

Introduction au système SYNTAX pour la génération de compilateurs et de traducteurs

Hubert Garavel

INRIA – Laboratoire d'Informatique de Grenoble

Université Grenoble Alpes, CNRS, Grenoble INP

<http://convecs.inria.fr>



Inria

Histoire (vue depuis Grenoble) du logiciel SYNTAX

Décennie 1970 (nuit des temps)

```
/******  
*  
*          S Y N T A X  
*-----  
* Copyright (C) 1972-2023 INRIA (Institut National de Recherche en  
* Informatique et Automatique)  
*-----  
* URL: http://sourcesup.renater.fr/projects/syntax  
*-----  
* The source code of SYNTAX is distributed with two different licenses,  
* depending on the file  
+-----
```

- Gros travail : Orléans, Paris 6, IRIA Rocquencourt
- P. Boullier et équipe de B. Lohro (époux Blaizot)
- Logiciel écrit en Algol 60, puis en PL/1 (Multics)
- Peu ou pas de publications disponibles en ligne

Conjecture : SYNTAX est le plus ancien logiciel INRIA encore utilisé aujourd'hui


Décennie 1980 (1/3)

- **1980** : les travaux convergent vers un outil SYNTAX

800 30

P. Boullier,
"Generation automatique d'analyseurs
syntaxiques, avec rattrapage d'erreurs",
Journées francophones sur la
production assistée de logiciel,
Geneve, janvier 1980.

- **1981** : l'outil SYNTAX est déployé à l'INRIA

23. Equipe "langage et traducteurs" de l'INRIA, manuel d'utilisation et de mise en oeuvre du système syntax sous Multics, listage par ordinateur, INRIA, 1981 (French). 

- **1983** : SYNTAX sert au langage Green (CII – Ichbiah)
le compilateur Ada d'Alslys utilise SYNTAX
- **1984** : thèse d'Etat de Pierre Boullier (Orléans)

Décennie 1980 (2/3)

- 1985 : Philippe Deschamp contribue à SYNTAX
- 1985 : paragrapheur automatique PARADIS
- 1985 : traducteur Pascal → Ada basé sur SYNTAX
- 1985 : SYNTAX porté de Multics vers Unix (SM90)
45 000 lignes de code PL/1 traduites en C
(traducteur lui-même basé sur SYNTAX)

**Paradis: un système de
paragraphage dirigé par la
syntaxe**

Ph. Deschamp , Pierre Boullier

RR-0455, INRIA. 1985

Rapport inria-00076099v1

Un traducteur de PL/1 vers C

M. Mazaud

RT-0065, INRIA. 1986, pp.40

Rapport inria-00070094v1

Décennie 1980 (3/3)

- Outil sémantique **TABC** (Bernard Lorho)
- Outil sémantique **FNC/FNC2** (grammaires attribuées) basé sur SYNTAX (Martin Jourdan)
- Concurrence entre **SYNTAX** (C) et **CENTAUR** (Lisp)

- **1987** : outil **YSX** qui traduit Yacc en SYNTAX
- **1987** : utilisation de SYNTAX pour un compilateur LOTOS (Ahmed Serhrouchni, puis Hubert Garavel)
- **1988** : version 3.0 de SYNTAX (portage vers VMS)

Décennie 1990

■ Grenoble : 5 compilateurs CADP basés sur SYNTAX

- ▶ CAESAR et CAESAR.ADT (Garavel et al.)
- ▶ XTL et EVALUATOR (Radu Mateescu)
- ▶ TRAIAN (Mihaela Sighireanu)

portages vers Motorola 680x0, Sparc, Alpha



■ Roquencourt : réorganisation thématique

- ▶ Langages et Traducteurs se scinde en trois équipes :
ATOLL, ChLoE-Compilation, ChLoE Programm. Logique
- ▶ ATOLL: virage vers le traitement des langues naturelles

Décennie 2000 (1/2)

- **Roquencourt** : Dépôt APP Syntax 3.9a en 2002
 - ▶ P. Boullier (75%) et Ph. Deschamp (25%)
- **2005** : SYNTAX migre sur la Gforge de l'INRIA
 - ▶ tous les commits CVS antérieurs semblent perdus
- Arrivée de **Benoît Sagot**
 - ▶ nouveaux outils **SxPipe** (2005), **SxPipe2** (2008)
 - ▶ préparation de SYNTAX 6.0, déposé à l'APP en 2009
- **Grenoble** : nouveaux outils CADP avec SYNTAX
 - ▶ **SVL** (F. Lang), **LNT2LOTOS** (D.Champelovier et al.)

Décennie 2000 (2/2)

- **2007-2008** : collaboration **Grenoble-Rocquencourt**
 - ▶ **Romain Lacroix** (747 commits, vérifiés par P. Boullier)
 - ▶ passage de SYNTAX sous licence libre CeCILL - CeCILL C
 - ▶ tri sélectif parmi les outils : "**trunk**" vs "**oldies**"
 - ▶ protection des interfaces : ACTION → SXACTION, etc.
 - ▶ nettoyage du code C selon "**gcc -Wall -Wextra**"
 - ▶ portage **Windows** et **macOS** (PowerPC, 32 bits)
 - ▶ portage **64 bits** : (x64, ia64, Sparc) x (Linux gcc, Solaris cc)
 - ▶ ajout du support pour compilation **multi-architectures**
 - ▶ ajout de demos : Lustre, formules chimiques...

Décennie 2010

■ Rocquencourt :

- ▶ création d'une branche "**devel**" (B. Sagot) orientée langues naturelles: RCG, caractères cyrilliques...
- ▶ départ en retraite de P. Boullier
- ▶ plus de commits depuis 2016

■ Grenoble :

- ▶ maintenance parallèle des branches "**trunk**" et "**devel**"
- ▶ corrections de **bugs**, dont plusieurs avec P. Boullier
- ▶ **portages** pour Gcc 6, 7, 8 et Clang (macOS Intel)
- ▶ simplification et extension de certaines **APIs**

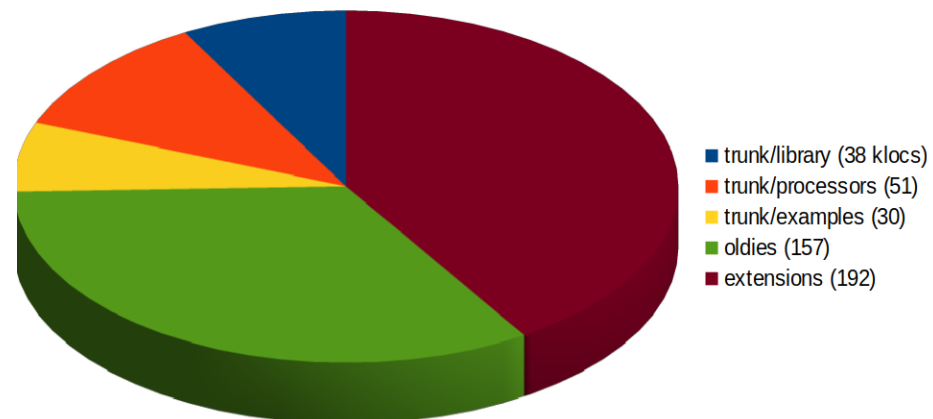
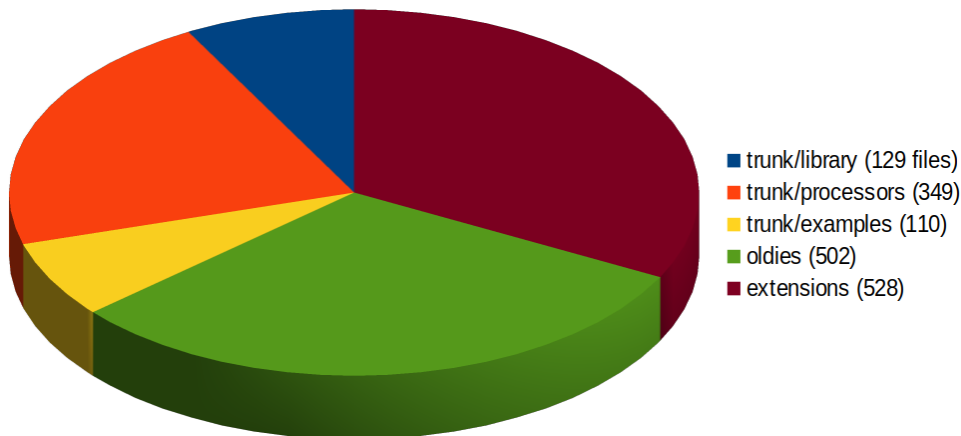
Décennie 2020

- Fermeture de la Gforge INRIA
 - ▶ migration vers sourcesup.renater.fr/projects/syntax
- Séparation entre "trunk" et "extensions"
 - ▶ **trunk** : outils (documentés) pour la compilation
 - ▶ **extensions** : outils (non documentés) pour les langues naturelles
- Nettoyage, rangement, simplification
 - ▶ commande **sxmake** unique pour compiler SYNTAX
- Préparation d'une version 7 en cours

Présentation (partielle) du système SYNTAX

SYNTAX en chiffres

- Au total : 1618 fichiers,
dont 980 fichiers de code C et SYNTAX (bootstrap)
- 468 248 lignes de code (lignes blanches exclues)
- Découpage : trunk (library + processors + examples)
+ oldies + extensions



Les processeurs SYNTAX

- C'est la partie de "haut niveau" de SYNTAX
- 10 processeurs principaux dans la partie "trunk"
- Taille : 349 fichiers – 51 686 lignes de code C
- Rôle d'un processeur SYNTAX :
 - ▶ prendre en entrée un ou plusieurs fichiers lisibles
 - ▶ produire en sortie un fichier illisible (binaire ou C)
- Typologie des processeurs :
 - ▶ lexicaux (LECL), syntaxiques (BNF, CSYNT)
 - ▶ lexico-syntaxiques (PRIO, RECOR, TABLES_C)
 - ▶ "sémantiques" (SEMAT, SEMACT, SEMC)

Le processeur BNF

- BNF vérifie qu'une grammaire hors-contexte est correcte (symboles bien définis, productive, etc.)

*extrait d'une grammaire
pour le langage Lustre*

<const> = "false" ;

<const> = "true" ;

<const> = %Integer ;

<const> = %Real ;

<exp> = <const> ;

<exp> = <exp> "and" <exp> ;

<exp> = <exp> "+" <exp> ;

<exp> = "pre" <exp> ;

<exp> = <exp> "->" <exp> ;

<exp> = "if" <exp> "then" <exp> "else" <exp> ;

<exp> = %Ident "(" <exp_list> ")" ;

Le processeur CSYNT

- CSYNT ("Constructeur SYNTaxique") produit l'automate correspondant à une grammaire BNF
 - ▶ analyse déterministe ascendante LR(1) ou LALR(1)
 - ▶ optimisation de l'automate produit
- En cas de conflit Shift/Reduce et Reduce/Reduce:
 - ▶ 1. on modifie la grammaire
 - ▶ 2. on introduit des prédicats et/ou actions syntaxiques
 - ▶ 3. on laisse CSYNT utiliser ses règles de résolution prédéfinies (notamment : Shift > Reduce)
 - ▶ 4. on utilise le processeur PRIO

Le processeur LECL

- LECL ("LE Constructeur Lexical") produit l'automate qui reconnaît les lexèmes du langage
- Langage très riche et très puissant (LECL > Lex)

commentaires en Ada

```
Comments = "-" "-" {^EOL} EOL ;
```

#include en C

```
Include = "#" &ls_First_Col {space} "include" {space}  
        QUOTE {^"\n"}+ QUOTE {space} EOL @1 ; @2 ;
```

Tokens

```
%Integer = {DIGIT}+;
```

```
Priority shift > reduce;
```

```
%Ident = LETTER [ {LETTER | DIGIT} ] ;
```

```
Context All But %Ident, %Integer;
```

Le processeur PRIO

- PRIO permet de désambigüer la grammaire BNF par des stratégies fournies dans un fichier séparé
- PRIO propose des moyens semblables à Yacc, mais avec des extensions :

règles de priorité pour un langage semblable à Lustre

```
%left "or"
```

```
%left "and"
```

```
%nonassoc "not"
```

```
%left "+" "-"
```

```
%left "*" "/" "div" "mod"
```

```
%nonassoc "<" "<=" "=" ">=" ">" "<>"
```

```
<exp> = "-" <exp> ; %prec "not"
```

Le processeur RECOR

■ Récupération automatique des erreurs

- ▶ **lexical** : insertion-destruction-permutation de caractères
- ▶ **syntaxe** : insertion-destruction-permutation de lexèmes

■ C'est un point fort de SYNTAX

```
5 :      A 1,2 := B(3*(I + 4, J*/K)
           ↑  ↑           ↑  ↑
           0  1           2  3
```

```
**** Warning (0) : "(" is inserted before "1".
**** Warning (1) : ")" is inserted before ":= ".
**** Warning (2) : ")" is inserted before ", ".
**** Error (3) :  "%IDENTIFIER" is inserted before "/".
```

```
6 :      if I = 1 then then go to UP ;
           ↑           ↑
           0           1
```

```
**** Warning (0) : ";" is inserted before "if".
**** Warning (1) : "then" is deleted.
```

Le processeur TABLES_C

- Contrairement à Lex/Yacc, SYNTAX ne génère pas un programme C contenant l'analyseur, mais des tables interprétées par un scanner/parser existant
- Ceci permet **plusieurs analyseurs simultanés** (un analyseur = un ensemble de tables)
- **Utilité n° 1** : un même outil lit des fichiers écrits dans des langages différents (ex.: model checker)
- **Utilité n° 2** : un outil lit un même fichier contenant des langages différents (ex.: HTML + JavaScript)

La bibliothèque libsx.a

- C'est la partie de "**bas niveau**" de SYNTAX
- Taille : 129 fichiers .c et .h – 37642 lignes de code
- Contenu :
 - ▶ **sxscanner** : automate d'analyse lexicale déterministe
 - ▶ **sxparser** : automate d'analyse syntaxique déterministe
 - ▶ leurs équivalents pour l'analyse non-déterministe
 - ▶ 24 composants ("**managers**")
 - ▶ divers utilitaires : fonctions de manipulation de bits...
- En pratique, seuls 4 managers sont à connaître

Les 4 composants essentiels de libsx.a

■ **sxerr_mngr** (*error manager*)

- ▶ affichage centralisé des erreurs et avertissements
- ▶ plusieurs formats disponibles (SYNTAX, Gcc, concis...)

■ **sxincl_mngr** (*include manager*)

- ▶ primitives pour empiler/déempiler des fichiers inclus
- ▶ détection d'inclusions cycliques

■ **sxsrc_mngr** (*source manager*)

- ▶ accès au caractère/lexème courant (et prédécesseurs)

■ **sxstr_mngr** (*string manager*)

- ▶ tables de hachage pour identificateurs (index $\in 1...N$)

Le processeur "sémantique" SEMAT

- Construction facile d'un arbre abstrait "élagué"
- Idée : décorer certaines règles de la grammaire BNF avec le nom des nœuds de l'arbre abstrait à construire

<code><exp> = <exp> "and" <exp></code>	<code>; "AND"</code>
<code><exp> = <exp> "or" <exp></code>	<code>; "OR"</code>
<code><exp> = "pre" <exp></code>	<code>; "PRE"</code>
<code><exp> = <exp> "->" <exp></code>	<code>; "ARROW"</code>
<code><exp> = "current" <exp></code>	<code>; "CURRENT"</code>
<code><exp> = <exp> "when" <exp></code>	<code>; "WHEN"</code>
<code><exp> = "if" <exp> "then" <exp> "else" <exp></code>	<code>; "IF"</code>
<code><exp> = %ident "(" <exp_list> ")"</code>	<code>; "CALL"</code>

*exemple tiré du
langage Lustre*

Le processeur "sémantique" SEMACT

- On décore certaines règles de la grammaire avec un indice qui indique un fragment de code C à exécuter lorsque la règle est reconnue

```
<PRIORITY> = "%left"           ; 1
<PRIORITY> = "%right"          ; 2
<PRIORITY> = "%nonassoc"       ; 3
<OP_LIST> = <OP>                ;
<OP_LIST> = <OP_LIST> <OP>      ;
<OP> = <X_TERMINAL>             ; 4
```

- Autre manière de construire un arbre abstrait
- Utile aussi pour analyser une ligne de commande

Le processeur "sémantique" SEMC

- On peut associer à chaque non-terminal de la grammaire des attributs typés synthétisés
- Chaque règle comprend un fragment de code C qui calcule les attributs du père en fonction des attributs des fils (SEMC > Yacc, car plusieurs attributs)

```
$NODE(<exp>): EXP_NODE;
```

```
$decl = #include "tree.h"
```

```
...
```

```
<exp> = "pre" <exp> ;          $NODE(<exp>)  
    { $NODE(<exp>) = BUILD_PRE ($NODE(<exp>')) ; }
```

```
<exp> = <exp> "->" <exp> ;    $NODE(<exp>)  
    { $NODE(<exp>) = BUILD_ARROW ($NODE(<exp>'), $NODE(<exp>'')) ; }
```

Utilisations possibles de SEMC

■ Objectifs :

- ▶ écrire le moins possible de code C
- ▶ utiliser des langages de plus haut niveau et plus sûr
- ▶ mais conserver l'efficacité du C et de SYNTAX

■ Approche 1 :

- ▶ générer avec SEMC un arbre abstrait XML, JSON, etc.
- ▶ quitter SYNTAX et relire cet arbre en Java, Python...

■ Approche 2 :

- ▶ décrire l'arbre abstrait en langage LNT
- ▶ générer avec SEMC cet arbre abstrait en mémoire
- ▶ analyser cet arbre par du code LNT (lui-même traduit en C)

Le processeur PARADIS

- Paragrapheur dirigé par la syntaxe (*pretty-printer*)
- But : obtenir un paragrapheur "à peu de frais"
- Idée :
 - ▶ on réutilise la grammaire BNF du langage considéré
 - ▶ on formate manuellement cette grammaire
 - ▶ on complète éventuellement par des directives de formatage (saut de ligne, saut de page, marge...)
 - ▶ PARADIS génère un paragrapheur selon ce formatage

Ref. : Rapport de recherche INRIA n° 455 (1985)

Qualités de SYNTAX

- Outil **stable**, qui fonctionne bien
- **Licence** permissive de droit français
- **Portage** vers de nombreux processeurs/systèmes
- Génération **rapide** d'analyseurs **rapides**
- **Expressivité** : LALR(1) + résolution des conflits
- **Rattrapage d'erreurs** automatique et configurable
- Gestion automatique des **tables de symboles**
- Possibilité d'avoir **plusieurs analyseurs** simultanés
 - ▶ **pôle d'expertise** à Grenoble (équipe CONVECS)

Défauts de SYNTAX

- **Domaines d'application** peut-être trop étendus
 - ▶ langages informatiques et langues humaines
 - ▶ deux classes d'utilisateurs aux besoins différents
 - ▶ dans le code : frontière incertaine entre ces domaines
- **Documentation** insuffisante pour l'international
 - ▶ manuel d'utilisation détaillé (100 pages) en français
 - ▶ manuels (au format Unix) pour certains outils
- **Code source** parfois difficile à lire
 - ▶ peu de commentaires, identificateurs peu clairs
 - ▶ style C devenu archaïque ("switch" par exemple)

Conclusion

SYNTAX, hier et aujourd'hui

- Un logiciel de plus de 50 ans, toujours en activité
- Une longue histoire, à plusieurs voix :
 - ▶ **Rocquencourt-Orléans** (créativité) : **Pierre Boullier**, **Philippe Deschamp**, **Benoît Sagot** + Martin Jourdan, Bernard Lorho, Monique Mazaud...
 - ▶ **Grenoble** (discipline) : **Hubert Garavel**, **Romain Lacroix**, **Wendelin Serwe** + David Champelovier, Rémi Héritelier, Frédéric Lang, Marie Vidal...
- Aucune publication d'ensemble sur SYNTAX
 - ▶ cet exposé vise à pallier cette lacune

SYNTAX et vous

- Un outil efficace pour développer les **compilateurs** (*notamment SYNTAX + XML ou SYNTAX + TRAIAN*)
- Un cas d'étude pour les outils d'**analyse statique**
 - ▶ code C qui compile proprement avec Gcc, Clang, SunC
 - ▶ avec: pointeurs fonctions, casts, champs de bits...
- Un domaine à explorer pour les **outils de preuve**
 - ▶ théorie des langages, grammaires, automates à pile
 - ▶ algorithmes non triviaux : rattrapage d'erreurs...