COMP 442 / 6421 Compiler Design

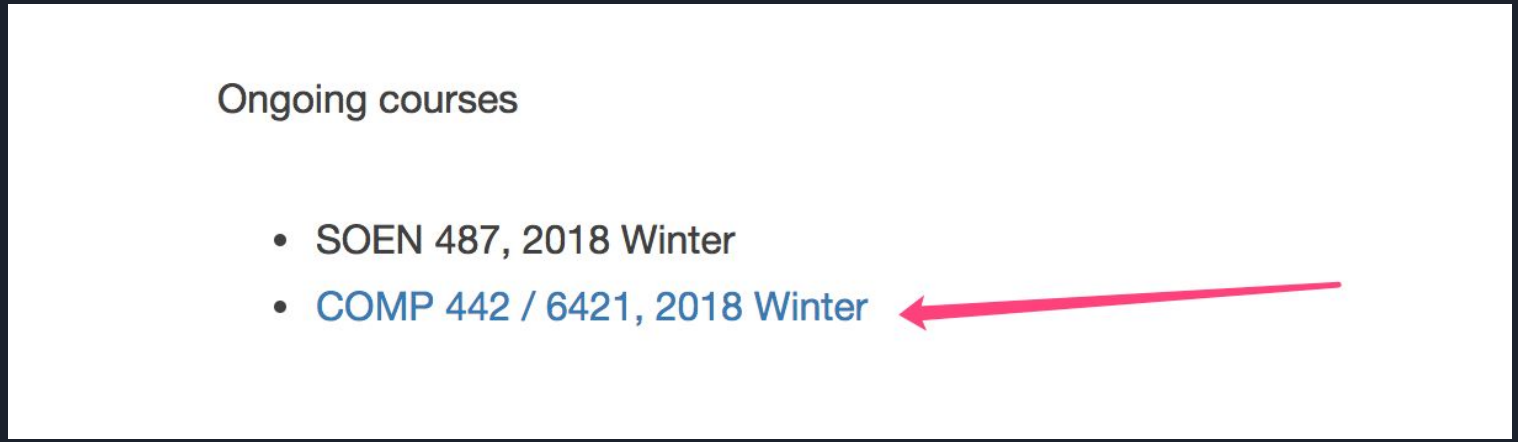
Tutorial 2

Instructor:	Dr. Joey Paquet	paquet@cse.concordia.ca
TAs:	Haotao Lai	h_lai@encs.concordia.ca
	Jashanjot Singh	s_jashan@cs.concordia.ca

Tutorial Slides

You can access the tutorial slide sets through the following link:

http://laihaotao.me/ta/



Assignment 2 : syntax analysis

- This lab material is about helping you achieve assignment #2, which is done in two stages:
  - Transform the grammar into an LL(1) grammar
  - Implement the parser
- The implementation absolutely cannot start before the grammar has been transformed.
  - We propose a set of tools to help achieve the transformation
  - Sample usage of these tools is depicted in this slide set

AtoCC kfgEdit

- Tool that allows you to analyze your grammar and locate possible ambiguities in the grammar.
- After you grammar is entered, it also allows you to enter a string representing a token stream and verify if this token stream is derivable from the grammar. If it is, it generates a parse tree and a derivation for it.

Installing AtoCC

Installing AtoCC

- AtoCC can be downloaded at the following web site:
    - http://www.atocc.de
- You can either download an installer, or precompiled applications.

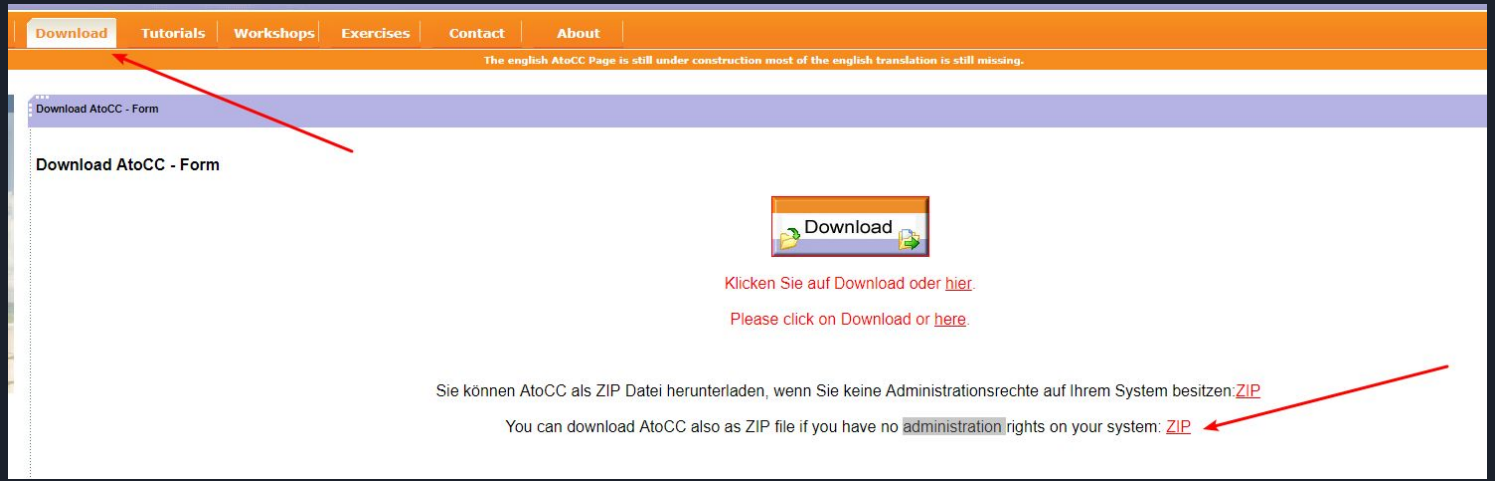
You don’t have Windows machine?

Check the following link out:

http://atocc.de/AtoCCFAQ/index.php?option=com_content&task=category&sectionid=11&id=25&Itemid=34

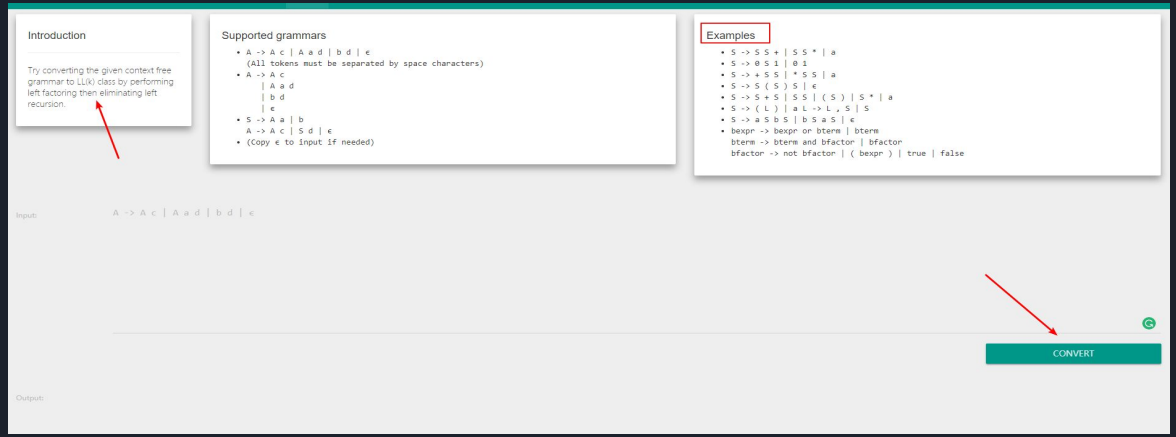
Works, for example, for macOS High Sierra version 10.13.1

Install AtoCC without administration rights ?



These are portable executables, but they often crash, so save your work frequently!

Automated grammar transformation tools

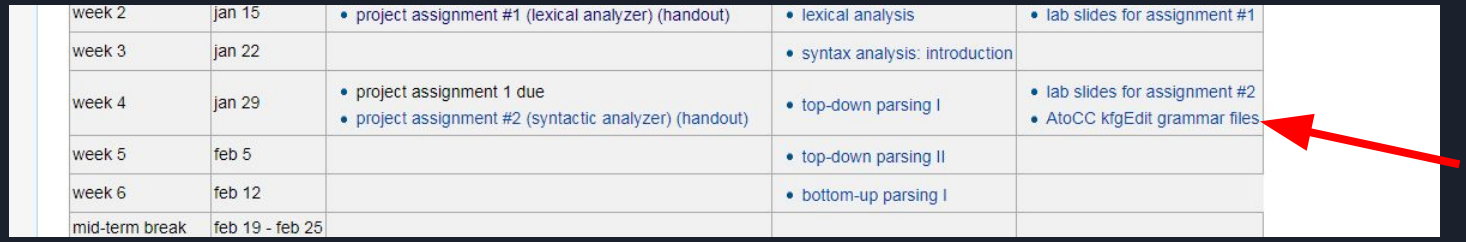


- CyberZHG's Compiler contruction toolkit:

  https://cyberzhg.github.io/toolbox/

- Can help you apply specific transformations
- Use in conjunction with kfgEdit
- However, it does not use the same grammar representation conventions

Example grammars in kfgEdit format



week 2	jan 15	• project assignment #1 (lexical analyzer) (handout)	• lexical analysis	• lab slides for assignment #1
week 3	jan 22		• syntax analysis: introduction	
week 4	jan 29	• project assignment 1 due • project assignment #2 (syntactic analyzer) (handout)	• top-down parsing I	• lab slides for assignment #2 • AtoCC kfgEdit grammar files
week 5	feb 5		• top-down parsing II	
week 6	feb 12		• bottom-up parsing I	
mid-term break	feb 19 - feb 25			

- Includes some before-after transformation examples.

The Goal of Assignment 2

1. Convert the given CFG to an LL(1) grammar
    a. Use tools to help your transformation procedure
    b. Remove the grammar from EBNF to non-EBNF representation
    c. Remove ambiguities and left recursions
    d. After each transformation step, verify that your grammar was not broken
2. Implement a LL(1) parser
    a. Recursive descent predictive parsing
    b. Table-driven predictive parsing

Example: removing EBNF constructs

Assume you was given a grammar as following, with EBNF repetition:

```
commaSeparatedList      -> a {,a}  |  EPSILON
```

You should remove the EBNF repetition and come up with the following grammar:

```
commaSeparatedList      -> a commaSeparatedListTail
                         | EPSILON
commaSeparatedListTail  -> ,a commaSeparatedListTail
                         | EPSILON
```

Example: removing left recursion

After removal of all EBNF format instances, assume you have something like:

```
expr   -> expr + term   | term
term   -> term * factor | factor
factor -> '(' expr ')'  | 'x'
```

Remove left recursions (on `expr` and `term`) using the transformation shown in class:

1- Isolate each set of productions of the form:

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \ldots$ (left-recursive)

$A \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots$ (non-left-recursive)

2- Introduce a new non-terminal A'

3- Change all the non-recursive productions on A to:

$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \ldots$

4- Remove the left-recursive production on A and substitute:

$A' \rightarrow \varepsilon \mid \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A' \mid \ldots$ (right-recursive)

How to come up with the proper grammar?

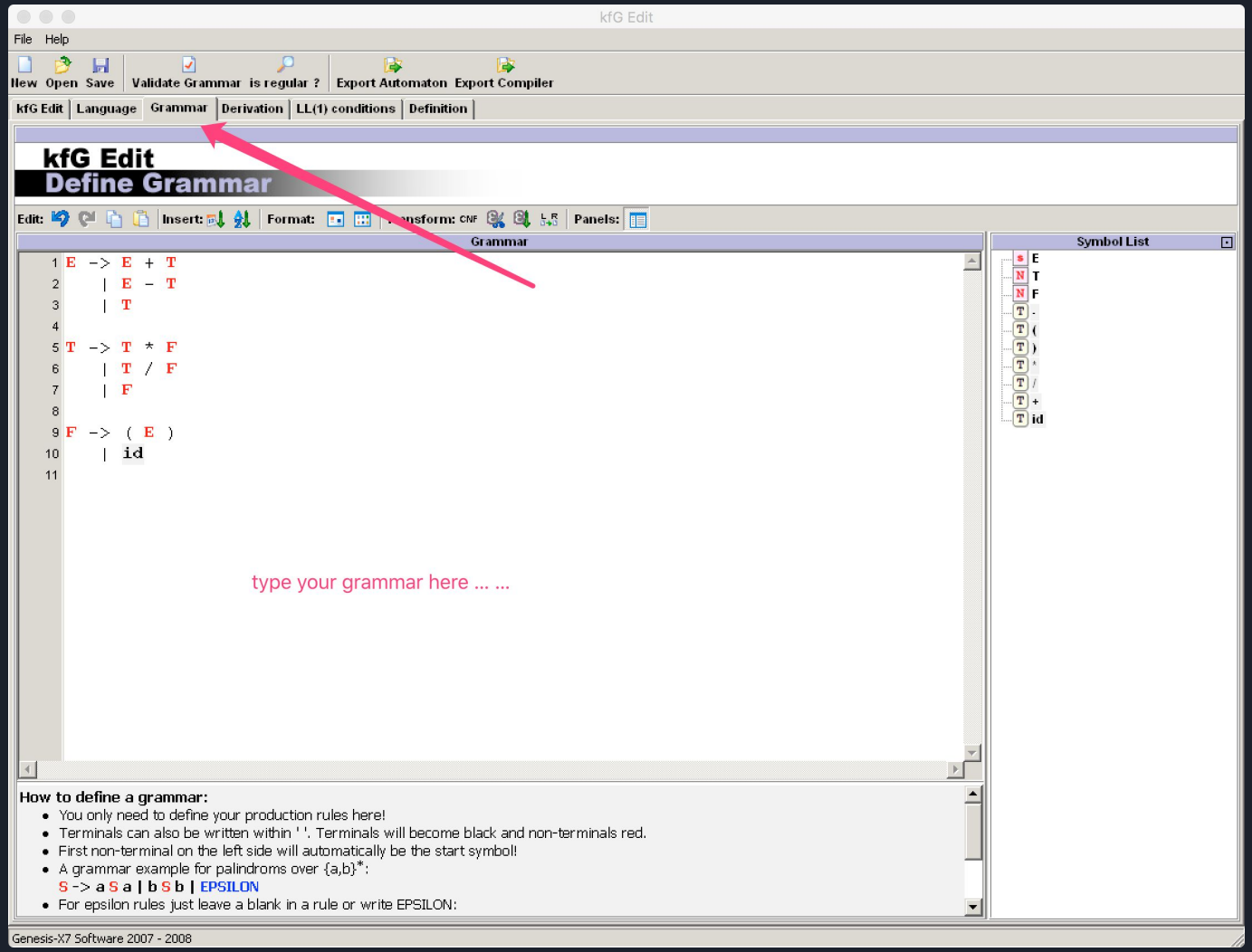
- You receive the initial grammar in EBNF in assignment 2 description already
- You need to remove the EBNF since AtoCC kfgEdit cannot understand this form
- Perform left factoring (if necessary)
- Remove left recursion (if exist, unfortunately, they exist in the given grammar)

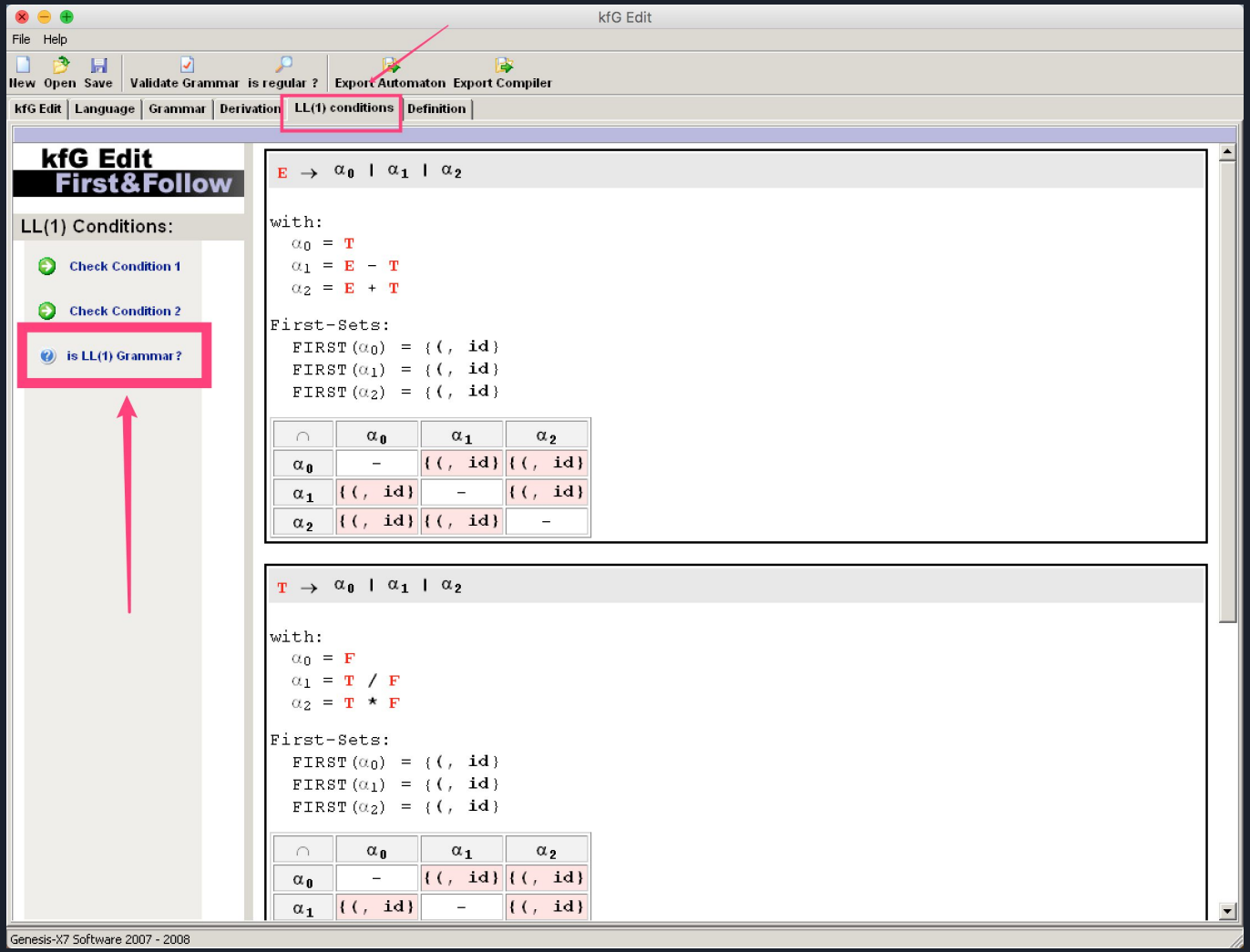
It is strongly suggested that every time you make a single transformation step, that you use AtoCC to check whether your transformation broke the grammar or not.

Don’t try to correct many errors in one shot, it is easy to get lost. Plus, if you make a mistake in one transformation step and you carry on without checking, your further transformation will be made on a wrong grammar and thus be invalid.

Example

--- How to use AtoCC for verification


kfG Edit
File Help
New Open Save Validate Grammar is regular ? Export Automaton Export Compiler
kfG Edit Language Grammar Derivation LL(1) conditions Definition
kfG Edit
Define Grammar
Edit: Insert: Format: Transform: CNF Panels:
Grammar
1 E -> E + T
2 | E - T
3 | T
4
5 T -> T * F
6 | T / F
7 | F
8
9 F -> (E)
10 | id
11
type your grammar here ... ...
Symbol List
S E
N T
N F
T -
T (
T)
T *
T /
T +
T id
How to define a grammar:
• You only need to define your production rules here!
• Terminals can also be written within ' '. Terminals will become black and non-terminals red.
• First non-terminal on the left side will automatically be the start symbol!
• A grammar example for palindroms over {a,b}*:
S -> a S a | b S b | EPSILON
• For epsilon rules just leave a blank in a rule or write EPSILON:
Genesis-X7 Software 2007 - 2008


kfG Edit
File
Help
New
Open
Save
Validate Grammar
is regular ?
Export Automaton
Export Compiler
kfG Edit
Language
Grammar
Derivation
LL(1) conditions
Definition
kfG Edit
First&Follow
LL(1) Conditions:
Check Condition 1
Check Condition 2
is LL(1) Grammar ?
E → α0 | α1 | α2
with:
α0 = T
α1 = E - T
α2 = E + T
First-Sets:
FIRST(α0) = {(, id}
FIRST(α1) = {(, id}
FIRST(α2) = {(, id}
∩ α0 α1 α2
α0 - {(, id} {(, id}
α1 {(, id} - {(, id}
α2 {(, id} {(, id} -
T → α0 | α1 | α2
with:
α0 = F
α1 = T / F
α2 = T * F
First-Sets:
FIRST(α0) = {(, id}
FIRST(α1) = {(, id}
FIRST(α2) = {(, id}
∩ α0 α1 α2
α0 - {(, id} {(, id}
α1 {(, id} - {(, id}
Genesis-X7 Software 2007 - 2008

$E \rightarrow \alpha_0 \mid \alpha_1 \mid \alpha_2$

with:
- α_0 = T
- α_1 = E - T
- α_2 = E + T

First-Sets:
- FIRST(α_0) = {(, id}
- FIRST(α_1) = {(, id}
- FIRST(α_2) = {(, id}

∩	α_0	α_1	α_2
α_0	-	{(, id}	{(, id}
α_1	{(, id}	-	{(, id}
α_2	{(, id}	{(, id}	-

$T \rightarrow \alpha_0 \mid \alpha_1 \mid \alpha_2$

with:
- α_0 = F
- α_1 = T / F
- α_2 = T * F

First-Sets:
- FIRST(α_0) = {(, id}
- FIRST(α_1) = {(, id}
- FIRST(α_2) = {(, id}

∩	α_0	α_1	α_2
α_0	-	{(, id}	{(, id}
α_1	{(, id}	-	{(, id}
α_2	{(, id}	{(, id}	-

first set intersection

F → α_0 | α_1

with:
  α_0 = id
  α_1 = (E)

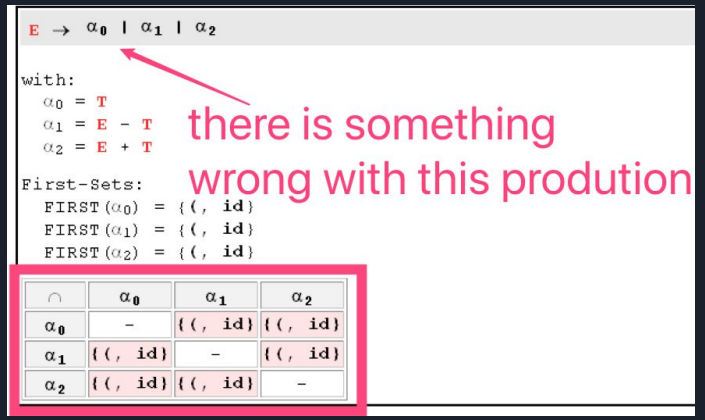
First-Sets:
  FIRST(α_0) = {id}
  FIRST(α_1) = {(}

∩	α_0	α_1
α_0	-	∅
α_1	∅	-

go to the very end of the page

LL(1) first condition not fulfilled!

What you should do?



1. Locate a specific error and identify the faulty productions (shown in red)
2. Copy the related productions into the grammar transformation tool mentioned above (https://cyberzhg.github.io/toolbox/cfg2ll).
3. Copy the correction from the tool and paste it into AtoCC
4. Do some modification to adapt to AtoCC format
5. Check the grammar again

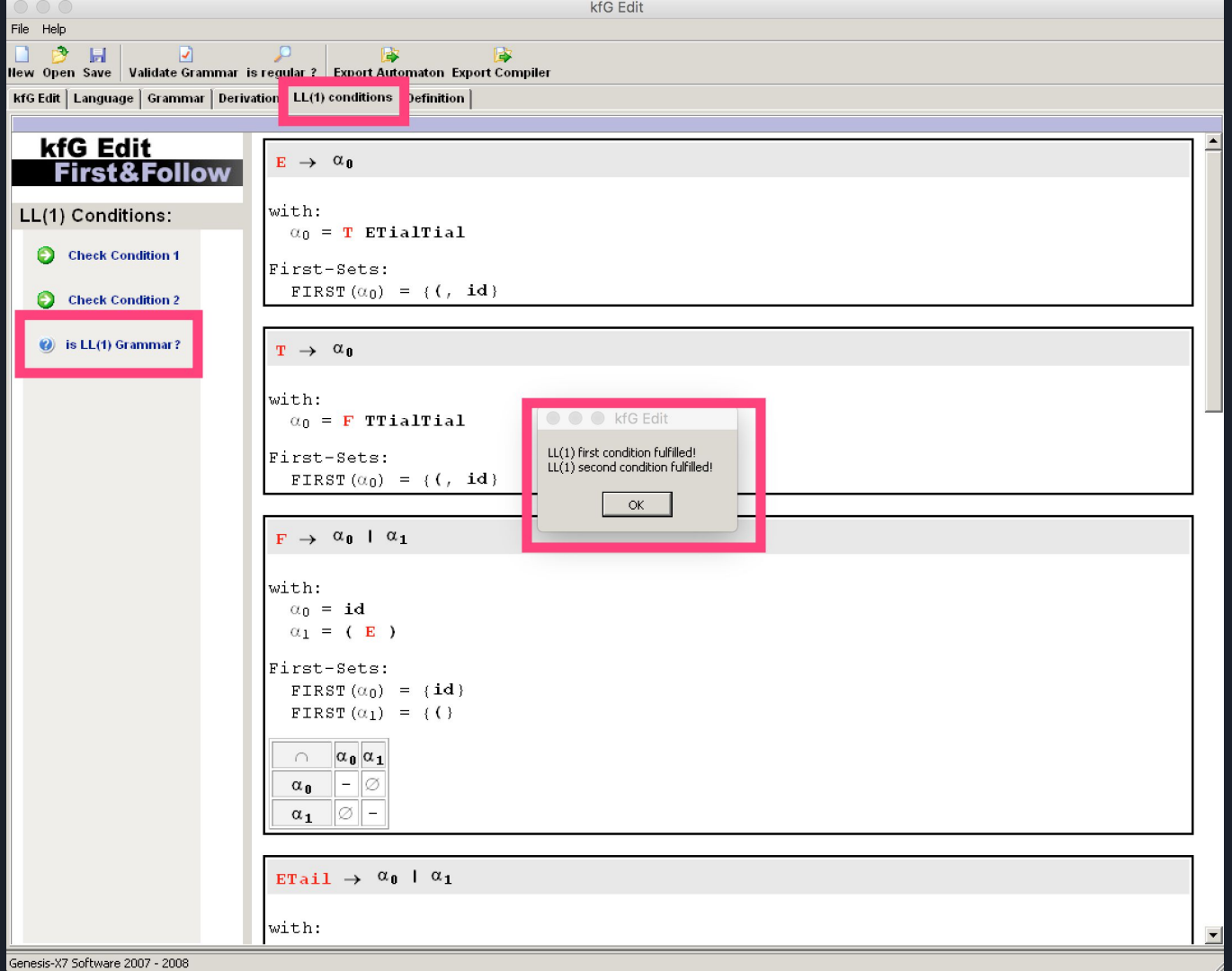
Note: Don't try to solve more than one production at a time. When you solve one production's error, use the tool to check to make sure you are not bringing new errors.

```
14 E -> T E''
15 T -> F T''
16 F -> ( E )
17    | id
18 E' -> + T
19     | - T
20 T' -> * F
21     | / F
22 E'' -> E' E''
23      | ?
24 T'' -> T' T''
25      | ?
```

result from the tool

```
1  E -> T ETailTail
2  T -> F TTailTail
3  F -> ( E )
4     | id
5  ETail -> + T
6         | - T
7  TTail -> * F
8         | / F
9  ETailTail -> ETail ETailTail
10            | EPSILON
11 TTailTail -> TTail TTailTail
12            | EPSILON
13
```

after modification, adapted to AtoCC


kfG Edit
File Help
New Open Save Validate Grammar is regular ? Export Automaton Export Compiler
kfG Edit Language Grammar Derivation LL(1) conditions Definition
kfG Edit
First&Follow
LL(1) Conditions:
Check Condition 1
Check Condition 2
is LL(1) Grammar ?
E → α0
with:
α0 = T ETialTial
First-Sets:
FIRST(α0) = {(, id}
T → α0
with:
α0 = F TTialTial
First-Sets:
FIRST(α0) = {(, id}
kfG Edit
LL(1) first condition fulfilled!
LL(1) second condition fulfilled!
OK
F → α0 | α1
with:
α0 = id
α1 = (E)
First-Sets:
FIRST(α0) = {id}
FIRST(α1) = {(}
∩ α0 α1
α0 - ∅
α1 ∅ -
ETail → α0 | α1
with:
Genesis-X7 Software 2007 - 2008

LL(1) first condition fulfilled!

```
FIRST (ETailTail) = {+, -, EPSILON}
FOLLOW(ETailTail) = {$, )}
FIRST (ETailTail) ∩ FOLLOW(ETailTail) = ∅
```

```
FIRST (TTailTail) = {*, /, EPSILON}
FOLLOW(TTailTail) = {$, ), +, -}
FIRST (TTailTail) ∩ FOLLOW(TTailTail) = ∅
```

LL(1) second condition fulfilled!

Thanks