Workshop TAG+5, Paris, 25-27 May 2000

Practical aspects in compiling tabular TAG parsers

Miguel A. Alonso[†], Djamé Seddah[‡], and Éric Villemonte de la Clergerie[⋆]

[†]Departamento de Computación, Universidad de La Coruña
Campus de Elviña s/n, 15071 La Coruña (Spain)
alonso@dc.fi.udc.es
[‡]LORIA, Technopôle de Nancy Brabois,
615, Rue du Jardin Botanique - B.P. 101, 54602 Villers les Nancy (France)
Djame.Seddah@loria.fr
[⋆]INRIA, Domaine de Voluceau
Rocquencourt, B.P. 105, 78153 Le Chesnay (France)
Eric.De_La_Clergerie@inria.fr

Abstract

This paper describes the extension of the system DyALog to compile tabular parsers from Feature Tree Adjoining Grammars. The compilation process uses intermediary 2-stack automata to encode various parsing strategies and a dynamic programming interpretation to break automata derivations into tabulable fragments.

1. Introduction

This paper describes the extension of the system DyALog in order to produce tabular parsers for Tree Adjoining Grammars [TAGs] and focuses on some practical aspects encountered during the process. By tabulation, we mean that traces of (sub)computations, called *items*, are tabulated in order to provide computation sharing and loop detection (as done in Chart Parsers).

The system DyALog[1] handles logic programs and grammars (DCG). It has two main components, namely an abstract machine that implements a generic fix-point algorithm with subsumption checking on objects, and a bootstrapped compiler. The compilation process first compiles a grammar into a Push-Down Automaton [PDA] that encodes the steps of a parsing strategy. PDAs are then evaluated using a Dynamic Programming [DP] interpretation that specifies how to break the PDA derivations into elementary tabulable fragments, how to represent, in an optimal way, these fragments by *items*, and how to combine items and transitions to retrieve all PDA derivations. Following this DP interpretation, the transitions of the PDAs are analyzed at compile time to emit application code as well as to build code for the skeletons of items and transitions that may be needed at run-time.

Recently, (Villemonte de la Clergerie & Alonso Pardo, 1998) has presented a variant of 2-stack automata [2SA] and presented a DP interpretation for them. These 2SAs allow the encoding of many parsing strategies for TAGs, ranging from pure bottom-up ones to valid-prefix top-down ones. For all strategies, the DP interpretation ensures worst-case complexities in time $O(n^6)$ and space $O(n^5)$, where n denotes the length of the input string.

This theoretical work has been implemented in DyALog with minimum effort. Only a few files have been added to the DyALog compiler and no modification was necessary in the DyALog machine. Several extensions and optimizations were added: handling of Feature TAGs,

[1]Freely available at http://atoll.inria.fr/~clerger

use of more sophisticated parsing strategies, use of meta-transitions to compact sequences of transitions, use of more efficient items, and possibility to escape to logic predicates.

2. Tree Adjoining Grammars

We assume the reader to be familiar with TAGs (Joshi, 1987) and with the basic notions in Logic Programming (substitution, unification, subsumption, . . .). Let us just recall that Feature TAGs are TAGs where a pair of first-order arguments `top` T_ν and `bottom` B_ν may be attached to each node ν labeled by a non-terminal.

We have chosen a Prolog-like linear representation of trees. For instance, the grammar `count` (Fig. 1) recognizes the language $\mathtt{a}^n\mathtt{b}^n\mathtt{e}\mathtt{c}^n\mathtt{d}^n$ with $n > 0$ and returns the number n of performed adjunctions. It corresponds (omitting the top and bottom arguments) to the trees on the right side. By default, the nodes are adjoinable, except when they are leaves or are prefixed with $-$. Obligatory Adjunction [OA] nodes are prefixed with ++ and foot nodes by $*$. Node arguments are introduced with the operators **at**, **and**, **top**, and **bot** and escapes to Prolog are enclosed with {} (as done in DCGs).

```
tree top=s(X) and bot=s(0) at ++s("e").
auxtree top=s(XpI)
      at -s("a",
             top=s(X) and bot=s(Y)
          at s("b", bot=s(Y) at *s, "c"),
            {XpI is X+1},
            "d").
```

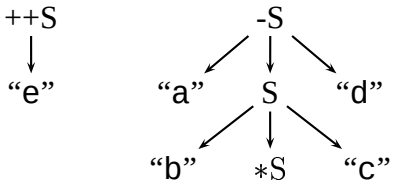

Figure 1: Concrete representation of grammar `count` and corresponding trees

3. Compiling into 2SAs following a modulated Call/Return Strategy

2SAs (Becker, 1994) are extensions of PDAs working on a pair of stacks and having the power of a Turing Machines. We restrict them by considering asymmetric stacks, one being the Master Stack **MS** where most of the work is done and the other being the Auxiliary Stack **AS** only used for restricted "bookkeeping" (Villemonte de la Clergerie & Alonso Pardo, 1998). When parsing TAGs, **MS** is used to save information about the different elementary tree traversals that are under way while **AS** saves information about adjunctions, as suggested in figure 2.

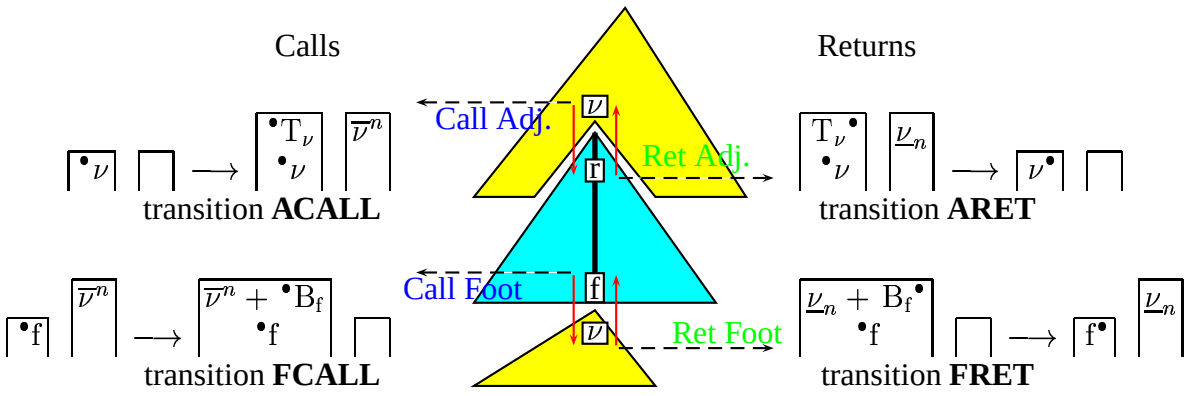

Figure 2: Illustration of some steps

Figure 2 also illustrates the notion of *modulated Call/Return strategy*: an elementary tree α is traversed in a pre-order way (skipping Null Adjunction [NA] internal nodes) and when predicting an adjunction on node ν, the traversal is suspended and some *prediction information*

${}^{\bullet}\mathrm{T}_{\nu}$ relative to the top argument T_{ν} of ν is pushed on MS (step **Call**) and used to select some auxiliary tree β (step **Select**). Some information $\overline{\nu}^{n}$ (partially) identifying ν is also pushed on $\mathbf{AS}$ and propagated to the foot of the auxiliary tree. Then $\overline{\nu}^{n}$ is popped, combined with some information ${}^{\bullet}\mathrm{B}_{\mathrm{f}}$ and pushed on $\mathbf{MS}$ in order to select the traversal of the subtree α_{ν} rooted at ν. Once α_{ν} has been traversed, we pop $\mathbf{MS}$ and resume the suspended traversal of β. We also push propagation information $\underline{\nu}_{n}$ on $\mathbf{AS}$ about the adjunction node. Once β has been traversed, we publish some propagation information about β (step **Publish**). We then pop both $\mathbf{MS}$ and $\mathbf{AS}$ and resume the suspended traversal of α, checking with $\underline{\nu}_{n}$ and $\mathrm{T}_{\nu}{}^{\bullet}$ that the adjunction has been correctly handled (step **Return**).

For each kind of suspension that may occur during a tree traversal, we get a pair of Call/Return transitions and a related pair of Select/Publish transitions. For TAGs, we have three kinds of suspension, occurring at substitution, adjunction, and foot nodes. We explicit here the transitions relative to an adjunction on node ν and to an auxiliary tree of root r and foot f. A transition τ of the form $(\Xi, \xi) \longmapsto (\Theta, \theta)$ applies on any *configuration* $(\mathbf{MS}, \mathbf{AS}) = (\Psi\Xi', \psi\xi')$ and returns $((\Psi\Theta)\sigma, (\psi\theta)\sigma)$ whenever the most general unifier $\sigma = \mathrm{mgu}(\Xi\xi, \Xi'\xi')$ exists. [2]

ACALL	$({}^{\bullet}\nu, \epsilon) \longmapsto ({}^{\bullet}\nu\ {}^{\bullet}\mathrm{T}_{\nu}, \overline{\nu}^{n})$	**ARET**	$({}^{\bullet}\nu\ \mathrm{T}_{\nu}{}^{\bullet}, \underline{\nu}_{n}) \longmapsto (\nu^{\bullet}, \epsilon)$
ASEL	$({}^{\bullet}\mathrm{T}_{\mathrm{r}}, \epsilon) \longmapsto ({}^{\bullet}\mathrm{r}, \epsilon)$	**APUB**	$(\mathrm{r}^{\bullet}, \epsilon) \longmapsto (\mathrm{T}_{\mathrm{r}}{}^{\bullet}, \epsilon)$
FCALL	$({}_{\bullet}\mathrm{f}, X) \longmapsto ({}_{\bullet}\mathrm{f}\ ({}_{\bullet}\mathrm{B}_{\mathrm{f}} + X), \epsilon)$	**FRET**	$({}_{\bullet}\mathrm{f}\ (\mathrm{B}_{\mathrm{f}\bullet} + Y), \epsilon) \longmapsto (\mathrm{f}_{\bullet}, Y)$
FSEL	$(({}_{\bullet}\mathrm{B}_{\nu} + \overline{\nu}^{n}), \epsilon) \longmapsto ({}_{\bullet}\nu, \epsilon)$	**FPUB**	$({}_{\nu\bullet}, \epsilon) \longmapsto ((\mathrm{B}_{\nu\bullet} + \underline{\nu}_{n}), \epsilon)$

In these transitions, X, Y denote free variables and *dotted atoms* ${}^{\bullet}\nu$, $\nu^{\bullet}$ (resp. ${}_{\bullet}\nu$, $\nu_{\bullet}$) denote computation points during a traversal that are just left and right of ν including (resp. not including) adjunction. The *prediction atoms* ${}^{\bullet}\mathrm{T}_{\nu}$, ${}_{\bullet}\mathrm{B}_{\nu}$ and *propagation atoms* $\mathrm{T}_{\nu}{}^{\bullet}$, $\mathrm{B}_{\nu\bullet}$ are built using *modulations* from the node arguments T_{ν} and B_{ν} completed by position variables $({}^{\bullet}\mathrm{P}_{\nu}, \mathrm{P}_{\nu}{}^{\bullet})$ and $({}_{\bullet}\mathrm{P}_{\nu}, \mathrm{P}_{\nu\bullet})$ used to delimit the span of ν including or not adjunction. [3] A *modulation* (Barthélemy & Villemonte de la Clergerie, 1998) is formalized as a pair of projection morphisms $(\overline{\ }, \underline{\ })$ such that, for all atoms A, B, $\mathrm{mgu}(A, B) = \mathrm{mgu}(\overline{A}\underline{A}, \overline{B}\underline{B})$. For TAGs, we offer the possibility to have distinct modulations $(\overline{\ }^{t}, \underline{\ }_{t})$ and $(\overline{\ }^{b}, \underline{\ }_{b})$ for top and bottom arguments, as well as a third one $(\overline{\ }^{n}, \underline{\ }_{n})$ for nodes, leading to the following definitions: [4]

$$
\begin{array}{ll}
{}^{\bullet}\mathrm{T}_{\nu} = \overline{T_{\nu}[\,{}^{\bullet}\mathrm{P}_{\nu};\ \mathrm{P}_{\nu}{}^{\bullet}]}^{t} & {}_{\bullet}\mathrm{B}_{\nu} = \overline{B_{\nu}[\,{}_{\bullet}\mathrm{P}_{\nu};\ \mathrm{P}_{\nu\bullet}]}^{b} \\
\mathrm{T}_{\nu}{}^{\bullet} = \underline{T_{\nu}[\,{}^{\bullet}\mathrm{P}_{\nu};\ \mathrm{P}_{\nu}{}^{\bullet}]}_{t} & \mathrm{B}_{\nu\bullet} = \underline{B_{\nu}[\,{}_{\bullet}\mathrm{P}_{\nu};\ \mathrm{P}_{\nu\bullet}]}_{b}
\end{array}
$$

Modulation is useful to tune the top-down prediction of trees and the bottom-up propagation of recognized trees. It allows an uniform description of a wide family of parsing strategies, ranging from pure bottom-up ones to prefix-valid top-down ones and also including mixed strategies such as Earley-like. In practice, a directive **tag_mode**(s/1,top,+(−),+,−) states that, for a node argument $T = \mathtt{s}(X)$ and position variables $({}^{\bullet}\mathrm{P}, \mathrm{P}^{\bullet}) = (L, R)$, we get ${}^{\bullet}\mathrm{T} = \mathtt{call\_s\_1}(L)$ and $\mathrm{T}^{\bullet} = \mathtt{ret}(R, X)$.

In practice, the transitions built following a Call/Return strategy may be grouped in *meta-transitions* by (a) grouping pairs of related call and return transition and (b) considering dotted nodes as *continuations*. For instance, figure 3 shows the skeleton of a meta-transition representing the traversal of an auxiliary tree with root r and some adjoinable node ν. [5]

[2] Transitions and configurations have been simplified in this paper for sake of clarity and space.

[3] Of course, there are many congruence relations between the position variables. For instance, if ν immediately precedes μ in the traversal, we have $\mathrm{P}_{\nu}{}^{\bullet} \equiv {}^{\bullet}\mathrm{P}_{\mu}$. When ν is not adjoinable, we have ${}^{\bullet}\mathrm{P}_{\nu} \equiv {}_{\bullet}\mathrm{P}_{\nu}$ and $\mathrm{P}_{\nu}{}^{\bullet} \equiv \mathrm{P}_{\nu\bullet}$.

[4] There is also a specific modulation for substitution nodes.

[5] Actually, we have considered the case of a mandatory adjunction at ν. To handle non mandatory ones, we add disjunction points in the meta-transitions and share common continuations between alternatives.

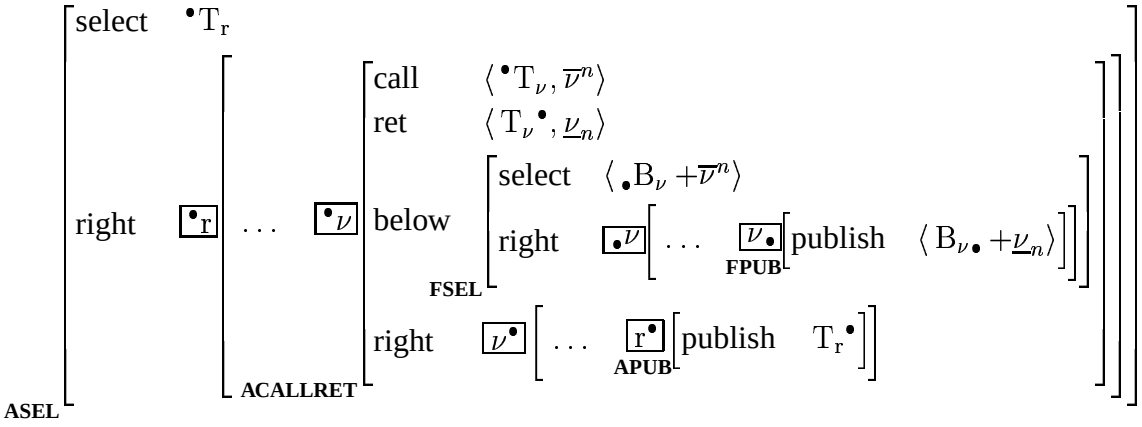

Figure 3: Sketch of a meta-transition for an auxiliary tree

4. Preparing the Dynamic Programming evaluation

The next compilation phase unfolds the meta-transitions and identifies which objects (items or transitions) may arise at run-time. This analysis is based on a DP interpretation for 2SAs.

Items We consider two kinds of items, namely Context-Free [CF] Items $\boldsymbol{ABC}$ (also represented by $\boldsymbol{AB}[\diamond\diamond]\boldsymbol{C}$) and Escaped Context-Free [xCF] Items $\boldsymbol{AB}[\boldsymbol{DE}]\boldsymbol{C}$ representing subderivations passing by configurations $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$. [6] Computation sharing stems from the fact that we don't save a full configuration $\mathcal{X} = (\Xi X, \xi x)$ but rather a *mini configuration* $\langle X, x\rangle$ or a *micro configuration* $\langle X\rangle$. Better, it is possible to keep, in some cases, only a fraction $\epsilon(X)$ of the information available in a stack element X. For instance, we take $\epsilon(\,^{\bullet}\nu) = \,^{\bullet}\mathrm{T}_{\nu}$ for an adjoinable node ν, and $\epsilon(\,_{\bullet}\nu) = \,_{\bullet}\mathrm{B}_{\nu}$ for a foot node. Finally, $\boldsymbol{A} = \langle \epsilon A\rangle$, $\boldsymbol{B} = \langle \epsilon B, b\rangle$ (when $\boldsymbol{B} \neq \boldsymbol{A}$), and $\boldsymbol{D} = \langle \epsilon D, d\rangle$ (when $\boldsymbol{D} \neq \diamond$). Table 1 shows some items relative to an adjunction at ν. If ν dominates some foot g in an adjunction on μ, $[\boldsymbol{DE}] = [\langle\, _{\bullet}\mathrm{B}_{\mathrm{g}}, a\rangle\langle\, \mathrm{B}_{\mathrm{g}\bullet} + b\rangle]$ and $(a, b) = (\overline{\mu}^n, \underline{\mu}_n)$; otherwise $[\boldsymbol{DE}] = [\diamond\diamond]$ and $(a, b) = (\diamond, \diamond)$.

	after CALL	before RET
on ADJ ν	$\langle\,^{\bullet}\mathrm{T}_{\nu}\rangle\langle\,^{\bullet}\mathrm{T}_{\nu}\rangle\langle\,^{\bullet}\mathrm{T}_{\nu}, \overline{\nu}^n\rangle$	$\langle\,^{\bullet}\mathrm{T}_{\nu}\rangle\langle\,^{\bullet}\mathrm{T}_{\nu}\rangle[DE]\langle\,\mathrm{T}_{\nu}{}^{\bullet}, \underline{\nu}_n\rangle$
on FOOT f	$\langle\,^{\bullet}\mathrm{T}_{\nu}\rangle\langle\,_{\bullet}\mathrm{B}_{\mathrm{f}}, \overline{\nu}^n\rangle\langle\,_{\bullet}\mathrm{B}_{\mathrm{f}} + \overline{\nu}^n, a\rangle$	$\langle\,^{\bullet}\mathrm{T}_{\nu}\rangle\langle\,_{\bullet}\mathrm{B}_{\mathrm{f}}, \overline{\nu}^n\rangle[\boldsymbol{DE}]\langle\,\mathrm{B}_{\mathrm{f}\bullet} + \underline{\nu}_n, b\rangle$

Table 1: Refined items at adjunction and foot nodes

Application rules Figure 4 shows (some of) the application rules used to combine items and transitions. The antecedent transition and items are (implicitly) correlated using unification with the resulting most general unifier applied to the consequent item. Component that need not be consulted are replaced by holes $\star$. Similar items occurring in different rules have been supscripted by I, J, K and L. Note that these rules derive from more abstract ones, independent of any strategy, that we have instantiated, to be more concrete, for the Call/Return strategies.

Projections Time complexity may be reduced by removing from objects the components that are not consulted (marked by $\star$), leading to tabulate one or more *projected* objects instead of the original one. In particular, instead of tabulating $K = \langle\,^{\bullet}\mathrm{T}_{\nu}\rangle\langle\,_{\bullet}\mathrm{B}_{\mathrm{f}}, \overline{\nu}^n\rangle[\boldsymbol{DE}]\langle\,\mathrm{B}_{\nu\bullet} + \underline{\nu}_n, b\rangle$

[6] The different conditions satisfied by these configurations are outside the scope of this paper.

$$\text{(ACALL)}\quad \frac{(\ ^\bullet\nu, \epsilon) \longmapsto (\ ^\bullet\nu\ ^\bullet\mathrm{T}_\nu, \overline{\nu}^n) \qquad \boldsymbol{A}\star\langle\ ^\bullet\nu, \star\rangle^I}{\langle\ ^\bullet\mathrm{T}_\nu\rangle\langle\ ^\bullet\mathrm{T}_\nu\rangle\langle\ ^\bullet\mathrm{T}_\nu, \overline{\nu}^n\rangle}$$

$$\text{(FCALL)}\quad \frac{(\ _\bullet\mathrm{f}, \overline{\nu}^n) \longmapsto (\ _\bullet\mathrm{f}\ (\ _\bullet\mathrm{B_f} + \overline{\nu}^n), \epsilon) \qquad \begin{array}{r}\boldsymbol{A}\star\langle\ ^\bullet\nu, a\rangle^I\\ \langle\ ^\bullet\mathrm{T}_\nu\rangle\star\langle\ _\bullet\mathrm{f}, \overline{\nu}^n\rangle^J\end{array}}{\boldsymbol{A}\langle\ _\bullet\mathrm{B_f}, \overline{\nu}^n\rangle\langle\ _\bullet\mathrm{B_f} + \overline{\nu}^n, a\rangle}$$

$$\text{(FRET)}\quad \frac{(\ _\bullet\mathrm{f}\ (\mathrm{B_f}\bullet + \underline{\nu}_n), \epsilon) \longmapsto (\mathrm{f}_\bullet, \underline{\nu}_n) \qquad \begin{array}{r}\langle\ ^\bullet\mathrm{T}_\nu\rangle\boldsymbol{O}\langle\ _\bullet\mathrm{f}, \overline{\nu}^n\rangle^J\\ \boldsymbol{A}\star\langle\ ^\bullet\nu, a\rangle^I\\ \boldsymbol{A}\langle\ _\bullet\mathrm{B_f}, \overline{\nu}^n\rangle[\boldsymbol{D}\star]\langle\ _\bullet\mathrm{B_f} + \underline{\nu}_n, \star\rangle^K\end{array}}{\langle\ ^\bullet\mathrm{T}_\nu\rangle\boldsymbol{O}[\langle\ _\bullet\mathrm{B_f}, \overline{\nu}^n\rangle\langle\ _\bullet\mathrm{B_f} + \underline{\nu}_n\rangle]\langle\ \mathrm{f}_\bullet, \underline{\nu}_n\rangle} \quad \boldsymbol{D} = \langle\star, a\rangle$$

$$\text{(dxCF)}\quad \frac{\begin{array}{r}\star\langle\ _\bullet\mathrm{B_f}, \overline{\nu}^n\rangle[\boldsymbol{DE}]\langle\ \mathrm{B_f}\bullet + \underline{\nu}_n, \star\rangle^K\\ \langle\ ^\bullet\mathrm{T}_\nu\rangle\langle\ ^\bullet\mathrm{T}_\nu\rangle[\langle\ _\bullet\mathrm{B_f}, a\rangle\langle\ \mathrm{B_f}\bullet + \underline{\nu}_n\rangle]\langle\ \mathrm{T}_\nu\ ^\bullet\rangle\end{array}}{\langle\ ^\bullet\mathrm{T}_\nu\rangle\langle\ ^\bullet\mathrm{T}_\nu\rangle[[\boldsymbol{DE}]]\langle\ \mathrm{T}_\nu\ ^\bullet\rangle^L}$$

$$\text{(ARET)}\quad \frac{(\ ^\bullet\nu\ \mathrm{T}_\nu\ ^\bullet, \underline{\nu}_n) \longmapsto (\ \nu^\bullet, \epsilon) \qquad \begin{array}{r}\boldsymbol{AN}\langle\ ^\bullet\nu, a\rangle^I\\ \boldsymbol{A}\star[\boldsymbol{DE}]\langle\star, b\rangle^K\\ \langle\ ^\bullet\mathrm{T}_\nu\rangle\langle\ ^\bullet\mathrm{T}_\nu\rangle[[\boldsymbol{DE}]]\langle\ \mathrm{T}_\nu\ ^\bullet, \underline{\nu}_n\rangle^L\end{array}}{\boldsymbol{AN}[\boldsymbol{DE}]\langle\ \nu^\bullet, b\rangle} \quad \boldsymbol{D} = \langle\ _\bullet\mathrm{B_f}, a\rangle$$

Figure 4: Some application rules for the Dynamic Programming interpretation

corresponding to the traversal of the subtree rooted at ν, we tabulate 3 projections used with Rules (FRET), (dxCF), and (ARET). The effectiveness of projections is still to be validated!

Partial and immediate applications To further reduce worst-case complexity, DyALog is configured to combine, at each step, a single item with a single transition. It is therefore necessary to decompose the application rules to follow this scheme, which is done by introducing intermediate pseudo transitions materializing partial applications. Subsumption checking can be done on these intermediate transitions, leading to better computation sharing. We get cascades of partial applications as illustrated by figure 5. An object $(Rule)\alpha_1 \ldots \alpha_k$ represents the (intermediate) structure associated to the partial application with *(Rule)* of $\alpha_1, \ldots, \alpha_k$ and α_i being either the transition τ or the ith item (from the top) in the rule.
Another advantage of partial application w.r.t. meta-transitions is the possibility to do *immediate partial application*, with no tabulation of some items. For instance, when reaching an adjunction node ν, we should tabulate item $I = \boldsymbol{A}\star\langle\ ^\bullet\nu, a\rangle$ and wait for other items to apply Rules (FCALL), (FRET), and (ARET). Instead, we immediately perform partial applications and tabulate the intermediate objects (see the underlined objects in fig. 5). We also apply Rule (ACALL) and tabulate "Call Aux Item" $\mathbf{CAI} = \langle\ ^\bullet\mathrm{T}_\nu\rangle\langle\ ^\bullet\mathrm{T}_\nu\rangle\langle\ ^\bullet\mathrm{T}_\nu, \overline{\nu}^n\rangle$. Immediate applications are also done when reaching a foot f with item $J = \langle\ ^\bullet\mathrm{T}_\nu\rangle\boldsymbol{O}\langle\ _\bullet\mathrm{f}, \overline{\nu}^n\rangle$.

Shared Derivation Forest They are extracted from tabulated objects by recursively following *typed backpointers* to their parents and are expressed as Context-Free Grammars(Vijay-Shanker & Weir, 1993). Parsing `aabbeccdd` with the grammar of figure 1 returns the following forest:

```
s(2)(0,9)              1 <-- 2 % Der. of elem tree with adj.
s(2)(0,9) * s(0)(4,5)  2 <-- 3 % Der. of aux. tree with adj.
s(1)(1,8) * s(0)(3,6)  3 <--   % Der. of aux. tree
```

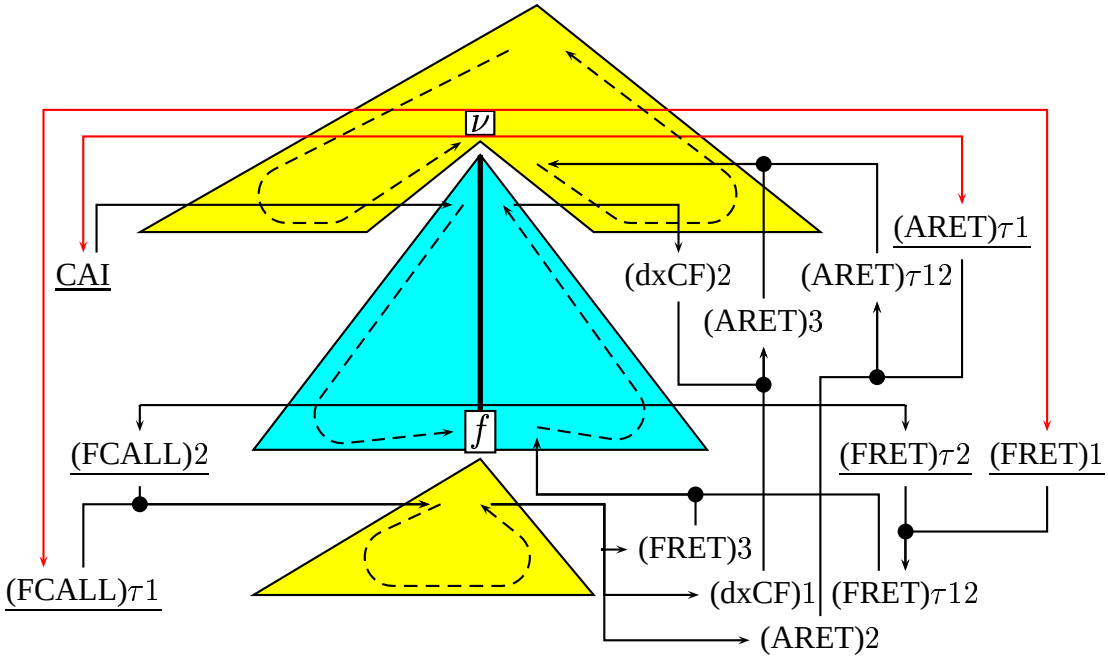

Figure 5: Cascades of partial evaluations related to an adjunction

5. Analysis and conclusion

In the worst case and in the case of TAGs without features, the number of created objects (using subsumption checking) is in $O(n^5)$ and the number of tried applications is in $O(n^6)$. These complexities come from the number of different instantiated position variables which may occur in objects or be consulted (for applications). [7] These complexities remain polynomial when dealing with DATALOG features (no symbol of functions) and may be exponential otherwise. Time complexity is directly related to the number of tried applications and created objects if objects can be accessed and added in constant time. The indexing scheme of DyALog based on trees of hashed tables ensures this property only for pure and DATALOG TAGs.
These different remarks about complexity have been confirmed for small "pathological" grammars. However, some recent experimentations done with a prefix-valid top-down parser compiled from a French XTAG-like non lexicalized grammar of 50 trees (with DATALOG features) have shown a much better behavior (0.5s to 2s for sentences of 3 to 15 words on a Pentium-II 450Mhz). We hope to improve these figures by factorizing tree traversals and using (when possible) specialized left and right adjunctions.

References

BARTHÉLEMY F. & VILLEMONTE DE LA CLERGERIE E. (1998). Information flow in tabular interpretations for generalized push-down automata. *Theoretical Computer Science*, **199**, 167–198.

BECKER T. (1994). A new automaton model for TAGs: 2-SA. *Computational Intelligence*, **10** (4).

JOSHI A. K. (1987). An introduction to tree adjoining grammars. In A. MANASTER-RAMER, Ed., *Mathematics of Language*, p. 87–115. Amsterdam/Philadelphia: John Benjamins Publishing Co.

VIJAY-SHANKER K. & WEIR D. J. (1993). The use of shared forest in tree adjoining grammar parsing. In *Proc. of the 6th Conference of the European Chapter of ACL*, p. 384–393: EACL.

VILLEMONTE DE LA CLERGERIE E. & ALONSO PARDO M. (1998). A tabular interpretation of a class of 2-stack automata. In *Proc. of ACL/COLING'98*.

[7] Note that uninstantiated or duplicated instantiated position variables may occur.