TD2 : Grammars and Automata

Thibaut Benjamin[1] and Burkhart Wolff[2][0000−0002−9648−7663]

[1] LMF, Université Paris-Saclay, France
thibaut.benjamin@universite-paris-saclay.fr
[2] LMF, Université Paris-Saclay, France
wolff@lmf.cnrs.fr

Abstract. This TD deepens the understanding of grammars. Handling the concept of priority paves the way to cope with examples coming from realistic programming languages. We deepen our understanding on SLR grammars for highly efficient parsers.
The corresponding TD ends with an introduction to functional programming.
The accompanying material can be found on the the following site: [1]

Keywords: Grammars, Parsers, Railroad Diagrams, Programming in SML

1 Main Part

1.1 Exo 6

We reconsider an expression grammar.

$$\text{E ::= Ident} \mid \text{E} + \text{E} \mid \text{E} - \text{E} \mid \text{E} * \text{E} \mid \text{E / E} \mid - \text{F} \mid \text{(F)}$$

1. How many syntax trees are possible for the word : $E + E * E - E$?
   Mark those that correspond to the priorities noted above.
2. Give an equivalent non ambiguous grammar with the required priorities and associativities.
3. Calculate the *First* and *Follow* sets of the initial grammar extended by the rule: $S ::= E\ \$$

Complete this expression language and incorporate:

– equality and inequality operators (== and !=)
– comparison operators (<, <=,>,>=)
– logical connectives (&&,|| and !)
– unary + and − operators

while respecting the priorities and associativity of the C language.

1.2 Exo 7

We reconsider our expression grammar.

$$\text{E} ::= \text{Ident} \mid \text{E} + \text{E} \mid \text{E} - \text{E} \mid \text{E} * \text{E} \mid \text{E} \;/\; \text{E} \mid - \text{F} \mid (\ \text{F}\ )$$

After constructing a non-ambiguous version, will extend this grammar allows identifiers, constants, and access to one-dimensional arrays (e.g., `*(p+1) = a[x*(2+y)][z+3] = **z*2`) and function calls. Not all aspects of a language can be expressed syntactically, but if a check can be performed at the syntactic level, it should be done at this stage.

1.3 Exo 8

Drawing inspiration from the algorithms in the course for detecting unproductive or unreachable symbols, provide an algorithm to compute the set of non-terminals in a grammar that can derive, directly or indirectly, the empty word ε. Apply your algorithm to the following grammar:

- $S ::= a\ X\ Y \mid X\ W\ Z$
- $X ::= Y\ Z \mid \varepsilon$
- $Z ::= b\ Z\ T \mid \varepsilon$
- $Y ::= a\ Y \mid X\ Z$
- $T ::= b\ T \mid b$
- $W ::= \varepsilon$

Show that we can write an equivalent grammar in which at most the root has a production with ε on the right-hand side (in the case where ε is in the language of the grammar).

1.4 Exo 9

Calculate the First and Follow sets, construct the automaton LR(0) show that the grammar is indeed SLR(1) :

G1: $A ::= S\ \$$
$S ::= (\ S\ )\ S \mid \varepsilon$

Present the stages of the SLR(1) parser with the words below (note that omly one word is correct):

1. ()()\$
2. ())\$

2 Preparatory Part

2.1 An Introduction to SML

Standard ML (SML) is a general-purpose, high-level, modular, functional programming language with compile-time type checking and type inference. It is popular for writing compilers, for programming language research, and for developing theorem provers.

Standard ML is a modern dialect of ML, the language used in the Logic for Computable Functions (LCF) theorem-proving project. It is distinctive among widely used languages in that it has a formal specification, given as typing rules and operational semantics in 'The Definition of Standard ML'.

ML and SML has been developed for the implementation of symbolic computations, so compilers, analysers and theorem provers. We will use it in this class for small experiments in compiler construction. We will use it as stand-alone shell tool as well as inside the Isabelle IDE.

Major implementations: SML/NJ, MLton, Poly/ML (which lead to 'Isabelle/SML')

Dialects: Alice, Concurrent ML, Dependent ML

influenced: Elm, F#, F∗, Haskell, OCaml, Python, Rust, Scala

References to Tutorials and Books:

A really nice Quick-Starter is:

– `https://learnxinyminutes.com/standard-ml/`

Alternative material (deeper):

– `https://www.cs.cmu.edu/~rwh/isml/book.pdf`
– `https://www.lix.polytechnique.fr/~catuscia/teaching/sml/GIML/manual~1.htm`
– `https://www.scribd.com/document/439835125/sml-tutorial-pdf`
– `http://mlton.org/StandardMLTutorials`
– `https://dokumen.pub/qdownload/ml-for-the-working-programmer-2nbsped-9781107268494-11072`
`html`

2.2 Exo 10

We will use Isabelle as an advanced IDE for functional programming in SML. After starting Isabelle in a shell by:

– `isabelle jedit`

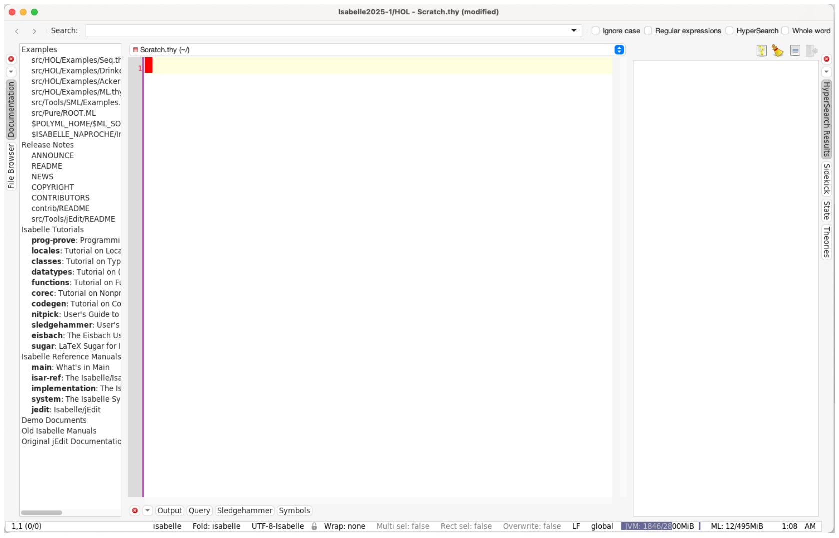
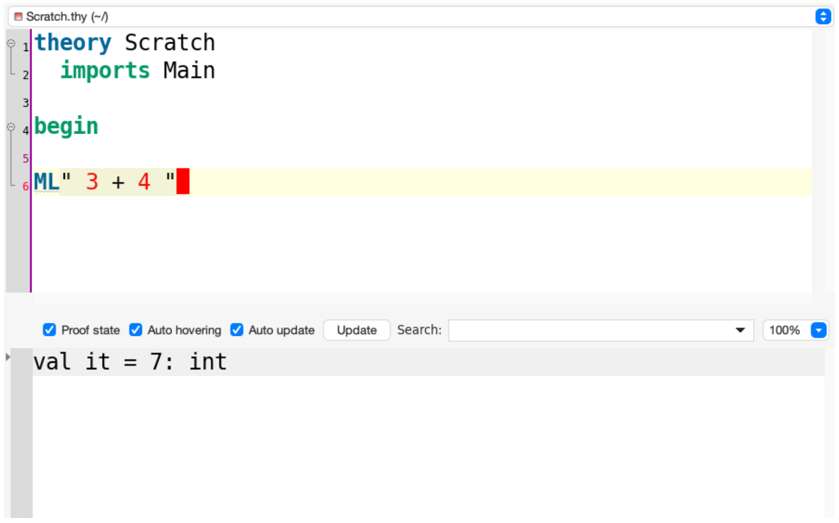


Fig. 1: The global Isabelle Window.



```
theory Scratch
  imports Main

begin

ML" 3 + 4 "
```

```
val it = 7: int
```

Fig. 2: The editing window with basic content.

(where `isabelle` is set to be the alias to your Isabelle2025-2 installation) a window should appear as shown in Fig. 1.

The central window is the editing window; by klicking on the "output"-button one gets additionally another window that shows the evaluation results corresponding to the cursor in the editing window.

Inserting the setup `theory Scratch imports Main begin` brings the system into a mode where you can start the `ML " ... "` (`ML` ‹› ... ‹›)sub-environment that we will use. This will look like Fig. 2

You may open up `ML` one after the other to separate them; and also put your SML snippets in a file (like "tp2.sml") and include this inside Isabelle by:

– `ML_file "tp1.sml"`

Some tasks:

1. Define the factorial function recursively and compute `fac 10000`.
2. Define the fibonacci function recursively and compute `fib 100`.
3. Define a recursive polymorphic data-type for binary trees with the constructors *Leaf* and *Node*.
4. Define a recursive function over these trees that 'reflects' a tree, i.e. that transforms it that for each node left and right sub-tree were exchanged.

References

1. Wolff, B.: Teaching Website: PolyTech Compiler Course. `https://usr.lmf.cnrs.fr/~wolff/teach-material/2025-2026/ET4-Compil/index.html` (2025), [Online; accessed 8-Dec-2025]