., Chind, W-N., Cavadini, S.V. Self-embedded context-free grammars with regular counterparts. Acta Informatica 40(5): 349–365. 2004

[3] Bar-Hillel, Y., Perles, M., Shamir, E. On formal properties of simple phrase-structure grammars. Z. Phonetik, Sprachwiss. Kommunikationsforsch 14, 143-172. 1961 (Reprinted in Bar-Hillel. Language and Information, Addison-Wesley, 1964)

[4] Paull, M.C., Unger, S.H. Structural Equivalence of context-free grammars. Journal of Comp. and System Sc., 2(4), 427-463. 1968

[5] Temprado-Battad, B., Sarasa, A., Sierra, J.L. Modular Specifications of XML Processing Tasks with Attribute Grammars Defined on Multiple Syntactic Views. 5th International Workshop on Flexible Database and Information Systems. Bilbao, Spain. 2010