

Learning and Refining Input Grammars for Effective Fuzzing

Rahul Gopinath



THE UNIVERSITY OF
SYDNEY

Prerequisites

<https://github.com/vrthra/SBST22-tutorial#readme>

- Install Python 3.10
- Install Graphviz
- Install Jupyter
- Start Jupyter

Prerequisites

- Download [Python 3.10](#)

Some variation of

```
$ sudo apt-get install python3.10
```

- Install [Graphviz](#) for your operating system.

Some variation of

```
$ sudo apt install graphviz
```

- Make a virtual environment (recommended)

```
$ python3 -m venv sbt2022  
$ cd sbt2022  
$ source bin/activate
```

- Install [Jupyter](#).

```
$ ./bin/python3 -m pip install "jupyter==1.0.0"
```

- Checkout this repository

```
git clone git@github.com:vrthra/SBST22-tutorial.git
```

- Start the Jupyter server in the repository directory

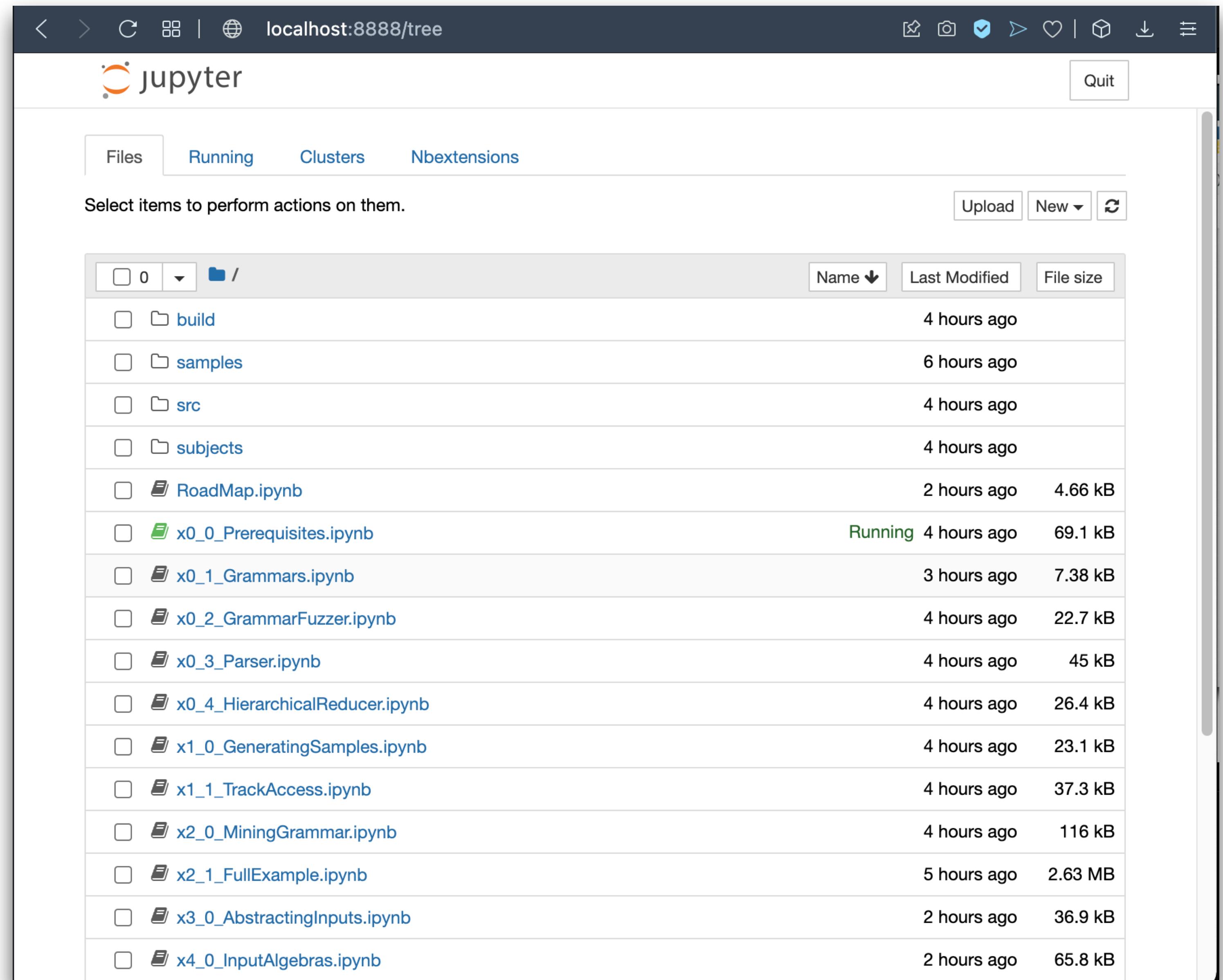
```
$ cd SBST22-tutorial  
$ jupyter-notebook
```

This opens a browser window at <http://localhost:8888/tree>

- Proceed to [x0_0_Prerequisites.ipynb](#) and execute the page completely.

Prerequisites

- Install Python 3.10
- Install Graphviz
- Install Jupyter
- Start Jupyter



`http://localhost:8888/tree`

< > C ☰ | 🌐 localhost:8888/tree

jupyter

Files Running Clusters Nbex

Select items to perform actions on them.

0 / build samples src subjects RoadMap.ipynb x0_0_Prerequisites.ipynb x0_1_Grammars.ipynb x0_2_GrammarFuzzer.ipynb x0_3_Parser.ipynb x0_4_HierarchicalReducer.ipynb x1_0_GeneratingSamples.ipynb x1_1_TrackAccess.ipynb x2_0_MiningGrammar.ipynb x2_1_FullExample.ipynb x3_0_AbstractingInputs.ipynb x4_0_InputAlgebras.ipynb

< > C ☰ | 🌐 localhost:8888/notebooks/RoadMap.ipynb

jupyter RoadMap Last Checkpoint: 2 hours ago (autosaved)

File Edit View Insert Cell Kernel Navigate Widgets Help Trusted

Contents ⚙️

- 1 The Roadmap
- 2 Prerequisites
 - 2.1 x0_1_Grammars
 - 2.2 x0_2_GrammarFuzzer
 - 2.3 x0_3_Parser
 - 2.4 x0_4_HierarchicalReducer
- 3 Generating Samples
 - 3.1 x1_0_GeneratingSamples
 - 3.2 x1_1_TrackAccess
- 4 Mining Grammar
 - 4.1 x2_0_MiningGrammar
- 5 Abstracting Inputs
 - 5.1 x3_0_AbstractingInputs
- 6 Input Algebras
 - 6.1 x4_0_InputAlgebras

1 The Roadmap

Please see the [README](#) for the basic prerequisites.

2 Prerequisites

2.1 x0_1 Grammars

This notebook describes the grammar format that we use in this tutorial, and provides a few sample grammars.

2.2 x0_2 GrammarFuzzer

This notebook contains a partial implementation of the `F1` GrammarFuzzer (only in Python).

2.3 x0_3 Parser

This notebook contains an Earley parser implementation for any context-free grammar.

2.4 x0_4 HierarchicalReducer

This notebook contains a partial implementation of the Perses hierarchical-delta debugger

< > C ☰ | 🌐 localhost:8888/tree

jupyter

Files Running Clusters Nbextensions

Select items to perform actions on them.

0 /

- build
- samples
- src
- subjects
- RoadMap.ipynb
- x0_0_Prerequisites.ipynb
- x0_1_Grammars.ipynb
- x0_2_GrammarFuzzer.ipynb
- x0_3_Parser.ipynb
- x0_4_HierarchicalReducer.ipynb
- x1_0_GeneratingSamples.ipynb
- x1_1_TrackAccess.ipynb
- x2_0_MiningGrammar.ipynb
- x2_1_FullExample.ipynb
- x3_0_AbstractingInputs.ipynb
- x4_0_InputAlgebras.ipynb

< > C ☰ | 🌐 localhost:8888/notebooks/x0_0_Prerequisites.ipynb

jupyter x0_0_Prerequisites Last Checkpoint: 4 hours ago (autosaved)

File Edit View Insert Cell Kernel Navigate Widgets Help Trusted Python

Kernel menu open:

- Interrupt [I, I]
- Restart [0, 0]
- Restart & Clear Output
- Restart & Run All
- Reconnect
- Shutdown
- Change kernel ▾

Prerequisites

This is tested completely only on Python 3.10. Other 3.x versions may likely work, but not tested so far.

```
In [ ]: major, minor, *_ = sys.version_info  
assert (major, minor) == (3, 10)
```

1.1 Python Packages

1.2 Cleanup

Cleanup if we have already done this before.

1.3 Utils.py

Utils.py contains a few helper routines that are used multiple times from other parts.

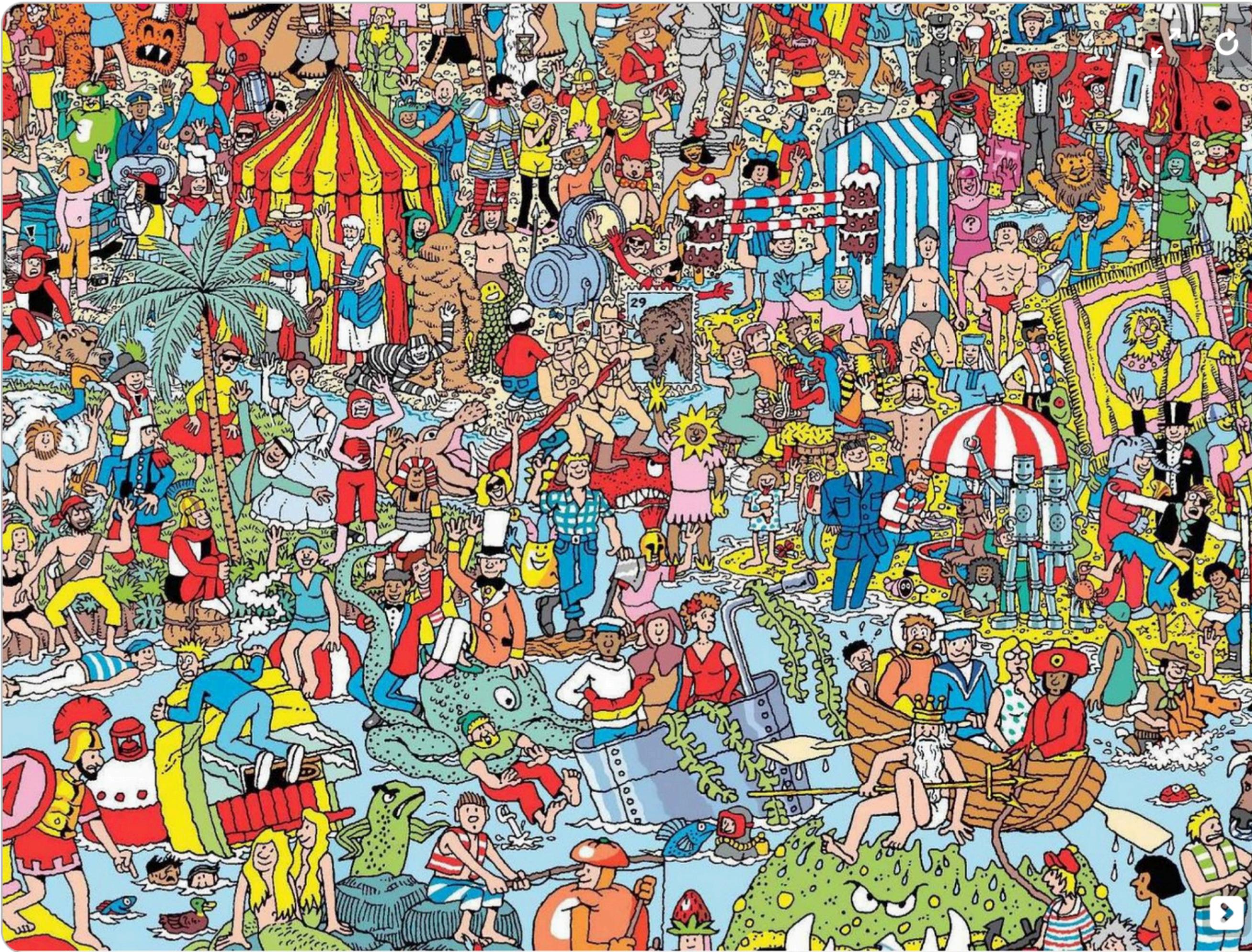
1.3.1 do()

do() allows us to execute a system command and capture its output in a log file.

localhost:8888/notebooks/x0_0_Prerequisites.ipynb#

http://localhost:8888/notebooks/x0_0_Prerequisites.ipynb

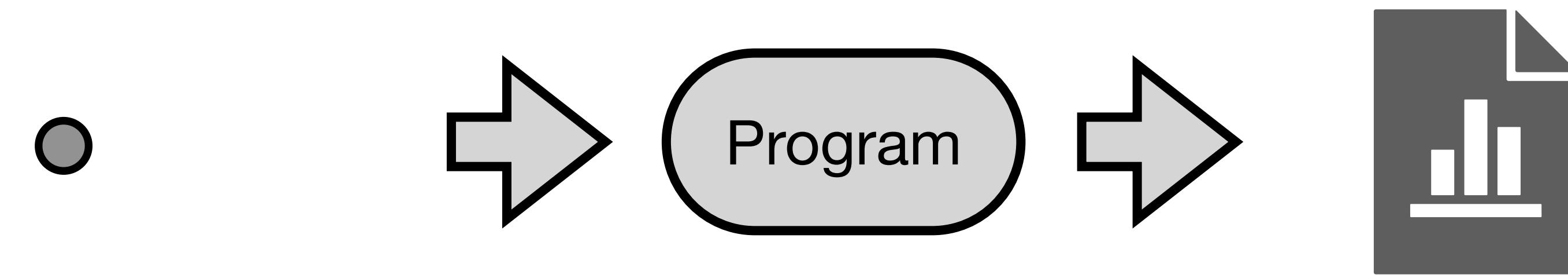
Search



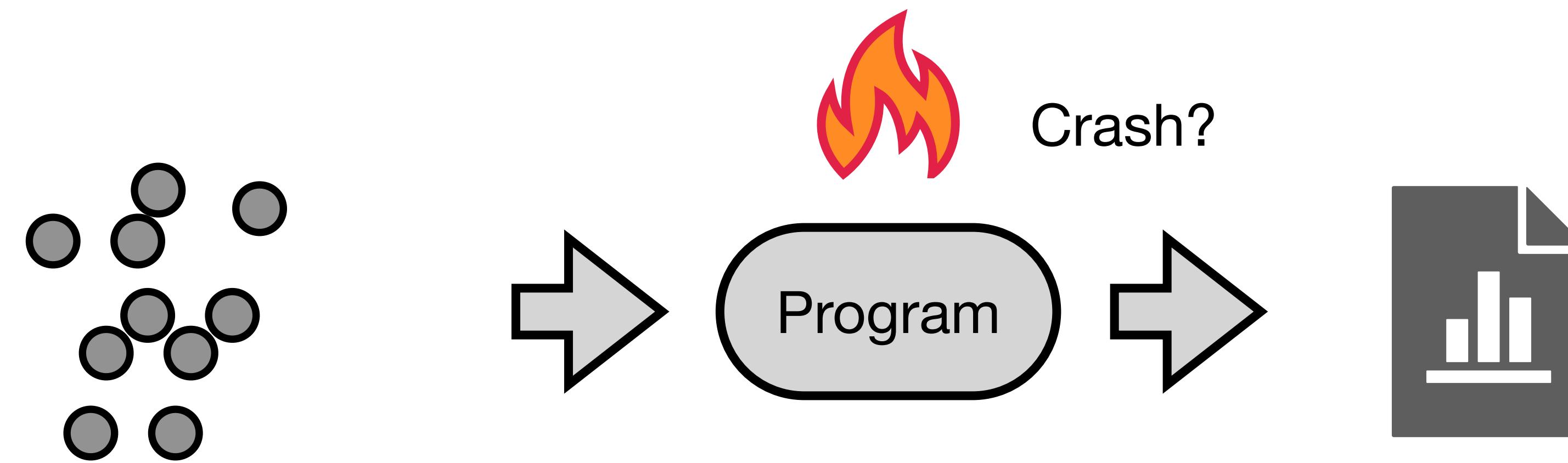
Fuzzing

Program

Fuzzing



Fuzzing



Random Fuzzing

Random Fuzzing

```
$ ./fuzz
```

```
[ ;x1-GPZ+wcckc ] ; ,N9J+?#6^6\ e?]9lu  
2_ %'4GX"0VUB[ E/r ~fApu6b8<{ %siq8Z  
h.6{V,hr?;{Ti.r3PIxMMv6 {xS^+'Hq!  
AXB"YXRS@!Kd6;wtAMefFWM(`|J_<1~o}  
z3K(CCzRH JIIvHz>_*.\>JrlU32~eGP?  
lR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{'  
)KC-i,c{<[~m!]o;{.'}Gj\ (X}EtYetrp  
bY@aGZ1{P!AZU7x#4(Rtn!q4nCwqol^y6  
)0|Ko=*JK~;zMKV=9Nai:wxu{J&UV#HaU  
)*BiC<),`+t*gka<W=Z.%T5WGHZpI30D<  
Pq>&]BS6R&j?#tP7iaV}-}^?\ ?[_[ Z^LBM  
PG-FKj '\xwuZ1=Q`^`5,$N$Q@[ !CuRzJ2  
D|vBy!^zkhdf3C5PAkR?V( (-%><hn|3='  
i2Qx]D$qs4O`1@fevnG'2\11Vf3piU37@  
5:dfd45*(7^%5ap\zIyl"'f,$ee,J4Gw:  
cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Sc  
un&sBCS,T[/3]KAeEnQ7lU)3Pn,0)G/6N  
-wyzj/MTd#A;r*(ds./df3r80daf?/<r
```



Feedback Driven Fuzzing

```
$ ./fuzz -int | program
634111569742810193727424069509
741355925061499451162464719526
61595733192482655590537407605
181400079803446874252046374716
740973770255348279425601333144
152724057932073828569041216191
099859446496509919024810271242
622974988671421938012464630138
735355134599327240920259675263
574528613057084231370741920902
794677842164654990353575580453
777282305855352378119038096476
699871306655084953377039862387
924957554389878352934547664240
082431556093837288597262675598
630851919061829885048834738832
677022429414980917053939970795
722006987916088650168665471731 yes
```

```
def is_prime(n: int) -> bool:
    """Primality test using 6k+-1 optimization."""
    if n <= 3:
        return n > 1
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i ** 2 <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

def main():
    num = stdin.read()
    print(num, is_prime(num))
```

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** localhost:8888/notebooks/x1_0_GeneratingSamples.ipynb
- Header:** jupyter x1_0_GeneratingSamples Last Checkpoint: a minute ago (autosaved) Python 3 (ipykernel)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, Help, Trusted, Python 3 (ipykernel).
- Buttons:** Save, New, Cell Types, Run, Cell, Cell, Run, Markdown.
- Contents Sidebar:** Shows a tree structure:
 - 1 Generating Samples
 - 1.1 Examples
 - 1.1.1 Calculator.py
 - 1.2 The error handler
 - 1.3 A random fuzzer.
 - 1.4 Adding Feedback**
 - 1.5 Building the Evolutionary Algorithm
 - 1.5.1 JSON
 - 2 Done
- Main Content Area:**

1 Generating Samples

In this notebook we will show how to generate valid samples for a given parser without using a grammar.

1.1 Examples

First we import the convenience utilities.

```
In [ ]: import src.utils as utils
```

1.1.1 Calculator.py

```
In [ ]: calculator = utils.load_file('subjects/calculator.py', 'calculator')
```

1.2 The error handler

We often need to interpret the error we get back. We use a simple exception class to capture the error.

```
In [ ]: with utils.ExpectError():
    calculator.main('xyz')
```

Fuzzing Parsers

```
$ ./fuzz
```

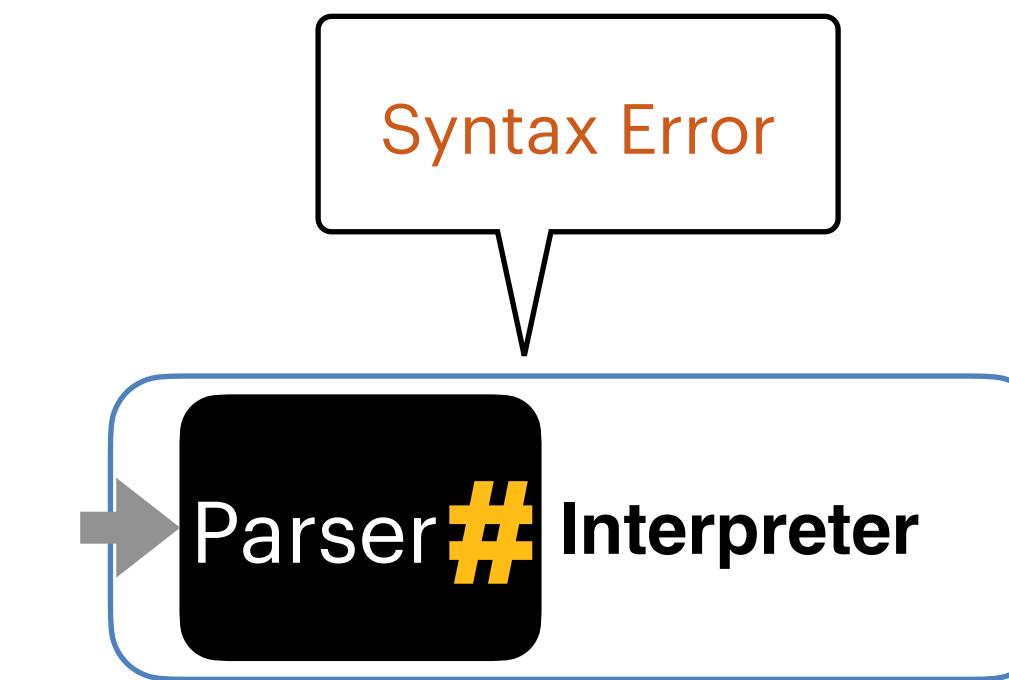
```
[ ;x1-GPZ+wcckc ] ; ,N9J+?#6^6\ e?]9lu
2_ %' 4GX" 0VUB[ E/r ~fApu6b8<{ %siq8Z
h.6{V,hr?; {Ti.r3PIxMMv6{ xS^+' Hq!
Ax B" YXRS@!Kd6; wtAMefFWM(`| J_<1~o}
z3K(CCzRH JIIvHz>_*.\>JrlU32~eGP?
lR=bF3+; y$3lodQ<B89!5 "W2fK*vE7v{ '
)KC-i,c{<[~m!]o;{.'}Gj\ (X}EtYetrp
bY@aGZ1{P!AZU7x#4(Rtn!q4nCwqol^y6
}0| Ko= *JK~; zMKV=9Nai:wxu{J&UV#HaU
)*BiC<),`+t*gka<W=Z.%T5WGHZpI30D<
Pq>& ]BS6R&j?#tP7iaV}-} ` \ ? [ _[ Z^LBM
PG-FKj '\xwuZ1=Q` ^` 5, $N$Q@[ !CuRzJ2
D| vBy! ^zkhdf3C5PAkR?V( (-%><hn| 3='
i2Qx]D$qs4O` 1@fevnG' 2\11Vf3piU37@
5:dfd45*( 7^%ap\zIyl"'f,$ee,J4Gw:
cgNKLie3nx9(`efSlg6#[ K" @WjhZ}r[ Sc
un&sBCS,T[ /3]KAeEnQ7lU)3Pn,0)G/6N
-wyzj/MTd#A;r*(ds./df3r80daf?/<#r
```

Interpreter

Fuzzing Parsers

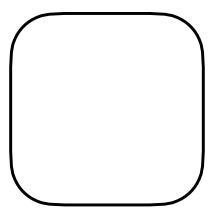
```
$ ./fuzz
```

```
[ ;x1-GPZ+wcckc ];,N9J+?#6^6\e?]9lu
2_ %'4GX"0VUB[E/r ~fApu6b8<{ %siq8Z
h.6{V,hr?;{Ti.r3PIxMMv6{xS^+'Hq!
Ax B"YXRS@!Kd6;wtAMefFWM(`|J_<1~o}
z3K(CCzRH JIIvHz>_*.\>JrlU32~eGP?
lR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{
)KC-i,c{<[~m!]o;{.'}Gj\{X}EtYetrp
bY@aGZ1{P!AZU7x#4(Rtn!q4nCwqol^y6
}0|Ko=*JK~;zMKV=9Nai:wxu{J&UV#HaU
)*BiC<),`+t*gka<W=Z.%T5WGHZpI30D<
Pq>&]BS6R&j?#tP7iaV}-}`\?\[_[Z^LBM
PG-FKj`\xwuZ1=Q`^`5,$N$Q@[!CuRzJ2
D|vBy!^zkhdf3C5PAkR?V((-%)<hn|3='
i2Qx]D$qs4O`1@fevnG'2\11Vf3piU37@
5:dfd45*(7^%5ap\zIyl"'f,$ee,J4Gw:
cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Sc
un&sBCS,T[/3]KAeEnQ7lU)3Pn,0)G/6N
-wyzj/MTd#A;r*(ds./df3r8Odaf?/<r
```

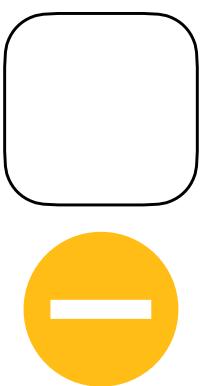


Sepcification Free Generators

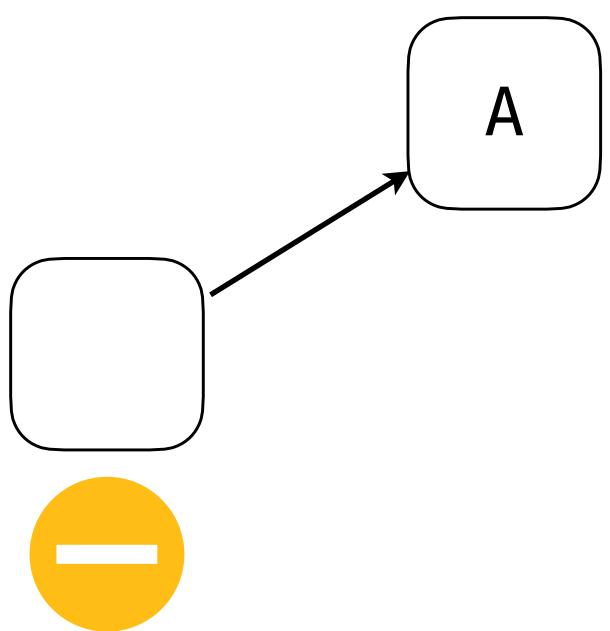
Sepcification Free Generators



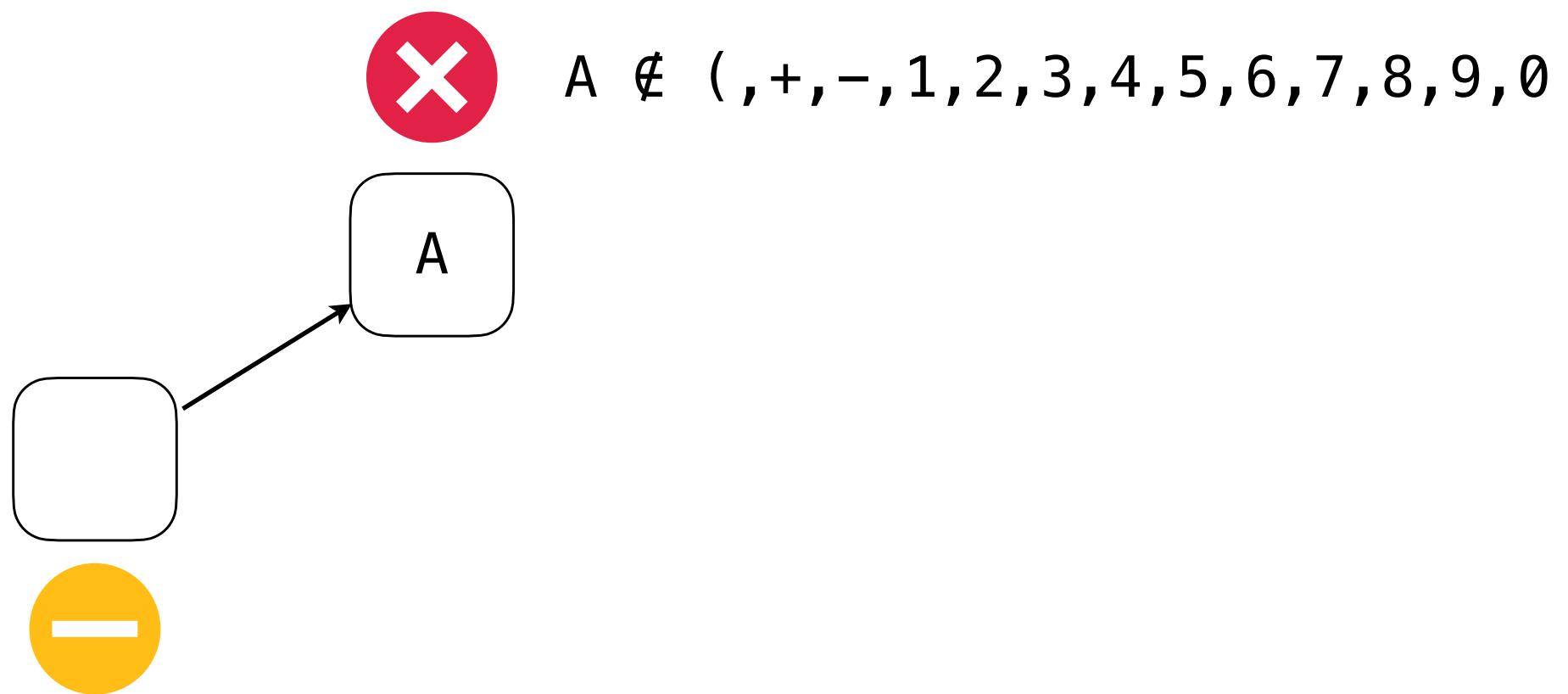
Sepcification Free Generators



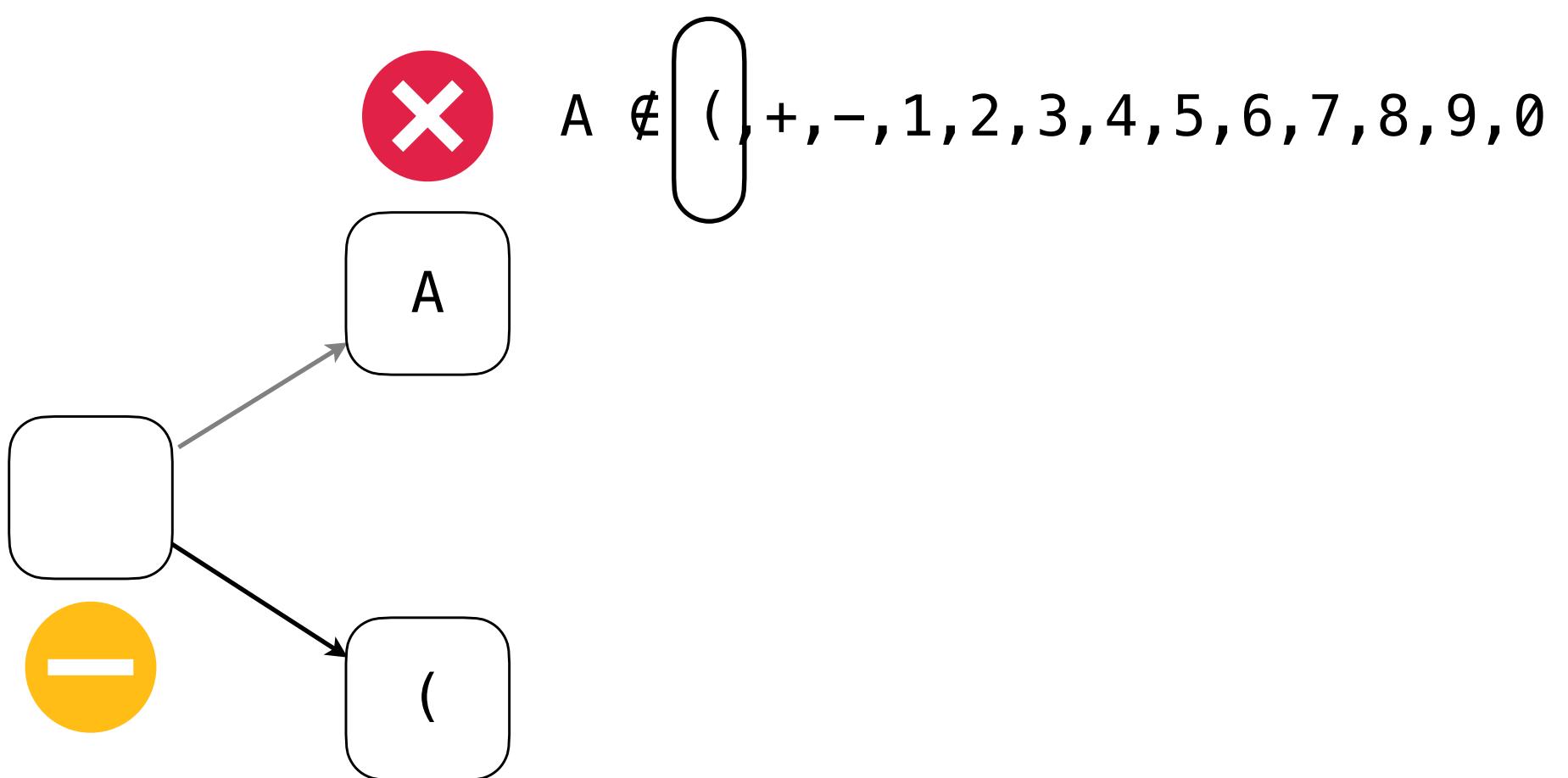
Sepcification Free Generators



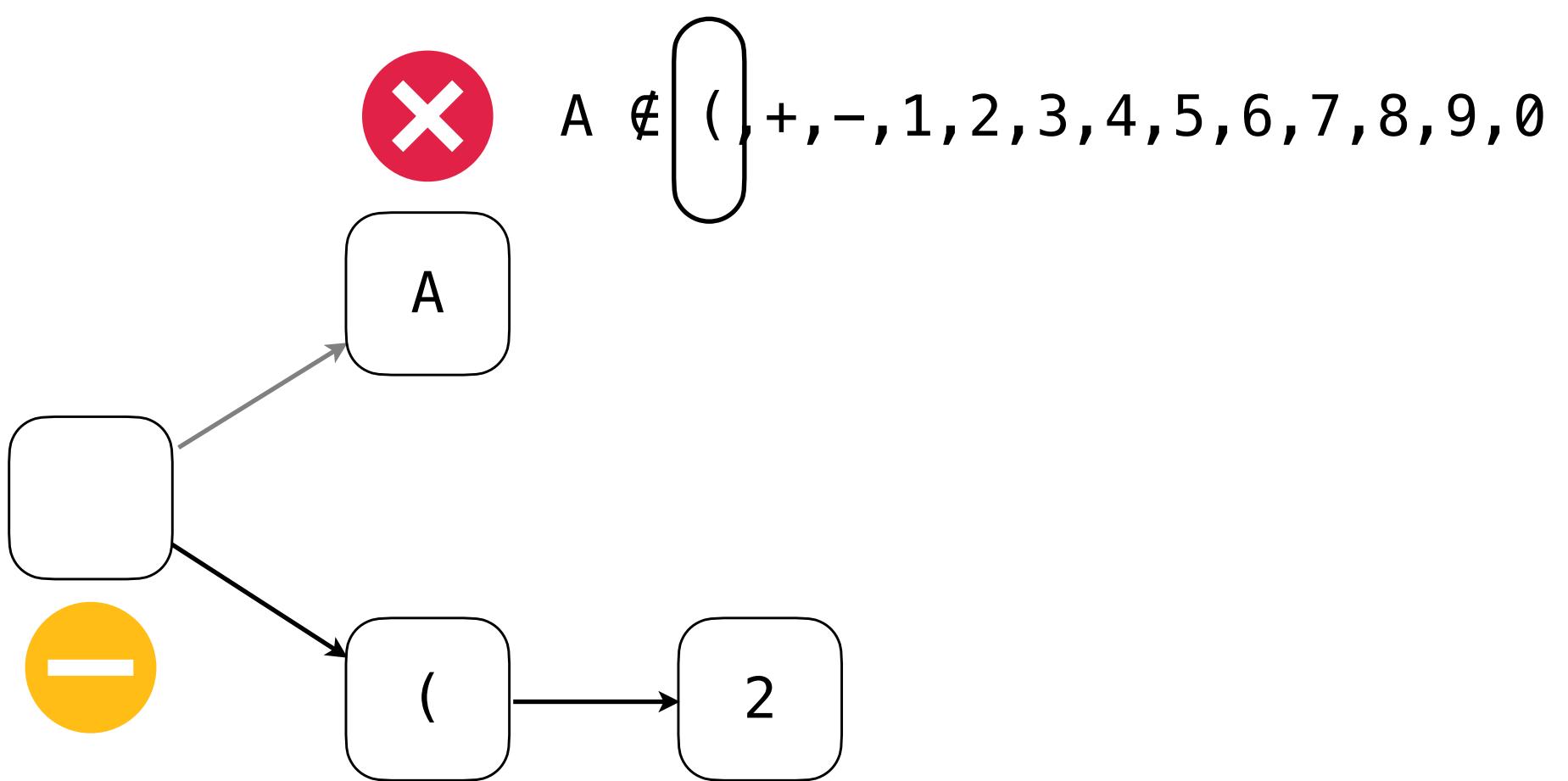
Sepcification Free Generators



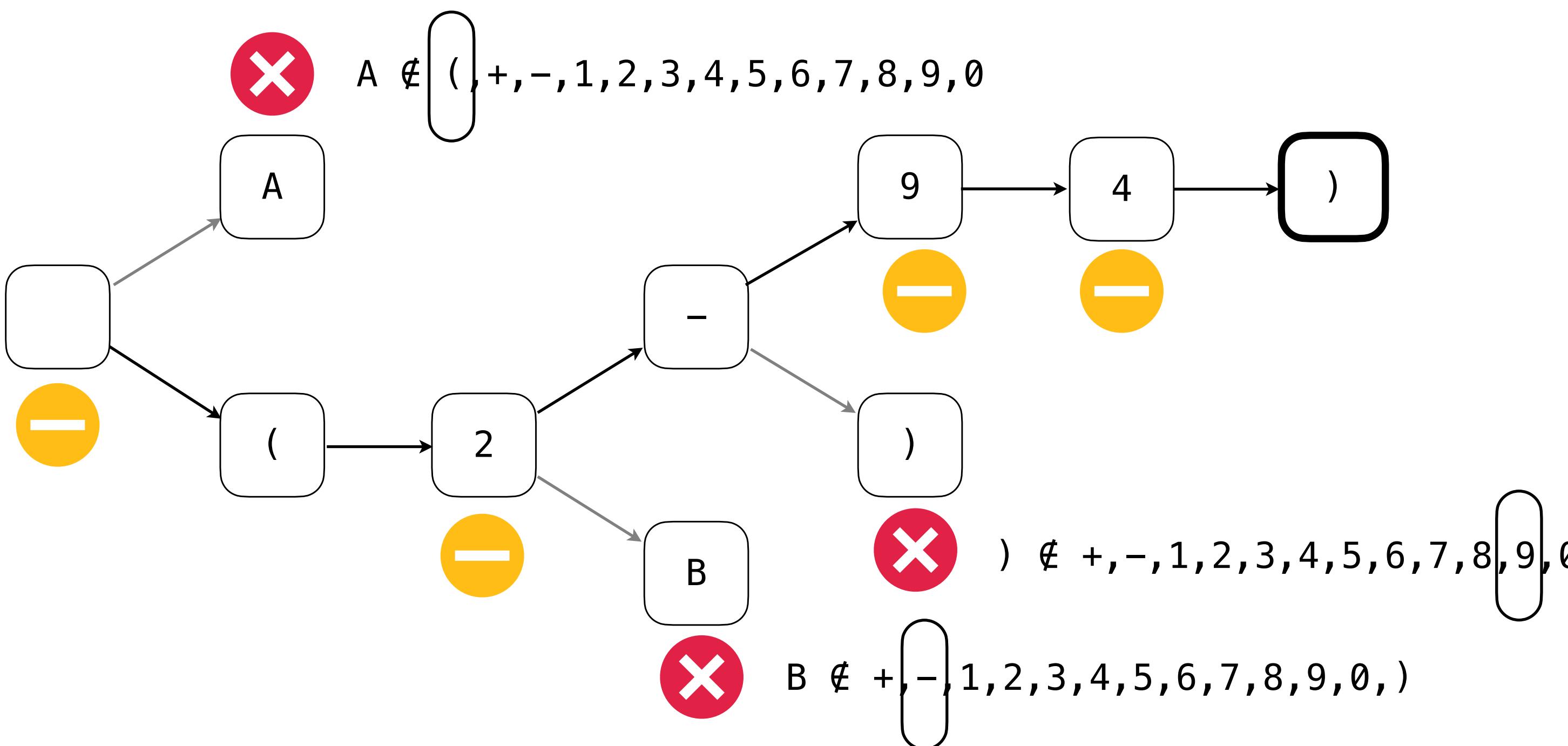
Sepcification Free Generators



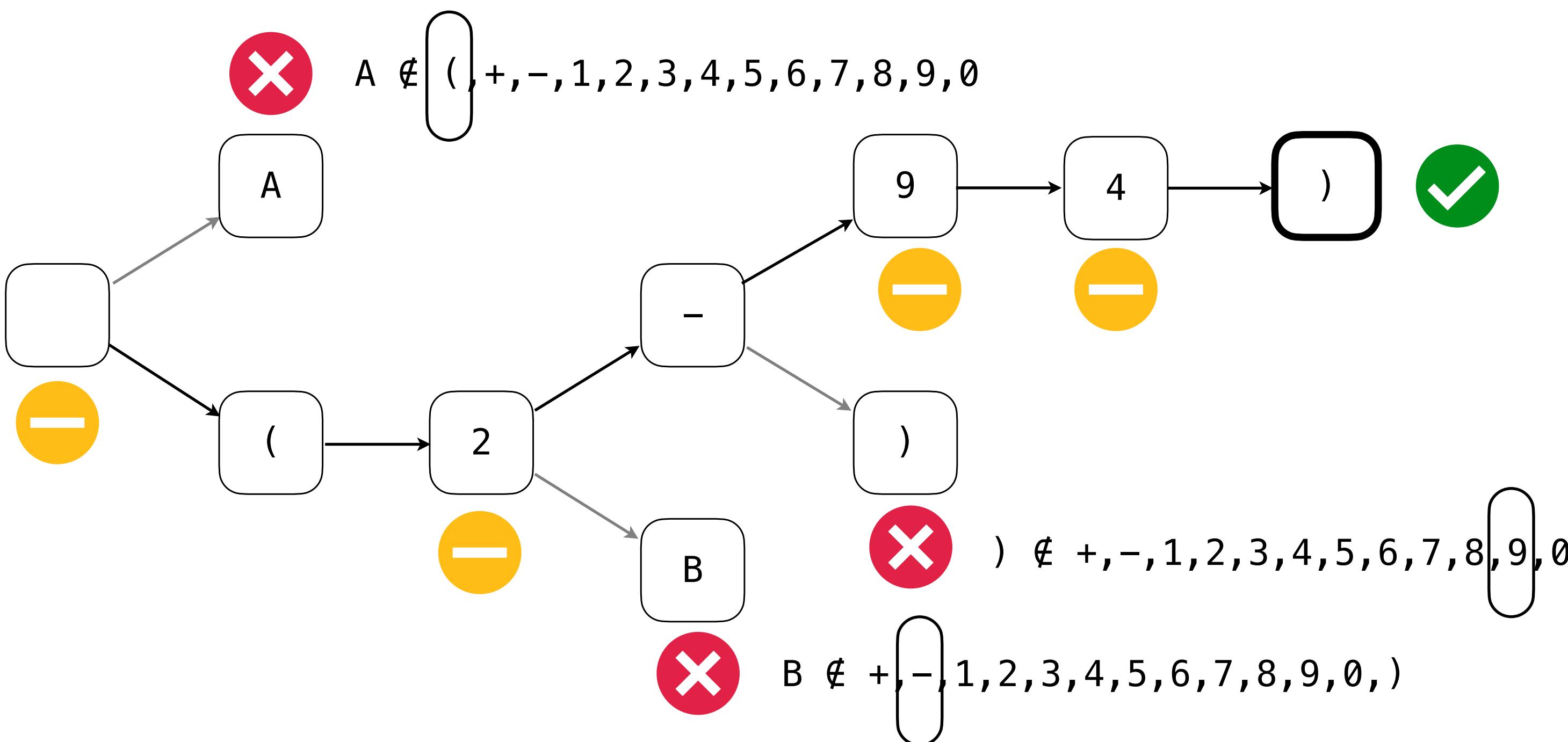
Sepcification Free Generators



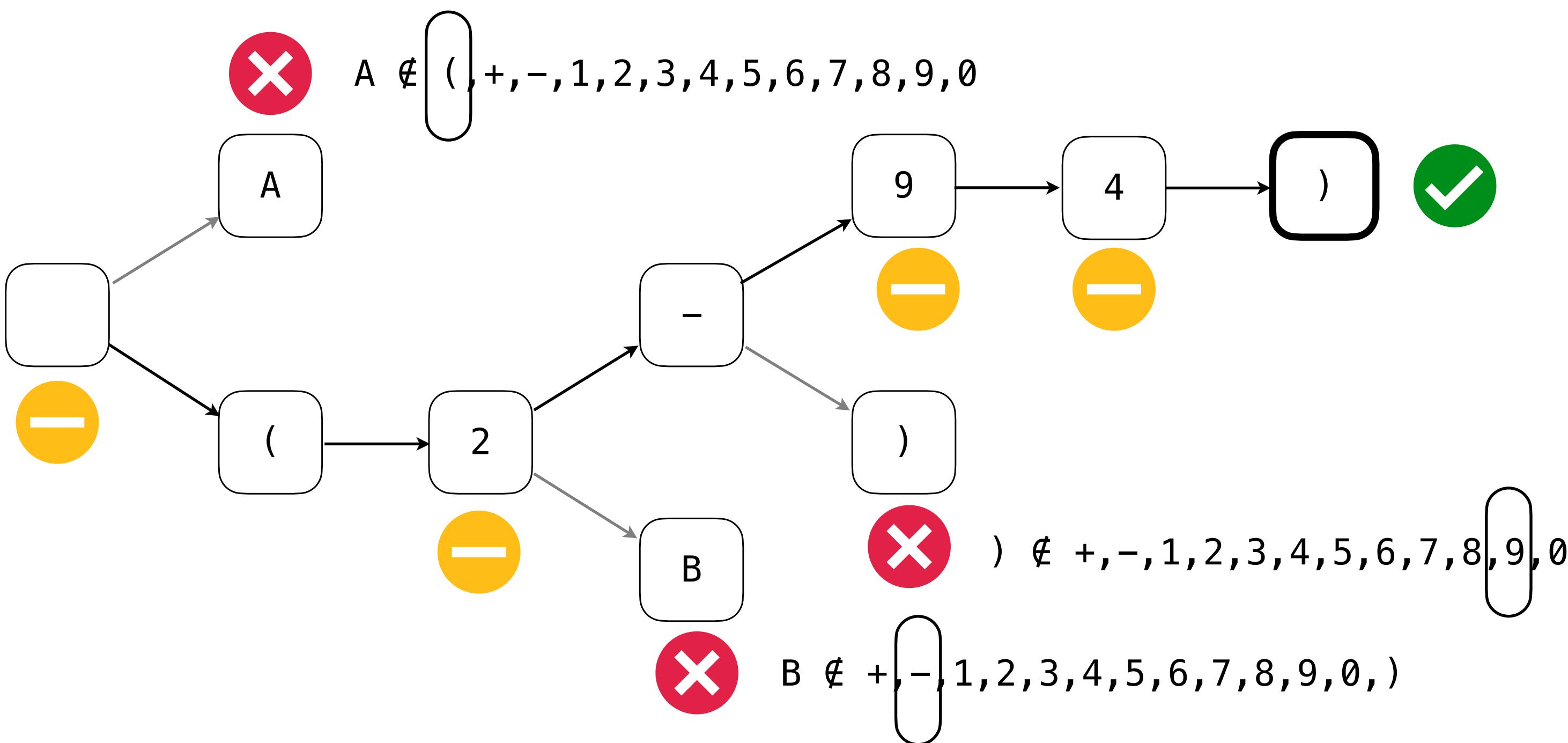
Sepcification Free Generators



Sepcification Free Generators



Sepcification Free Generators



(2-94)

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** localhost:8888/notebooks/x1_1_TrackAccess.ipynb
- Header:** jupyter x1_1_TrackAccess Last Checkpoint: 5 hours ago (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, Help, Trusted, Python 3 (ipykernel)
- Buttons:** Save, New, Cut, Copy, Paste, Run, Stop, Kernel, Markdown, Cell Type, Cell Layout.
- Contents Sidebar:** Shows a tree structure:
 - 1 Track Access
 - 1.1 tstr
 - 1.2 xtstr
 - 1.3 helpers
 - 1.4 calculator.py
 - 1.5 InRewriter
 - 2 Generator
 - 3 Done
- Main Content Area:**

1 Track Access

IMPORTANT. Not all methods are implemented. A more detailed treatment of `tstr` can be found in the FuzzingBook [Tracking Information Flow](#) chapter.

```
In [ ]: import sys
import random
import enum
import src.utils as utils
```

1.1 tstr

`tstr` is a simple proxy for strings. It tracks the origin of any given descendent string.

```
In [ ]: import inspect
import enum

class tstr_(str):
    def __new__(cls, value, *args, **kw):
        return super(tstr_, cls).__new__(cls, value)

    class tstr(tstr_):
        def __init__(self, value, taint=None, parent=None, **kwargs):
            self.parent = parent
            l = len(self)
            if l > 1000:
                self._taint = taint
                self._parent = parent
            else:
                self._taint = None
                self._parent = None
```

Limitation: Lack of control

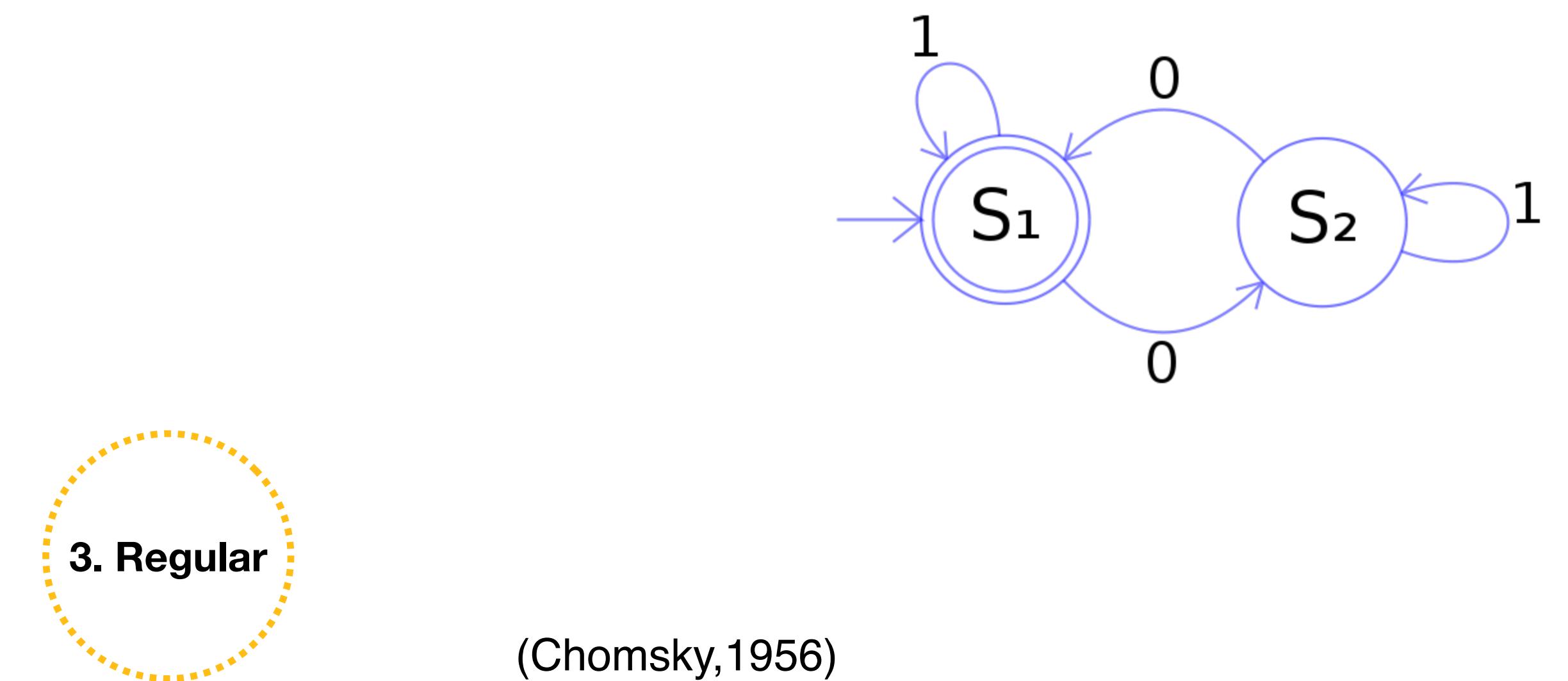
Constraining the Search Space with Input Grammars

Grammar

Formal Languages

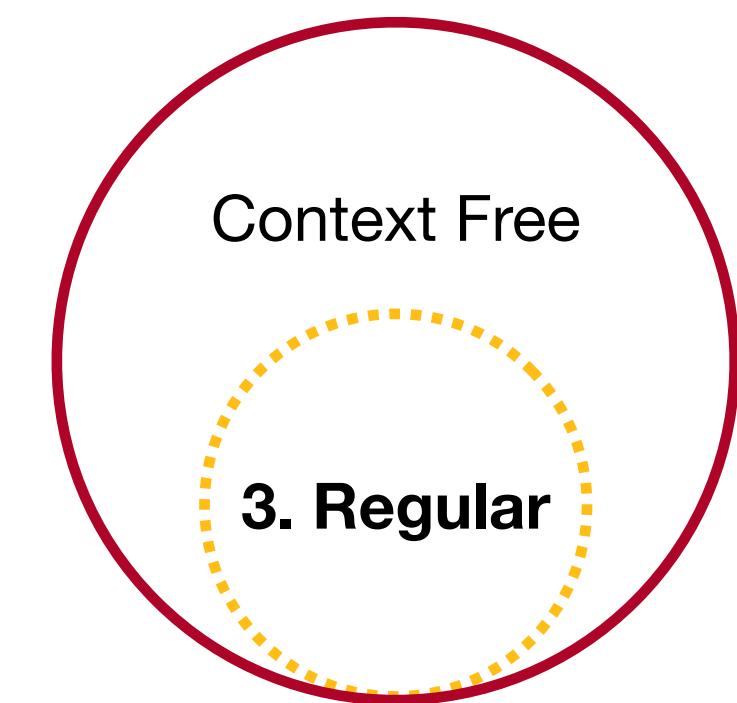
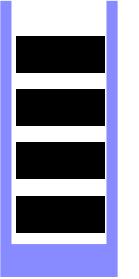
Formal Language Descriptions

Formal Languages

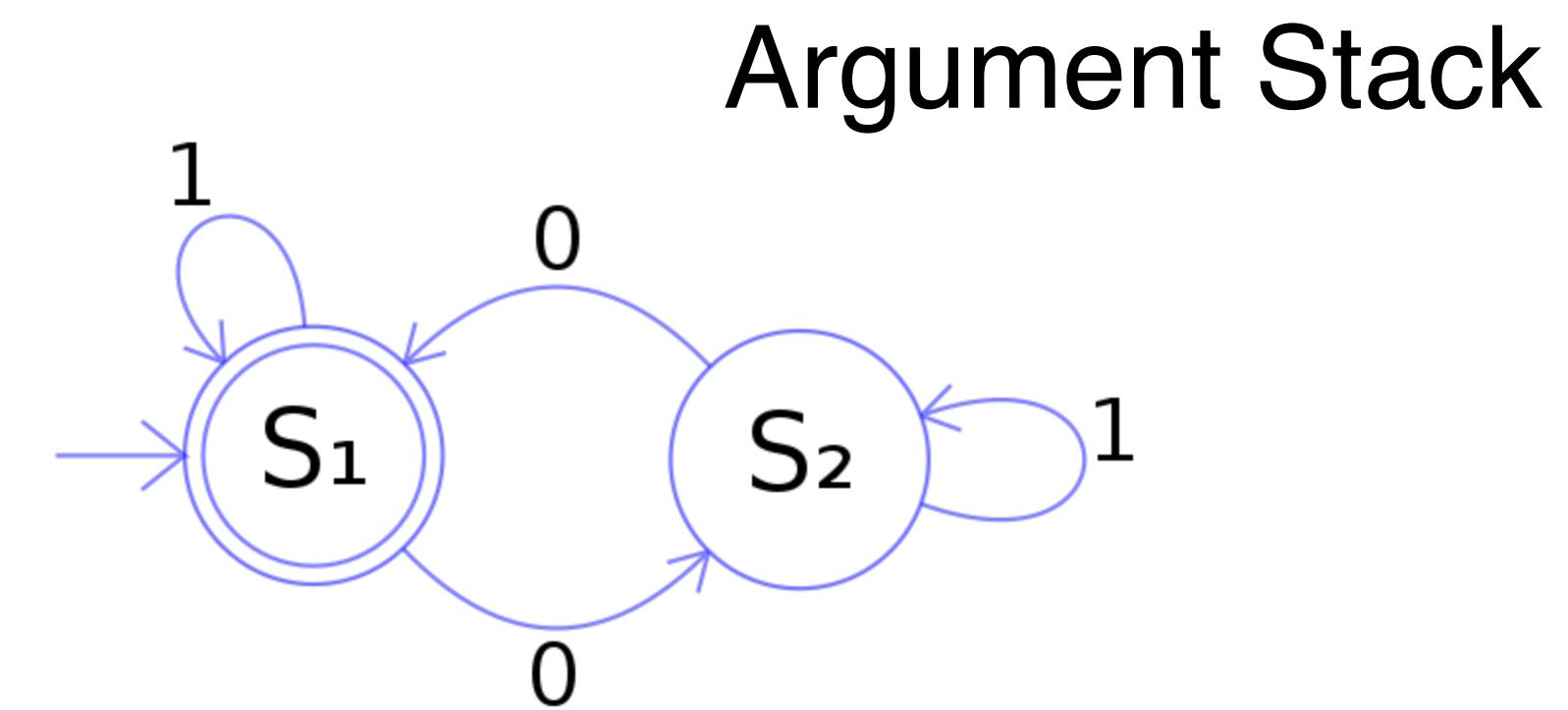


Formal Language Descriptions

Formal Languages

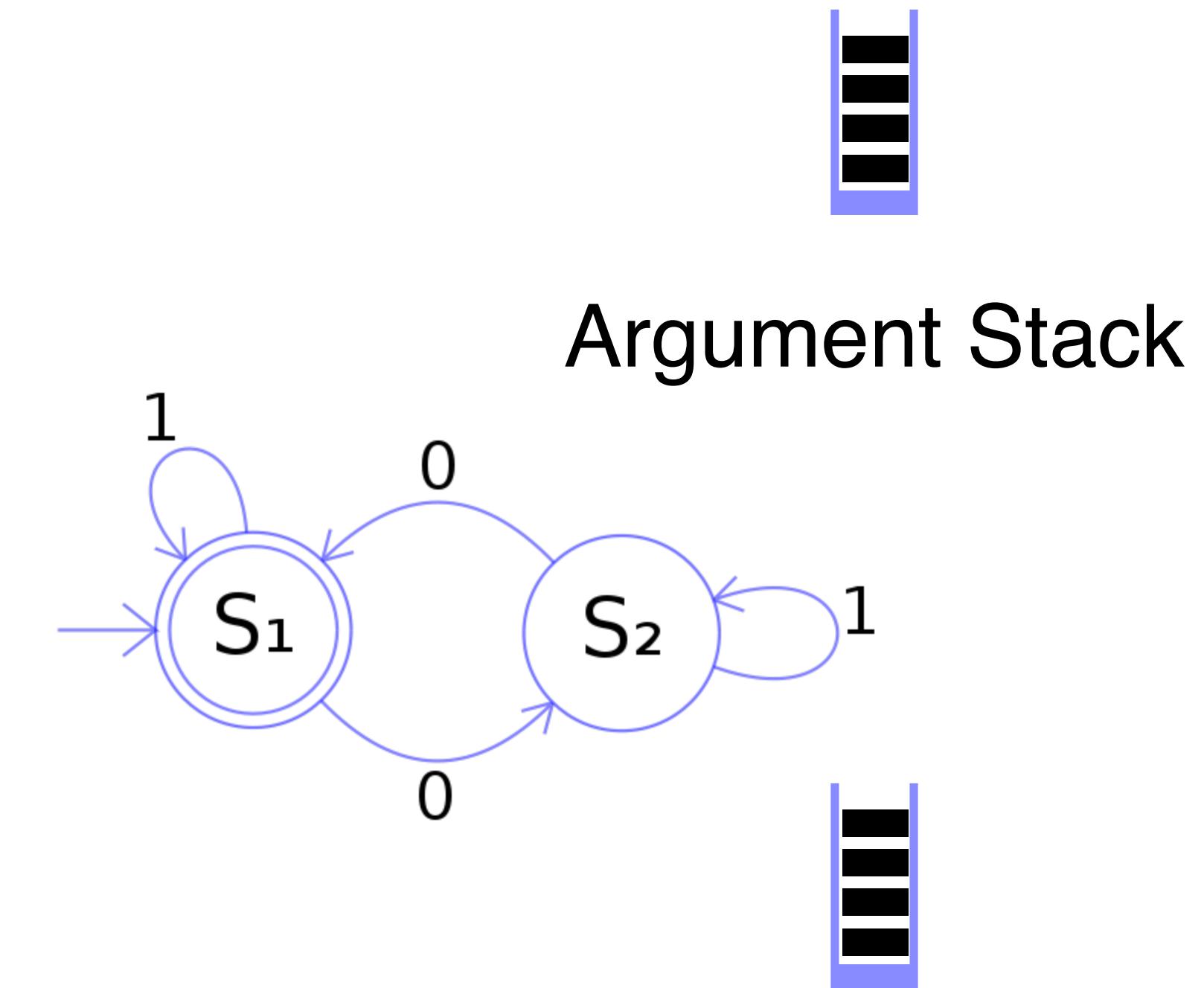
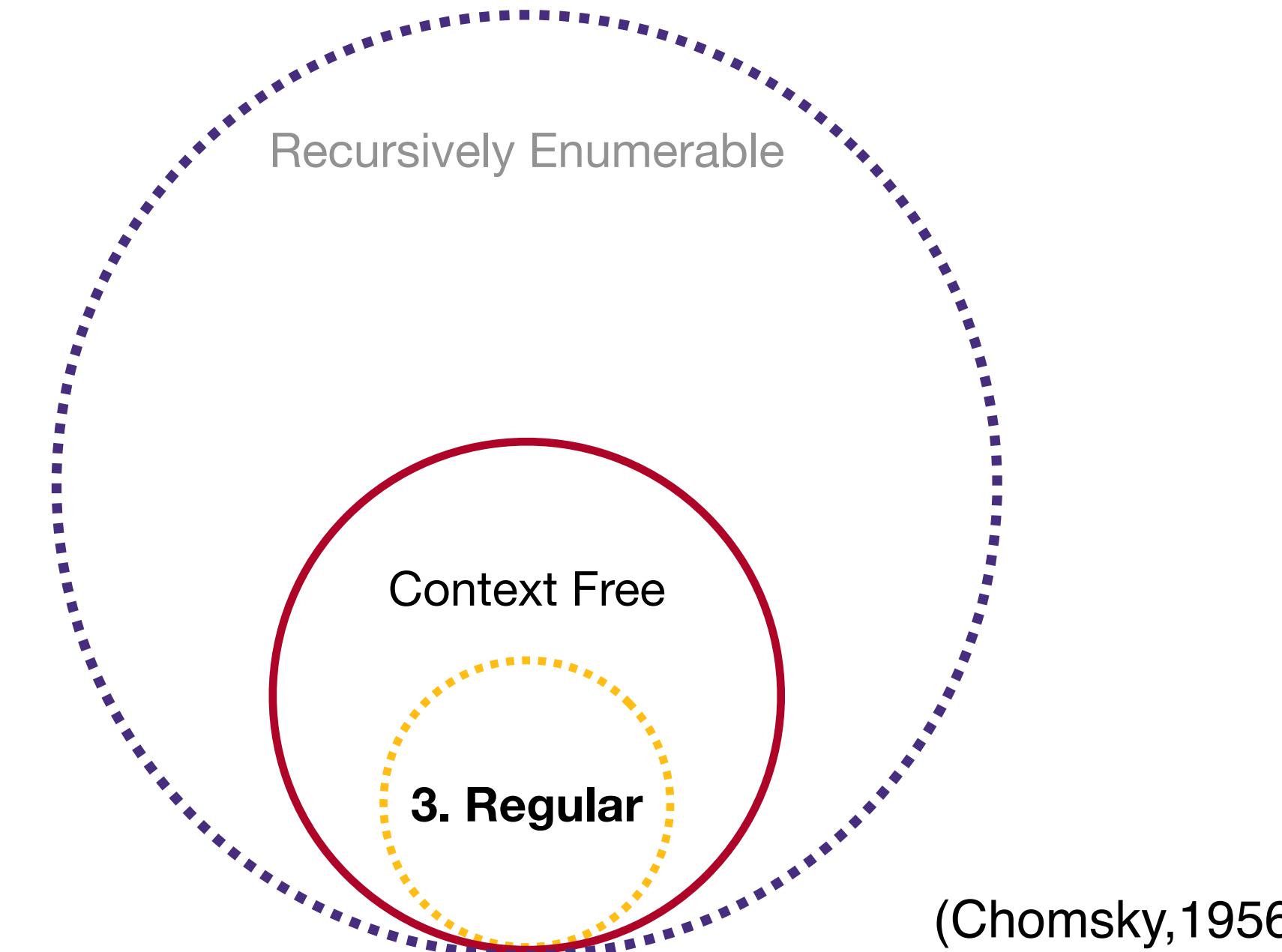


(Chomsky, 1956)



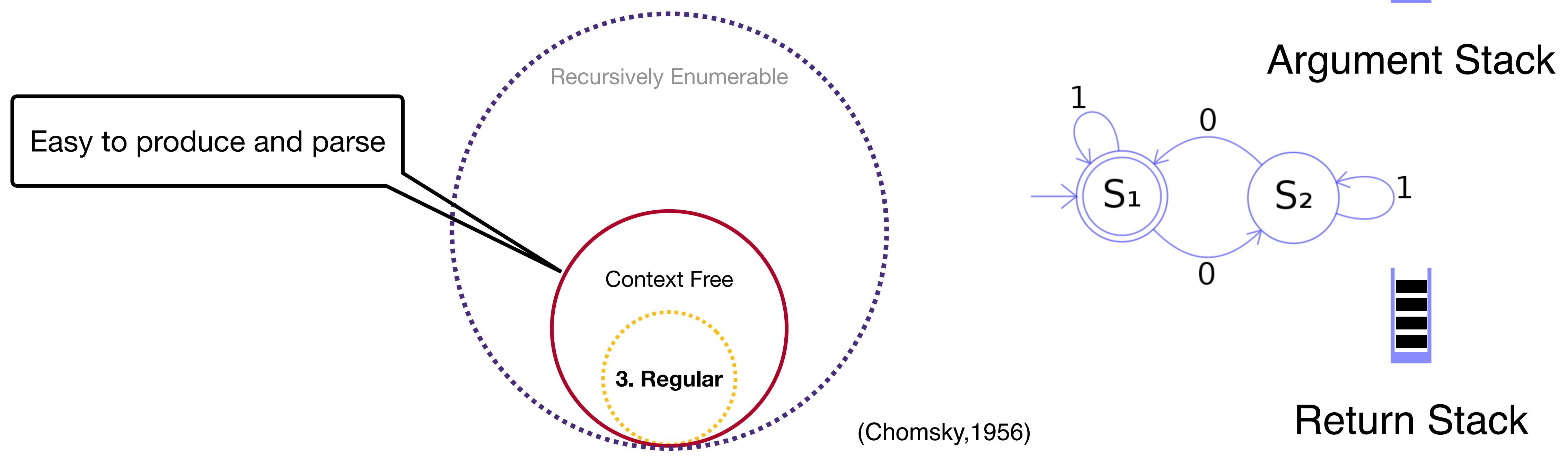
Formal Language Descriptions

Formal Languages



Formal Language Descriptions

Formal Languages



Formal Language Descriptions

Grammar

```
<start>      := <expr>

<expr>       := <expr> '+' <expr>
                  | <expr> '-' <expr>
                  | <expr> '/' <expr>
                  | <expr> '*' <expr>
                  | '(' <expr> ')'
                  | <number>

<number>     := <integer>
                  | <integer> '.' <integer>

<integer>:= <digit> <integer>
                  | <digit>

<digit>   := [0-9]
```

Arithmetic expression grammar

Grammar

`<start> := <expr>`

```
<expr>      := <expr> '+' <expr>
              | <expr> '-' <expr>
              | <expr> '/' <expr>
              | <expr> '*' <expr>
              | '(' <expr> ')'
              | <number>
```

`<number> := <integer>`
 `| <integer> '.' <integer>`

`<integer>:= <digit> <integer>`
 `| <digit>`

`<digit> := [0-9]`

Arithmetic expression grammar

Grammar

`<start> := <expr>`

`<expr> := <expr> '+' <expr>`
| `<expr> '-' <expr>`
| `<expr> '/' <expr>`
| `<expr> '*' <expr>`
| `'(' <expr> ')'`
| `<number>`

`<expr> key`

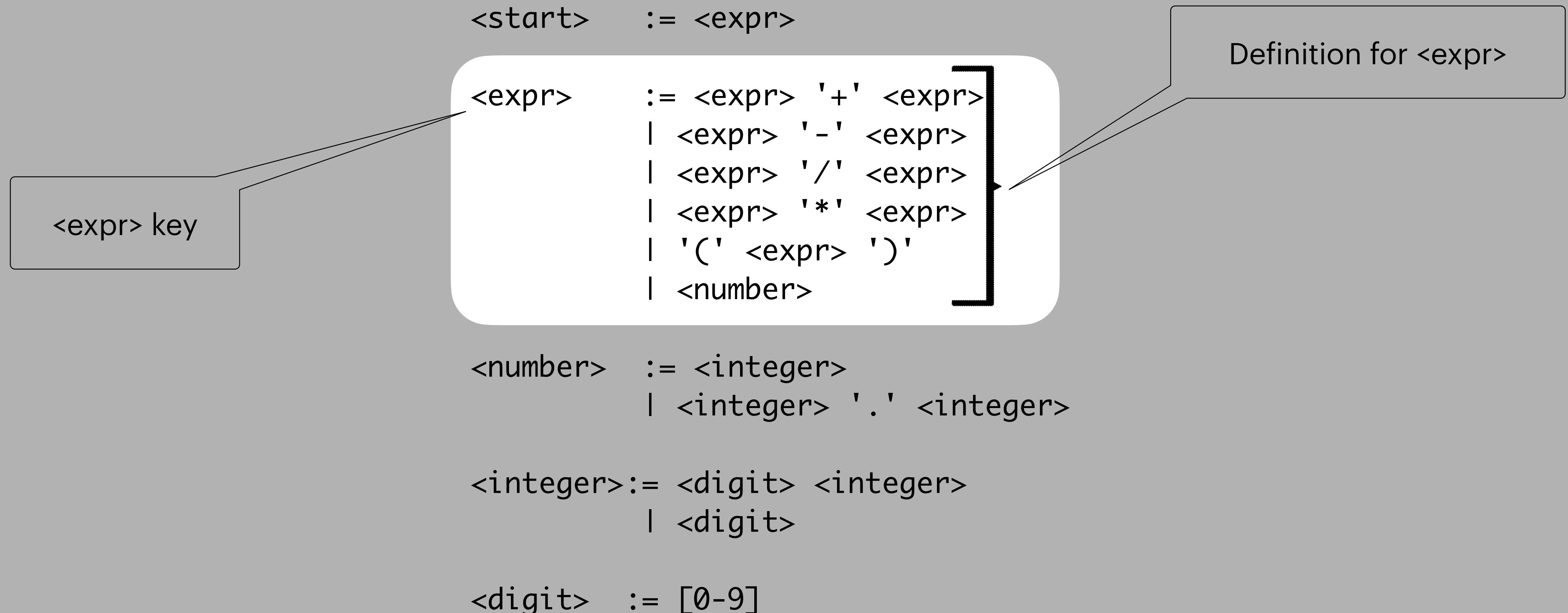
`<number> := <integer>`
| `<integer> '.' <integer>`

`<integer> := <digit> <integer>`
| `<digit>`

`<digit> := [0-9]`

Arithmetic expression grammar

Grammar



Arithmetic expression grammar

Grammar

```
<start>      := <expr>

<expr>       := <expr> '+' <expr>
                  | <expr> '-' <expr>
                  | <expr> '/' <expr>
                  | <expr> '*' <expr>
                  | '(' <expr> ')'
                  | <number>

<number>     := <integer>
                  | <integer> '.' <integer>

<integer>:= <digit> <integer>
                  | <digit>

<digit>      := [0-9]
```

Arithmetic expression grammar

Grammar

`<start> := <expr>`

`<expr> := <expr> '+' <expr>`
| `<expr> '-' <expr>`
| `<expr> '/' <expr>`
| `<expr> '*' <expr>`
| `'(' <expr> ')'`
| `<number>`

Expansion Rule

`<number> := <integer>`
| `<integer> '.' <integer>`

`<integer> := <digit> <integer>`
| `<digit>`

`<digit> := [0-9]`

Arithmetic expression grammar

Grammar

`<start> := <expr>`

`<expr> := <expr> '+' <expr>`
| `<expr> '-' <expr>`
| `<expr> '/' <expr>`
| `<expr> '*' <expr>`
| `'(' <expr> ')'`
| `<number>`

Expansion Rule

Terminal Symbol

`<number> := <integer>`
| `<integer> '.' <integer>`

`<integer> := <digit> <integer>`
| `<digit>`

`<digit> := [0-9]`

Arithmetic expression grammar

Grammar

`<start> := <expr>`

`<expr> := <expr> '+' <expr>`
| `<expr> '-' <expr>`
| `<expr> '/' <expr>`
| `<expr> '*' <expr>`
| `'(' <expr> ')'`
| `<number>`

Expansion Rule

Terminal Symbol

`<number> := <integer>`
| `<integer> '.' <integer>`

Nonterminal Symbol

`<integer> := <digit> <integer>`
| `<digit>`

`<digit> := [0-9]`

Arithmetic expression grammar

Grammar

`<start> := <expr>`

`<expr> := <expr> '+' <expr>`
| `<expr> '-' <expr>`
| `<expr> '/' <expr>`
| `<expr> '*' <expr>`
| `'(' <expr> ')'`
| `<number>`

Expansion Rule

Terminal Symbol

`<number> := <integer>`
| `<integer> '.' <integer>`

Nonterminal Symbol

`<integer> := <digit> <integer>`
| `<digit>`

`<digit> := [0-9]`

Arithmetic expression grammar

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Shows tabs for Home Page, RoadMap, x0_1_Grammars (the active tab), x0_3_Parser..., x0_2_Gramm..., x0_4_Hierarc..., x2_0_Mining..., x3_0_Abstra..., and x4_0_InputAl...
A Python logo icon and Logout button are also present.
- Toolbar:** Includes File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, Help, Trusted, and Python 3 (ipykernel) buttons.
- Contents Sidebar:** Shows a tree structure:
 - 1 Grammars
 - 1.1 Definitons (highlighted)
 - 2 Done
- Main Content Area:** Displays the following text:

1 Grammars

1.1 Definitons

We use the following terms:

 - The *alphabet* is the set all of symbols in the input language. For example, in this post, we use all ASCII characters as alphabet.
 - A *terminal* is a single alphabet symbol. Note that this is slightly different from usual definitions (done here for ease of parsing). (Usually a terminal is a contiguous sequence of symbols from the alphabet. However, both kinds of grammars have a one to one correspondence, and can be converted easily.)
For example, `x` is a terminal symbol.
 - A *nonterminal* is a symbol outside the alphabet whose expansion is *defined* in the grammar using *rules* for expansion.
For example, `<term>` is a nonterminal in the below grammar.
 - A *term* is a terminal or a nonterminal symbol. These are also sometimes called *tokens*. We note that a *token* is also the name used for lexical units in ANTLR. We use both interchangeably when there is no ambiguity.
 - A *rule* is a finite sequence of *terms* (two types of terms: terminals and nonterminals) that describe an expansion of a given terminal.
For example, `[<term>+<expr>]` is one of the expansion rules of the nonterminal `<expr>`.
 - A *definition* is a set of *rules* that describe the expansion of a given nonterminal.
For example, `[[<digit>,<digits>],[<digit>]]` is the definition of the nonterminal `<digits>`.
 - A *context-free grammar* is composed of a set of nonterminals and corresponding definitions that define the structure of the nonterminal.
The grammar given below is an example context-free grammar.
 - A terminal *derives* a string if the string contains only the symbols in the terminal. A nonterminal derives a string if the corresponding definition derives the string. A definition derives the string if one of the rules in the definition derives the string. A rule derives a string if the sequence of terms that make up the rule can derive the string, deriving one substring after another contiguously (also called *parsing*).

http://localhost:8888/notebooks/x0_1_Grammars.ipynb

Grammars

<start> := **<expr>**

<expr> := **<expr>** '+' **<expr>**
| **<expr>** '-' **<expr>**
| **<expr>** '/' **<expr>**
| **<expr>** '*' **<expr>**
| '(' **<expr>** ')'
| **<number>**

<number> := **<integer>**
| **<integer>** '.' **<integer>**

<integer> := **<digit>** **<integer>**
| **<digit>**

<digit> := [0-9]

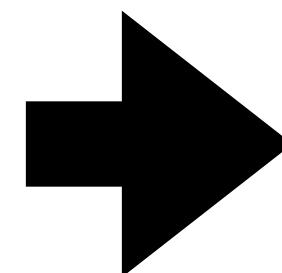
For Parsing

Grammars

<start> := **<expr>**

<expr> := **<expr>** '+' **<expr>**
| **<expr>** '-' **<expr>**
| **<expr>** '/' **<expr>**
| **<expr>** '*' **<expr>**
| '(' **<expr>** ')'
| **<number>**

(8 / 3) * 49



<number> := **<integer>**
| **<integer>** '.' **<integer>**

<integer> := **<digit>** **<integer>**
| **<digit>**

<digit> := [0-9]

For Parsing

Grammars

```

<start>    := <expr>

<expr>     := <expr> '+' <expr>
| <expr> '-' <expr>
| <expr> '/' <expr>
| <expr> '*' <expr>
| '(' <expr> ')'
| <number>

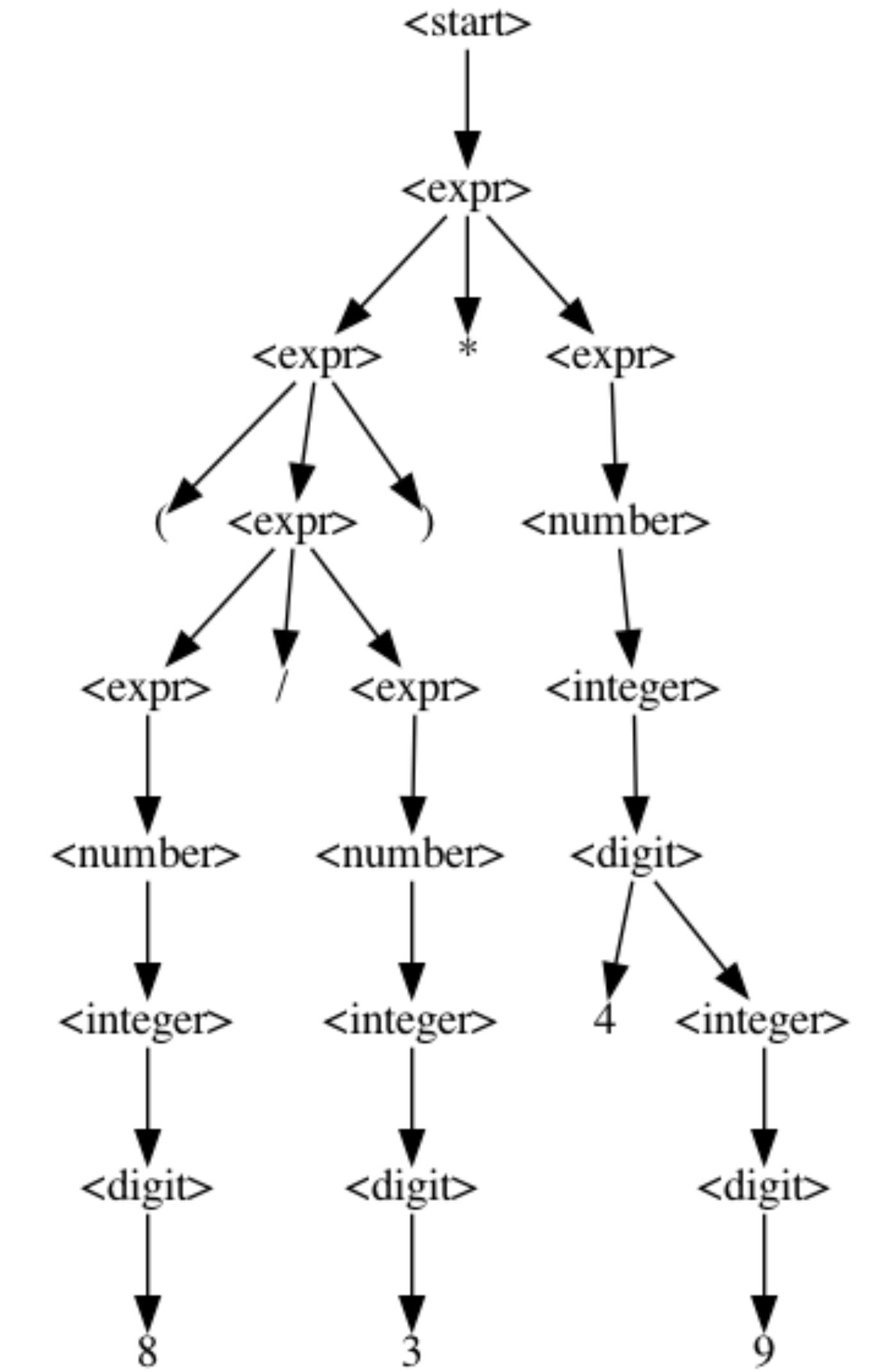
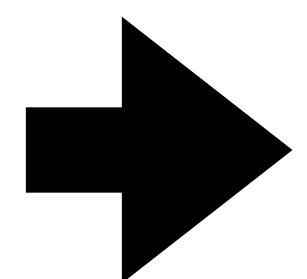
<number>   := <integer>
| <integer> '.' <integer>

<integer>  := <digit> <integer>
| <digit>

<digit>    := [0-9]

```

(8 / 3) * 49



For Parsing

The screenshot shows a Jupyter Notebook interface with the title "jupyter x0_3_Parser Last Checkpoint: 8 hours ago (autosaved)". The notebook has a "Trusted" status and is running on "Python 3 (ipykernel)". The left sidebar contains a "Contents" tree:

- 1 Parser
 - 1.1 Synopsis
 - 1.2 Summary
 - 1.3 The Column Data Structure
 - 1.4 The State Data Structure
 - 1.5 The Basic Parser Interface
 - 1.5.1 Nonterminals Deriving Empty
 - 1.6 The Chart Parser
 - 1.6.1 The Chart Construction
 - 1.6.1.1 Predict
 - 1.6.1.2 Scan
 - 1.6.1.3 Complete
 - 1.6.2 Filling The Chart
 - 1.7 Derivation trees
 - 1.7.1 parse_prefix
 - 1.7.2 parse_on
 - 1.7.3 parse_paths
 - 1.7.4 parse_forest
 - 1.7.5 extract_trees
 - 1.8 Ambiguous Parsing
 - 1.8.1 Example
- 2 Done

The main content area displays the "1 Parser" section:

1 Parser

The Earley parsing algorithm was invented by Jay Earley in 1970. It can be used to parse strings that conform to a context-free grammar. The algorithm uses a chart for parsing -- that is, it is implemented as a dynamic program relying on solving simpler sub-problems.

Earley parsers are very appealing for a practitioner because they can use any context-free grammar for parsing a string, and from the parse forest generated, one can recover all (even an infinite number) of parse trees that correspond to the given grammar.

Note. This notebook does not implement the Leo optimization. A more detailed worked out notebook that explains and implements the Leo optimization can be seen [here](#) from which this notebook has been adapted.

1.1 Synopsis

```
import earleyparser as P
my_grammar = {'<start>': [['1', '<A>'],
                               ['2'],
                               ],
              '<A>': [['a']]}
my_parser = P.EarleyParser(my_grammar)
for tree in my_parser.parse_on(text='1a', start_symbol='<start>'):
    print(P.format_parse_tree(tree))
```

Secondly, as per traditional implementations, there can only be one expansion rule for the `<start>` symbol. We work around this restriction by simply constructing as many charts as there are expansion rules, and returning all parse trees.

In []: `grammar = {<start>: [['<expr>']],
 '<expr>': [
 ['<term>', '+', '<expr>'],
 ['<term>', '-', '<expr>'],
 ['<term>'],
 '<term>': [
 ['+', '<expr>'],
 ['-',`

Grammars

`<start> := <expr>`

`<expr> := <expr> '+' <expr>`
| `<expr> '-' <expr>`
| `<expr> '/' <expr>`
| `<expr> '*' <expr>`
| `'(' <expr> ')'`
| `<number>`

`<number> := <integer>`
| `<integer> '.' <integer>`

`<integer> := <digit> <integer>`
| `<digit>`

`<digit> := [0-9]`

For Fuzzing

Grammars

```

<start>    := <expr>

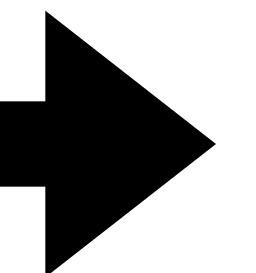
<expr>     := <expr> '+' <expr>
             | <expr> '-' <expr>
             | <expr> '/' <expr>
             | <expr> '*' <expr>
             | '(' <expr> ')'
             | <number>

<number>   := <integer>
             | <integer> '.' <integer>

<integer>  := <digit> <integer>
             | <digit>

<digit>    := [0-9]

```



8.2 - 27 - -9 / +((+9 * --2 + ---+-
((-1 * +(8 - 5 - 6)) * (-a-+(((+(4)
)))) - ++4) / +(-+---((5.6 - --(3 *
-1.8 * +(6 * +-(((--6) * ----+6)) /
+---(+--+7 * (-0 * (+((((2)) + 8 - 3
- ++9.0 + ---(--+7 / (1 / +++6.37)
+ (1) / 482) / +--+0))) + 8.2 - 27
- -9 / +((+9 * --2 + ---+-((-1 * +
(8 - 5 - 6)) * (-a-+(((+(4)))))) - +
+4) / +(-+---((5.6 - --(3 * -1.8 * +
(6 * +-(((--6) * ----+6)) / +---(+--+
7 * (-0 * (+((((2)) + 8 - 3 - ++9.0
+ ---(--+7 / (1 / +++6.37) + (1) /
482) / +--+0))) * -+5 + 7.513)))
- (+1 / ++((-84))))))) * ++5 / +-(-
-2 - ---9.0))) / 5 * ---+090 + * -
+5 + 7.513))) - (+1 / ++((-84)))))))
)) * 8.2 - 27 - -9 / +((+9 * --2 + -
---+((-1 * +(8 - 5 - 6)) * (-a-+((
+(4)))) - ++4) / +(-+---((5.6 - --
(3 * -1.8 * +(6 * +-(((--6) * ----+6
)) / +---(+--+7 * (-0 * (+((((2)) +
8 - 3 - ++9.0 + ---(--+7 / (1 / +++6
.37) + (1) / 482) / +--+0))) * -+5
+ 7.513))) - (+1 / ++((-84)))))))
* ++5 / +-(-2 - ---9.0))) / 5 *
---+090 ++5 / +-(-2 - ---9.0))) /
5 * ---+090

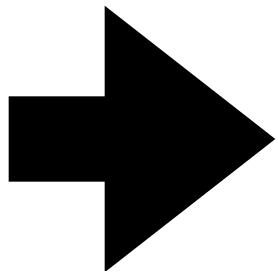
For Fuzzing

```

8.2 - 27 - -9 / +((+9 * --2 + ---+-
((-1 * +(8 - 5 - 6)) * (-a-+(((+(4)
)))) - ++4) / +(-+---((5.6 - --(3 *
-1.8 * +(6 * +-(((--6) * ----+6)) /
+--+(-+-7 * (-0 * (+((((2)) + 8 - 3
- ++9.0 + ---(--+7 / (1 / +++6.37)
+ (1) / 482) / +--+0))) + 8.2 - 27
- -9 / +((+9 * --2 + ---+-((-1 * +
(8 - 5 - 6)) * (-a-+(((+(4)))))) - +
+4) / +(-+---((5.6 - --(3 * -1.8 * +
(6 * +-(((--6) * ----+6)) / +--+(
7 * (-0 * (+((((2)) + 8 - 3 - ++9.0
+ ---(--+7 / (1 / +++6.37) + (1) /
482) / +--+0))) * -+5 + 7.513))) )
- (+1 / ++((-84))))))) * ++5 / +-(
-2 - -++9.0))) / 5 * ---+090 + * -
+5 + 7.513))) - (+1 / ++((-84)))))))
) * 8.2 - 27 - -9 / +((+9 * --2 + -
---+-((-1 * +(8 - 5 - 6)) * (-a-+(
+(4)))) - ++4) / +(-+---((5.6 - --
(3 * -1.8 * +(6 * +-(((--6) * ----+6
))) / +--+(-+-7 * (-0 * (+((((2)) +
8 - 3 - ++9.0 + ---(--+7 / (1 / +++6
.37) + (1) / 482) / +--+0))) * -+5
+ 7.513))) - (+1 / ++((-84)))))))
* ++5 / +-(-2 - -++9.0))) / 5 *
---+090 ++5 / +-(-2 - -++9.0))) /
5 * ---+090

```

Grammars



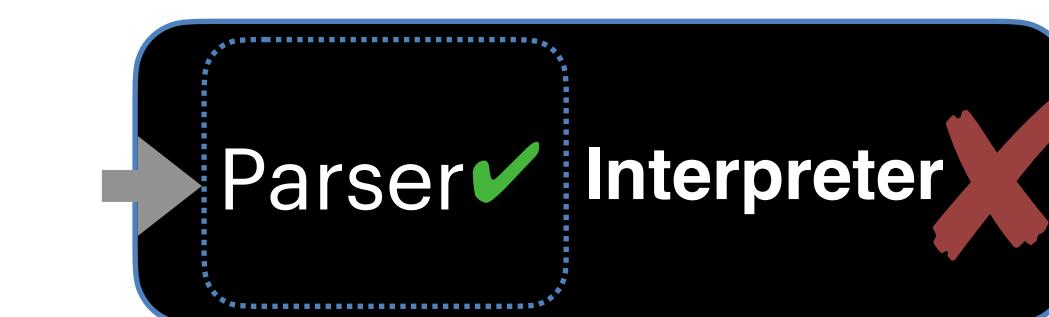
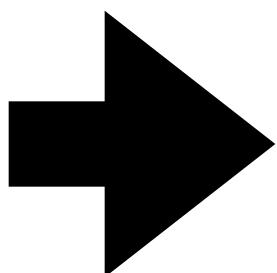
As effective producers

```

8.2 - 27 - -9 / +((+9 * --2 + ---+-
((-1 * +(8 - 5 - 6)) * (-a-+(((+(4)
)))) - ++4) / +(-+---((5.6 - --(3 *
-1.8 * +(6 * +-(((--6) * ----+6)) /
+--+(-+-7 * (-0 * (+((((2)) + 8 - 3
- ++9.0 + ---(--+7 / (1 / +++6.37)
+ (1) / 482) / +--+0))) + 8.2 - 27
- -9 / +((+9 * --2 + ---+-((-1 * +
(8 - 5 - 6)) * (-a-+(((+(4)))))) - +
+4) / +(-+---((5.6 - --(3 * -1.8 * +
(6 * +-(((--6) * ----+6)) / +--+(
7 * (-0 * (+((((2)) + 8 - 3 - ++9.0
+ ---(--+7 / (1 / +++6.37) + (1) /
482) / +--+0))) * -+5 + 7.513))) )
- (+1 / ++((-84))))))) * ++5 / +-(-
-2 - --+-9.0))) / 5 * ---+090 + * -
+5 + 7.513))) - (+1 / ++((-84)))))))
) * 8.2 - 27 - -9 / +((+9 * --2 + -
---+-((-1 * +(8 - 5 - 6)) * (-a-+(
+(4)))) - ++4) / +(-+---((5.6 - --
(3 * -1.8 * +(6 * +-(((--6) * ----+6
))) / +--+(-+-7 * (-0 * (+((((2)) +
8 - 3 - ++9.0 + ---(--+7 / (1 / +++6
.37) + (1) / 482) / +--+0))) * -+5
+ 7.513))) - (+1 / ++((-84)))))))
* ++5 / +-(-2 - --+-9.0))) / 5 *
---+090 ++5 / +-(-2 - --+-9.0))) /
5 * ---+090

```

Grammars



As effective producers

Grammars

<start> := <expr>

<expr> := <expr> '+' <expr>
| <expr> '-' <expr>
| <expr> '/' <expr>
| <expr> '*' <expr>
| '(' <expr> ')'
| <number>

<number> := <integer>
| <integer> '.' <integer>

<integer> := <digit> <integer>
| <digit>

<digit> := [0-9]

As efficient producers

Grammars

```

<start>    := <expr>

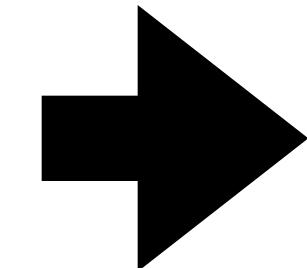
<expr>     := <expr> '+' <expr>
| <expr> '-' <expr>
| <expr> '/' <expr>
| <expr> '*' <expr>
| '(' <expr> ')'
| <number>

<number>   := <integer>
| <integer> '.' <integer>

<integer> := <digit> <integer>
| <digit>

<digit>    := [0-9]

```



Compiled Grammar (F1)

```

def start():
    expr()

def expr():
    match (random() % 6):
        case 0: expr(); print('+'); expr()
        case 1: expr(); print('-'); expr()
        case 2: expr(); print('/'); expr()
        case 3: expr(); print('*'); expr()
        case 4: print('('); expr(); print(')')
        case 5: number()

def number():
    match (random() % 2):
        case 0: integer()
        case 1: integer(); print('.'); integer()

def integer():
    match (random() % 2):
        case 0: digit(); integer()
        case 1: digit()

def digit():
    match (random() % 10):
        case 0: print('0')
        case 1: print('1')
        case 2: print('2')
        case 3: print('3')
        case 4: print('4')
        case 5: print('5')
        case 6: print('6')
        case 7: print('7')

```

As efficient producers

The screenshot shows a Jupyter Notebook interface with the title "x0_2_GrammarFuzzer.ipynb". The notebook contains the following content:

1 Grammar Fuzzer

This is a fairly simple grammar fuzzer. For a detailed treatment, please see my blog [here](#).

```
In [ ]: import string
import src.utils as utils
import random
```

```
In [ ]: import string
import src.grammars as grammars
```

We allow the grammar to contain a few lexical definitions.

```
In [ ]: ASCII_MAP = {
    '[__WHITESPACE__]': string.whitespace,
    '[__DIGIT__]': string.digits,
    '[__ASCII_LOWER__]': string.ascii_lowercase,
    '[__ASCII_UPPER__]': string.ascii_uppercase,
    '[__ASCII_PUNCT__]': string.punctuation,
    '[__ASCII LETTER__]': string.ascii_letters,
    '[__ASCII_ALPHANUM__]': string.ascii_letters + string.digits,
    '[__ASCII_PRINTABLE__]': string.printable
}
FUZZRANGE = 10
```

1.1 The fuzzer interface

Here is the fuzzer interface

```
In [ ]: class Fuzzer:
    def __init__(self, grammar):
        self.grammar = grammar

    def fuzz(self, key='<start>', max_num=None, max_depth=None):
        raise NotImplemented()
```

Where to Get the Input Grammar From?

The screenshot shows a Jupyter Notebook interface with the title bar "localhost:8888/notebooks/x2_0_MiningGrammar.ipynb". The notebook is titled "Jupyter x2_0_MiningGrammar" and indicates it was last checked 9 hours ago (autosaved). The top menu includes File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, and Help. Below the menu is a toolbar with various icons for file operations like Open, Save, and Run. The status bar shows "Trusted" and "Python 3 (ipykernel)".

The left sidebar contains a "Contents" tree:

- 1 Mining Grammars
 - 1.1 Imports
 - 1.2 Subject Registry
 - 1.3 Context Managers
 - 1.3.1 Method context
 - 1.3.2 Stack context
 - 1.3.3 Scope context
 - 1.4 Rewriting the source to track context
 - 1.4.1 Rewriter
 - 1.4.1.1 The method context wrapper
 - 1.4.1.2 The stack wrapper
 - 1.4.1.3 The scope wrapper
 - 1.4.1.4 Rewriting `If` conditions
 - 1.4.1.5 Rewriting `while` loops
 - 1.4.1.6 Combining both
 - 1.4.2 Generating the complete instruction
 - 1.4.2.1 Using It
 - 1.4.3 Generate Transformed Source
 - 1.5 Generating Traces
 - 1.6 Mining the Traces Generated
 - 1.6.1 Reconstructing the Method Tree
 - 1.6.1.1 Identifying last comparison
 - 1.6.1.2 Attaching characters to the tree
 - 1.6.1.3 Removing Overlap
 - 1.6.1.4 Generate derivation tree
 - 1.6.2 The Complete Miner
 - 1.7 Generalize Nodes
 - 1.7.1 Generalize Methods
 - 1.7.2 Generalize Loops
 - 1.8 Generating a Grammar
 - 1.8.1 Trees to grammar
 - 1.8.2 Inserting Empty Alternatives for the grammar
 - 1.8.3 Learning Regular Expressions
 - 1.8.3.1 The modified Fernau algorithm
 - 1.8.4 Remove duplicate and redundant nodes
 - 1.9 Accio Grammar

The main content area displays the "1 Mining Grammars" section:

1 Mining Grammars

Please see the [Mimid notebook](#) for examples of mining grammars for a number of grammars.

1.1 Imports

```
In [ ]: import src.utils as utils
```

```
In [ ]: import ast
import sys
```

```
In [ ]: import random
```

```
In [ ]: random.seed(0)
```

```
In [ ]: import json
```

1.2 Subject Registry

We store all our subject programs in `program_src`.

```
In [ ]: program_src = {
    'calculator.py': utils.slurp('subjects/calculator.py'),
    'microjson.py': utils.slurp('subjects/microjson.py')
}
```

1.3 Context Managers

The context managers are probes inserted into the source code so that we know when execution enters and exits specific control flow structures such as conditionals and loops. Note that source code for these probes are not really a requirement. They can be inserted directly on binaries too, or even dynamically inserted using tools such as `dtrace`. For now, we make our life simple using AST editing.

Where to Get the Grammar From?

Hand-written parsers already encode the grammar

Where to Get the Grammar From?

Hand-written parsers already encode the grammar

1. Extract the input string accesses
2. Attach control flow information (context-managers)

How to Extract This Grammar?

How to Extract This Grammar?

- Inputs + control flow -> Dynamic Control Dependence Trees

How to Extract This Grammar?

- Inputs + control flow -> Dynamic Control Dependence Trees
- DCD Trees -> Parse Tree

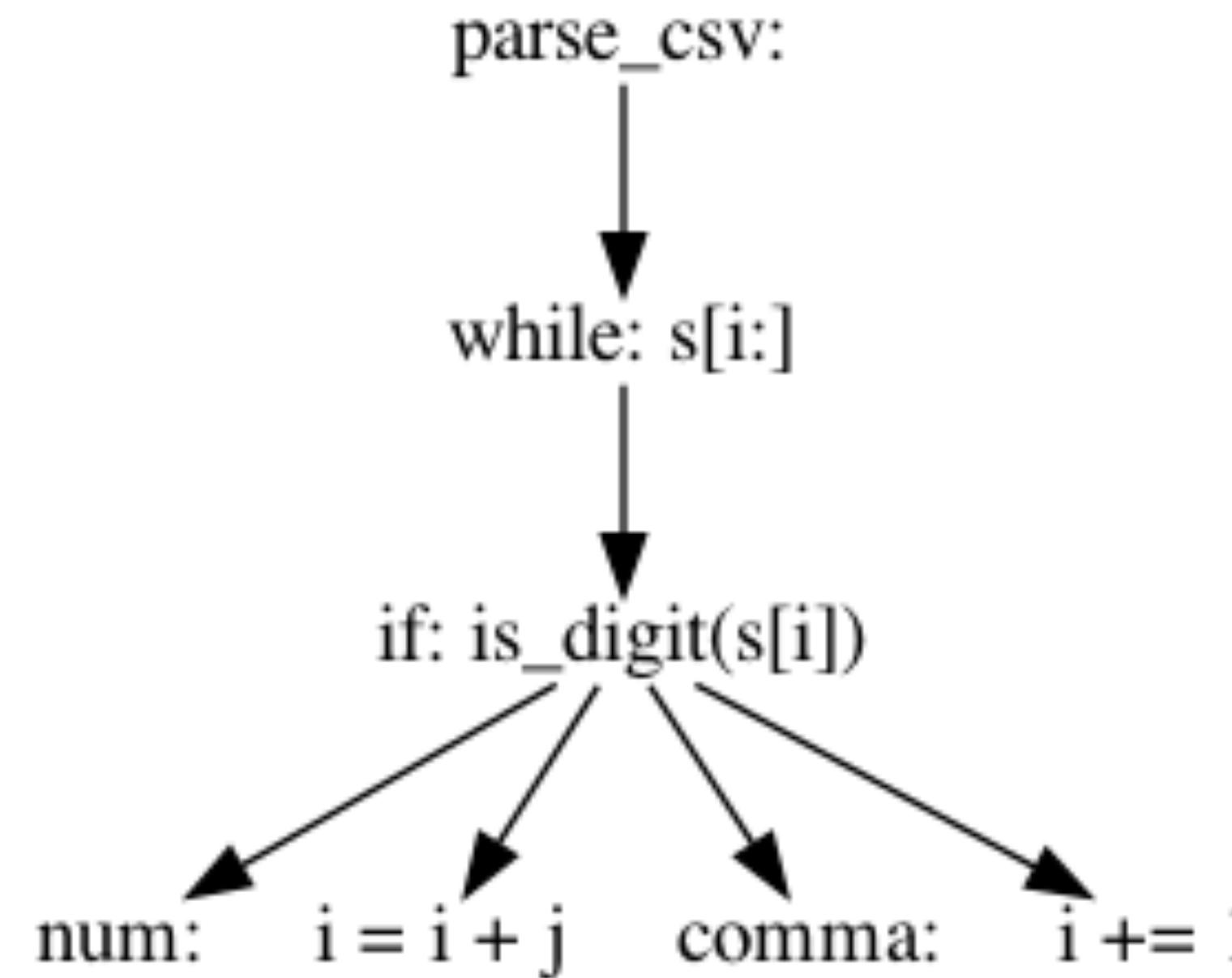
Control Dependence Graph

```
def parse_csv(s,i):
    while s[i:]:
        if is_digit(s[i]):
            n,j = num(s[i:])
            i = i+j
        else:
            comma(s[i])
            i += 1
```

Statement **B** is control dependent on **A** if A determines whether B executes.

Control Dependence Graph

```
def parse_csv(s,i):
    while s[i:]:
        if is_digit(s[i]):
            n,j = num(s[i:])
            i = i+j
        else:
            comma(s[i])
            i += 1
```

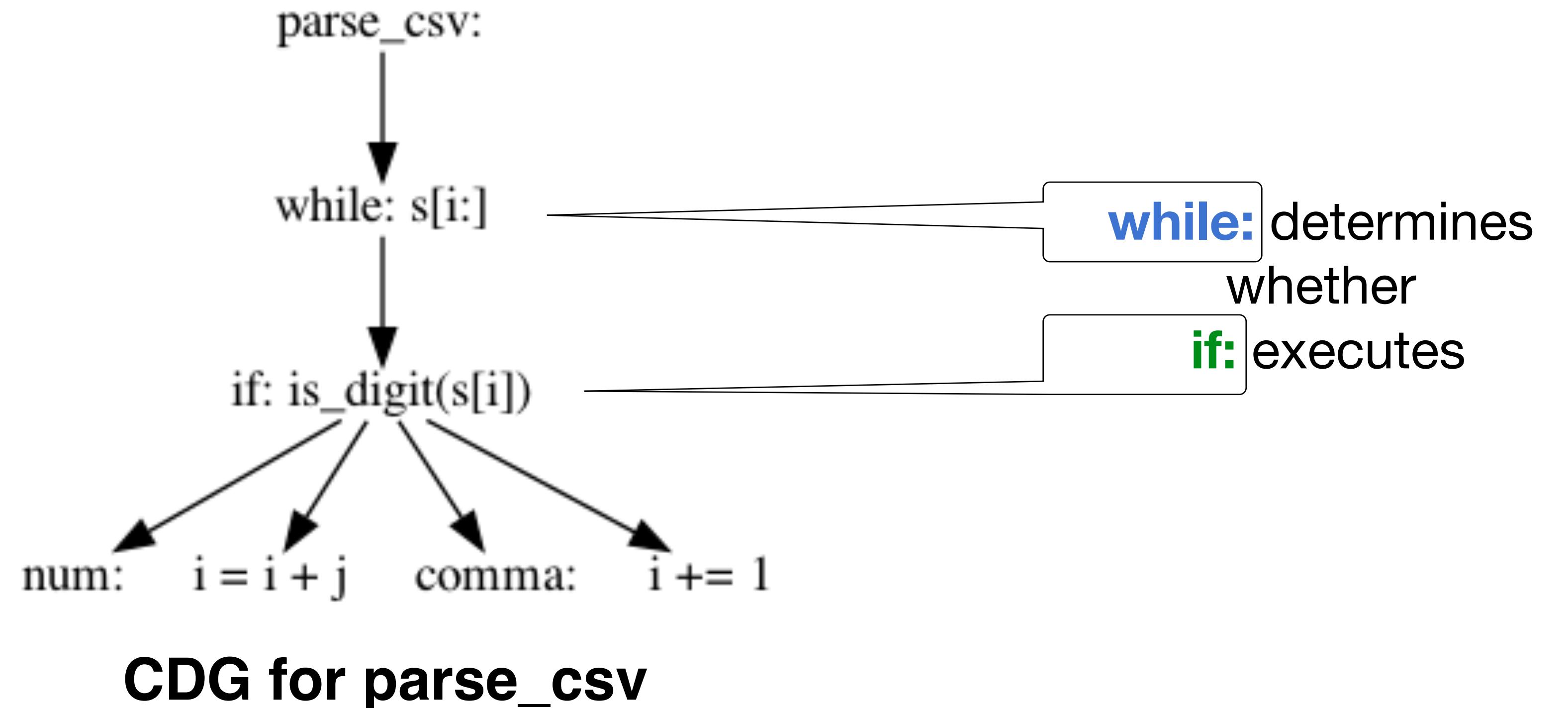


CDG for parse_csv

Statement **B** is control dependent on **A** if A determines whether B executes.

Control Dependence Graph

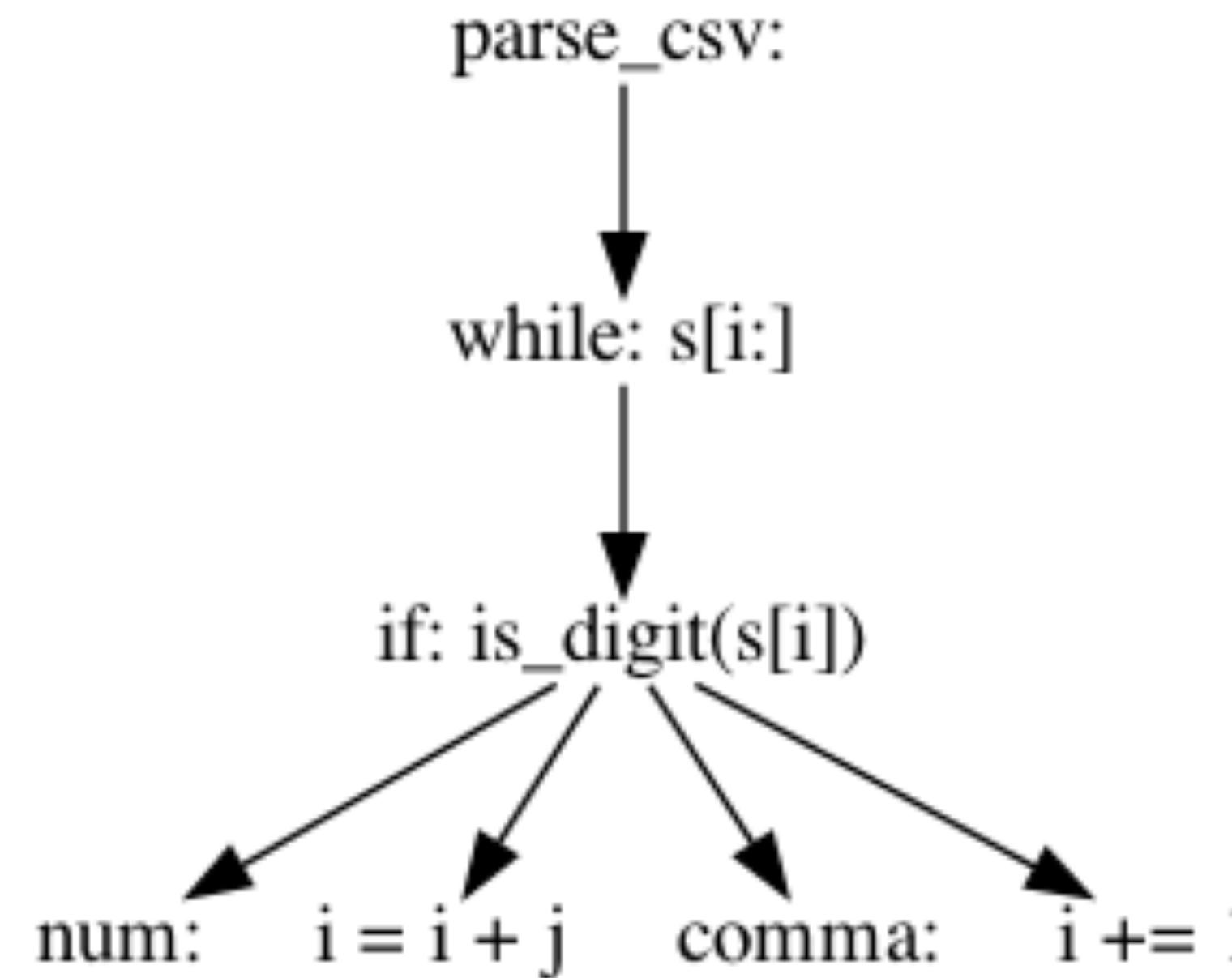
```
def parse_csv(s,i):
    while s[i:]:
        if is_digit(s[i]):
            n,j = num(s[i:])
            i = i+j
        else:
            comma(s[i])
            i += 1
```



Statement **B** is control dependent on **A** if A determines whether B executes.

Dynamic Control Dependence Tree

```
def parse_csv(s,i):
    while s[i:]:
        if is_digit(s[i]):
            n,j = num(s[i:])
            i = i+j
        else:
            comma(s[i])
            i += 1
```

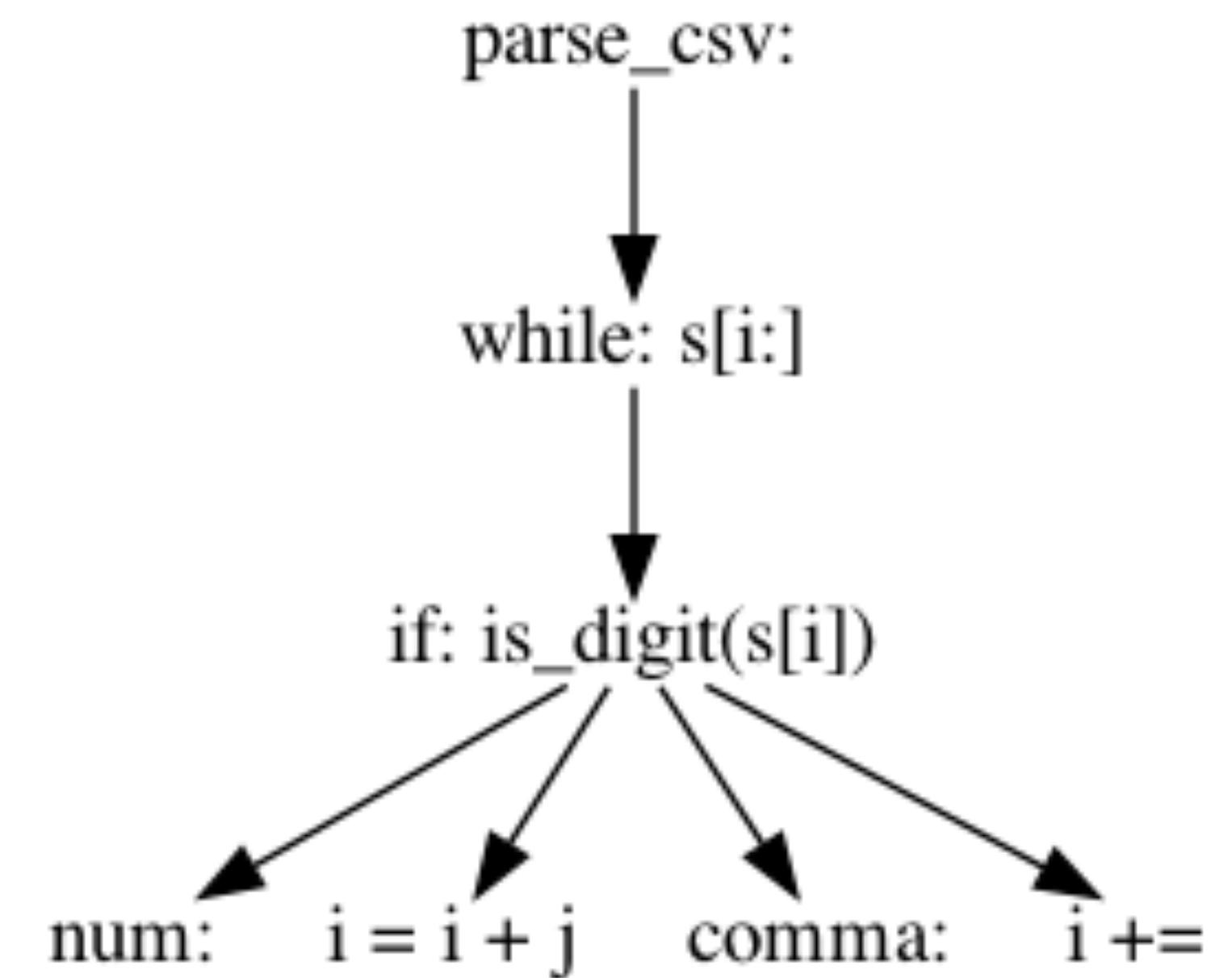


CDG for `parse_csv`

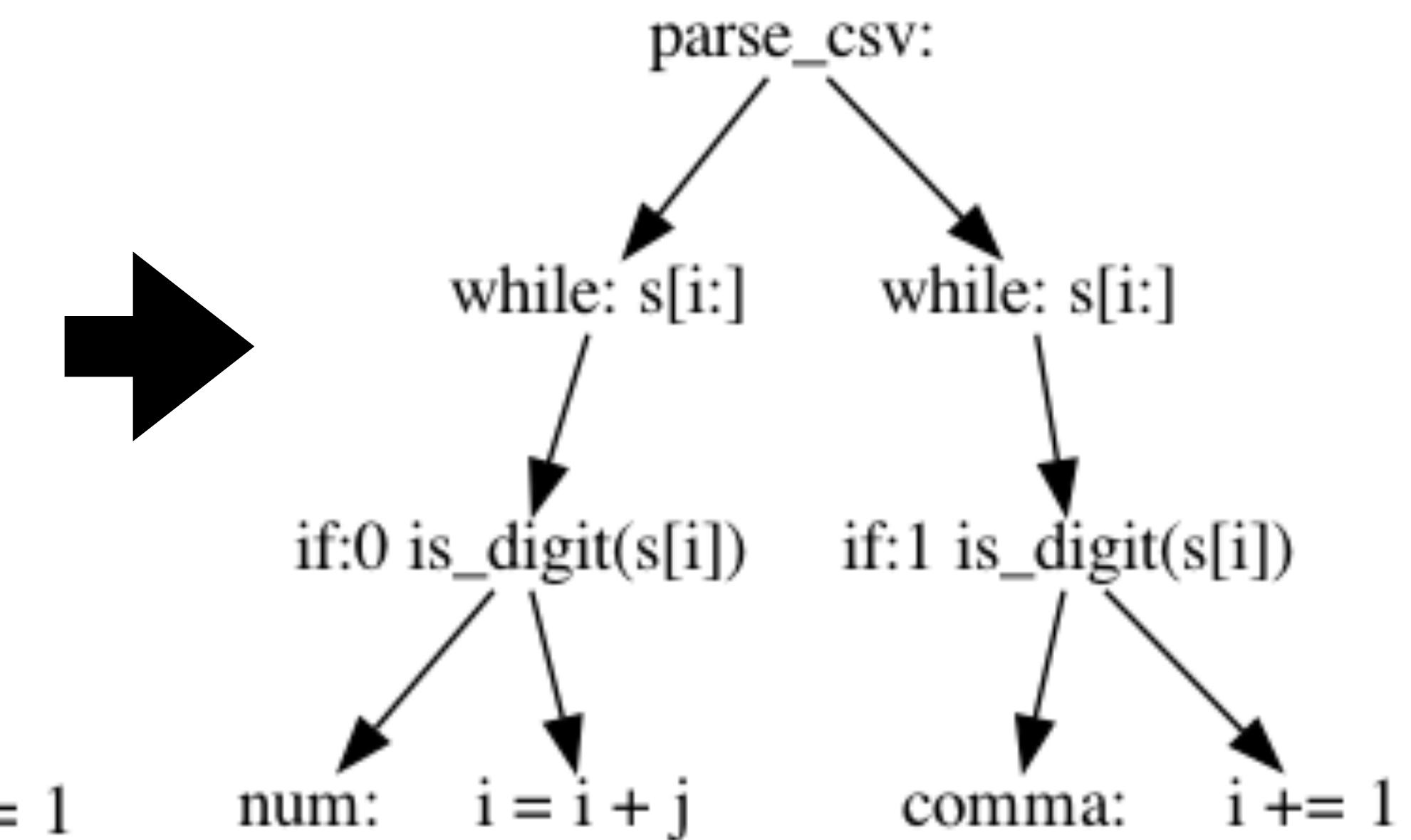
Each statement execution is represented as a separate node

Dynamic Control Dependence Tree

```
def parse_csv(s, i):
    while s[i:]:
        if is_digit(s[i]):
            n, j = num(s[i:])
            i = i+j
        else:
            comma(s[i])
            i += 1
```



CDG for `parse_csv`

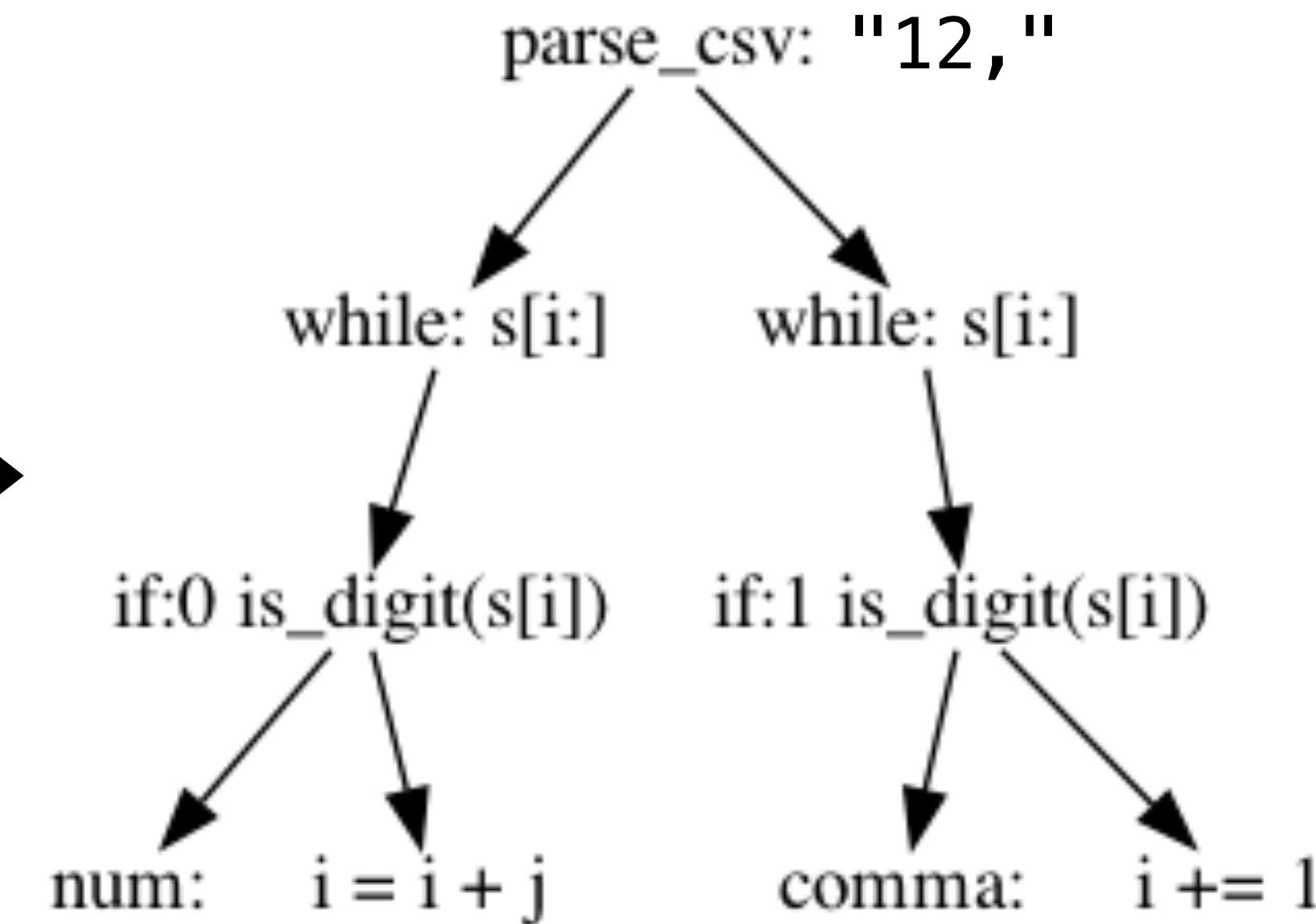
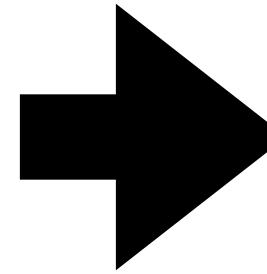


DCD Tree for `call parse_csv()`

Each statement execution is represented as a separate node

DCD Tree ~ Parse Tree

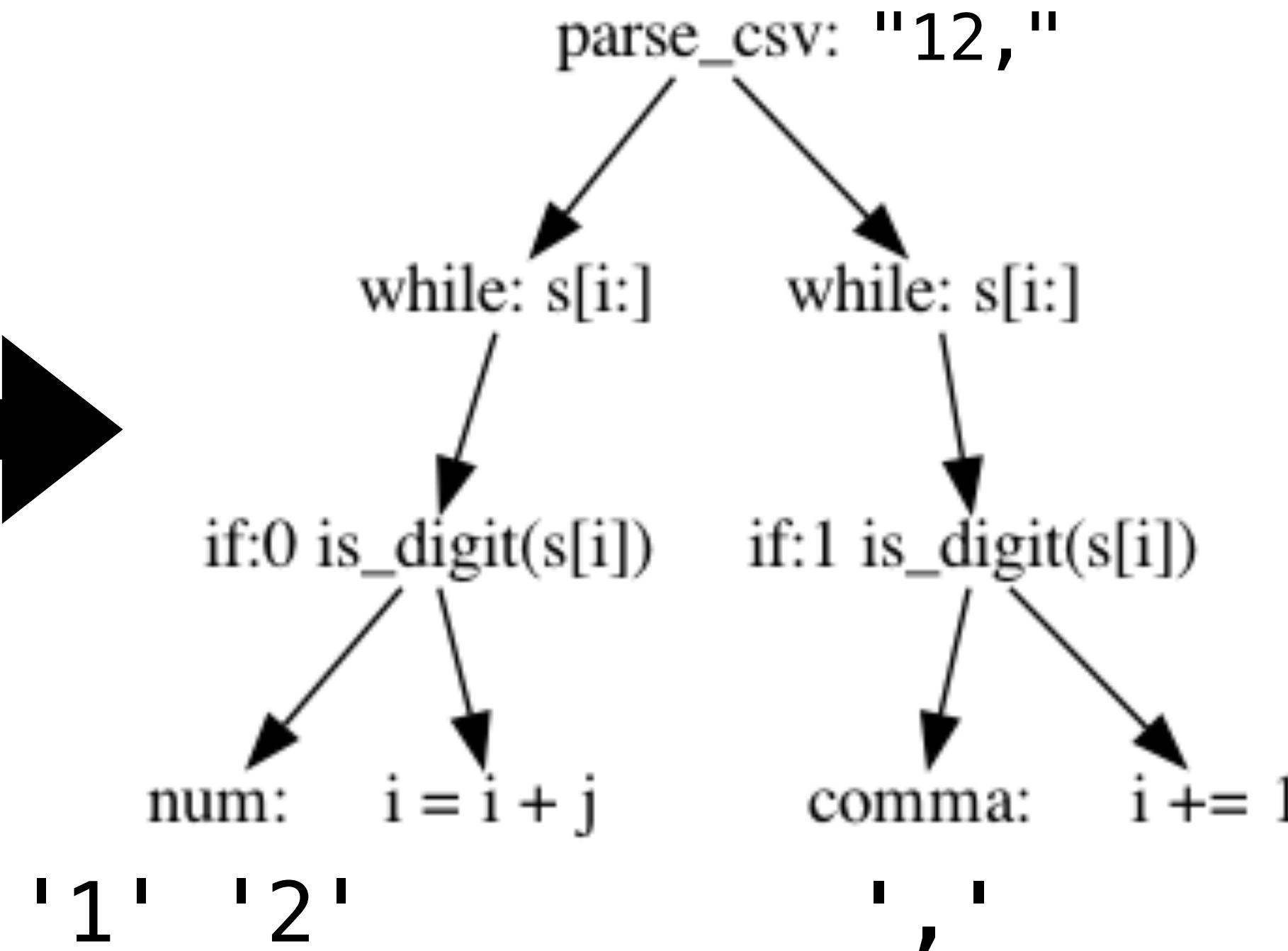
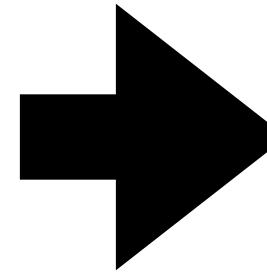
```
def parse_csv(s,i): "12,"
    while s[i:]:
        if is_digit(s[i]):
            n,j = num(s[i:])
            i = i+j
        else:
            comma(s[i])
            i += 1
```



- No tracking beyond input buffer
- Characters are attached to nodes where they are **accessed last**

DCD Tree ~ Parse Tree

```
def parse_csv(s,i): "12,"
    while s[i:]:
        if is_digit(s[i]):
            n,j = num(s[i:])
            i = i+j
        else:
            comma(s[i])
            i += 1
```



- No tracking beyond input buffer
- Characters are attached to nodes where they are **accessed last**

```

def is_digit(i): return i in '0123456789'
def parse_num(s,i):
    n = ''
    while s[i:] and is_digit(s[i]):
        n += s[i]
        i = i +1
    return i,n

def parse_paren(s, i):
    assert s[i] == '('
    i, v = parse_expr(s, i+1)
    if s[i:] == '': raise Ex(s, i)
    assert s[i] == ')'
    return i+1, v

def parse_expr(s, i = 0):
    expr, is_op = [], True
    while s[i:]:
        c = s[i]
        if isdigit(c):
            if not is_op: raise Ex(s,i)
            i,num = parse_num(s,i)
            expr.append(num)
            is_op = False
        elif c in ['+', '-', '*', '/']:
            if is_op: raise Ex(s,i)
            expr.append(c)
            is_op, i = True, i + 1
        elif c == '(':
            if not is_op: raise Ex(s,i)
            i, cexpr = parse_paren(s, i)
            expr.append(cexpr)
            is_op = False
        elif c == ')': break
        else: raise Ex(s,i)
    if is_op: raise Ex(s,i)
    return i, expr

```

$9+3/4$

Parse tree for `parse_expr('9+3/4')`

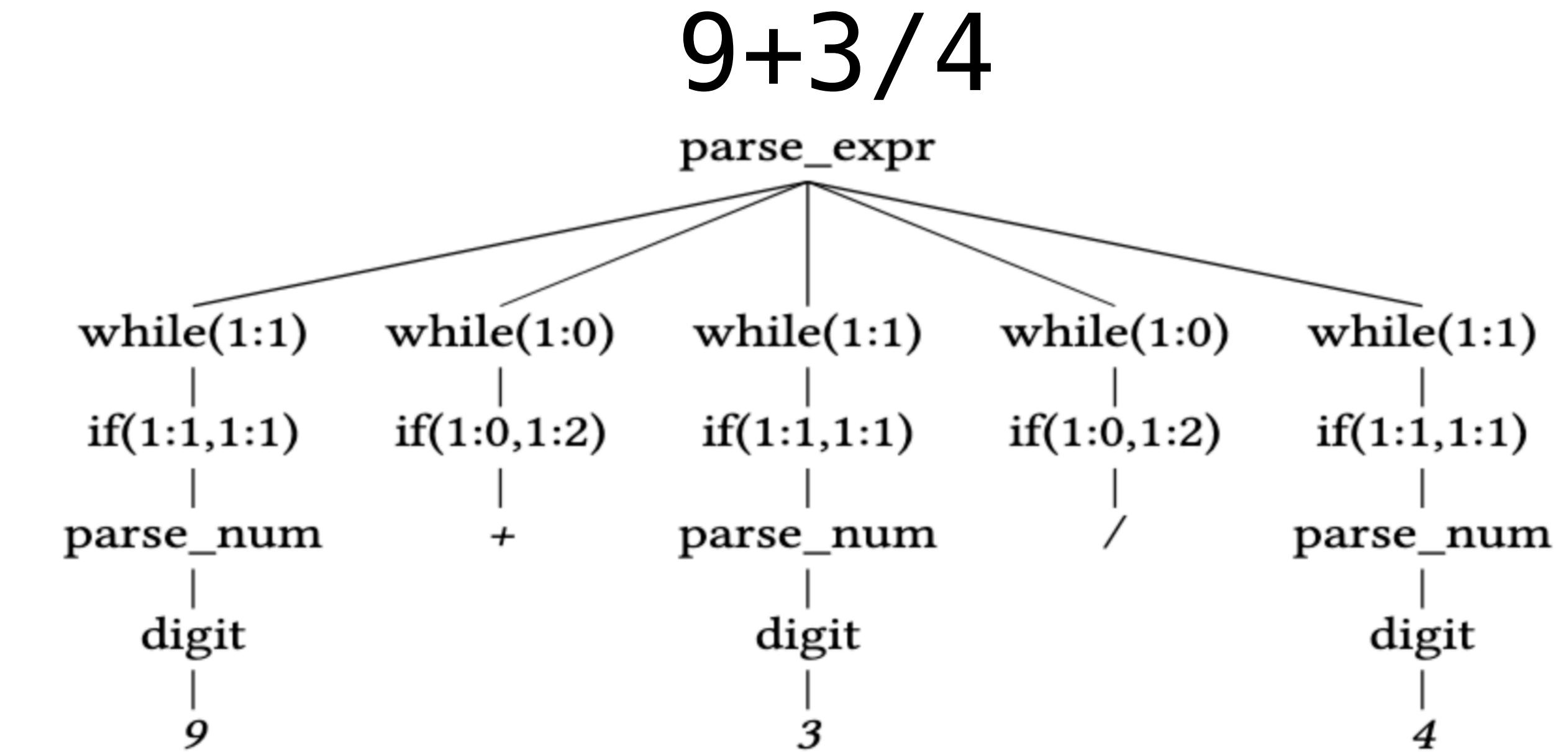
```

def is_digit(i): return i in '0123456789'
def parse_num(s,i):
    n = ''
    while s[i:] and is_digit(s[i]):
        n += s[i]
        i = i +1
    return i,n

def parse_paren(s, i):
    assert s[i] == '('
    i, v = parse_expr(s, i+1)
    if s[i:] == '': raise Ex(s, i)
    assert s[i] == ')'
    return i+1, v

def parse_expr(s, i = 0):
    expr, is_op = [], True
    while s[i:]:
        c = s[i]
        if isdigit(c):
            if not is_op: raise Ex(s,i)
            i,num = parse_num(s,i)
            expr.append(num)
            is_op = False
        elif c in ['+', '-', '*', '/']:
            if is_op: raise Ex(s,i)
            expr.append(c)
            is_op, i = True, i + 1
        elif c == '(':
            if not is_op: raise Ex(s,i)
            i, cexpr = parse_paren(s, i)
            expr.append(cexpr)
            is_op = False
        elif c == ')': break
        else: raise Ex(s,i)
    if is_op: raise Ex(s,i)
    return i, expr

```



Parse tree for **parse_expr('9+3/4')**

< > C localhost:8888/notebooks/x2_0_MiningGrammar.ipynb

jupyter x2_0_MiningGrammar Last Checkpoint: a minute ago (autosaved)

File Edit View Insert Cell Kernel Navigate Widgets Help Trusted Python 3 (ipykernel) O

Contents ↗

- 1 Mining Grammars
 - 1.1 Imports
 - 1.2 Subject Registry
- 1.3 Context Managers
 - 1.3.1 Method context
 - 1.3.2 Stack context
 - 1.3.3 Scope context
- 1.4 Rewriting the source to track contexts
 - 1.4.1 Rewriter
 - 1.4.1.1 The method context wrapper
 - 1.4.1.2 The stack wrapper
 - 1.4.1.3 The scope wrapper
 - 1.4.1.4 Rewriting `if` conditions
 - 1.4.1.5 Rewriting `while` loops
 - 1.4.1.6 Combining both
 - 1.4.2 Generating the complete instruction
 - 1.4.2.1 Using It
 - 1.4.3 Generate Transformed Source
- 1.5 Generating Traces
- 1.6 Mining the Traces Generated
 - 1.6.1 Reconstructing the Method Tree
 - 1.6.1.1 Identifying last comparison
 - 1.6.1.2 Attaching characters to the tree
 - 1.6.1.3 Removing Overlap
 - 1.6.1.4 Generate derivation tree
 - 1.6.2 The Complete Miner
- 1.7 Generalize Nodes
 - 1.7.1 Generalize Methods
 - 1.7.2 Generalize Loops
- 1.8 Generating a Grammar
 - 1.8.1 Trees to grammar
 - 1.8.2 Inserting Empty Alternatives for loops
 - 1.8.3 Learning Regular Expressions
 - 1.8.3.1 The modified Fennau algorithm
 - 1.8.4 Remove duplicate and redundant nodes
- 1.9 Accio Grammar

1.7 Generalize Nodes

Caching and book keeping variables.

```
In [ ]: EXEC_MAP = {}
NODE_REGISTER = {}
```

```
In [ ]: def reset_generalizer():
    global EXEC_MAP
    global NODE_REGISTER
    EXEC_MAP.clear()
    NODE_REGISTER.clear()
```

Given that we are evaluating Python functions, we need a wrapper to make them executable.

A small library function to convert from tuple to lists so that we can modify a tree.

```
In [ ]: def to_modifiable(derivation_tree):
    node, children, *rest = derivation_tree
    return [node, [to_modifiable(c) for c in children], *rest]
```

1.7.1 Generalize Methods

The idea here is that, sometimes one finds that our central assumption -- that a fragment consumed by a function can be replaced by another fragment consumed by the same function elsewhere -- doesn't hold. This can be seen in functions that take an additional argument to specify what it should match. In such cases, we want to try and find out how to distinguish between these function invocations.

`node_include()` is a library function that checks whether the node `j` is within the boundary of `i`.

```
In [ ]: def node_include(i, j):
    name_i, children_i, s_i, e_i = i
    name_j, children_j, s_j, e_j = j
    return is_second_item_included((s_i, e_i), (s_j, e_j))
```

```

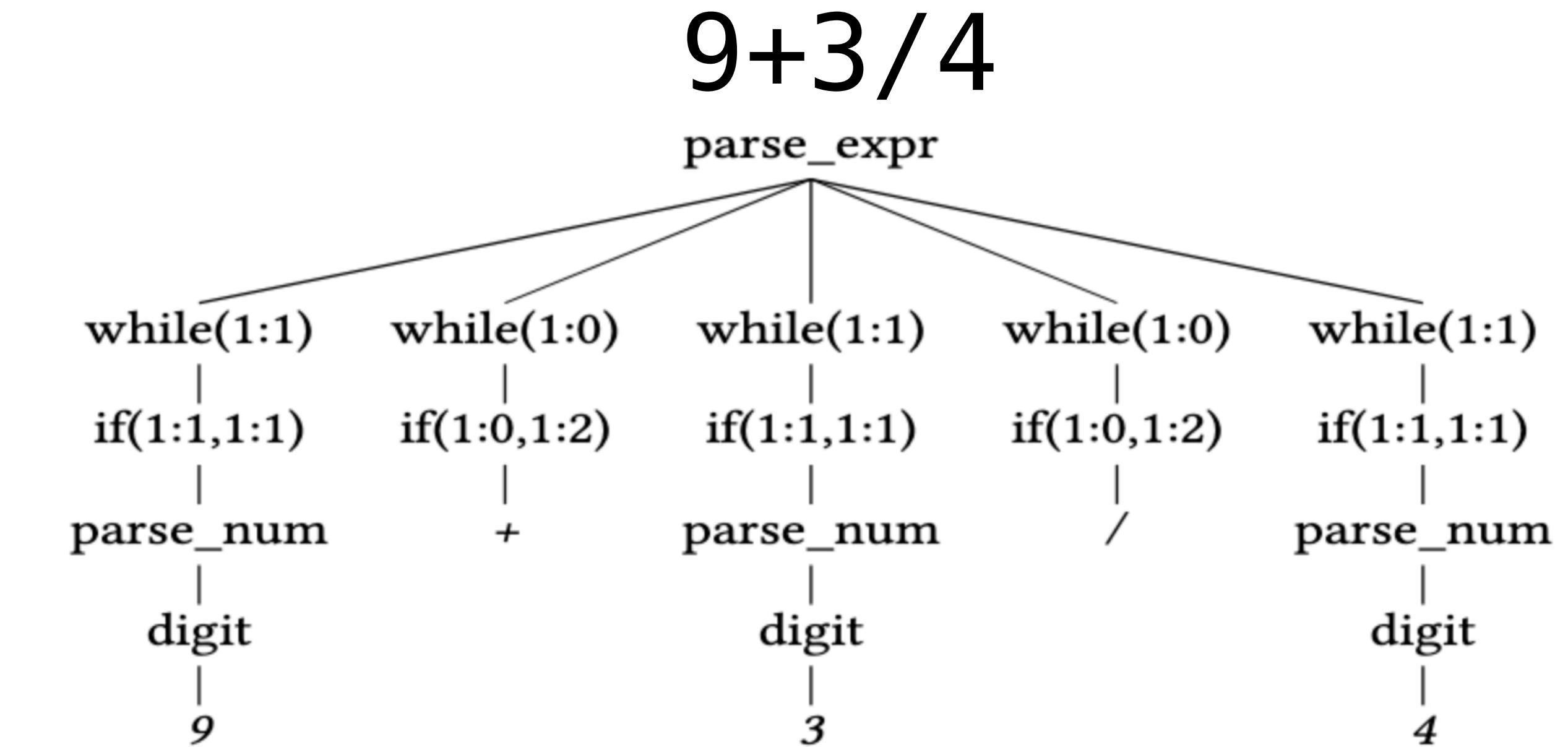
def is_digit(i): return i in '0123456789'
def parse_num(s,i):
    n = ''
    while s[i:] and s[i].isdigit():
        n += s[i]
        i = i + 1
    return i,n

def parse_paren(s, i):
    assert s[i] == '('
    i, v = parse_expr(s, i+1)
    if s[i:] == ''): raise Ex(s, i)
    assert s[i] == ')'
    return i+1, v

def parse_expr(s, i = 0):
    expr, is_op = [], True
    while s[i:]:
        c = s[i]
        if isdigit(c):
            if not is_op: raise Ex(s,i)
            i,num = parse_num(s,i)
            expr.append(num)
            is_op = False
        elif c in ['+', '-', '*', '/']:
            if is_op: raise Ex(s,i)
            expr.append(c)
            is_op, i = True, i + 1
        elif c == '(':
            if not is_op: raise Ex(s,i)
            i, cexpr = parse_paren(s, i)
            expr.append(cexpr)
            is_op=False
        elif c == ')': break
        else: raise Ex(s,i)
    if is_op: raise Ex(s,i)
    return i, expr

```

Identifying Compatible Nodes



Which nodes correspond to the same nonterminal

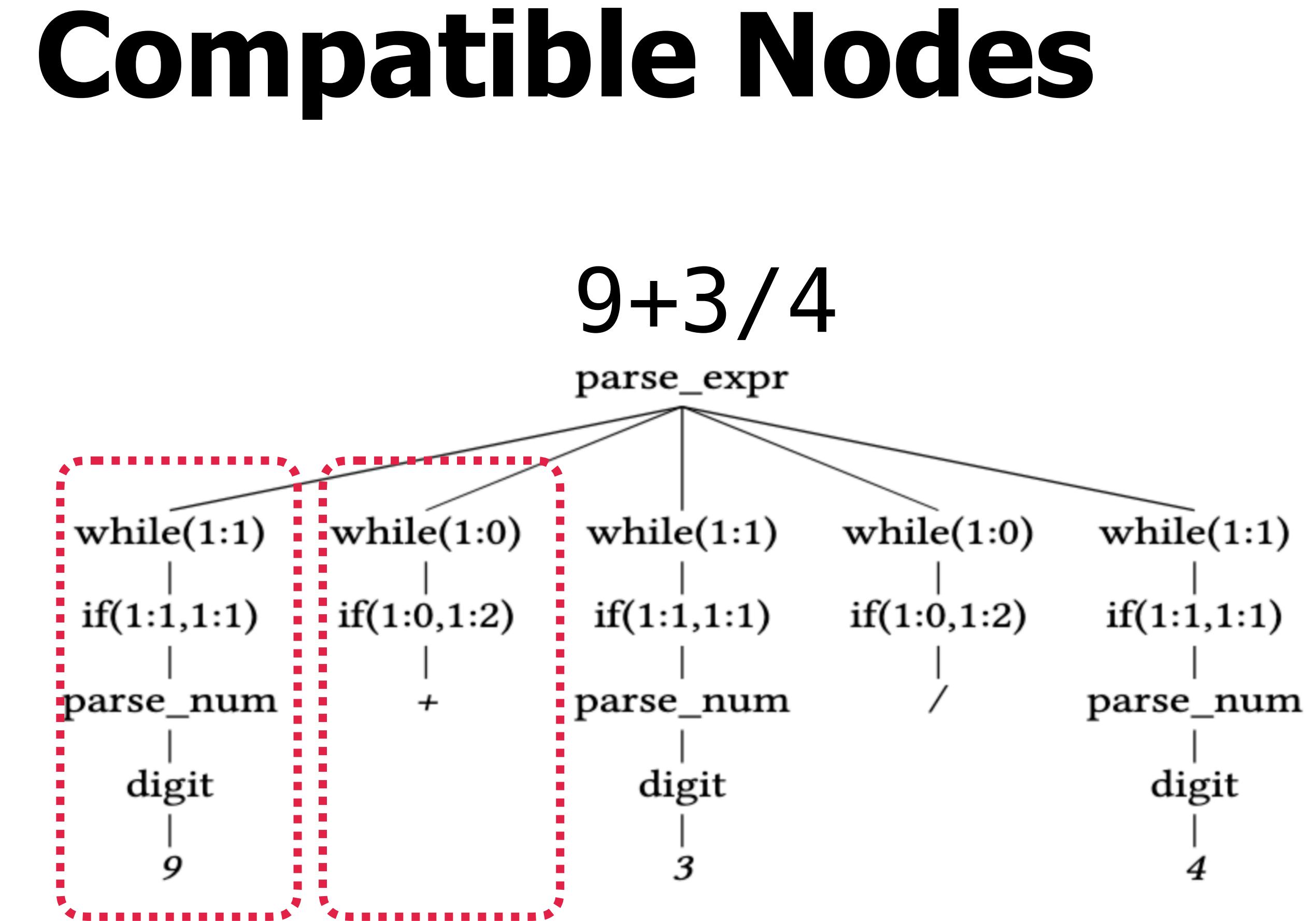
```

def is_digit(i): return i in '0123456789'
def parse_num(s,i):
    n = ''
    while s[i:] and s[i].isdigit():
        n += s[i]
        i = i + 1
    return i,n

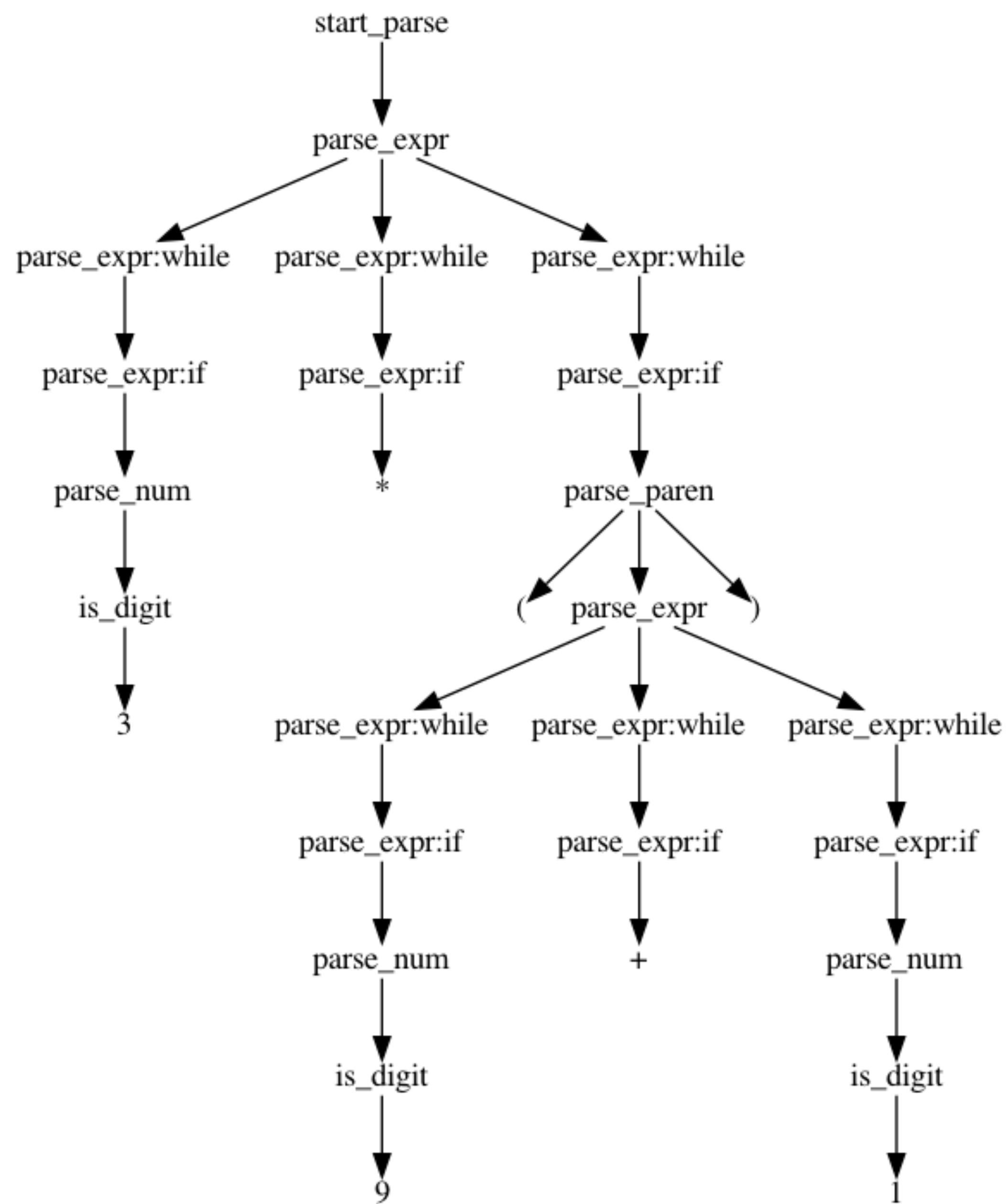
def parse_paren(s, i):
    assert s[i] == '('
    i, v = parse_expr(s, i+1)
    if s[i:] == ''): raise Ex(s, i)
    assert s[i] == ')'
    return i+1, v

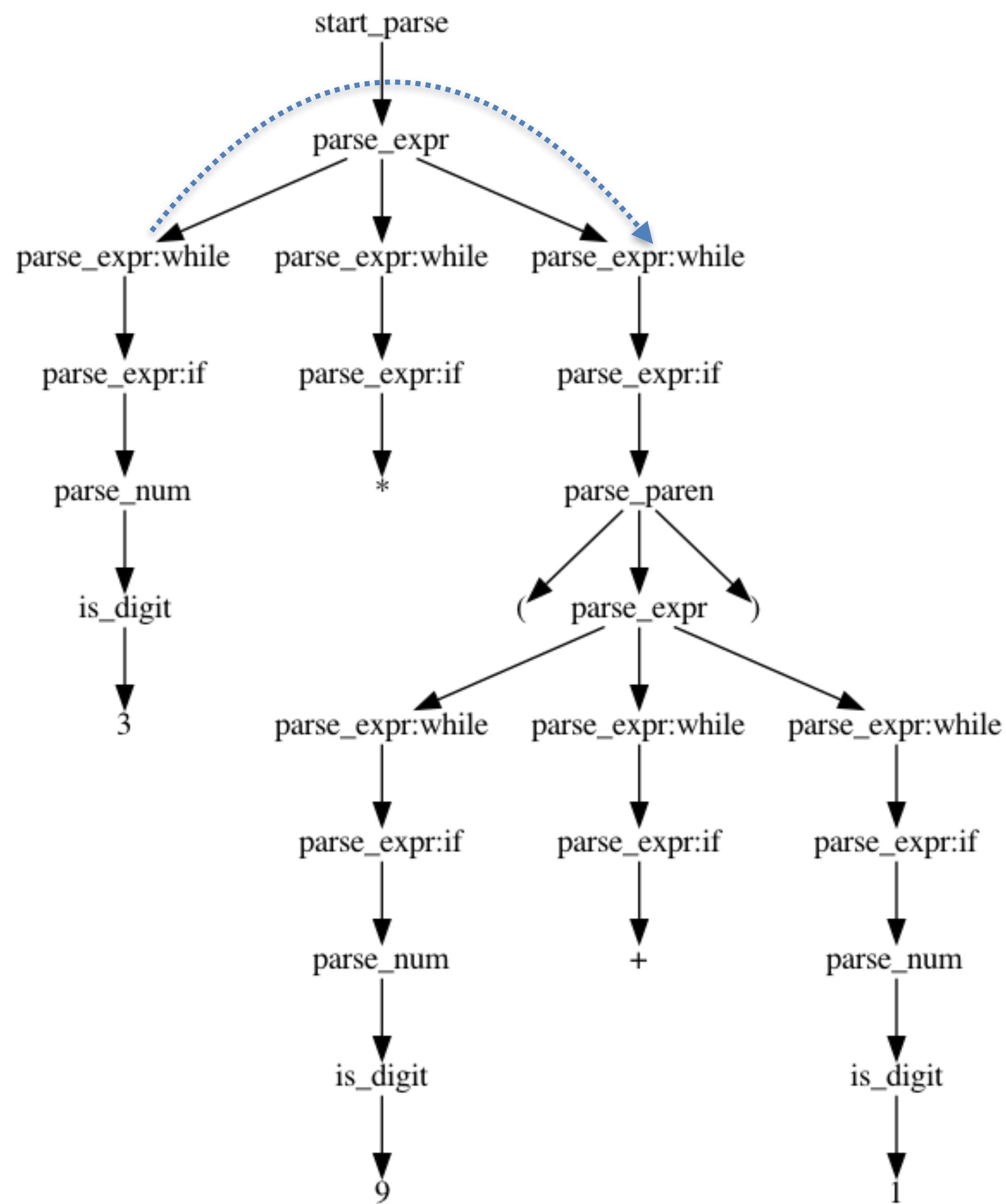
def parse_expr(s, i = 0):
    expr, is_op = [], True
    while s[i:]:
        c = s[i]
        if isdigit(c):
            if not is_op: raise Ex(s,i)
            i,num = parse_num(s,i)
            expr.append(num)
            is_op = False
        elif c in ['+', '-', '*', '/']:
            if is_op: raise Ex(s,i)
            expr.append(c)
            is_op, i = True, i + 1
        elif c == '(':
            if not is_op: raise Ex(s,i)
            i, cexpr = parse_paren(s, i)
            expr.append(cexpr)
            is_op=False
        elif c == ')': break
        else: raise Ex(s,i)
    if is_op: raise Ex(s,i)
    return i, expr

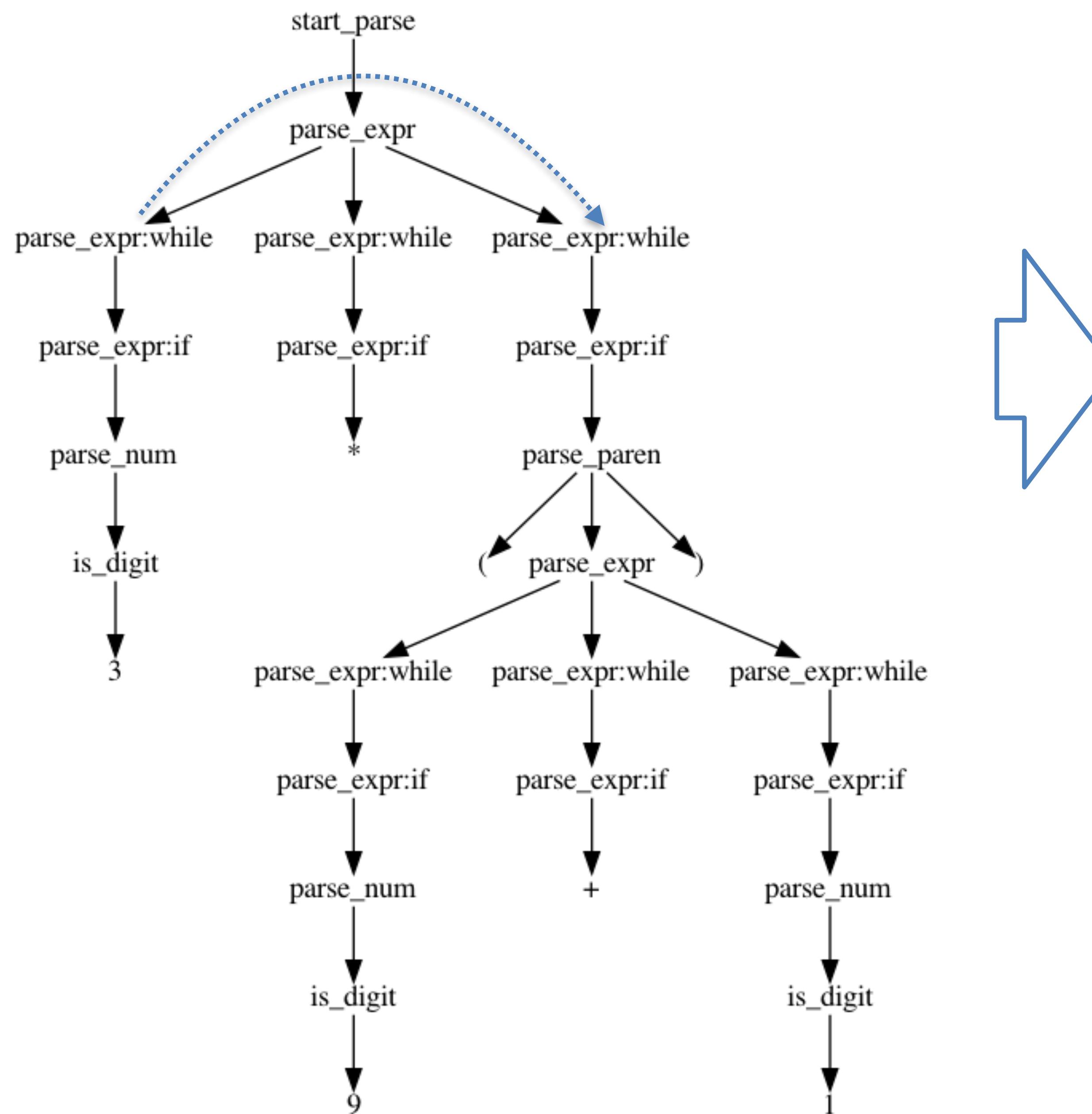
```

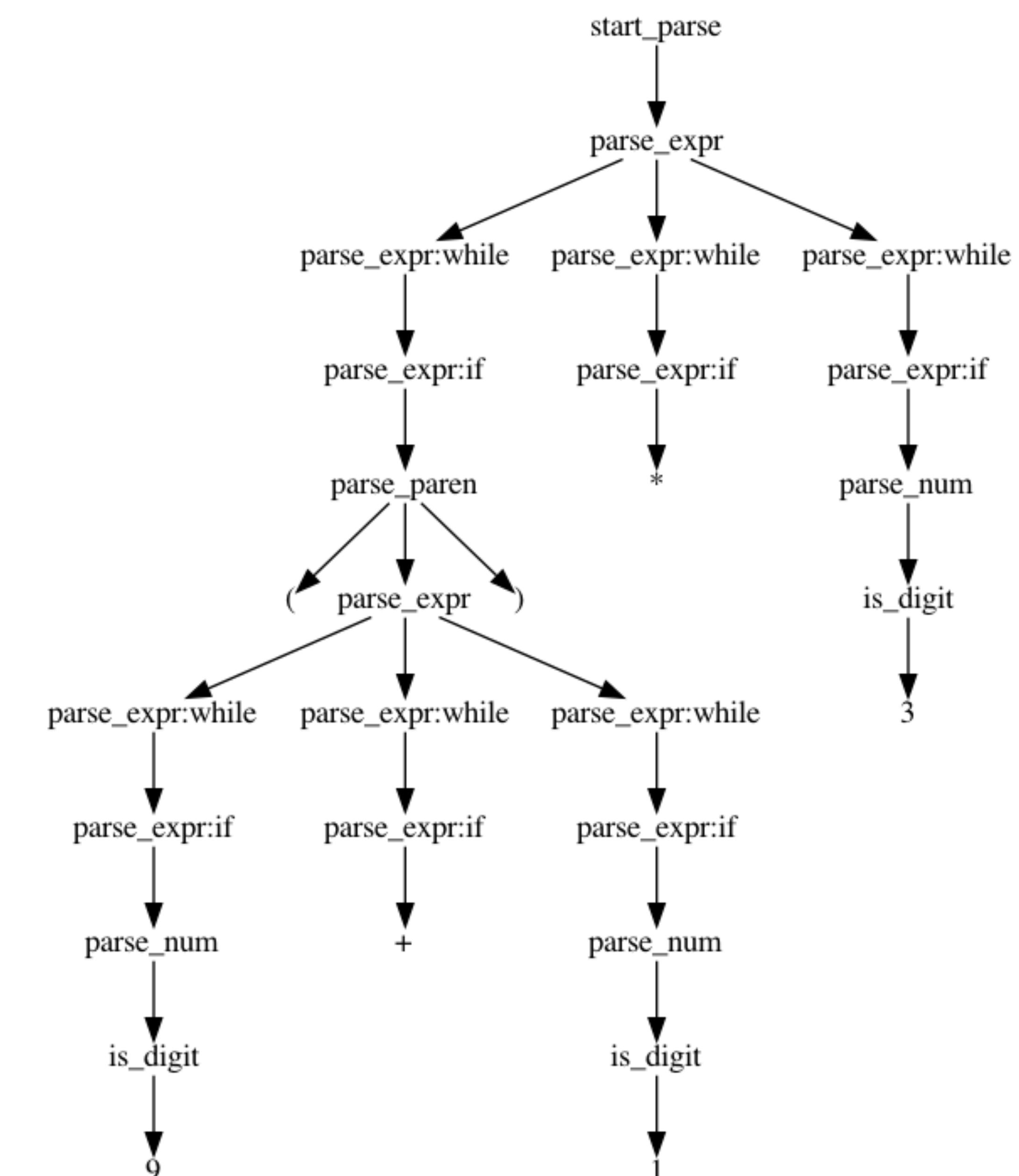


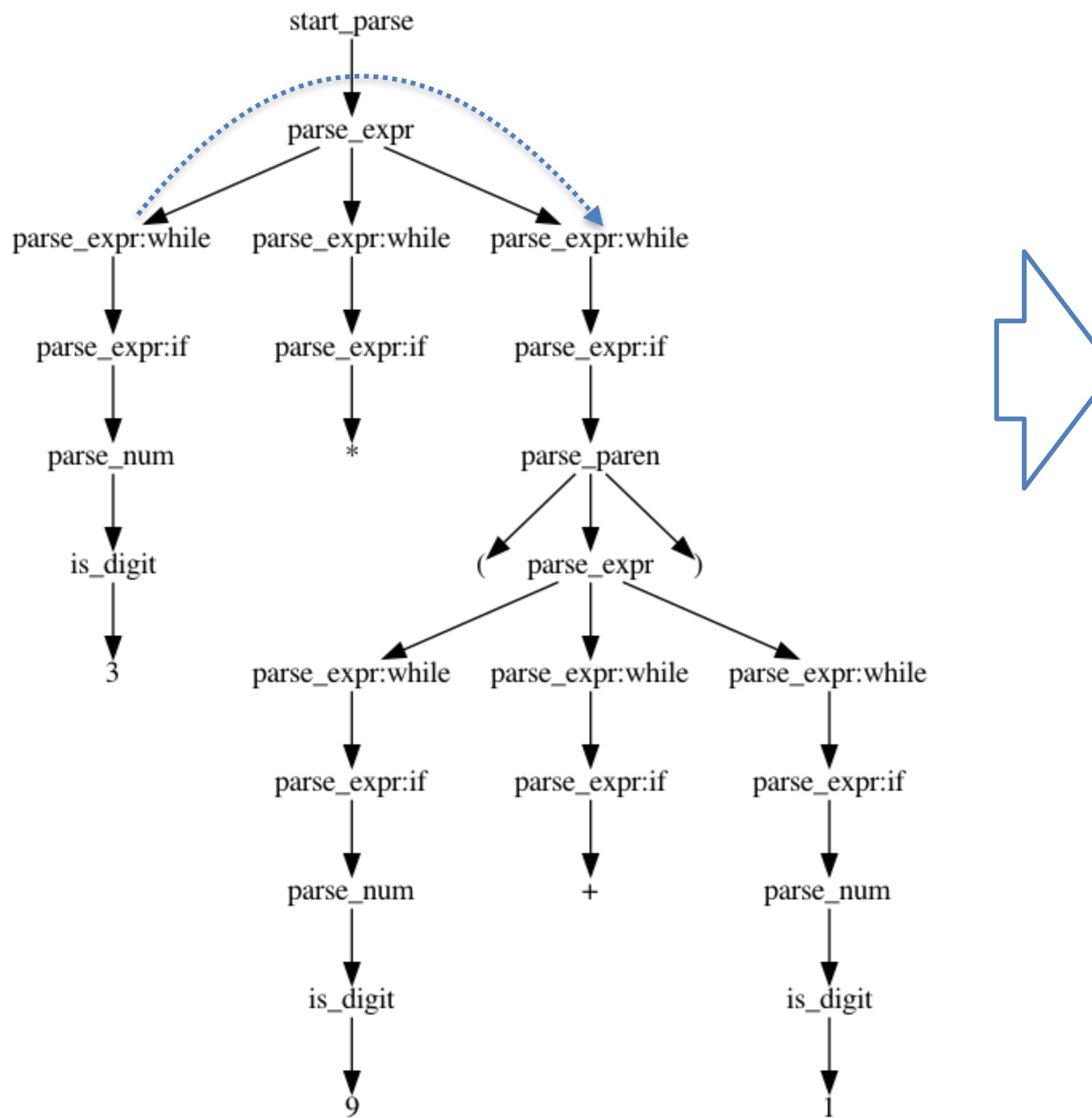
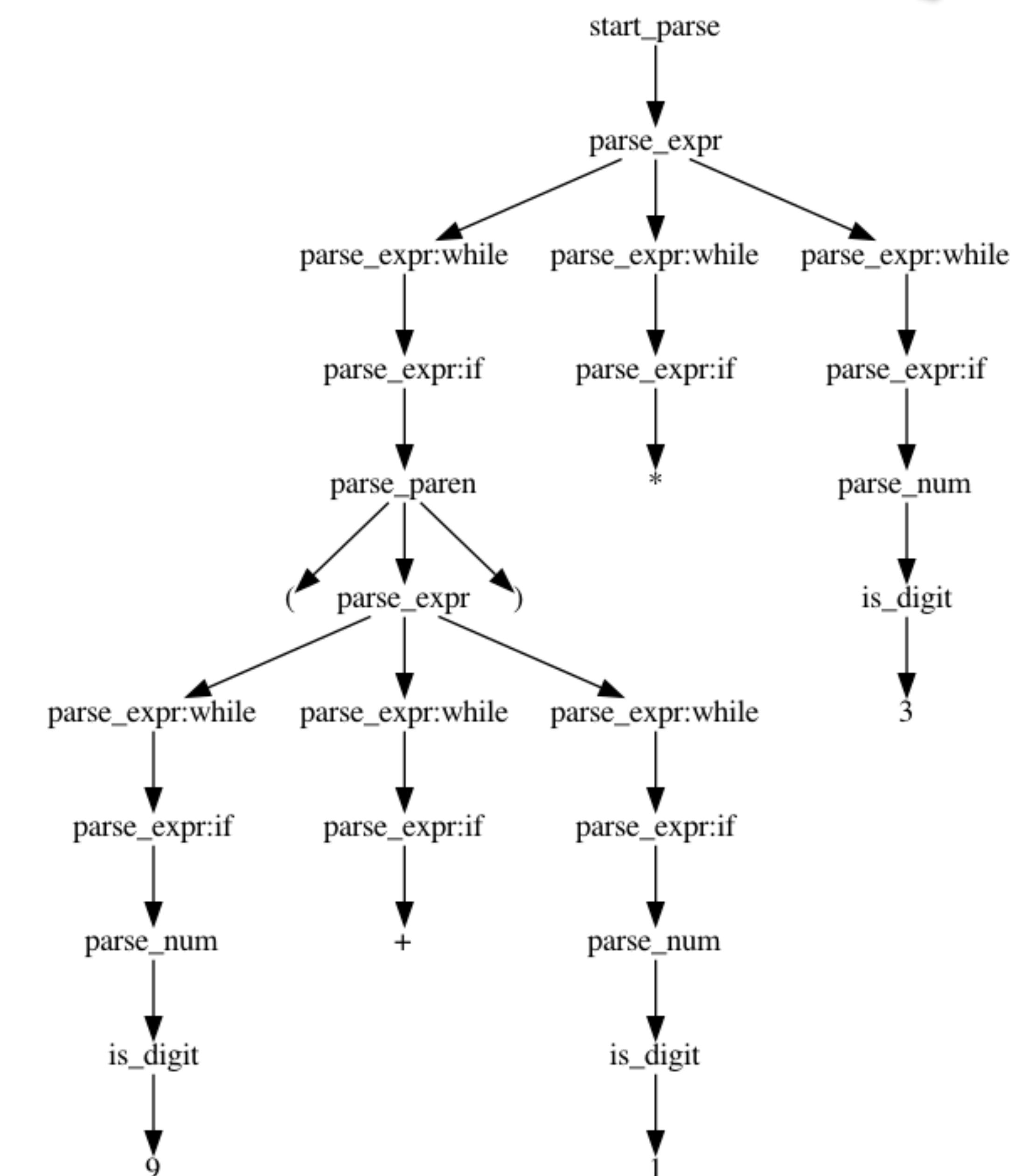
Which nodes correspond to the same nonterminal

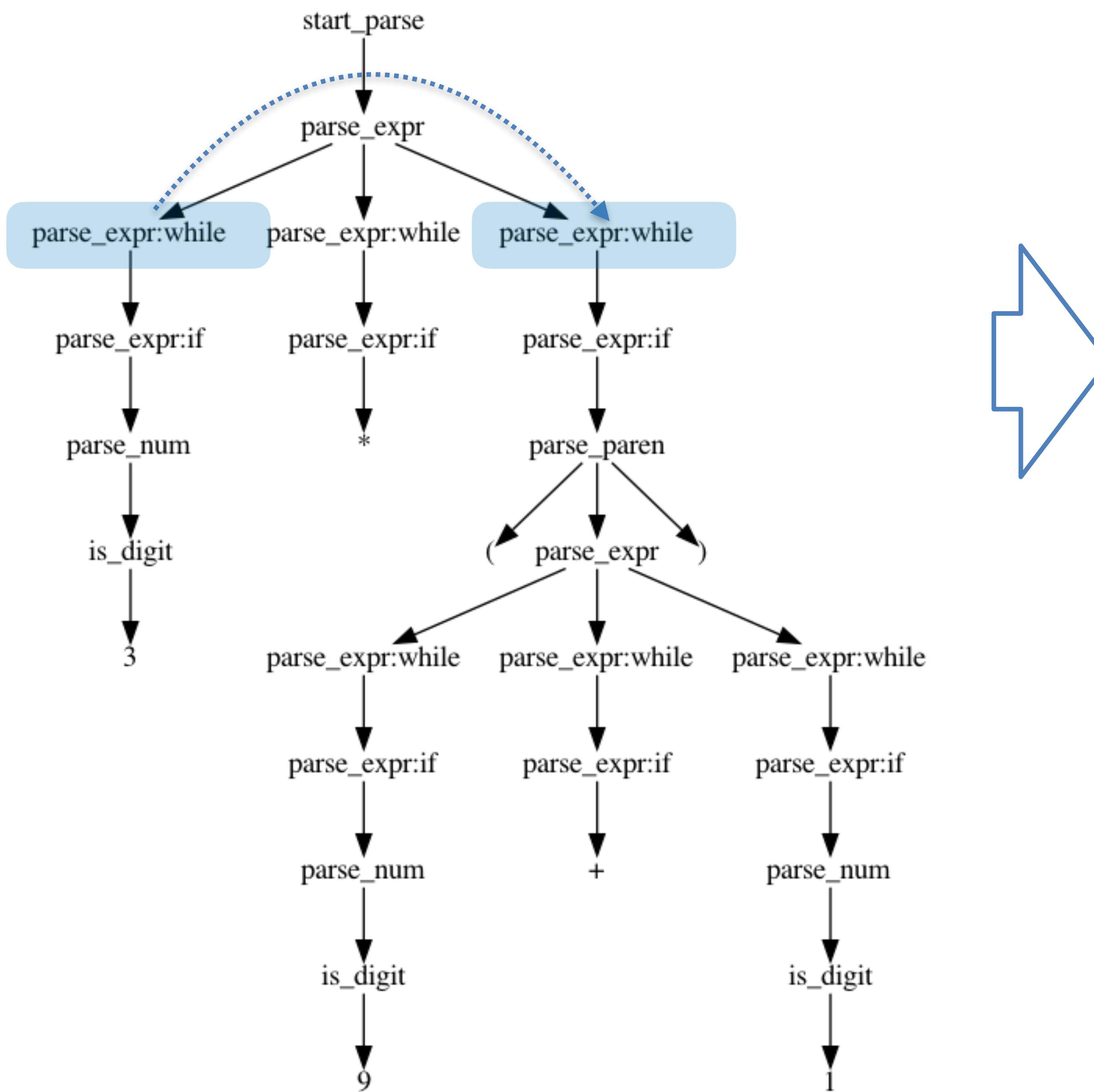
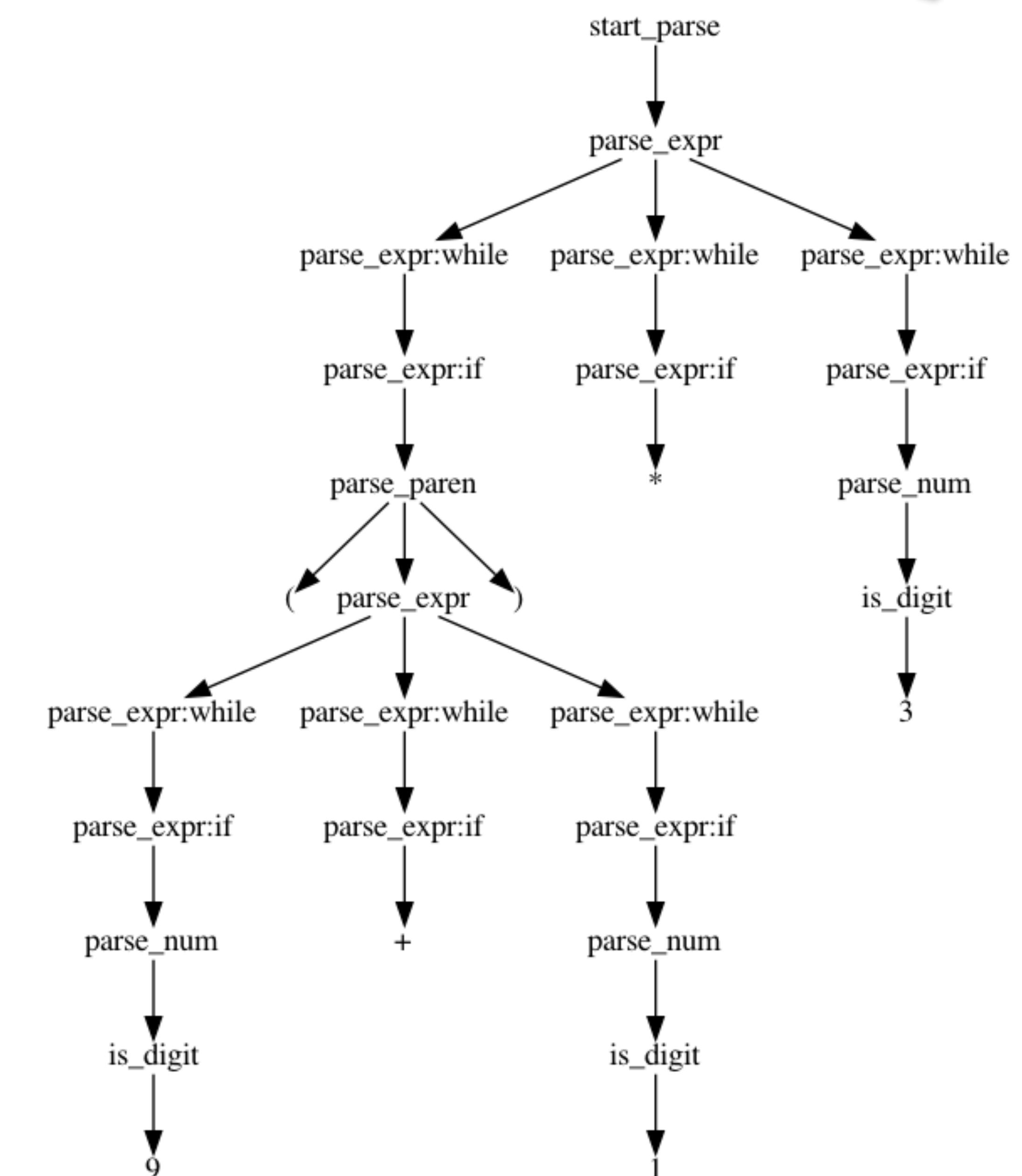
$$3 * (9 + 1)$$


$$3 * (9 + 1)$$


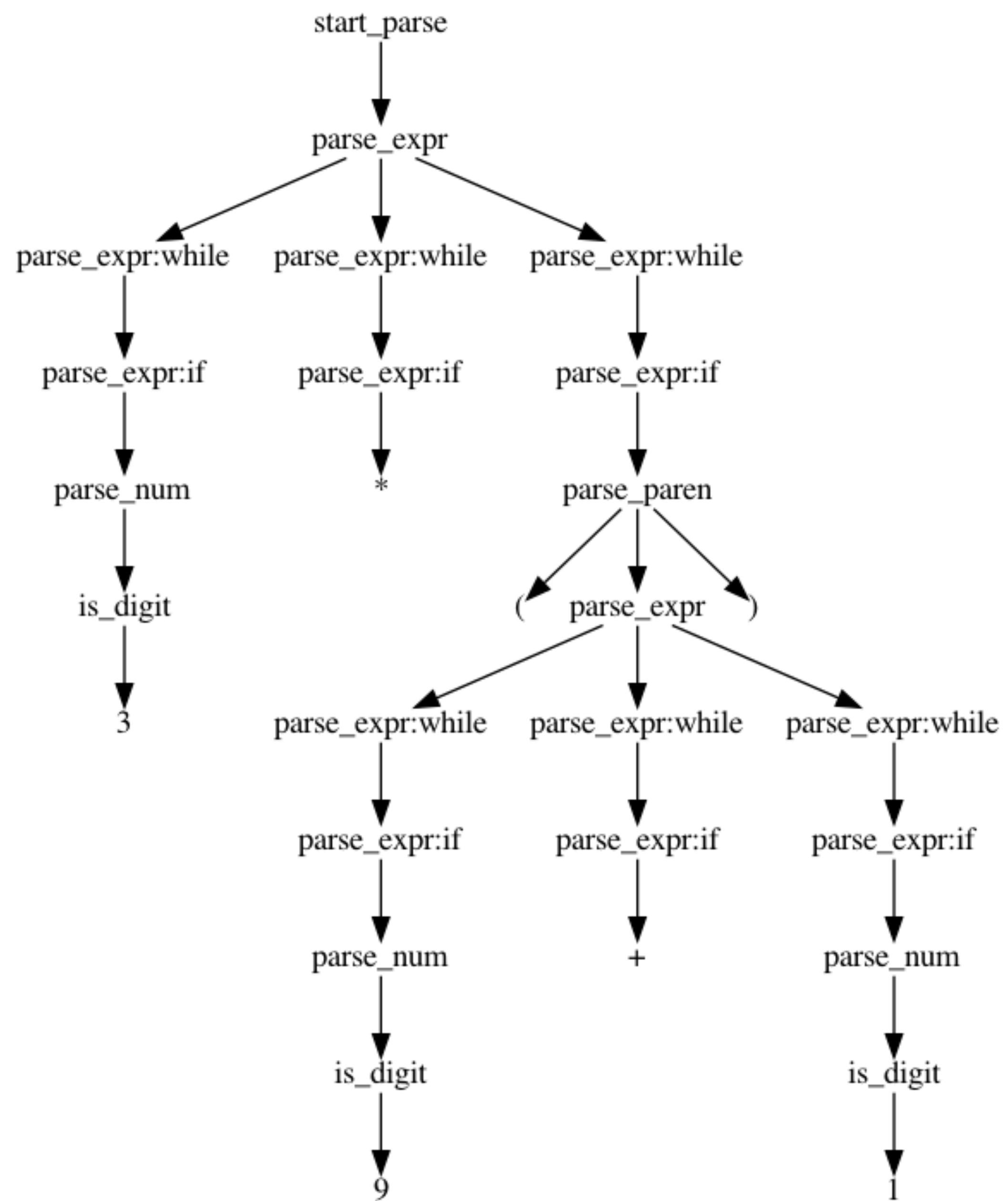
$$3 * (9 + 1)$$


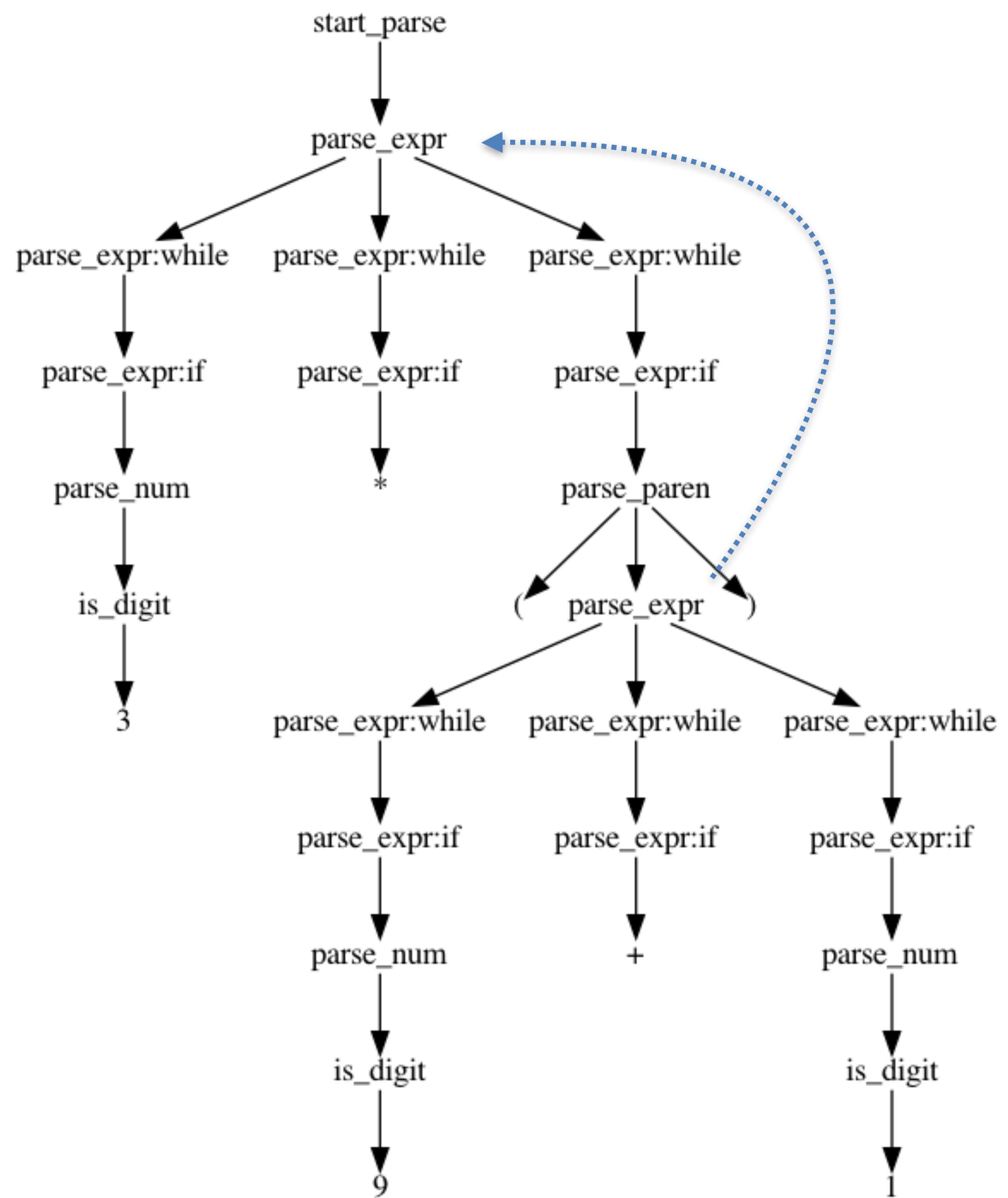
$$(9 + 1) * 3$$


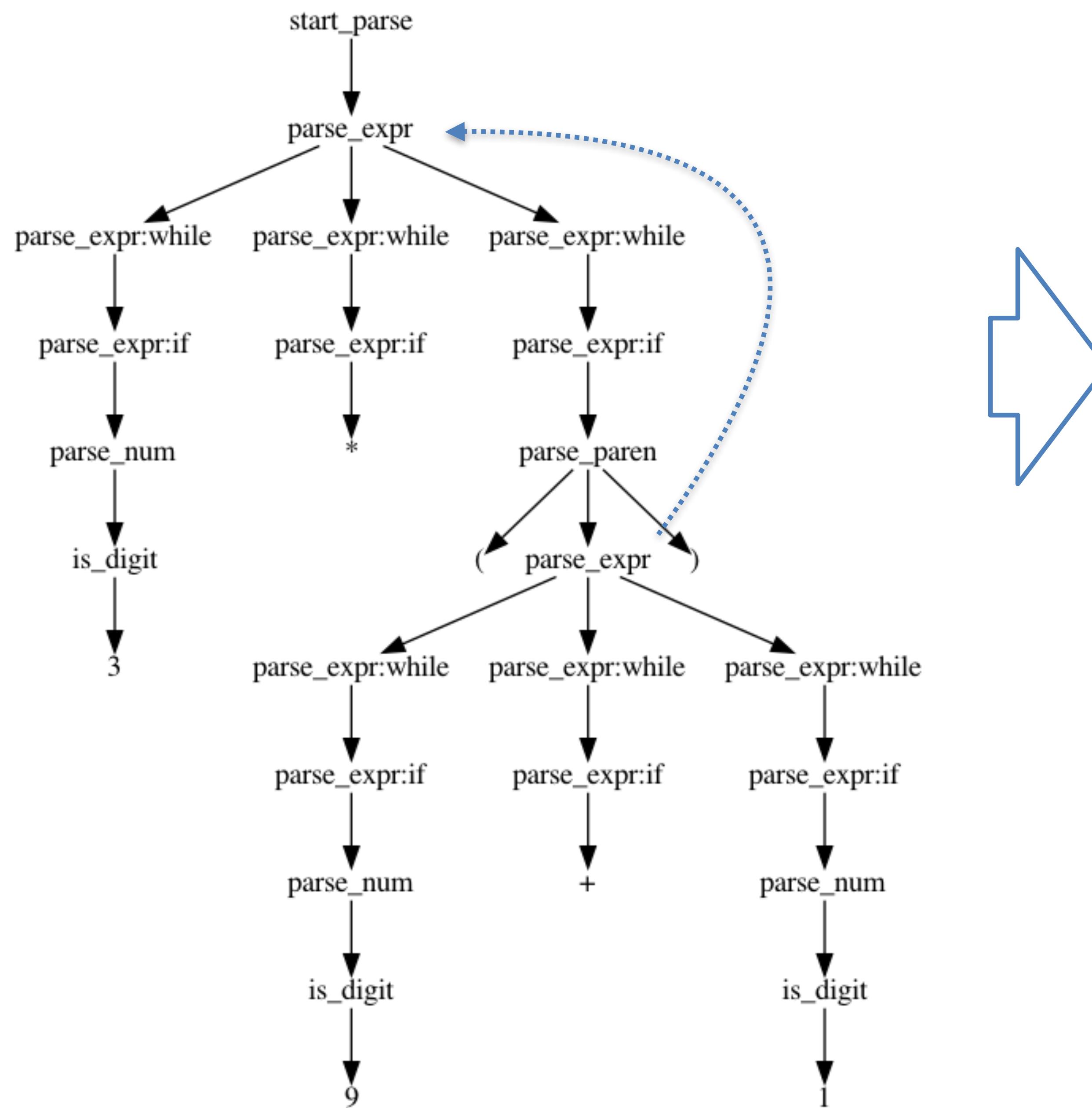
$3 * (9 + 1)$

 $(9 + 1) * 3$


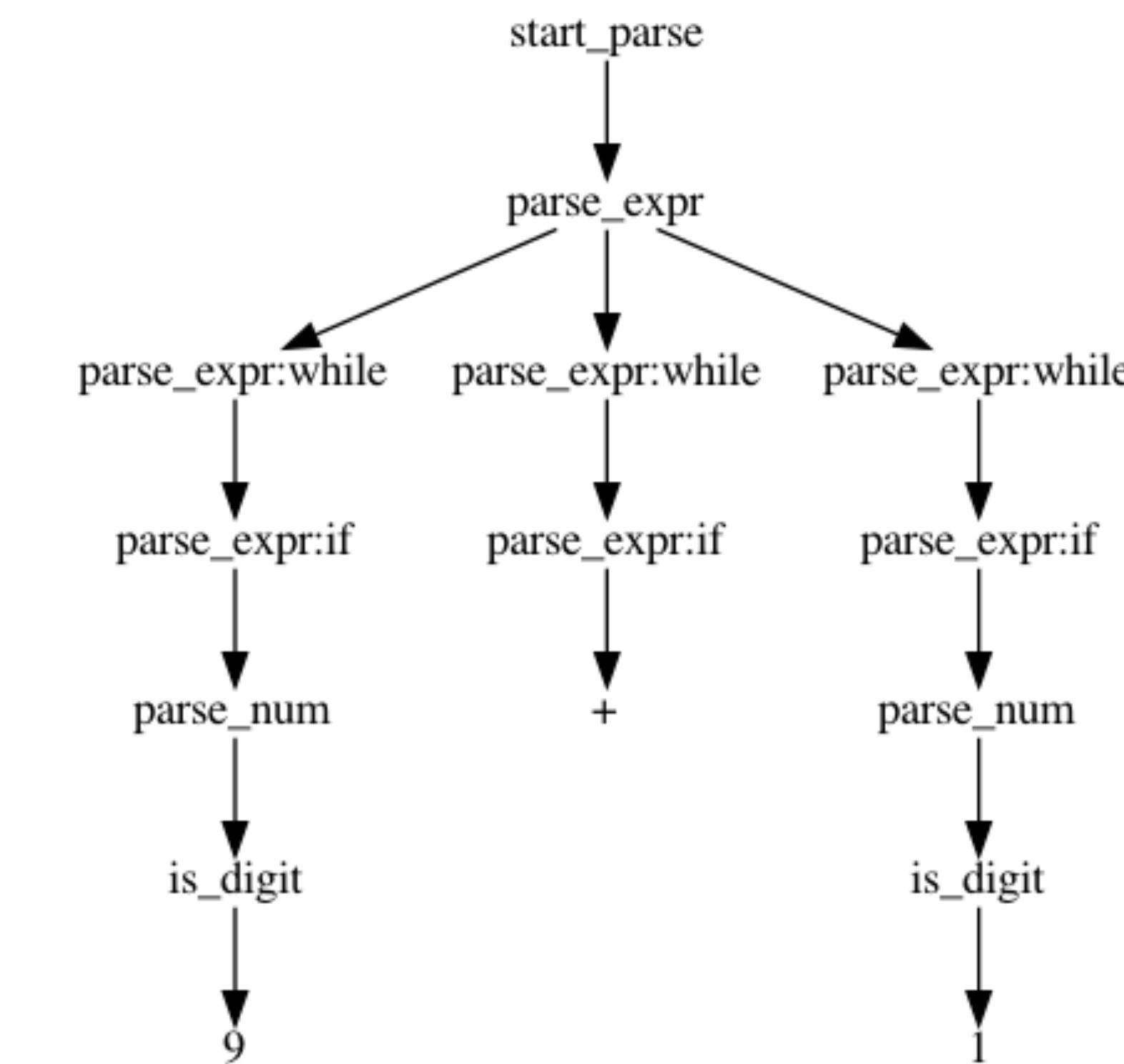
$3 * (9 + 1)$

 $(9 + 1) * 3$


$3 * (9 + 1)$

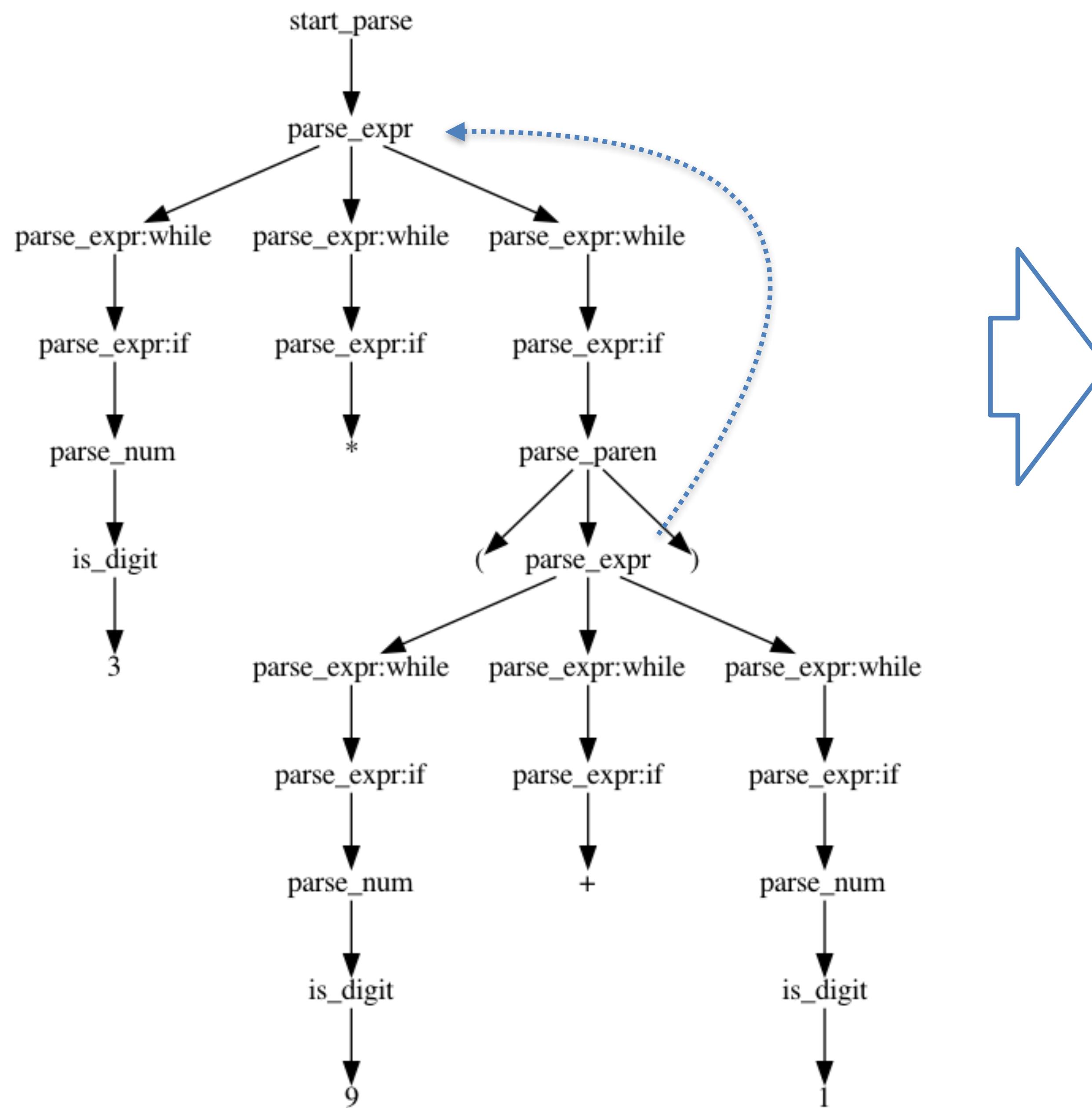


$$3 * (9 + 1)$$


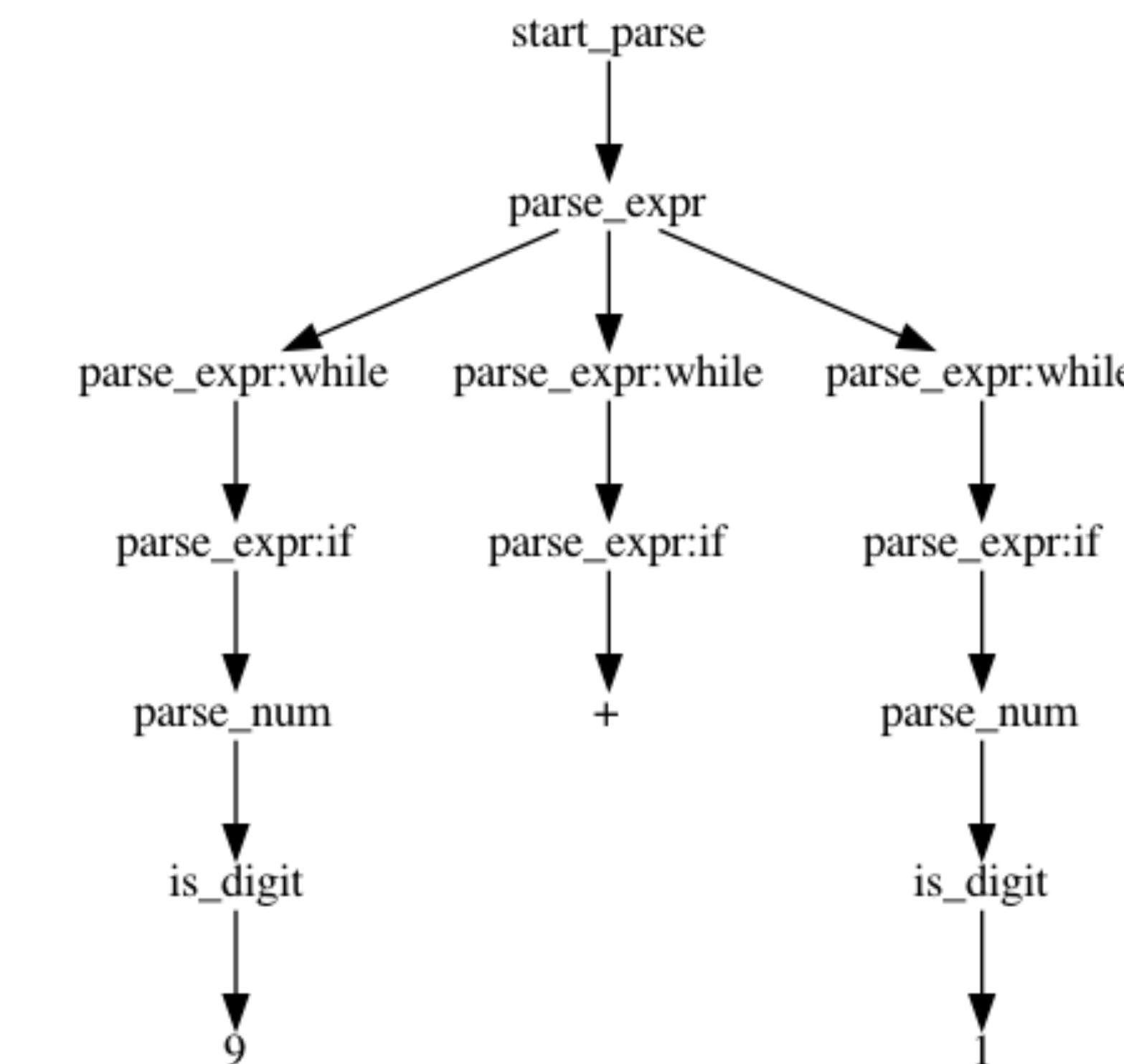
$$3 * (9 + 1)$$


$$9 + 1$$


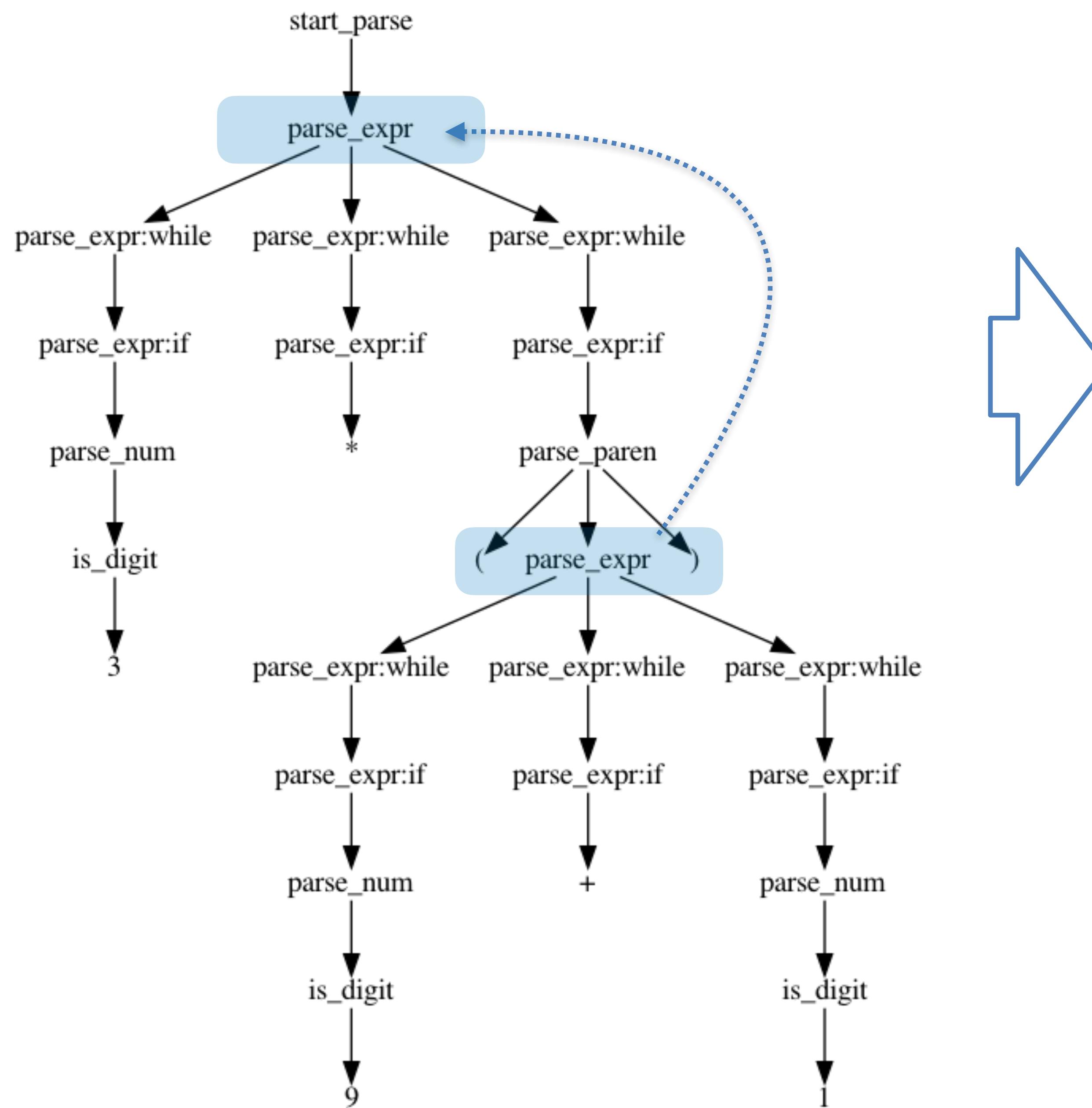
$3 * (9 + 1)$



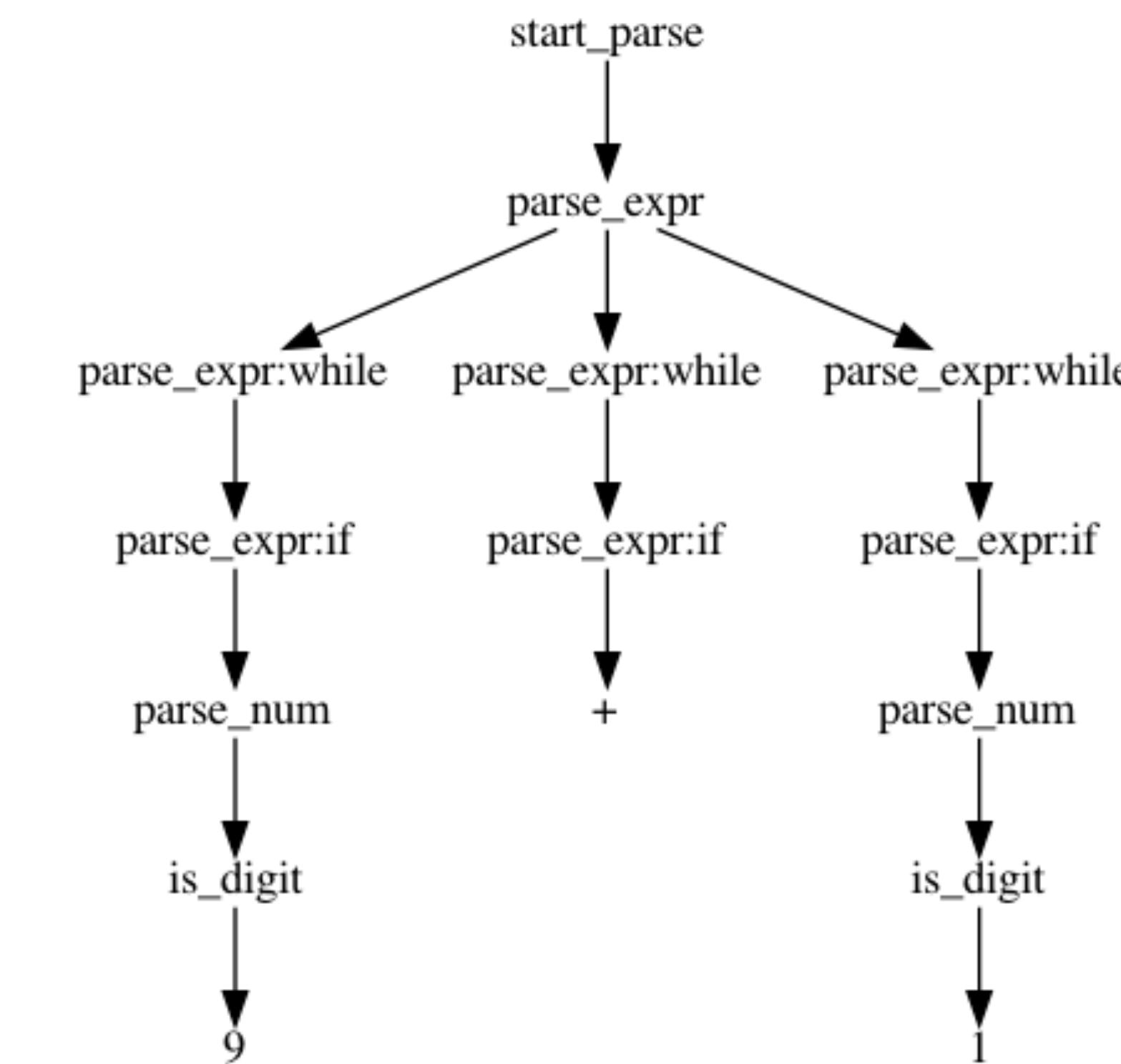
$9 + 1$

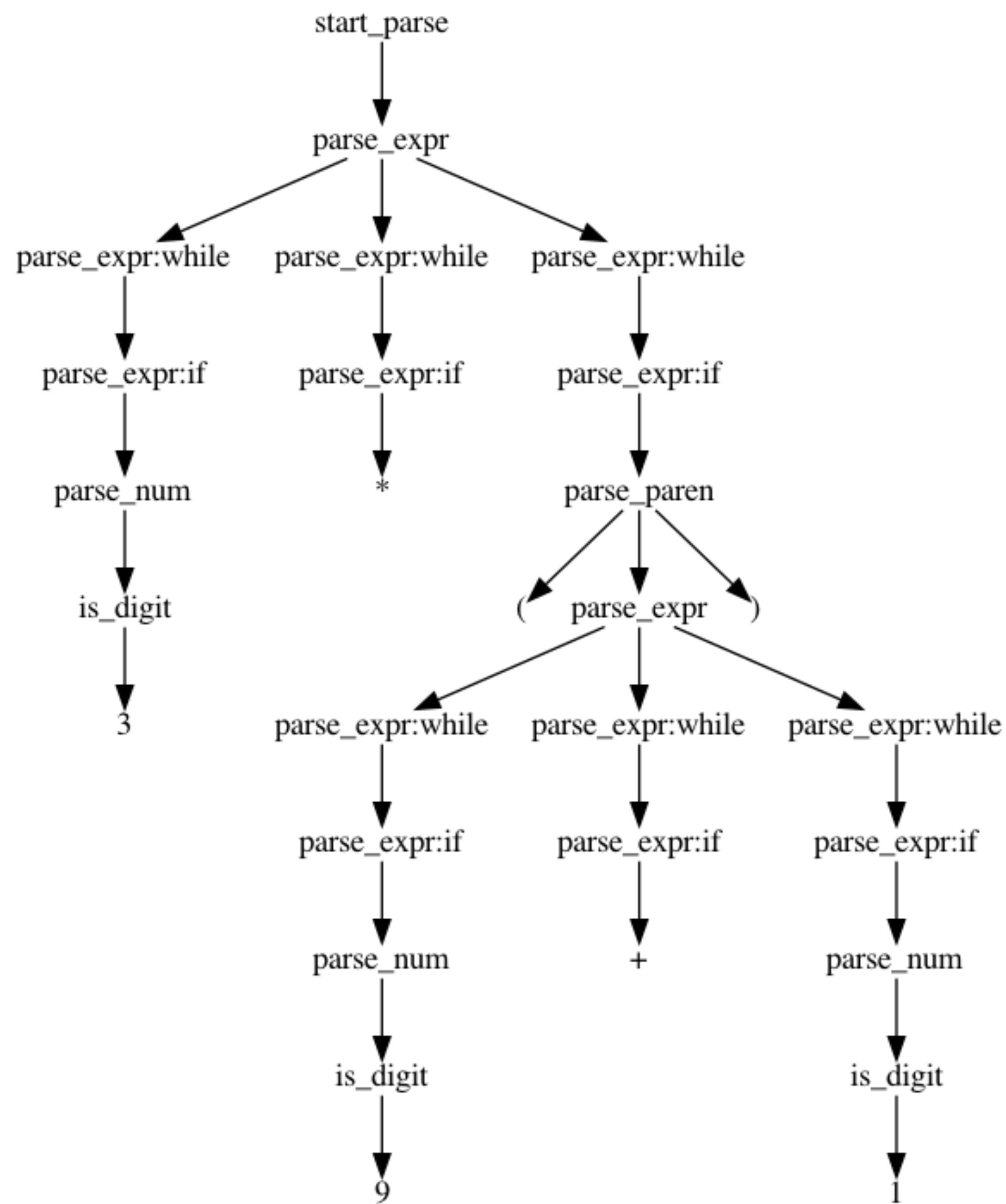


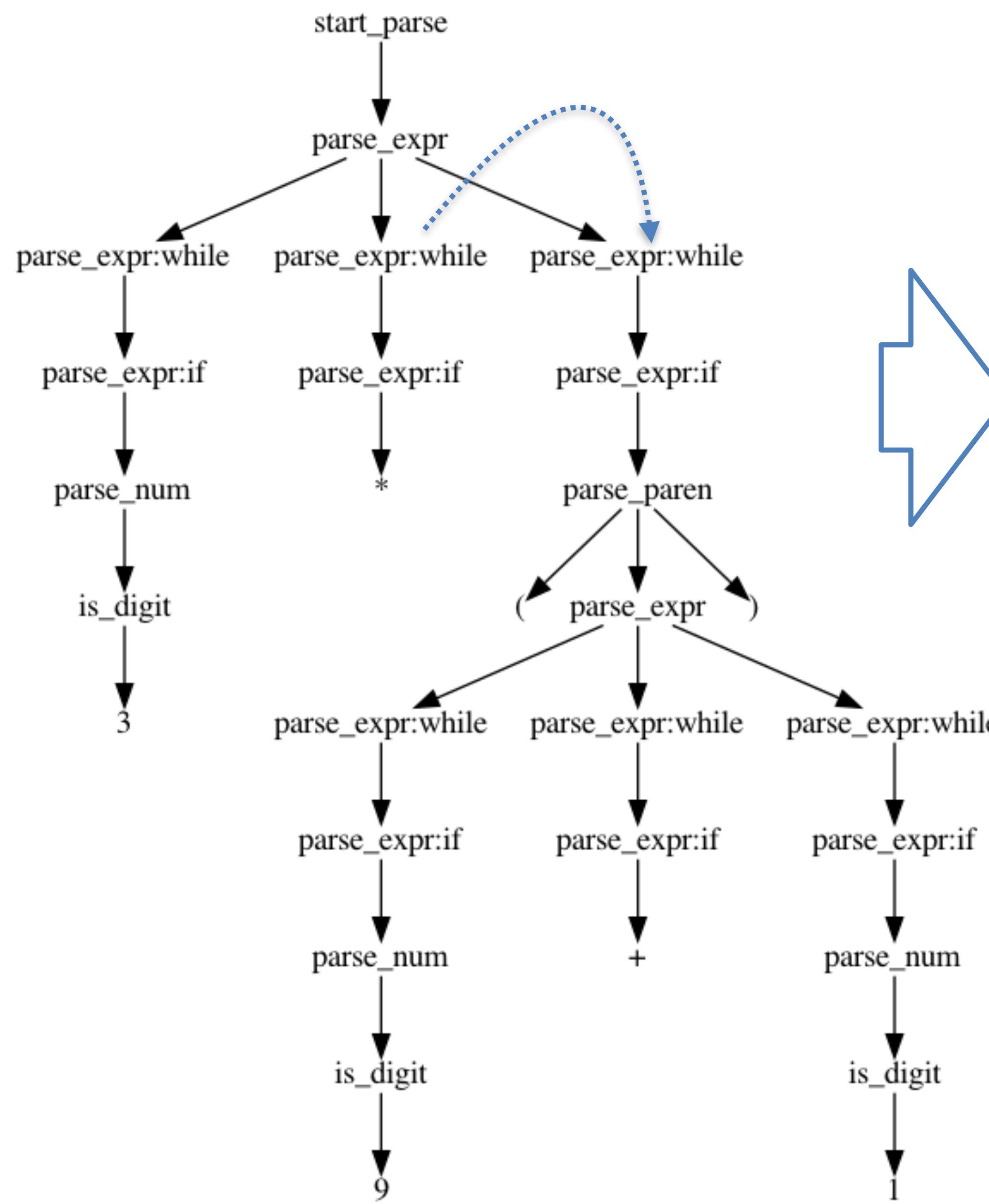
$3 * (9 + 1)$

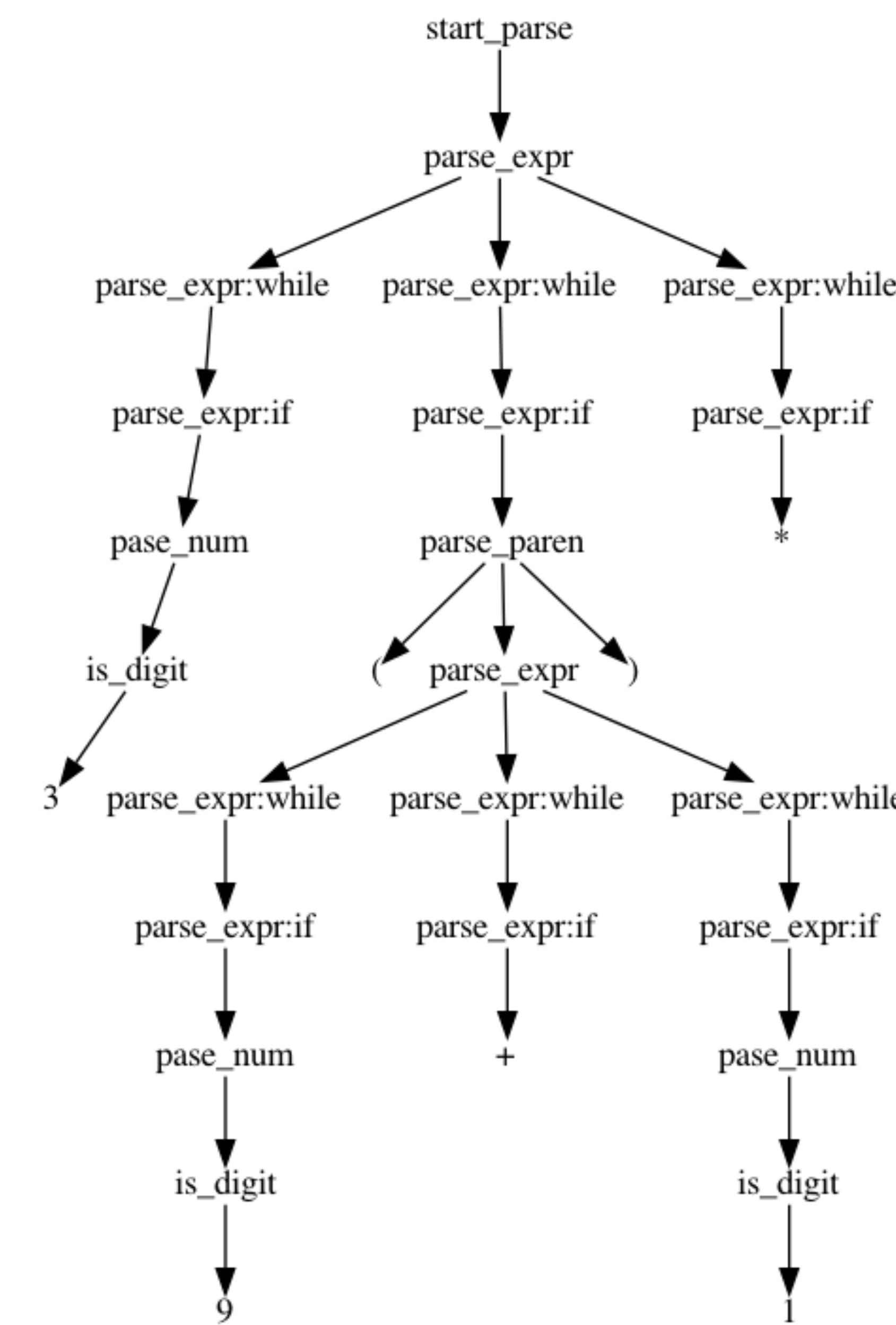


$9 + 1$

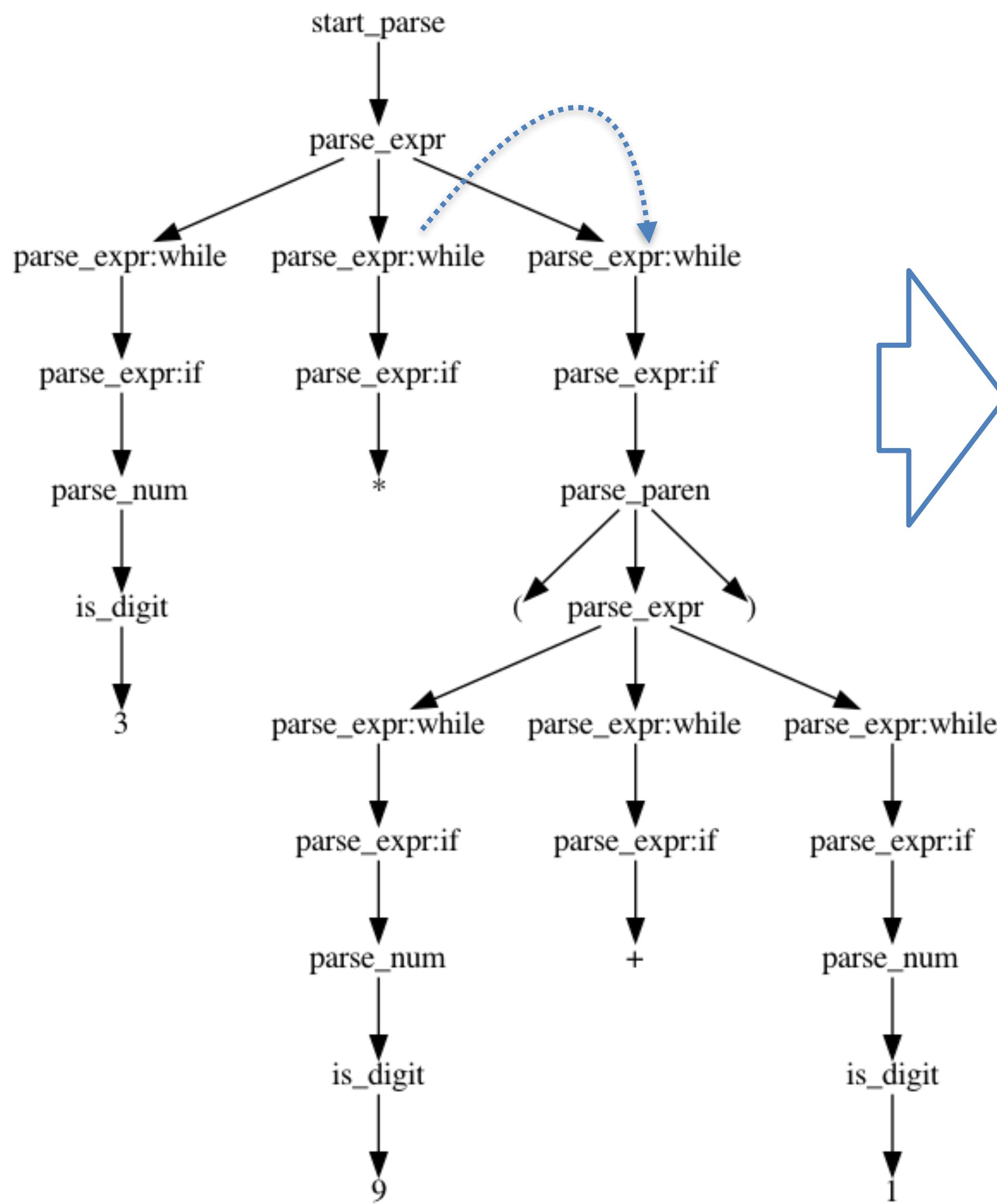


$$3 * (9 + 1)$$


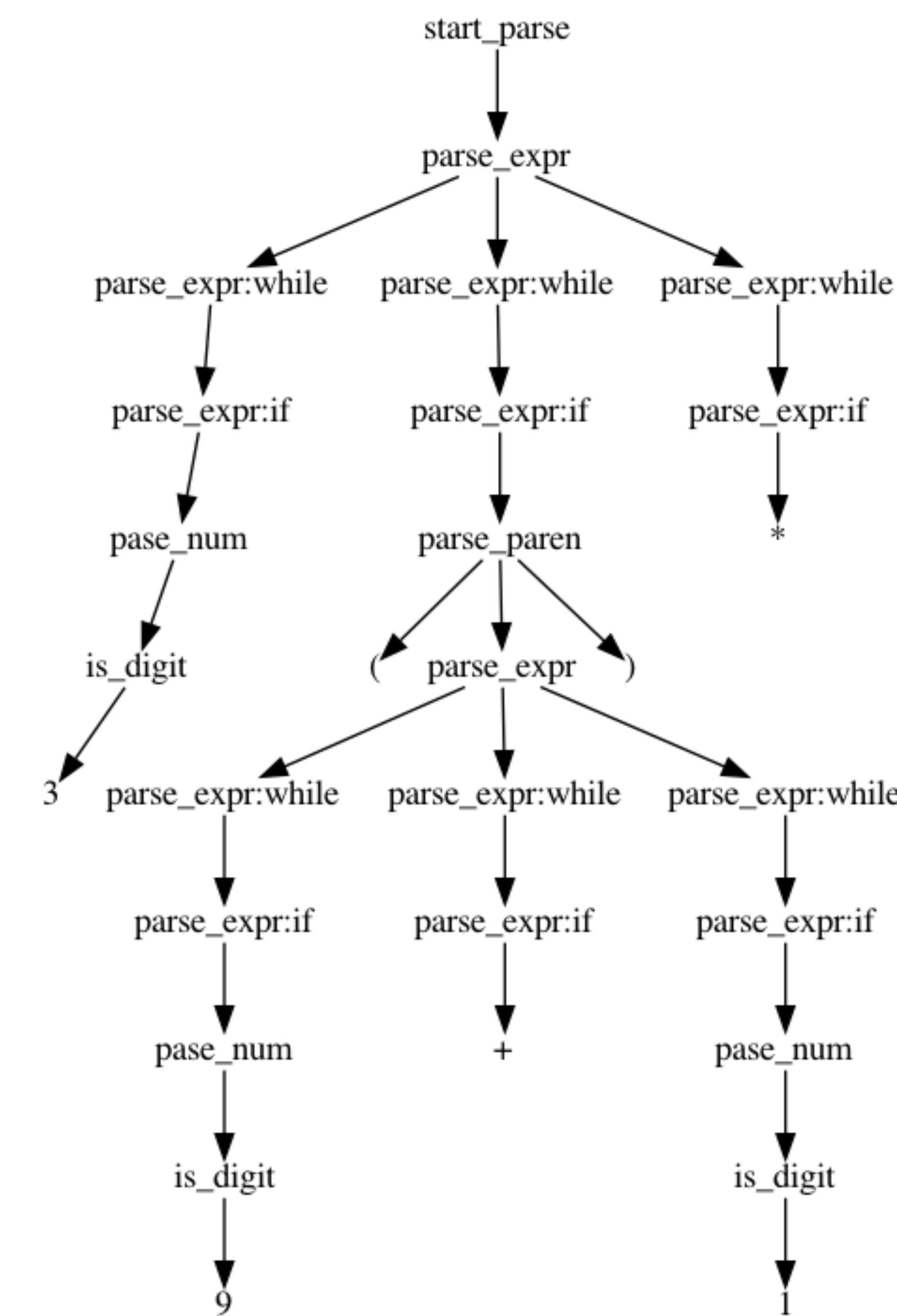
$$3 * (9 + 1)$$


$$3 (9 + 1) *$$


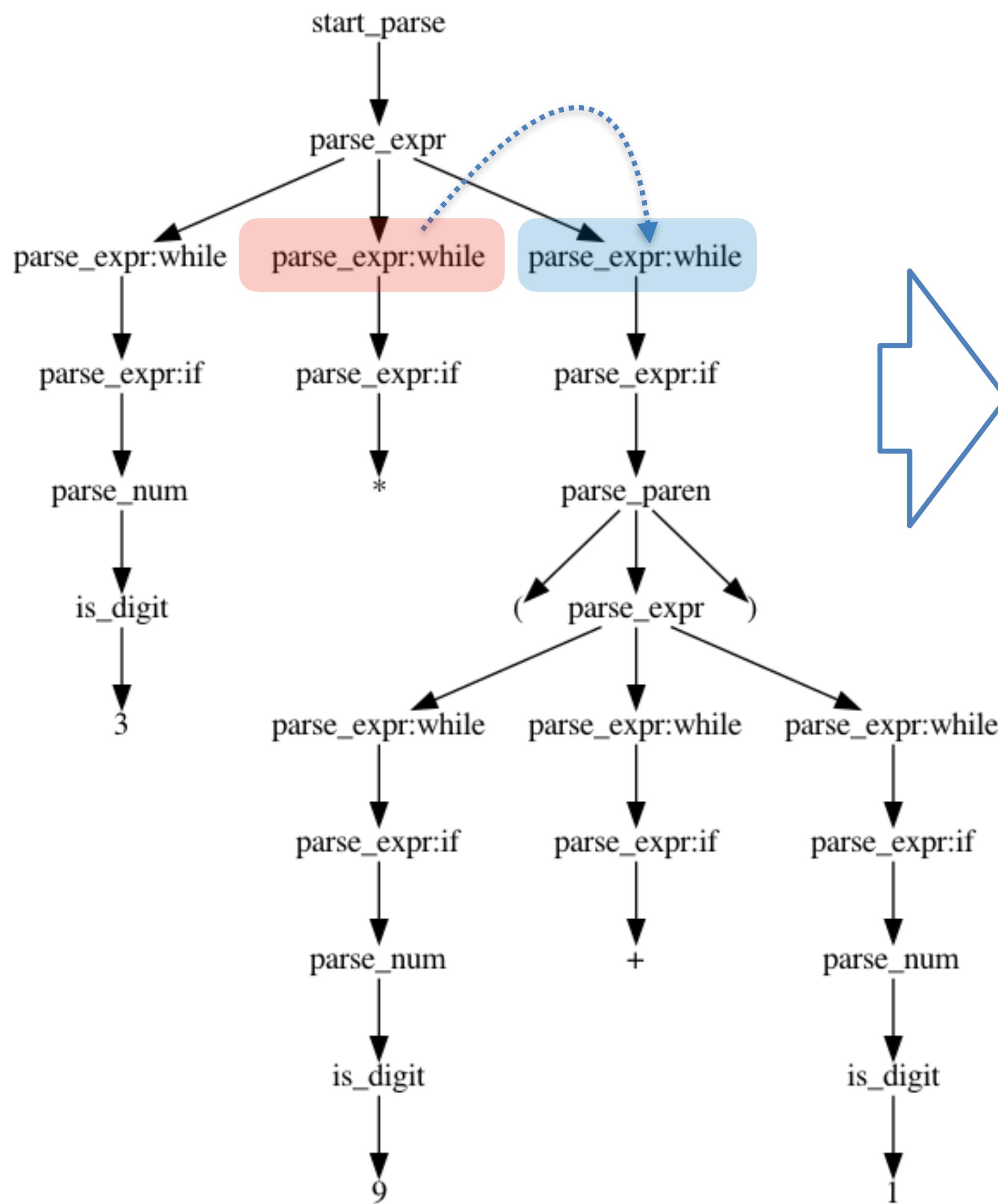
$3 * (9 + 1)$



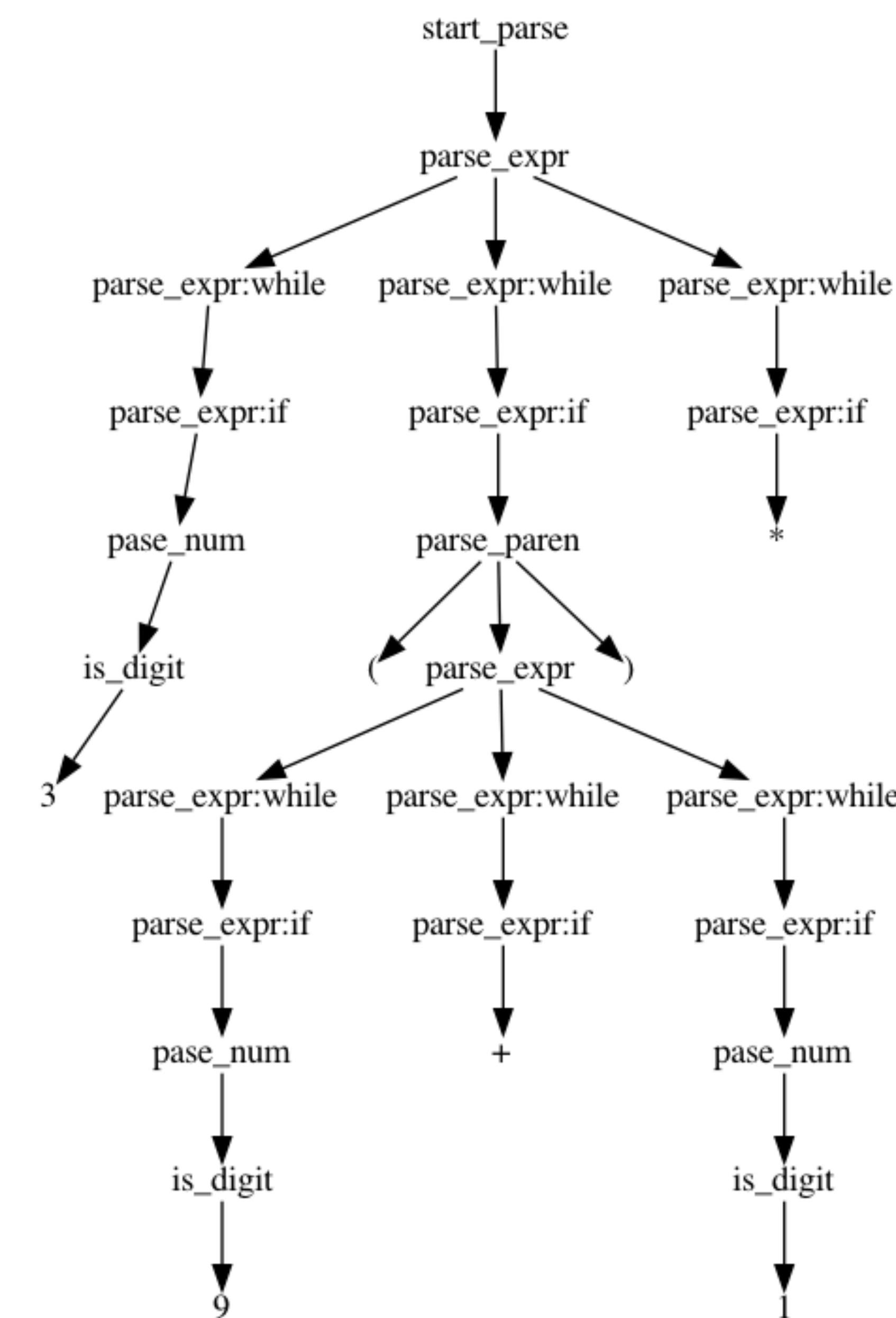
$3 (9 + 1) *$



$3 * (9 + 1)$



$3 (9 + 1) *$ ~~X~~



localhost:8888/notebooks/x2_0_MiningGrammar.ipynb

jupyter x2_0_MiningGrammar Last Checkpoint: 2 minutes ago (autosaved)

File Edit View Insert Cell Kernel Navigate Widgets Help Trusted Python 3 (ipykernel)

Contents

- 1.1 Imports
- 1.2 Subject Registry
- 1.3 Context Managers
 - 1.3.1 Method context
 - 1.3.2 Stack context
 - 1.3.3 Scope context
- 1.4 Rewriting the source to track context
 - 1.4.1 Rewriter
 - 1.4.1.1 The method context wrapper
 - 1.4.1.2 The stack wrapper
 - 1.4.1.3 The scope wrapper
 - 1.4.1.4 Rewriting If conditions
 - 1.4.1.5 Rewriting while loops
 - 1.4.1.6 Combining both
 - 1.4.2 Generating the complete instruction
 - 1.4.2.1 Using It
 - 1.4.3 Generate Transformed Source
 - 1.5 Generating Traces
 - 1.6 Mining the Traces Generated
 - 1.6.1 Reconstructing the Method Tree
 - 1.6.1.1 Identifying last comparison
 - 1.6.1.2 Attaching characters to the tree
 - 1.6.1.3 Removing Overlap
 - 1.6.1.4 Generate derivation tree
 - 1.6.2 The Complete Miner
 - 1.7 Generalize Nodes
 - 1.7.1 Generalize Methods
 - 1.7.2 Generalize Loops
 - 1.8 Generating a Grammar
 - 1.8.1 Trees to grammar
 - 1.8.2 Inserting Empty Alternatives for the start symbol
 - 1.8.3 Learning Regular Expressions
 - 1.8.3.1 The modified Fernau algorithm
 - 1.8.4 Remove duplicate and redundant nodes
 - 1.9 Accio Grammar
 - 2 Done

1.8 Generating a Grammar

Generating a grammar from the generalized derivation trees is pretty simple. Start at the start node, and any node that represents a method or a pseudo method becomes a nonterminal. The children forms alternate expansions for the nonterminal. Since all the keys are compatible, merging the grammar is simply merging the hash map.

First, we define a pretty printer for grammar.

```
In [ ]: import re
RE_NONTERMINAL = re.compile(r'(<[^> ]*>)')
```

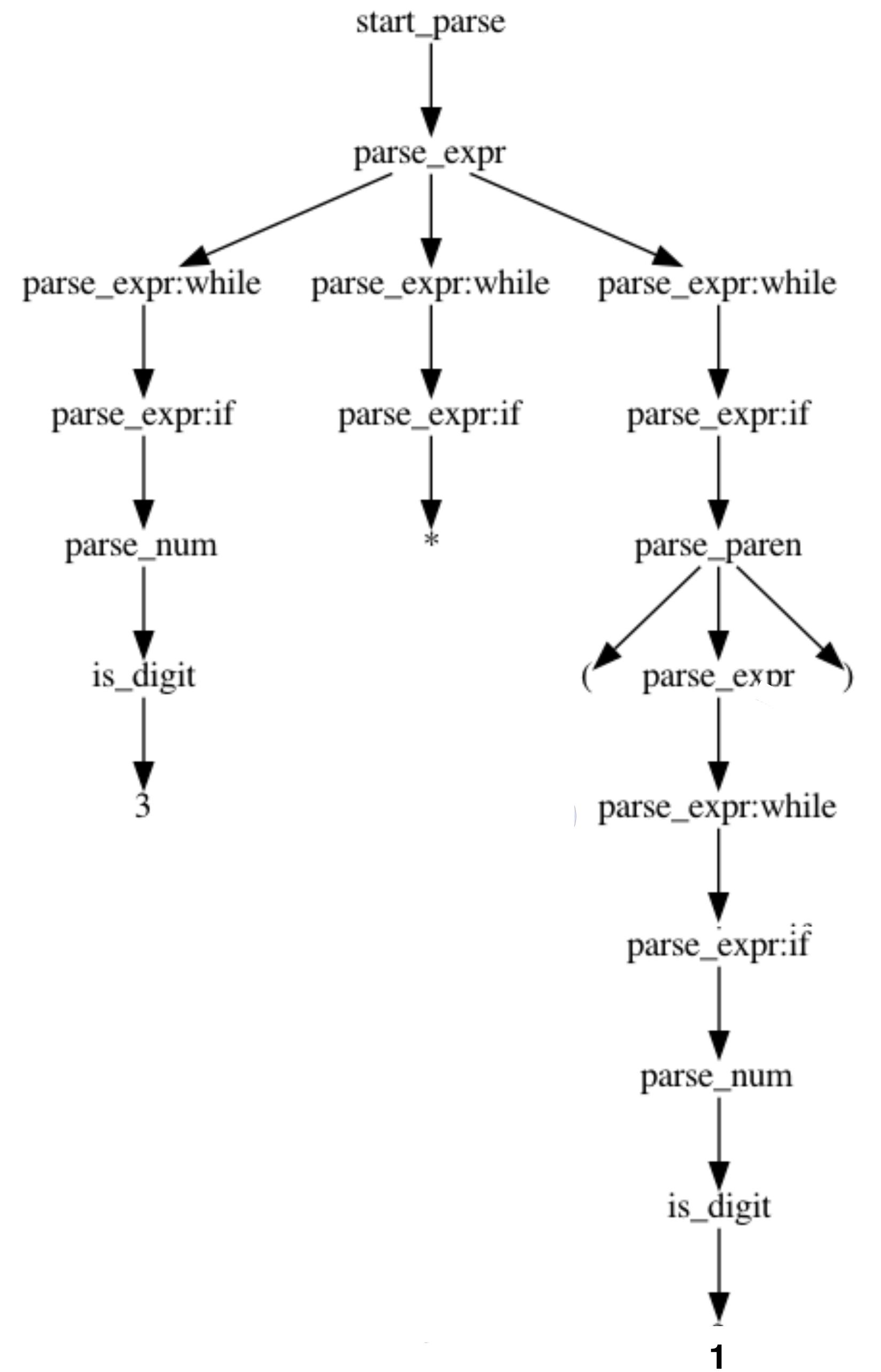
```
In [ ]: def recurse_grammar(grammar, key, order, canonical):
    rules = sorted(grammar[key])
    old_len = len(order)
    for rule in rules:
        if not canonical:
            res = re.findall(RE_NONTERMINAL, rule)
        else:
            res = rule
        for token in res:
            if token.startswith('<') and token.endswith('>'):
                if token not in order:
                    order.append(token)
    new = order[old_len:]
    for ckey in new:
        recurse_grammar(grammar, ckey, order, canonical)
```

```
In [ ]: def show_grammar(grammar, start_symbol='<START>', canonical=True):
    order = [start_symbol]
    recurse_grammar(grammar, start_symbol, order, canonical)
    if len(order) != len(grammar.keys()):
        assert len(order) < len(grammar.keys())
    return {k: sorted(grammar[k]) for k in order}
```

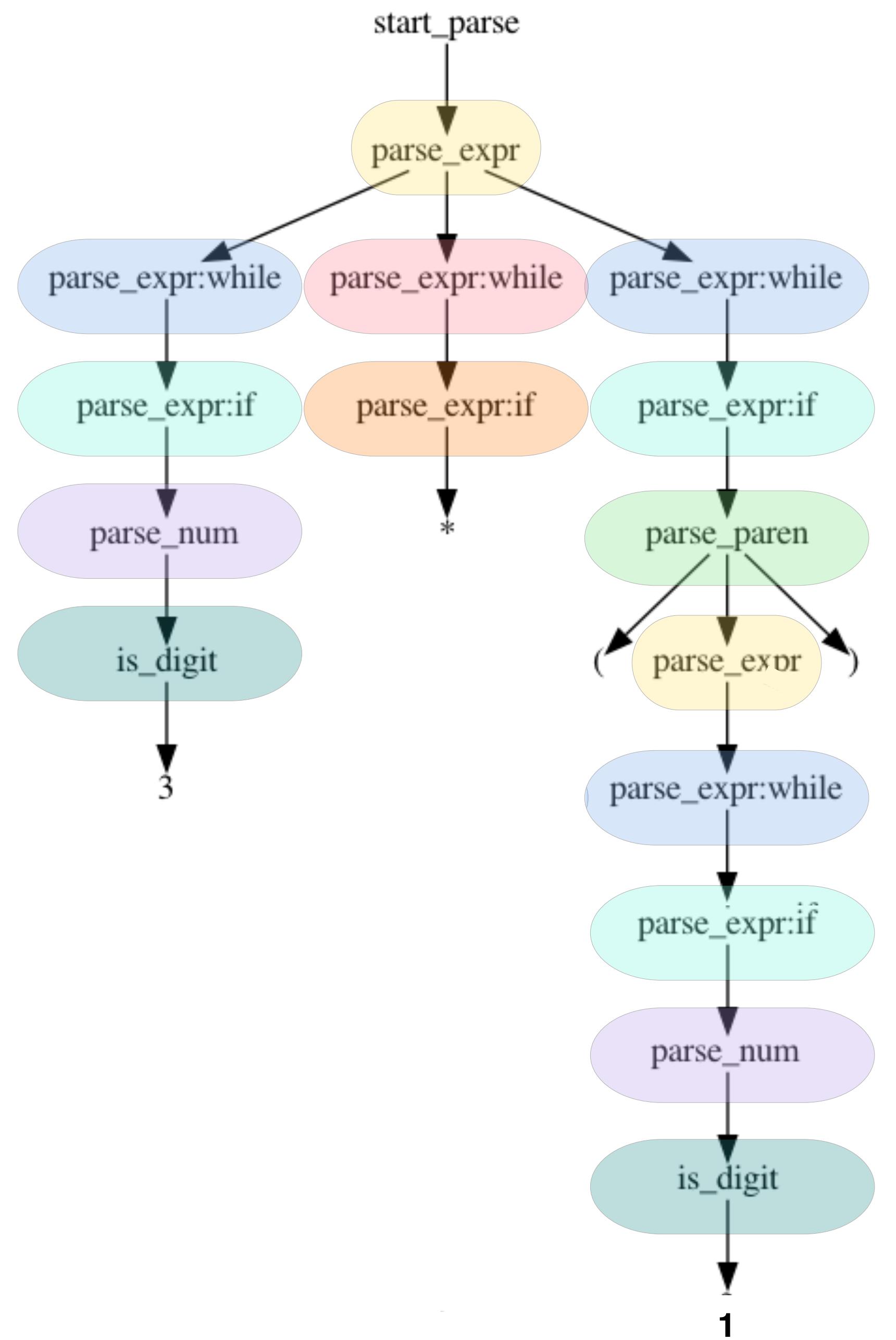
1.8.1 Trees to grammar

```
In [ ]: def to_grammar(tree, grammar):
    node, children, _ = tree
```

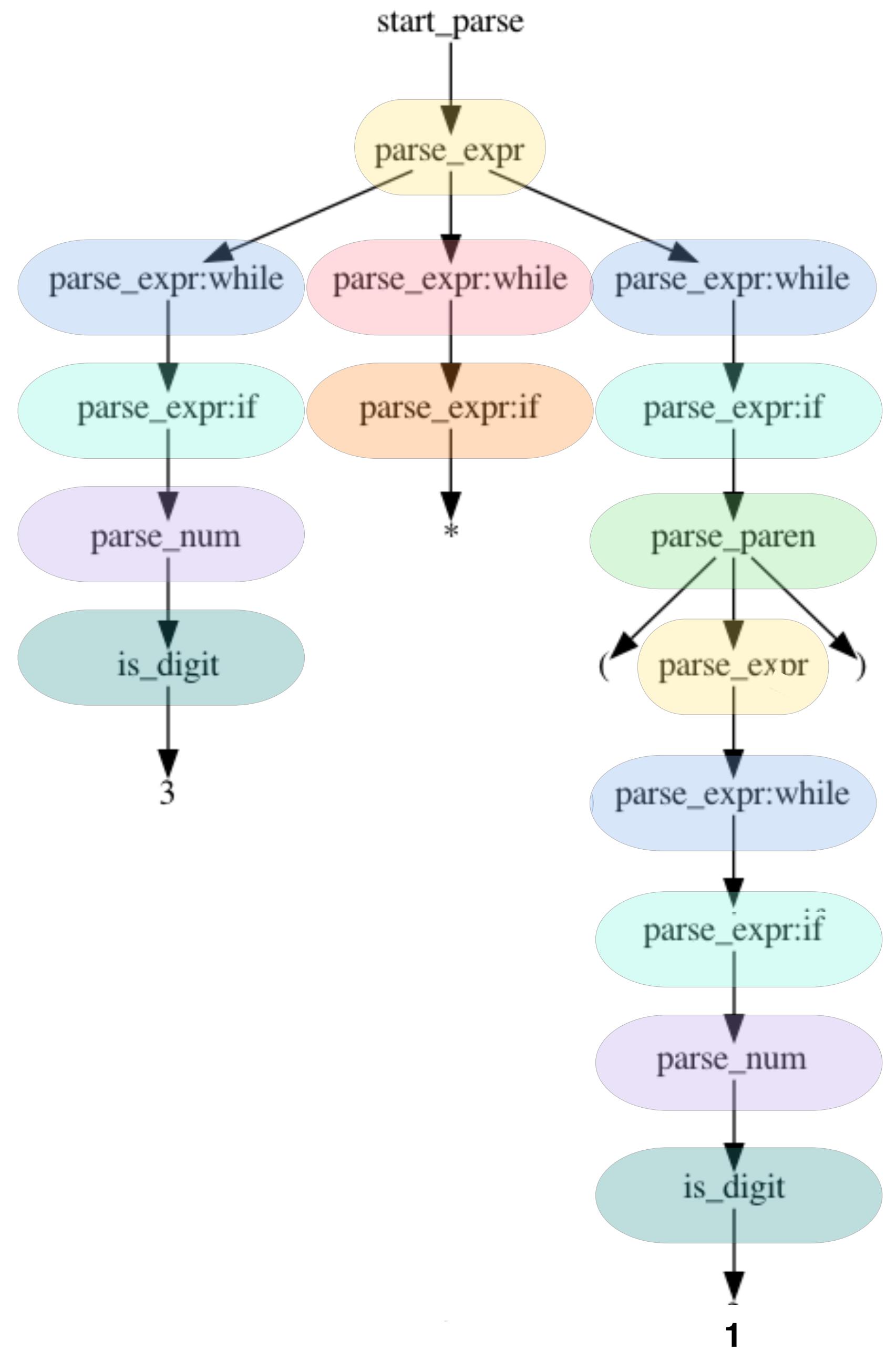
$3 * (1)$



3*(1)

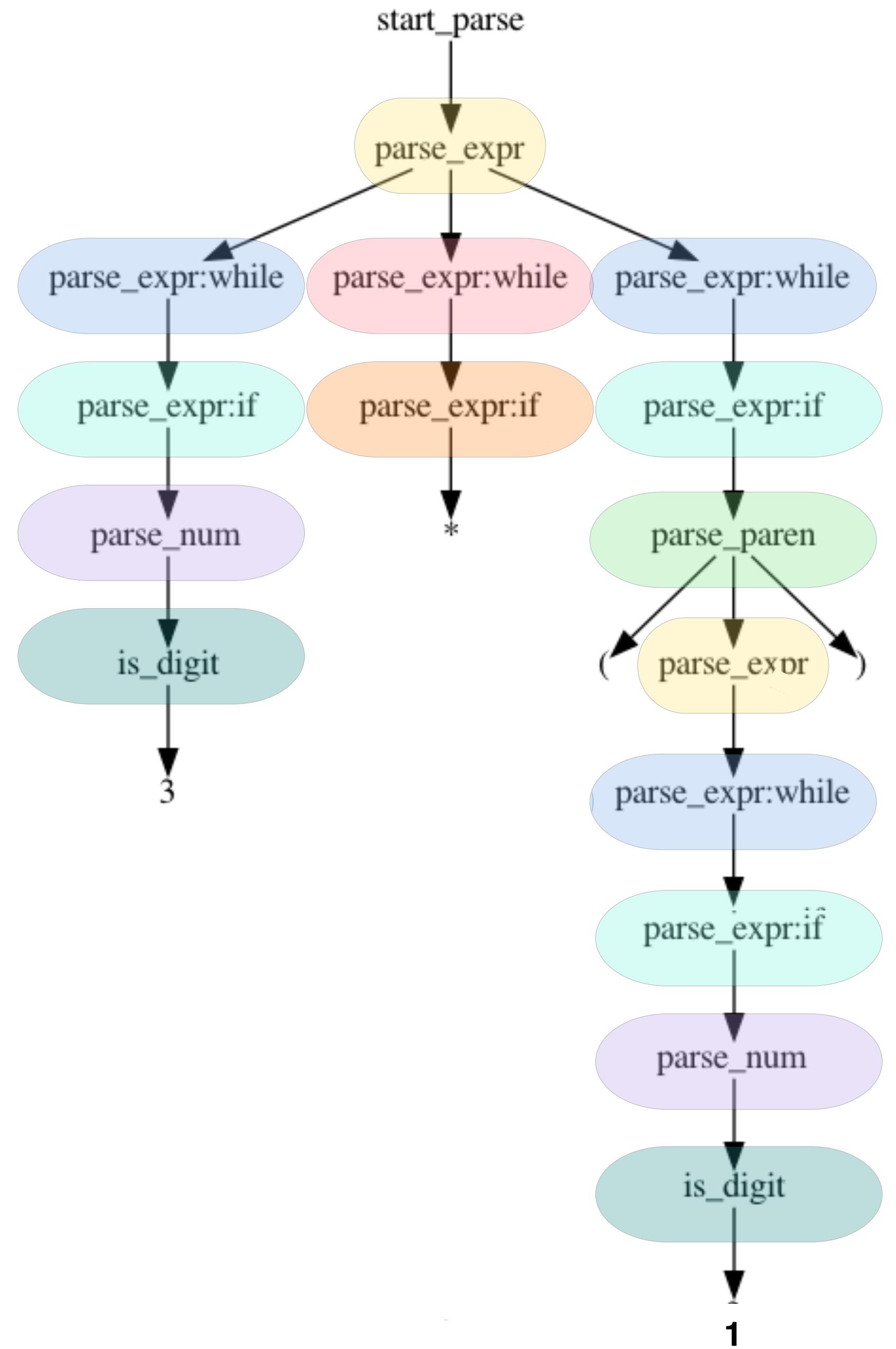


3*(1)



<parse_expr> := <while 1:1> <while 1:0> <while 1:1>

3*(1)



`<parse_expr>` := `<while 1:1>` `<while 1:0>` `<while 1:1>`

```
<parse_expr> := <while 1:1> <while 1:0> <while 1:1>
               | <while 1:1> <while 1:0> <while 1:1> <while 1:0> <while 1:1> <while 1:0> <while 1:1>
               | <while 1:1> <while 1:0> <while 1:1> <while 1:0> <while 1:1>
               | <while 1:1>

<while 1:1> := <if 1:1>
<if 1:1>   := <parse_num>
              | <parse_paren>
<parse_num> := <is_digit>
<is_digit>  := '3' | '1'

<parse_paren>:= '(' <parse_expr> ')'

<while 1:0>  := <if 1:0>
<if 1:0>    := '*'
```

```

<parse_expr> := <while 1:1> <while 1:0> <while 1:1>
               | <while 1:1> <while 1:0> <while 1:1> <while 1:0> <while 1:1> <while 1:0> <while 1:1>
               | <while 1:1> <while 1:0> <while 1:1> <while 1:0> <while 1:1>
               | <while 1:1>

<while 1:1> := <if 1:1>
<if 1:1>   := <parse_num>
              | <parse_paren>
<parse_num> := <is_digit>
<is_digit>  := '3' | '1'

<parse_paren>:= '(' <parse_expr> ')'

<while 1:0>  := <if 1:0>
<if 1:0>    := '*'

```

```

<parse_expr> := <while_s>

<while_s>   := <while_1:1> <while_1:0> <while_s>
               | <while_1:1>

```

```

def is_digit(i): return i in '0123456789'
def parse_num(s,i):
    n = ''
    while s[i:] and is_digit(s[i]):
        n += s[i]
        i = i +1
    return i,n

def parse_paren(s, i):
    assert s[i] == '('
    i, v = parse_expr(s, i+1)
    if s[i:] == '': raise Ex(s, i)
    assert s[i] == ')'
    return i+1, v

def parse_expr(s, i = 0):
    expr, is_op = [], True
    while s[i:]:
        c = s[i]
        if isdigit(c):
            if not is_op: raise Ex(s,i)
            i,num = parse_num(s,i)
            expr.append(num)
            is_op = False
        elif c in ['+', '-','*', '/']:
            if is_op: raise Ex(s,i)
            expr.append(c)
            is_op, i = True, i + 1
        elif c == '(':
            if not is_op: raise Ex(s,i)
            i, cexpr = parse_paren(s, i)
            expr.append(cexpr)
            is_op = False
        elif c == ')': break
        else: raise Ex(s,i)
    if is_op: raise Ex(s,i)
    return i, expr

```

<START>	:= <parse_expr.0-0-c>
<parse_expr.0-0-c>	:= <parse_expr.0-1-s><parse_expr.0> <parse_expr.0>
<parse_expr.0-1-s>	:= <parse_expr.0><parse_expr.0-2> <parse_expr.0><parse_expr.0-2><parse_expr.0-1-s>
<parse_expr.0>	:= '(' <parse_expr.0-0-c> ')' ' ' <parse_num.0-1-s>
<parse_expr.0-2>	:= '*' '+' '-' '/'
<parse_num.0-1-s>	:= <is_digit.0-0-c> <is_digit.0-0-c><parse_num.0-1-s>
<is_digit.0-0-c>	: [0-9]

```
<START>          := <parse_expr.0-0-c>

<parse_expr.0-0-c> := <parse_expr.0-1-s><parse_expr.0>
                     | <parse_expr.0>

<parse_expr.0-1-s> := <parse_expr.0><parse_expr.0-2>
                     | <parse_expr.0><parse_expr.0-2><parse_expr.0-1-s>

<parse_expr.0>    := '(' <parse_expr.0-0-c> ')'
                     | <parse_num.0-1-s>

<parse_expr.0-2>  := '*' | '+' | '-' | '/'

<parse_num.0-1-s> := <is_digit.0-0-c>
                     | <is_digit.0-0-c><parse_num.0-1-s>

<is_digit.0-0-c>  : [0-9]
```

```

<START>          := <parse_expr.0-0-c>
<parse_expr.0-0-c> := <parse_expr.0-1-s><parse_expr.0>
                     | <parse_expr.0>
<parse_expr.0-1-s> := <parse_expr.0><parse_expr.0-2>
                     | <parse_expr.0><parse_expr.0-2><parse_expr.0-1-s>
<parse_expr.0>    := '(' <parse_expr.0-0-c> ')'
                     | <parse_num.0-1-s>
<parse_expr.0-2>  := '*' | '+' | '-' | '/'
<parse_num.0-1-s> := <is_digit.0-0-c>
                     | <is_digit.0-0-c><parse_num.0-1-s>
<is_digit.0-0-c>  : [0-9]

```

```

8.2 - 27 - -9 / +((+9 * --2 + ---+-+
((-1 * +(8 - 5 - 6)) * (-a-+(((+(4)
)))) - ++4) / +(-+---((5.6 - --(3 *
-1.8 * +(6 * +-(((--6) * ----+6)) /
+--+(-+-7 * (-0 * (+((((2)) + 8 - 3
- ++9.0 + ---(--+7 / (1 / +++6.37)
+ (1) / 482) / +----+0)))) + 8.2 - 27
- -9 / +((+9 * --2 + ---+-+((-1 * +
(8 - 5 - 6)) * (-a-+(((+(4)))) - +
+4) / +(-+---((5.6 - --(3 * -1.8 * +
(6 * +-(((--6) * ----+6)) / +---(+-
7 * (-0 * (+((((2)) + 8 - 3 - ++9.0
+ ---(--+7 / (1 / +++6.37) + (1) /
482) / +----+0)))) * -+5 + 7.513)))) -
(+1 / ++((-84))))))) * ++5 / +-(-
-2 - ---+9.0)))) / 5 * ---+090 + * -
+5 + 7.513)))) - (+1 / ++((-84)))))))
)) * 8.2 - 27 - -9 / +((+9 * --2 + -
---+(-1 * +(8 - 5 - 6)) * (-a-+(
((4)))) - ++4) / +(-+---((5.6 - --
(3 * -1.8 * +(6 * +-(((--6) * ----+6
)))) / +--+(-+-7 * (-0 * (+((((2)) +
8 - 3 - ++9.0 + ---(--+7 / (1 / +++6
.37) + (1) / 482) / +----+0)))) * -+5
+ 7.513)))) - (+1 / ++((-84)))))))
* ++5 / +-(-2 - ---+9.0)))) / 5 *
---+090 ++5 / +-(-2 - ---+9.0)))) /
5 * ---+090

```

```

        stm.next()
    if expect_key:
        raise JSONError(E_DKEY, stm, stm.pos)
    if c == '}':
        return result
    expect_key = 1
    continue

# parse out a key/value pair
elif c == '"':
    key = _from_json_string(stm)
    stm.skipspaces()
    c = stm.next()
    if c != ':':
        raise JSONError(E_COLON, stm, stm.pos)
    stm.skipspaces()
    val = _from_json_raw(stm)
    result[key] = val
    expect_key = 0
    continue
raise JSONError(E_MALF, stm, stm.pos)

def _from_json_raw(stm):
    while True:
        stm.skipspaces()
        c = stm.peek()
        if c == '"':
            return _from_json_string(stm)
        elif c == '{':
            return _from_json_dict(stm)
        elif c == '[':
            return _from_json_list(stm)
        elif c == 't':
            return _from_json_fixed(stm, 'true', True, E_BOOL)
        elif c == 'f':
            return _from_json_fixed(stm, 'false', False, E_BOOL)
        elif c == 'n':
            return _from_json_fixed(stm, 'null', None, E_NULL)
        elif c in NUMSTART:
            return _from_json_number(stm)
        raise JSONError(E_MALF, stm, stm.pos)

def from_json(data):
    stm = JSONStream(data)
    return _from_json_raw(stm)

```

```

<START>      ::= <json_raw>
<json_raw>    ::= '"' <json_string>
| '[' <json_list>
| '{' <json_dict>
| <json_number>
| 'true'
| 'false'
| 'null'
| <json_number>+
| <json_number>+ 'e' <json_number>+
 ::= '+' | '-' | '.' | [0-9] | 'E' | 'e'
 ::= <json_string>* '"'
 ::= ']'
| <json_raw> (',' <json_raw>)* ']'
| (',' <json_raw>)+ (',' <json_raw>)* ']'
 ::= '}'
| ('"' <json_string> ':' <json_raw> ',' )*
| "''" <json_string> ':' <json_raw> '}'
 ::= ' ' | '!' | '#' | '$' | '%' | '&' | ''
| '*' | '+' | '-' | ',', | '.' | '/' | ':' | ';'
| '<' | '=' | '>' | '?' | '@' | '[' | ']' | '^'
| '_', "'", | '{' | '}' | '}' | '~'
| '[A-Za-z0-9]'
| '\' <decode_escape>
<decode_escape> ::= '"' | '/' | 'b' | 'f' | 'n' | 'r' | 't'

```

< > C ☰ | 🔒 nbviewer.org/github/vrthra/mimid/blob/master/src/PymimidBook.ipynb

JUPYTER FAQ </> ⚡ GitHub ⚡ ⚡ ⚡

mimid / src

Mimid : Inferring Grammars

- Code for subjects [here](#)
- Evaluation starts [here](#)
 - The evaluation on specific subjects starts [here](#)
 - [CGIDecode](#)
 - [Calculator](#)
 - [MathExpr](#)
 - [URLParse](#)
 - [Microjson](#)
 - [Lisp](#)
- Results are [here](#)
- Recovering parse tree from a recognizer is [here](#))
- Recovering parse tree from parser combinators is [here](#)
- Recovering parse tree from PEG parer is [here](#)

Please note that a complete run can take an hour and a half to complete.

We start with a few Jupyter magics that let us specify examples inline, that can be turned off if needed for faster execution. Switch `TOP` to `False` if you do not want examples to complete.

```
In [1]: TOP = __name__ == '__main__'
```

The magics we use are `%%var` and `%top`. The `%%var` lets us specify large strings such as file contents directly without too many escapes. The `%top` helps with examples.

```
In [2]: from IPython.core.magic import (Magics, magics_class, cell_magic, line_magic, line_cell_magic)
class B(dict):
    def __getattr__(self, name):
        return self.__getitem__(name)
@magics_class
class MyMagics(Magics):
    def __init__(self, shell=None, **kwargs):
        super().__init__(shell=shell, **kwargs)
```


We Found A Crash

We Found A Crash 🔥

Why Did My Program Crash?

Why Did My Program Crash?

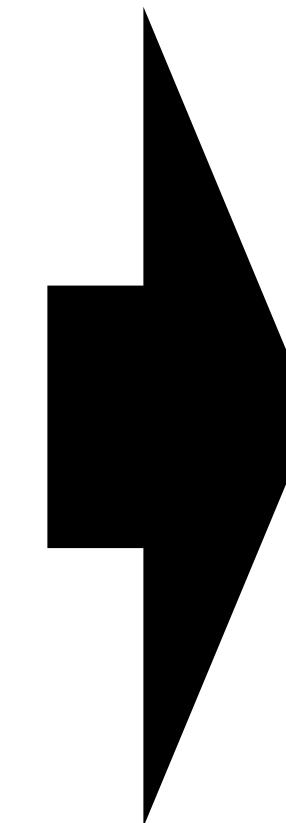


```
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4)))))) - ++4) / +(-+---((5.
6 - --(3 * -1.8 * +(6 * +-(((--6) * ----+6)) / +-
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) + +
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4)))))) - ++4) / +(-+---((5.6
- --(3 * -1.8 * +(6 * +-(((--6) * ----+6)) / +-
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) * -
+5 + 7.513)))) - (+1 / ++((-84))))))) * ++5 / +-
(--2 - -+-9.0)))) / 5 * ---+090 + * --+5 + 7.513)
))) - (+1 / ++((-84))))))) * 8.2 - 27 - -9 / +(
+9 * --2 + ---+-((-1 * +(8 - 5 - 6)) * (-(a-
(((+(4)))))) - ++4) / +(-+---((5.6 - --(3 * -1.8 *
+(6 * +-(((--6) * ----+6)) / +--+7 * (-0 * (
+((((2)) + 8 - 3 - ++9.0 + ---(+7 / (1 / +++6.
37) + (1) / 482) / +--+0)))) * --+5 + 7.513)))) -
(+1 / ++((-84))))))) * ++5 / +--(--2 - -+-9.0)))
) / 5 * ---+090 ++5 / +--(--2 - -+-9.0)))) / 5 *
---+090
```

Why Did My Program Crash?



```
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4)))))) - ++4) / +(-+---((5.
6 - --(3 * -1.8 * +(6 * +-(((--6) * ----+6)) / +
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) + +
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4)))))) - ++4) / +(-+---((5.6
- --(3 * -1.8 * +(6 * +-(((--6) * ----+6)) / +
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) * -
+5 + 7.513)))) - (+1 / ++((-84))))))) * ++5 / +
(--2 - -++-9.0)))) / 5 * ---+090 + * --+5 + 7.513)
))) - (+1 / ++((-84))))))) * 8.2 - 27 - -9 / +(
+9 * --2 + ---+-((-1 * +(8 - 5 - 6)) * (-(a-
(((+(4)))))) - ++4) / +(-+---((5.6 - --(3 * -1.8 *
+(6 * +-(((--6) * ----+6)) / +--+7 * (-0 * (
+((((2)) + 8 - 3 - ++9.0 + ---(+7 / (1 / +++6.
37) + (1) / 482) / +--+0)))) * --+5 + 7.513)))) -
(+1 / ++((-84))))))) * ++5 / +--(--2 - -++-9.0)))
) / 5 * ---+090 ++5 / +--(--2 - -++-9.0)))) / 5 *
---+090
```



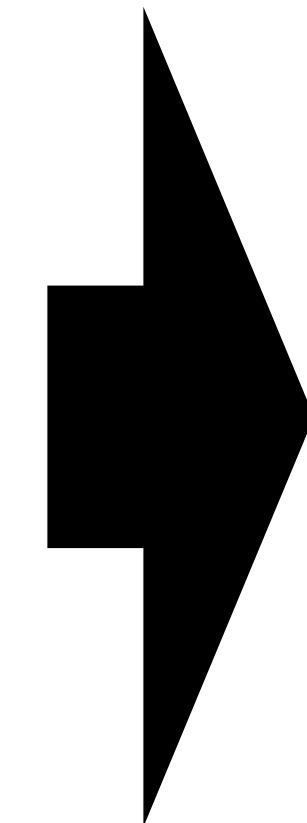
((4))

DD Minimized Input
50

Why Did My Program Crash?



```
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4)))))) - ++4) / +(-+---((5.
6 - --(3 * -1.8 * +(6 * +-(((--6) * ---+6)) / +-
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) + 
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4)))))) - ++4) / +(-+---((5.6
- --(3 * -1.8 * +(6 * +-(((--6) * ---+6)) / +-
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) * -
+5 + 7.513)))) - (+1 / ++((-84))))))) * ++5 / +-
(--2 - -+-9.0)))) / 5 * ---+090 + * --+5 + 7.513)
))) - (+1 / ++((-84))))))) * 8.2 - 27 - -9 / +(
+9 * --2 + ---+-((-1 * +(8 - 5 - 6)) * (-(a-
(((+(4)))) - ++4) / +(-+---((5.6 - --(3 * -1.8 *
+(6 * +-(((--6) * ---+6)) / +--+7 * (-0 * (
+((((2)) + 8 - 3 - ++9.0 + ---(+7 / (1 / +++6.
37) + (1) / 482) / +--+0)))) * --+5 + 7.513)))) -
(+1 / ++((-84))))))) * ++5 / +--(--2 - -+-9.0)))
) / 5 * ---+090 ++5 / +--(--2 - -+-9.0)))) / 5 *
---+090
```



((4))

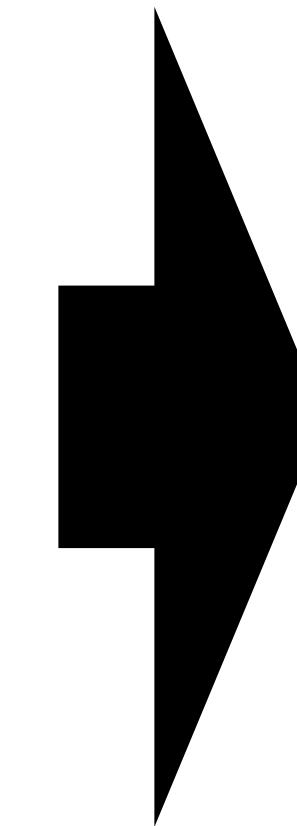
00000 ?

DD Minimized Input
50

Why Did My Program Crash?



```
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4)))))) - ++4) / +(-+---((5.
6 - --(3 * -1.8 * +(6 * +-(((--6) * ---+6)) / +-
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) + 
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4)))))) - ++4) / +(-+---((5.6
- --(3 * -1.8 * +(6 * +-(((--6) * ---+6)) / +-
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) * -
+5 + 7.513)))) - (+1 / ++((-84))))))) * ++5 / +-
(--2 - -+-9.0)))) / 5 * ---+090 + * --5 + 7.513)
))) - (+1 / ++((-84))))))) * 8.2 - 27 - -9 / +(
+9 * --2 + ---+-((-1 * +(8 - 5 - 6)) * (-(a-
(((+(4)))))) - ++4) / +(-+---((5.6 - --(3 * -1.8 *
+(6 * +-(((--6) * ---+6)) / +--+7 * (-0 * (
+((((2)) + 8 - 3 - ++9.0 + ---(+7 / (1 / +++6.
37) + (1) / 482) / +--+0)))) * --5 + 7.513)))) -
(+1 / ++((-84))))))) * ++5 / +--(--2 - -+-9.0)))
) / 5 * ---+090 ++5 / +--(--2 - -+-9.0)))) / 5 *
---+090
```



((4))

00000 ?

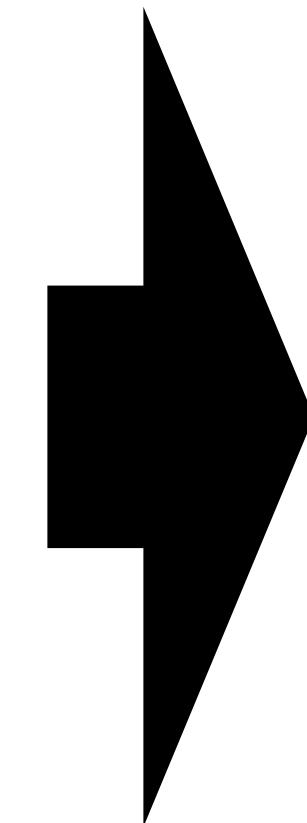
((5)) ?

DD Minimized Input

Why Did My Program Crash?



```
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4))))))) - ++4) / +(-+---((5.
6 - --(3 * -1.8 * +(6 * +-(((--6) * ----+6)) / +-
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) + 
8.2 - 27 - -9 / +((+9 * --2 + ---+-((-1 * +(8 -
5 - 6)) * (-(a-+(((+(4))))))) - ++4) / +(-+---((5.6
- --(3 * -1.8 * +(6 * +-(((--6) * ----+6)) / +-
- (+--+7 * (-0 * (+((((2)) + 8 - 3 - ++9.0 + ---(
--+7 / (1 / +++6.37) + (1) / 482) / +--+0)))) * -
+5 + 7.513)))) - (+1 / ++((-84))))))) * ++5 / +-
(--2 - -++-9.0)))) / 5 * ---+090 + * --+5 + 7.513)
))) - (+1 / ++((-84))))))) * 8.2 - 27 - -9 / +(
+9 * --2 + ---+-((-1 * +(8 - 5 - 6)) * (-(a-
(((+(4)))) - ++4) / +(-+---((5.6 - --(3 * -1.8 *
+(6 * +-(((--6) * ----+6)) / +--+7 * (-0 * (
+((((2)) + 8 - 3 - ++9.0 + ---(+7 / (1 / +++6.
37) + (1) / 482) / +--+0)))) * --+5 + 7.513)))) -
(+1 / ++((-84))))))) * ++5 / +--(--2 - -++-9.0)))
) / 5 * ---+090 ++5 / +--(--2 - -++-9.0)))) / 5 *
---+090
```



((4))

00000 ?

((5)) ?

(++5) ?

DD Minimized Input

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** localhost:8888/notebooks/x3_0_AbstractingInputs.ipynb
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, Help, Trusted, Python 3 (ipykernel)
- Contents Sidebar:** Shows a tree view of the notebook's structure:
 - 1 Abstracting Inputs
 - 1.1 Imports
 - 1.2 Grammars
 - 1.3 Predicate
 - 1.4 DDSets Simple
 - 1.4.1 Example
 - 1.5 Single Fault Grammar
 - 1.5.1 Reachable Grammar
 - 1.5.2 Reachable positions.
 - 1.5.3 Insertion Grammar
 - 1.5.4 Pattern Grammar
 - 1.5.5 Insert A Fault
 - 1.6 Number the nodes.
 - 1.7 Abstract Context- 2 Done

- Main Content Area:** Displays the first section of the notebook.

1 Abstracting Inputs

1.1 Imports

```
In [ ]: import src.utils as utils
```

```
In [ ]: import ipynb.fs.full.x0_4_HierarchicalReducer as hdd
```

```
In [ ]: import ipynb.fs.full.x0_2_GrammarFuzzer as fuzzer
```

```
In [ ]: import ipynb.fs.full.x0_3_Parser as parser
```

```
In [ ]: from ipynb.fs.full.x0_4_HierarchicalReducer import PRes
```

```
In [ ]: import ast
```

```
In [ ]: import json
```

```
In [ ]: import random
```

```
In [ ]: if __name__ == '__main__':  
    random.seed(0)
```

1.2 Grammars

```
In [ ]: import src.grammars as grammars
```

1.3 Predicate

```
In [ ]: import re
```

```
In [ ]: def expr_double_paren(inp):
```

```
var A = class extends (class {}){};
```

Issue 2937 from Closure

```
const [y,y] = [];
```

Issue 386 from Rhino

```
var {baz:{} = baz => {}} = baz => {};
```

Issue 385 from Rhino

```
{while ((l_0)){ if ((l_0)) {break;;var l_0; continue }0}}
```

Issue 2842 from Closure

```
var A = class extends (class {}){};
```

Issue 2937 from Closure

```
const [y,y] = [];
```

Issue 386 from Rhino

```
var {baz:{} = baz => {}} = baz => {};
```

Issue 385 from Rhino

```
{while ((l_0)){ if ((l_0)) {break;;var l_0; continue }0}}
```

Issue 2842 from Closure

```
var A = class extends (class {}){};
```

Issue 2937 from Closure

```
const [y,y] = [];
```

Issue 386 from Rhino



```
var {baz:{} = baz => {}} = baz => {};
```

Issue 385 from Rhino

```
{while ((l_0)){ if ((l_0)) {break;;var l_0; continue }0}}
```

Issue 2842 from Closure

```
var A = class extends (class {}){};
```

Issue 2937 from Closure

```
const [y,y] = [];
```

Issue 386 from Rhino

```
var {baz:{} = baz => {}} = baz => {};
```

Issue 385 from Rhino

```
{while ((l_0)){ if ((l_0)) {break;;var l_0; continue }0}}
```

Issue 2842 from Closure

```
var A = class extends (class {}){};
```

Issue 2937 from Closure

```
const [y,y] = [];
```

Issue 386 from Rhino

```
var {baz:{} = baz => {}} = baz => {};
```

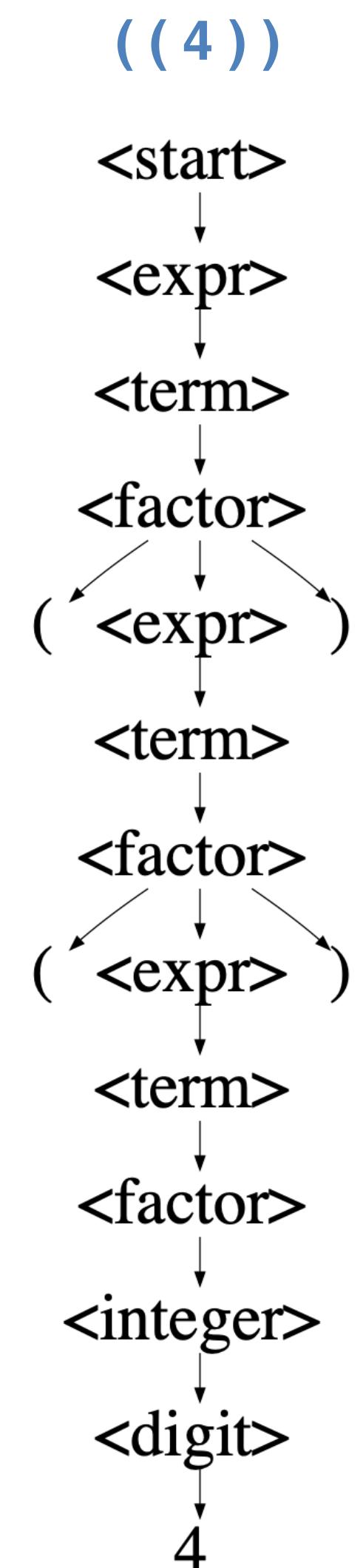
Issue 385 from Rhino

```
{while ((l_0)){ if ((l_0)) {break;;var l_0; continue }0}}
```

Issue 2842 from Closure

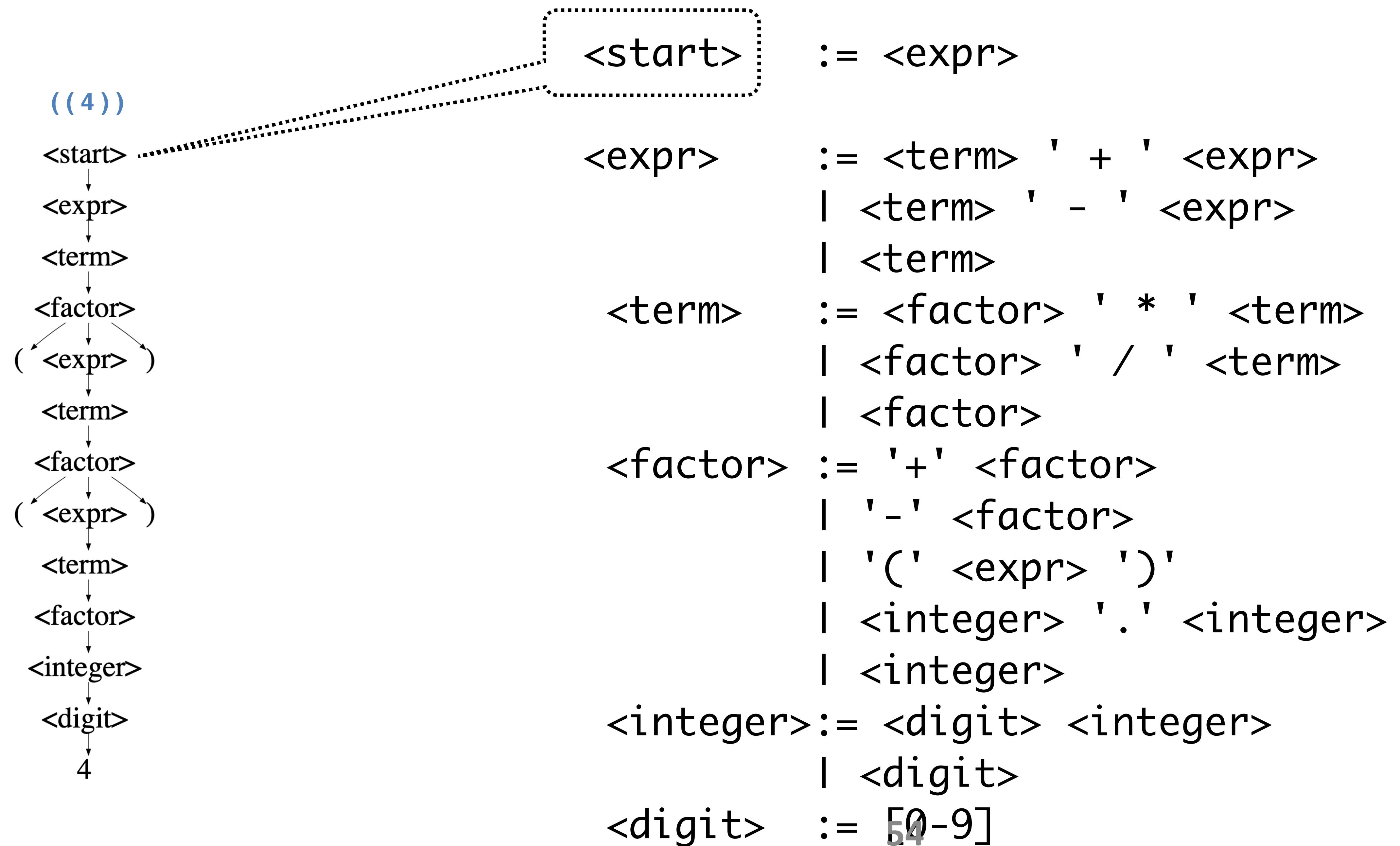
Delta Minimization is useful but not sufficient

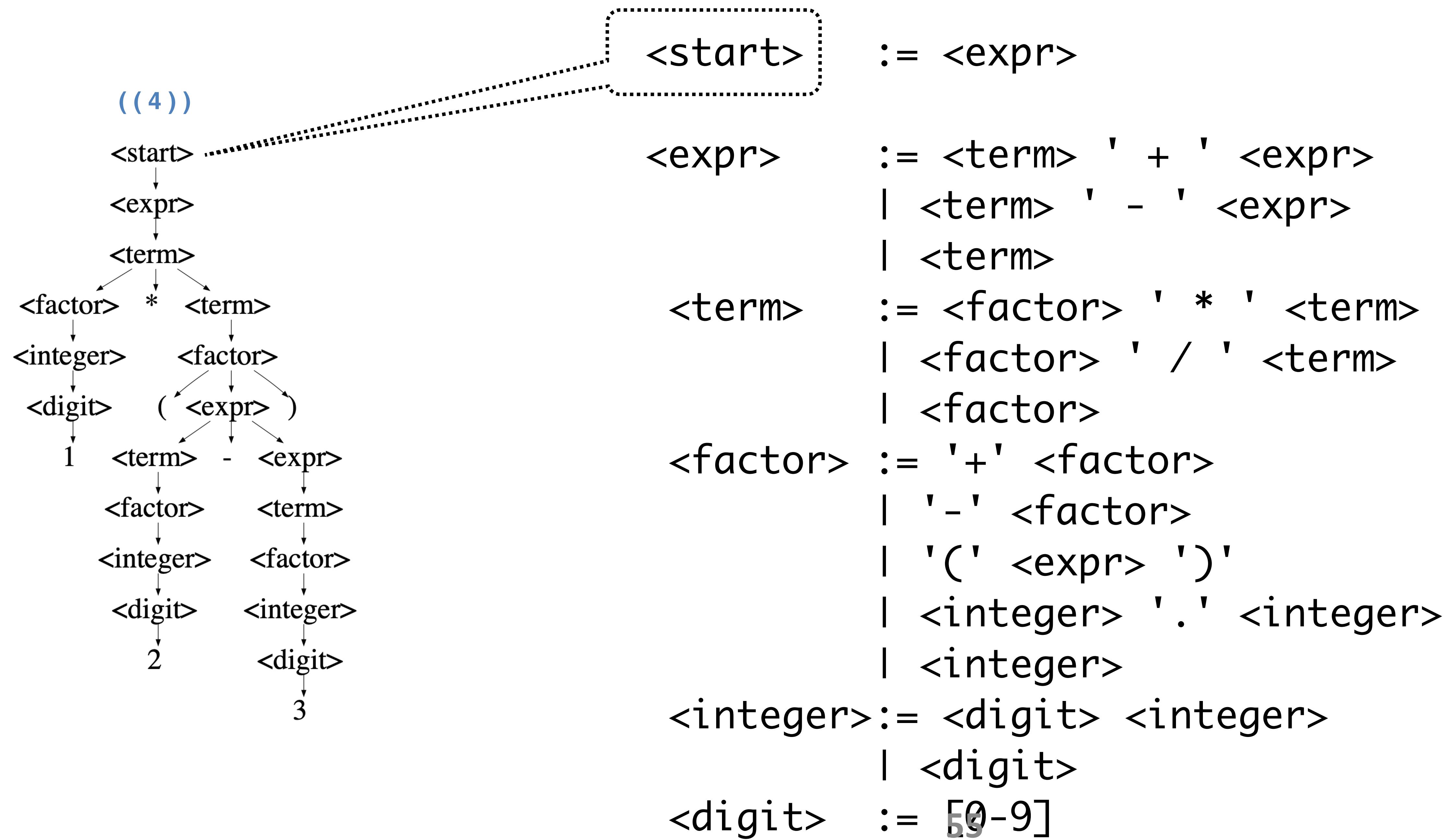
((4))

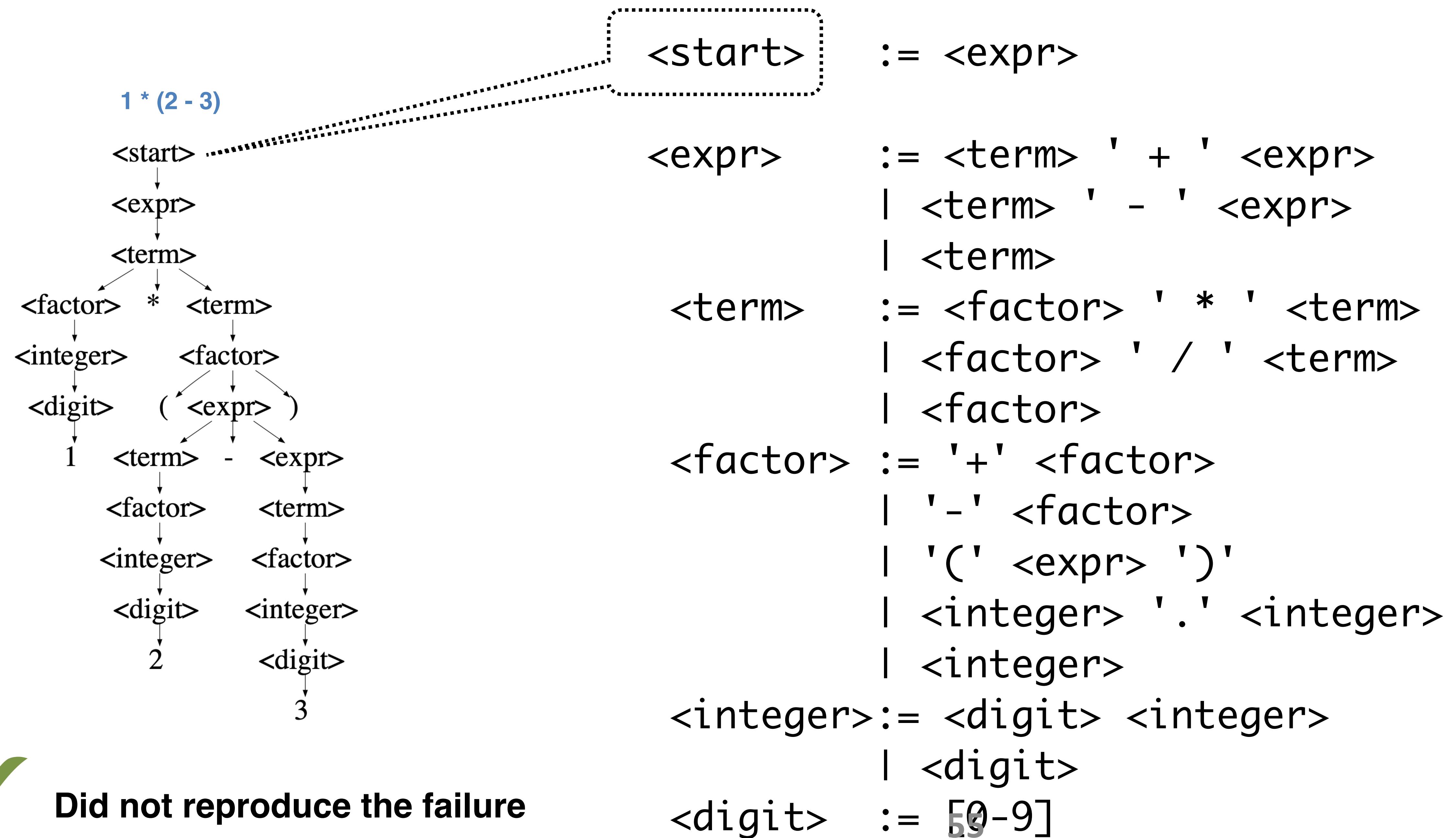


$\begin{array}{l} <\text{start}> := \text{<expr>} \\ \\ ((4)) \\ \\ \begin{array}{l} <\text{start}> \\ \downarrow \\ <\text{expr}> \\ \downarrow \\ <\text{term}> \\ \downarrow \\ <\text{factor}> \\ \swarrow \quad \searrow \\ (<\text{expr}>) \\ \downarrow \\ <\text{term}> \\ \downarrow \\ <\text{factor}> \\ \swarrow \quad \searrow \\ (<\text{expr}>) \\ \downarrow \\ <\text{term}> \\ \downarrow \\ <\text{factor}> \\ \downarrow \\ <\text{integer}> \\ \downarrow \\ <\text{digit}> \\ \downarrow \\ 4 \end{array} \end{array}$	$\begin{array}{l} <\text{expr}> := <\text{term}> ' + ' <\text{expr}> \\ <\text{term}> ' - ' <\text{expr}> \\ <\text{term}> \\ <\text{term}> := <\text{factor}> ' * ' <\text{term}> \\ <\text{factor}> ' / ' <\text{term}> \\ <\text{factor}> \\ <\text{factor}> := '+' <\text{factor}> \\ '-' <\text{factor}> \\ '(' <\text{expr}> ')' \\ <\text{integer}> '.' <\text{integer}> \\ <\text{integer}> \\ <\text{integer}> := <\text{digit}> <\text{integer}> \\ <\text{digit}> \\ <\text{digit}> := [0-9] \end{array}$
--	--

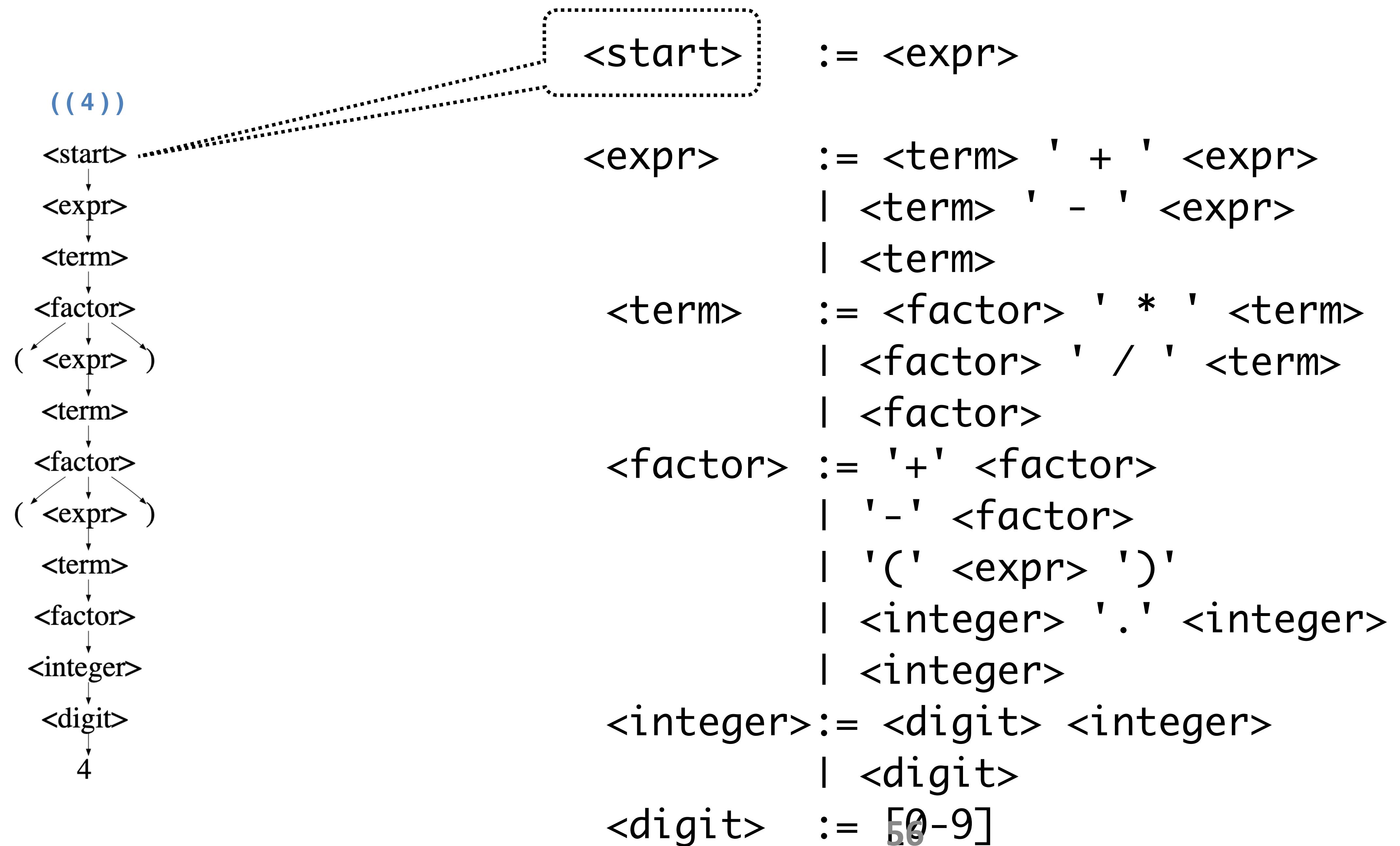
((4))

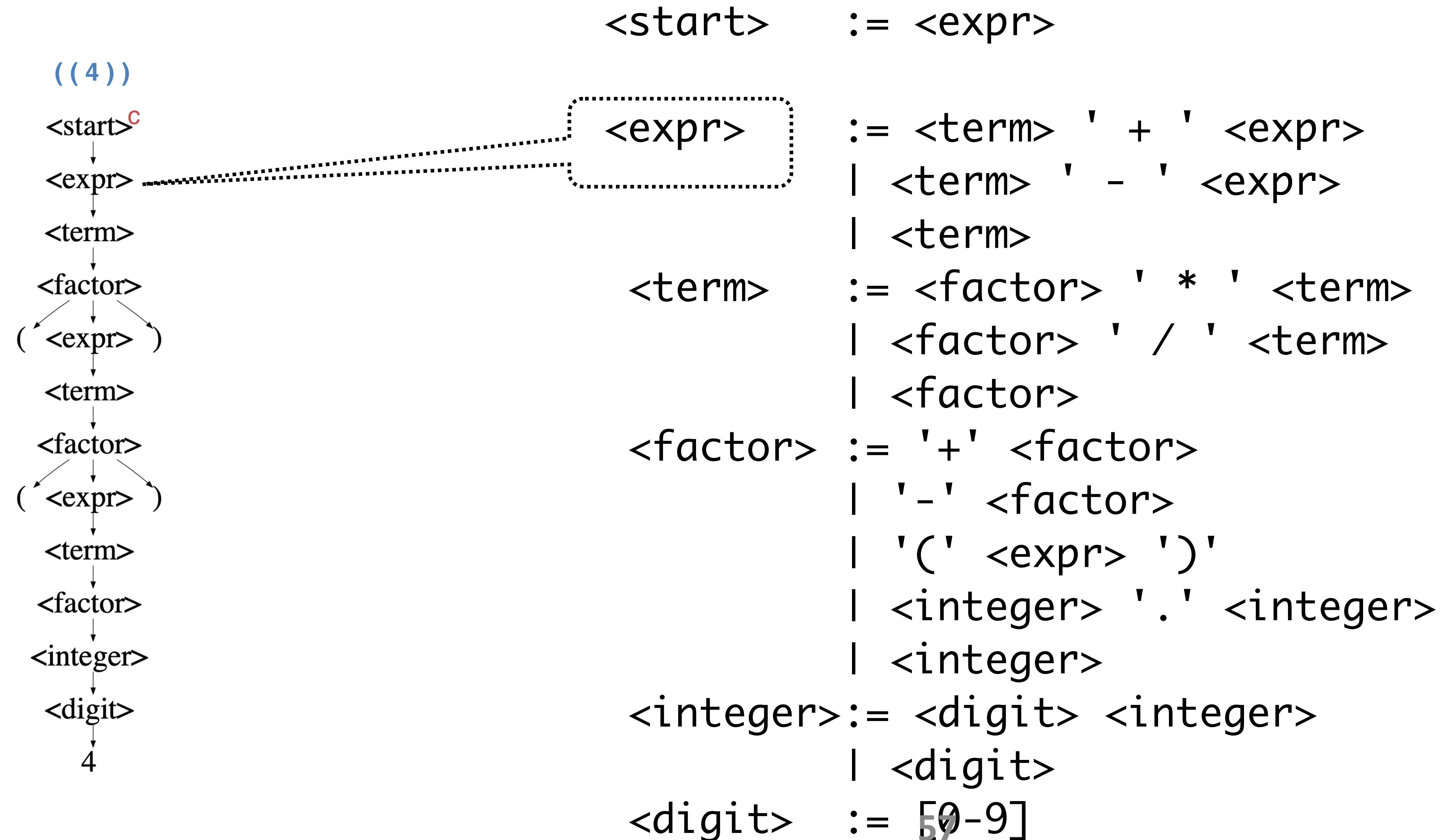


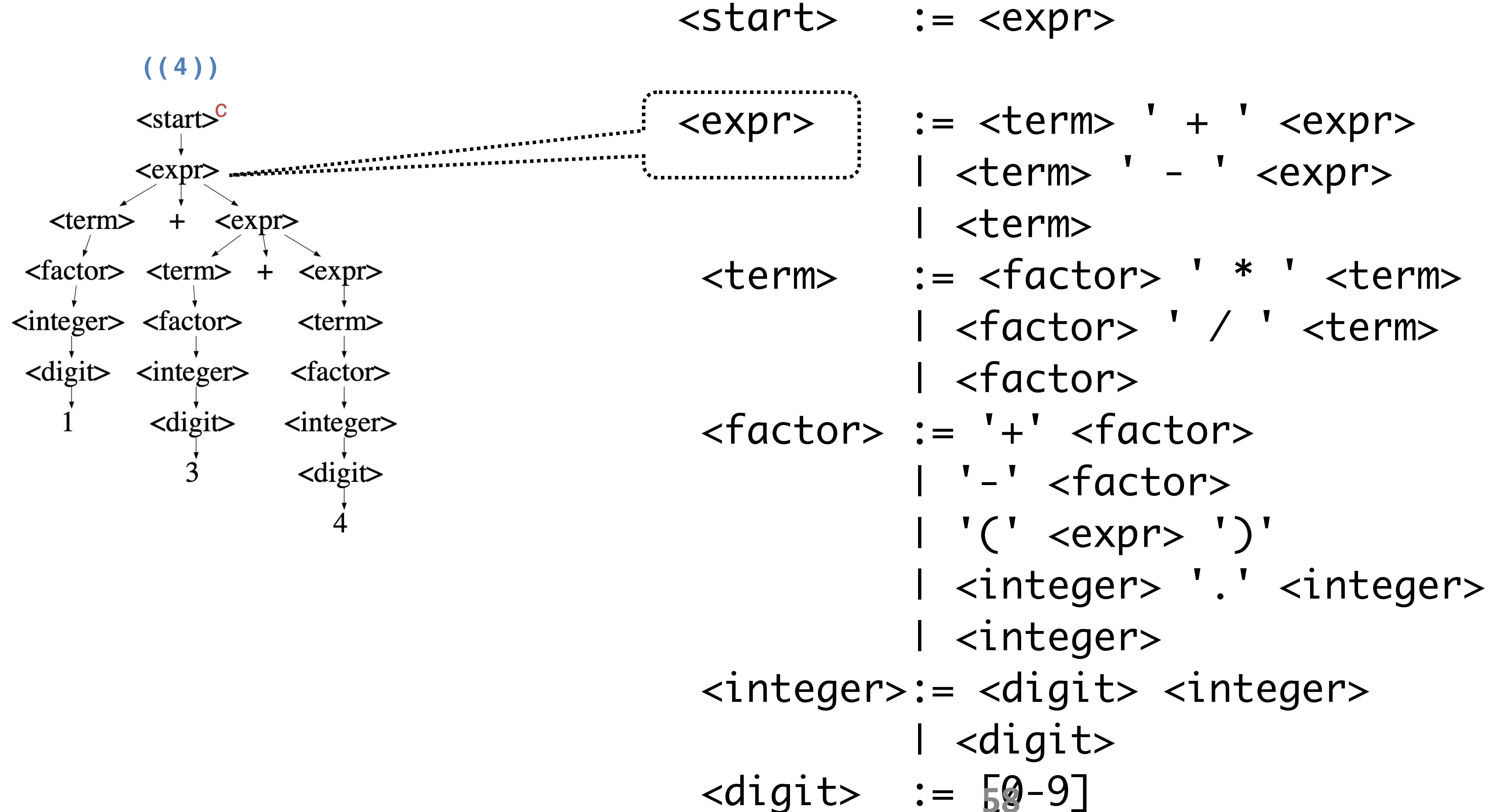


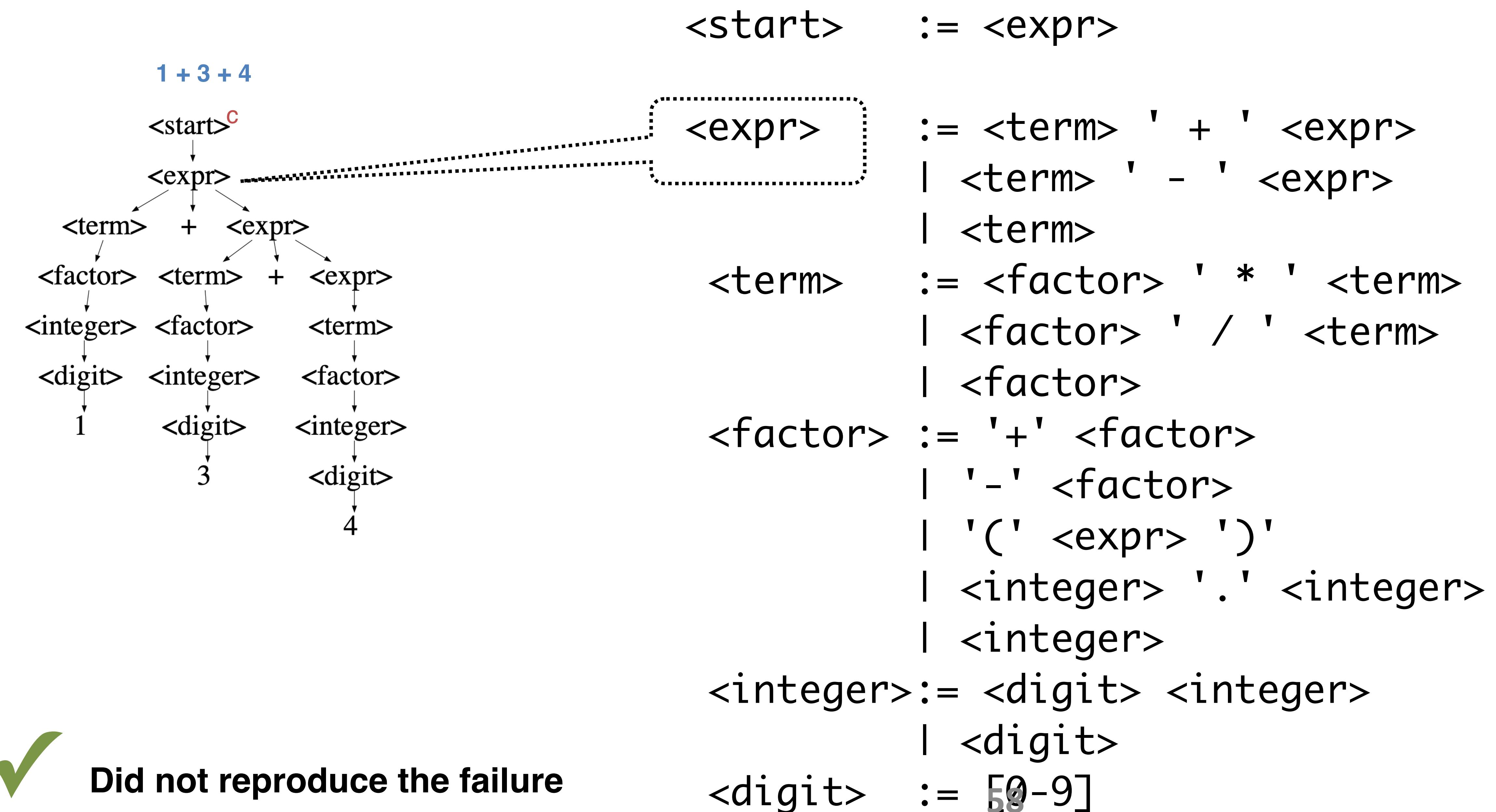


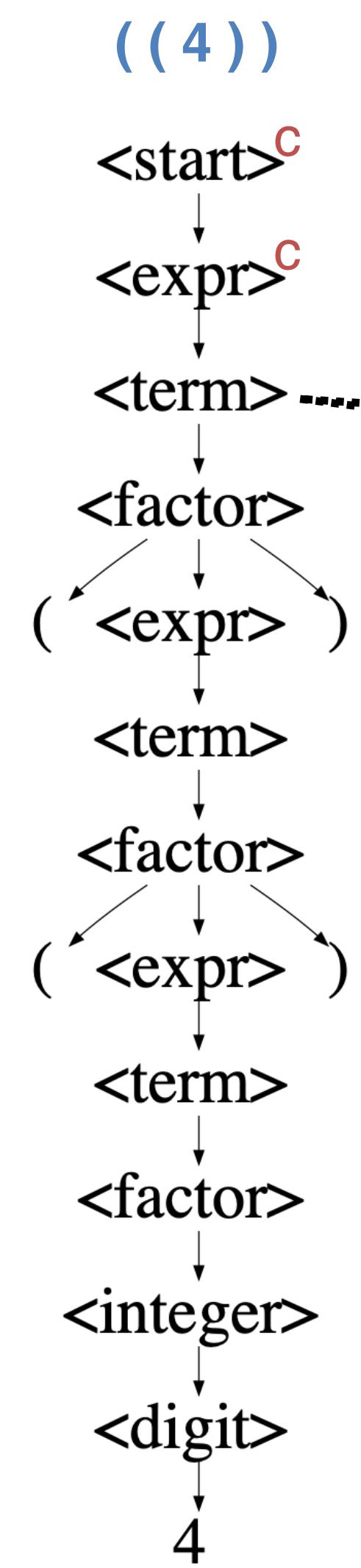
Did not reproduce the failure



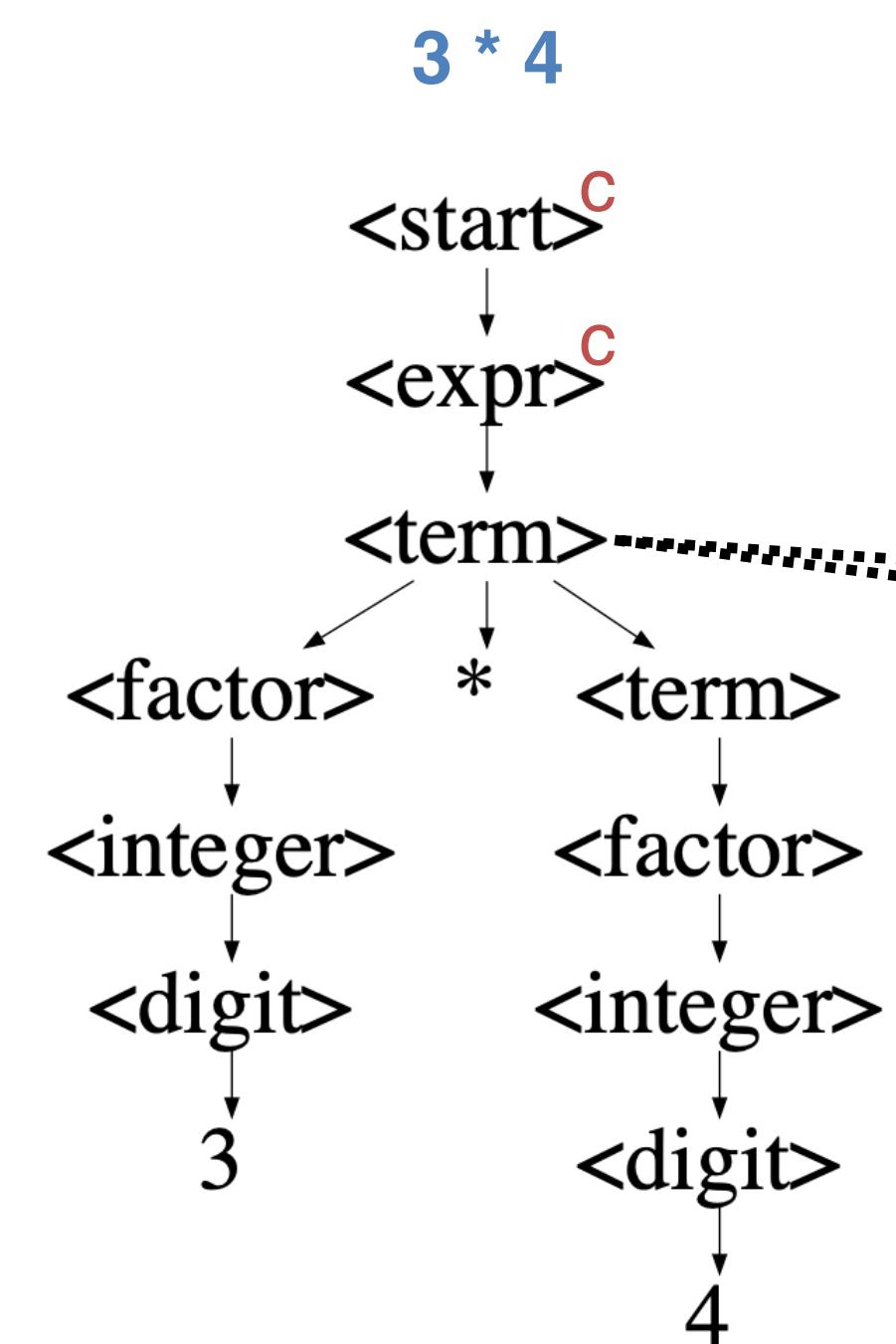




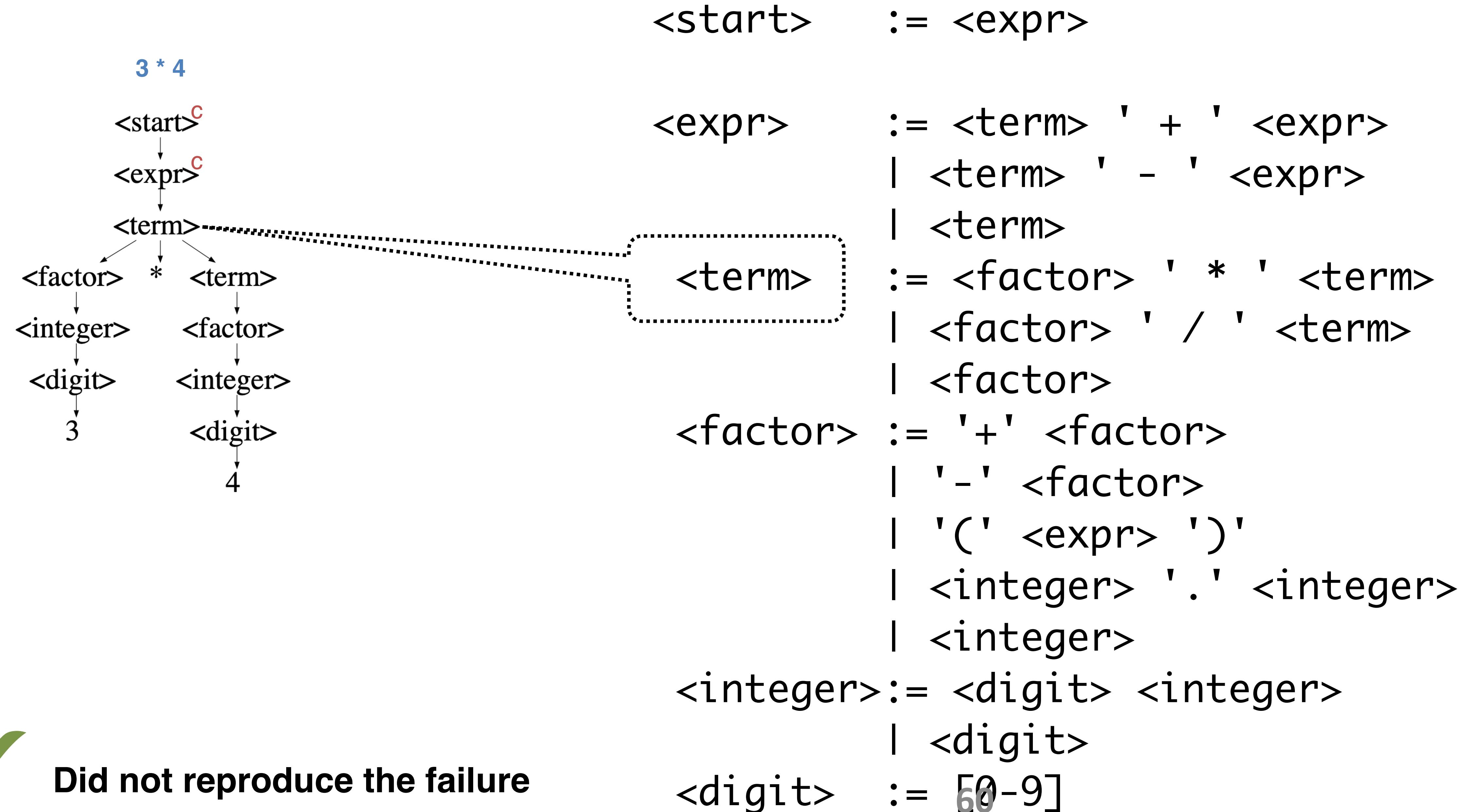




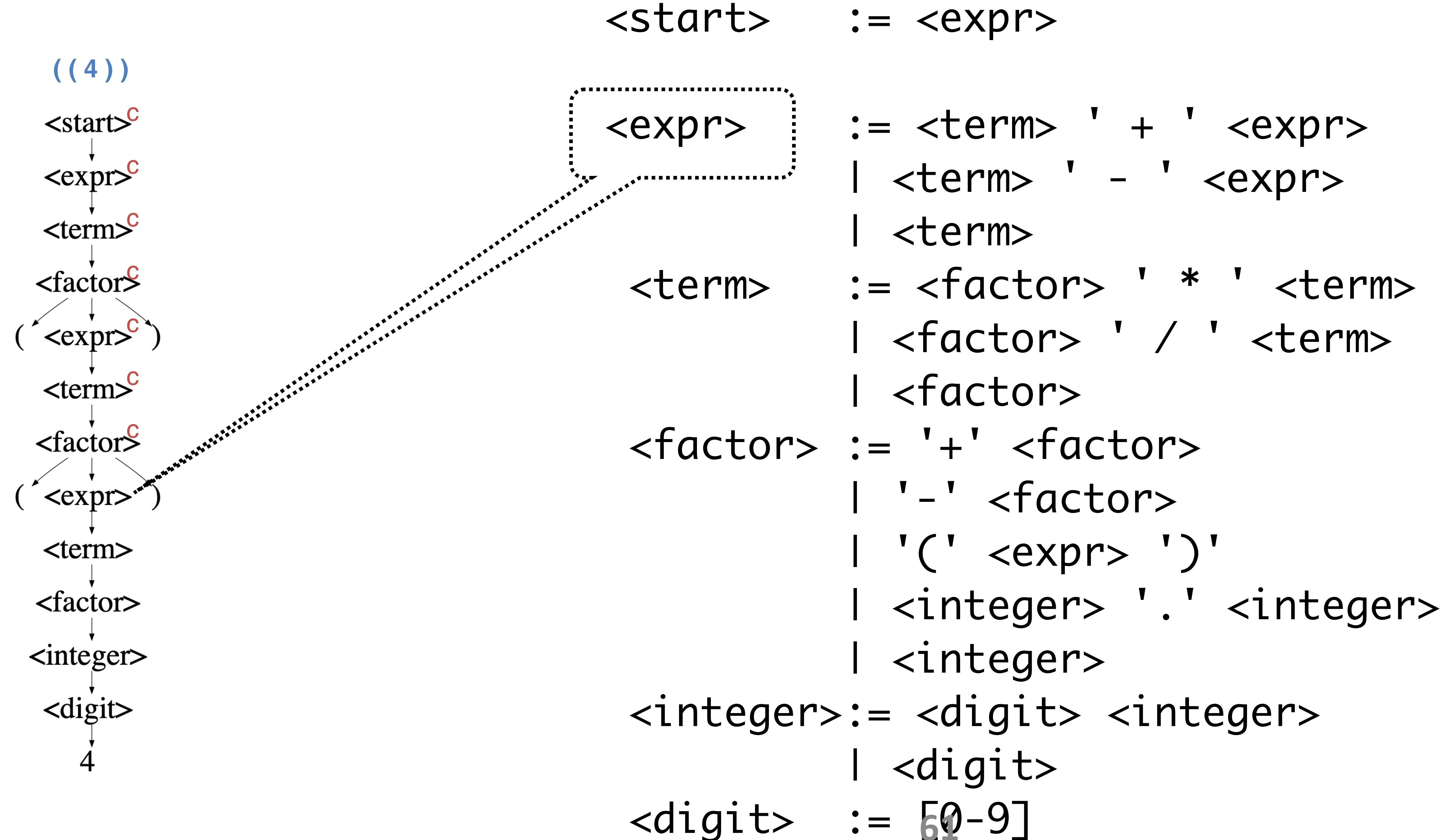
<code><start></code>	<code>:= <expr></code>
<code>((4))</code>	
<code><start>^C</code>	<code><expr> := <term> ' + ' <expr></code>
<code><expr>^C</code>	<code> <term> ' - ' <expr></code>
<code><term></code>	<code> <term></code>
<code><factor></code>	<code>:= <factor> ' * ' <term></code>
<code>(<expr>)</code>	<code> <factor> ' / ' <term></code>
<code><term></code>	<code> <factor></code>
<code><factor></code>	<code><factor> := '+ <factor></code>
<code>(<expr>)</code>	<code> '- <factor></code>
<code><term></code>	<code> '(<expr> ')'</code>
<code><factor></code>	<code> <integer> '.' <integer></code>
<code><integer></code>	<code> <integer></code>
<code><digit></code>	<code><integer> := <digit> <integer></code>
<code>4</code>	<code> <digit></code>
	<code><digit> := [0-9]</code>

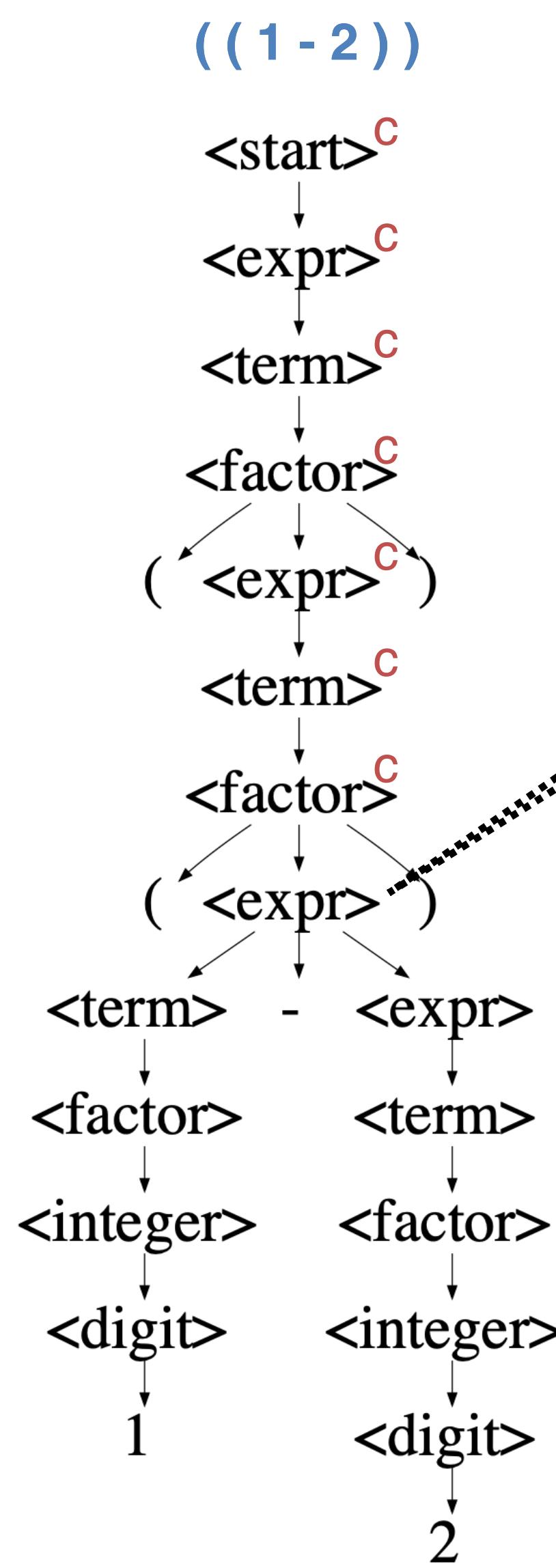


$\langle \text{start} \rangle := \langle \text{expr} \rangle$
 $\langle \text{expr} \rangle := \langle \text{term} \rangle \text{ ' + ' } \langle \text{expr} \rangle$
 $\mid \langle \text{term} \rangle \text{ ' - ' } \langle \text{expr} \rangle$
 $\mid \langle \text{term} \rangle$
 $\mid := \langle \text{factor} \rangle \text{ ' * ' } \langle \text{term} \rangle$
 $\mid \langle \text{factor} \rangle \text{ ' / ' } \langle \text{term} \rangle$
 $\mid \langle \text{factor} \rangle$
 $\mid := \text{ '+' } \langle \text{factor} \rangle$
 $\mid \text{ '-' } \langle \text{factor} \rangle$
 $\mid \text{ '(' } \langle \text{expr} \rangle \text{ ')' }$
 $\mid \langle \text{integer} \rangle \text{ '.' } \langle \text{integer} \rangle$
 $\mid \langle \text{integer} \rangle$
 $\langle \text{integer} \rangle := \langle \text{digit} \rangle \langle \text{integer} \rangle$
 $\mid \langle \text{digit} \rangle$
 $\langle \text{digit} \rangle := [0-9]$



Did not reproduce the failure

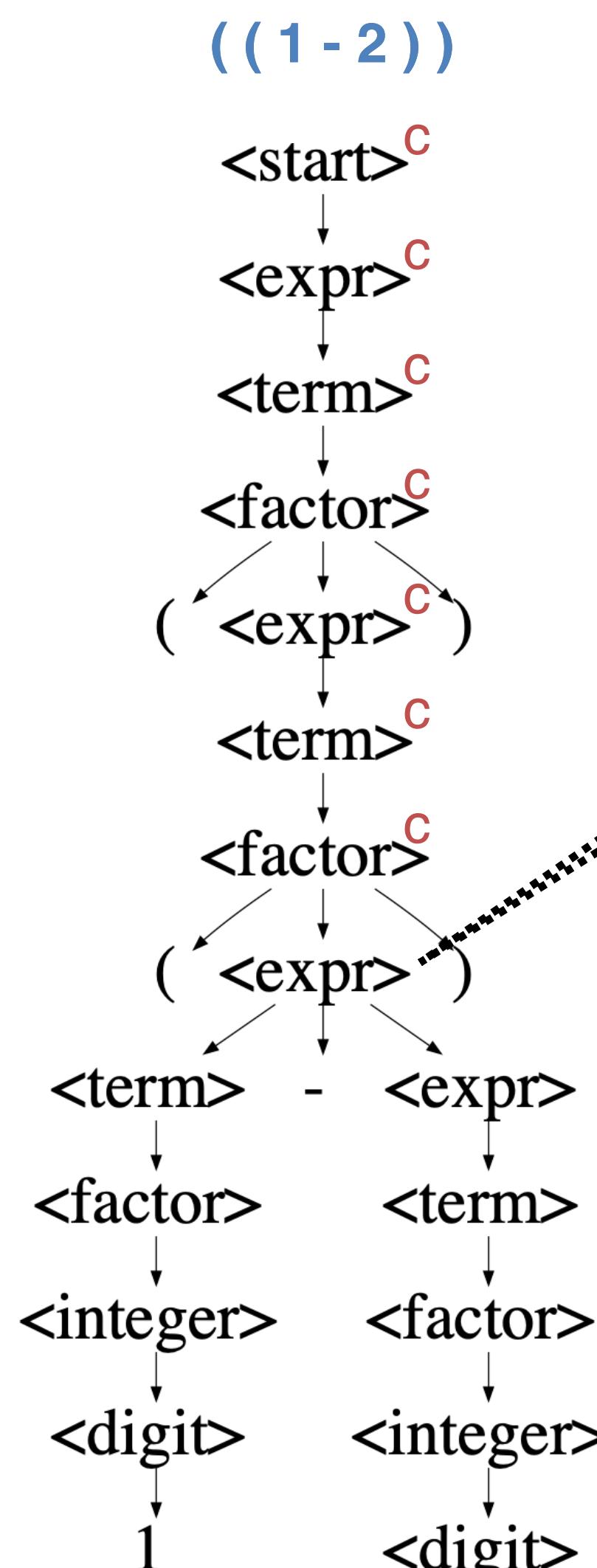




$<\text{start}>$	$::= <\text{expr}>$
	$<\text{expr}>$
	$::= <\text{term}> ' + ' <\text{expr}>$
	$ <\text{term}> ' - ' <\text{expr}>$
	$ <\text{term}>$
	$<\text{term}>$
	$::= <\text{factor}> ' * ' <\text{term}>$
	$ <\text{factor}> ' / ' <\text{term}>$
	$ <\text{factor}>$
	$<\text{factor}>$
	$::= '+' <\text{factor}>$
	$ '-' <\text{factor}>$
	$ '(' <\text{expr}> ')' $
	$ <\text{integer}> '.' <\text{integer}>$
	$ <\text{integer}>$
	$<\text{integer}>$
	$::= <\text{digit}> <\text{integer}>$
	$ <\text{digit}>$
	$<\text{digit}>$
	$::= [0-9]$

X

reproduced the failure



<start> := <expr>

<expr> := <term> ' + ' <expr>
| <term> ' - ' <expr>
| <term>

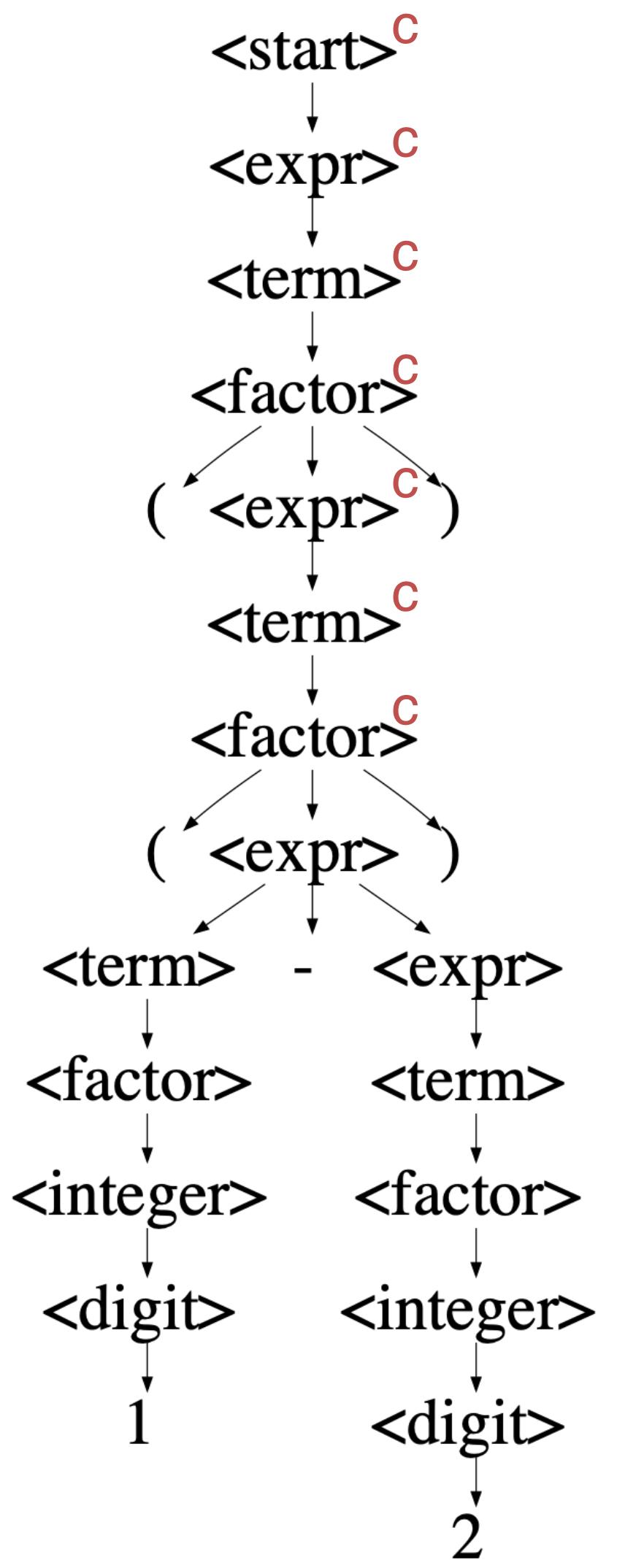
<term> := <factor> ' * ' <term>
| <factor> ' / ' <term>
| <factor>

<factor> := '+' <factor>
| '-' <factor>
| '(' <expr> ')'
| <integer> '.' <integer>
| <integer>

<integer> := <digit> <integer>
| <digit>

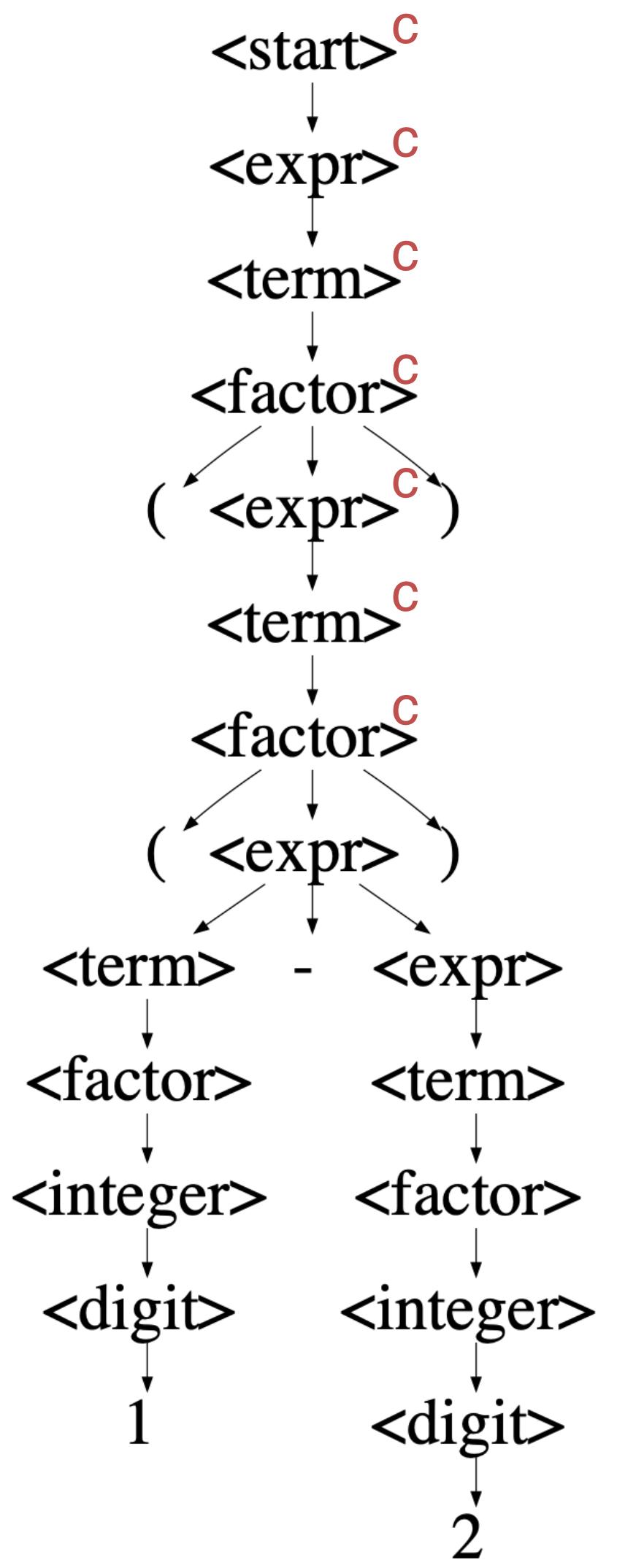
<digit> := [0-9]

((1 - 2))

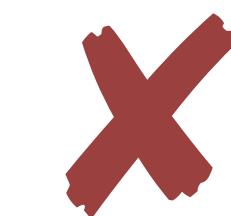


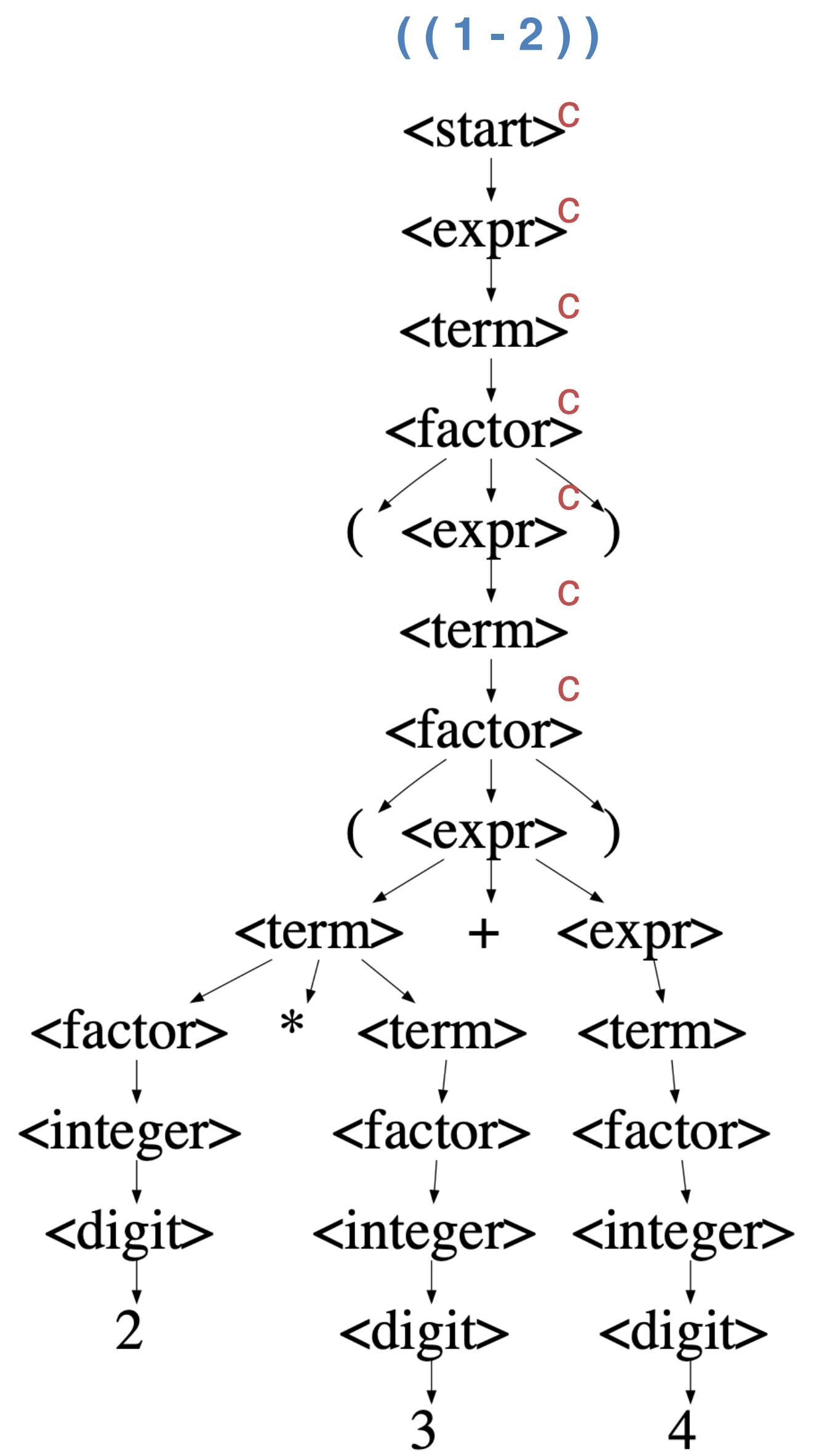
((1 - 2))

((1 - 2))

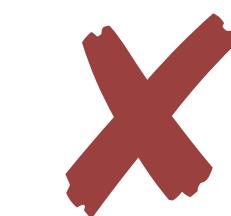


((1 - 2))

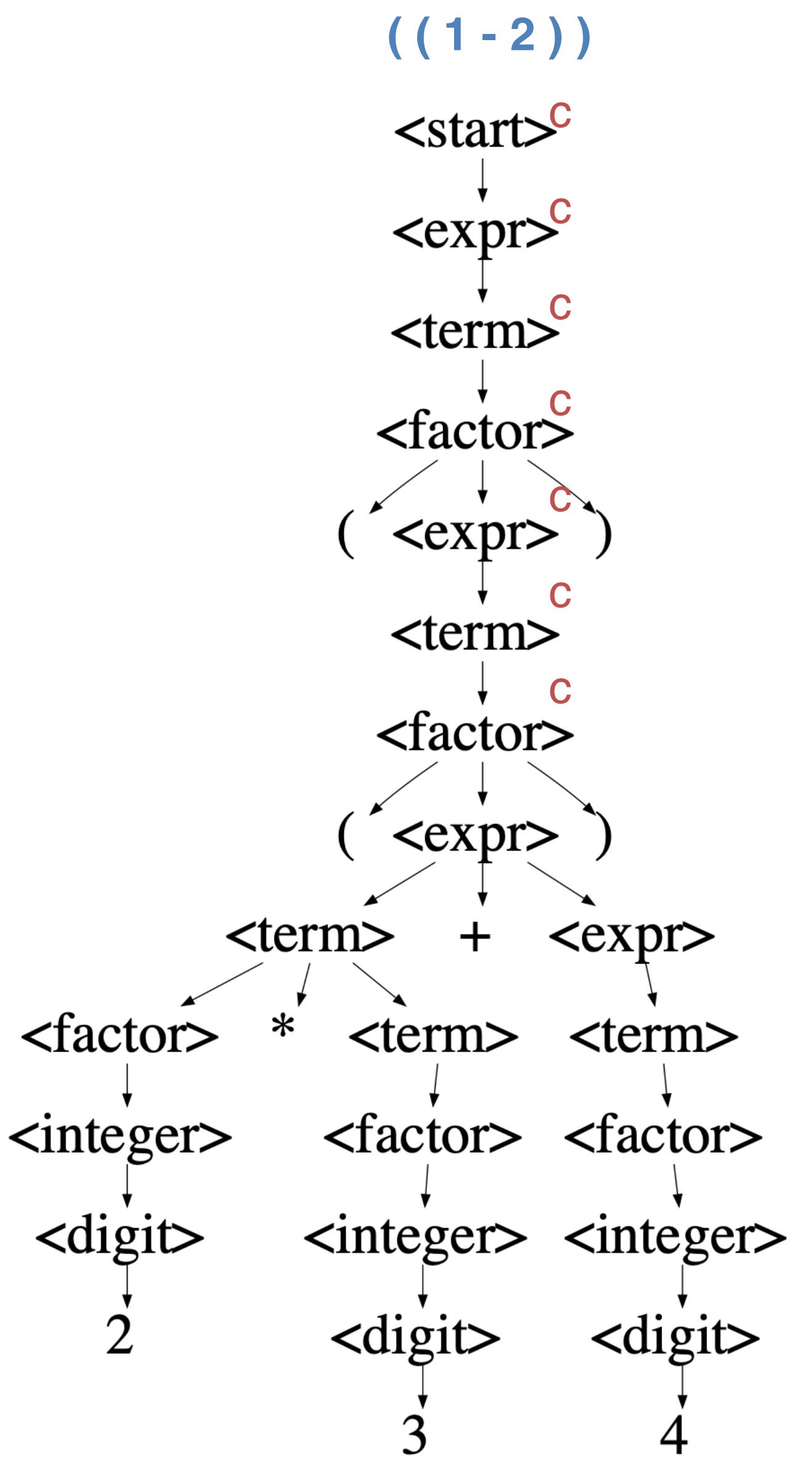




((1 - 2))



((2 * 3 + 4))



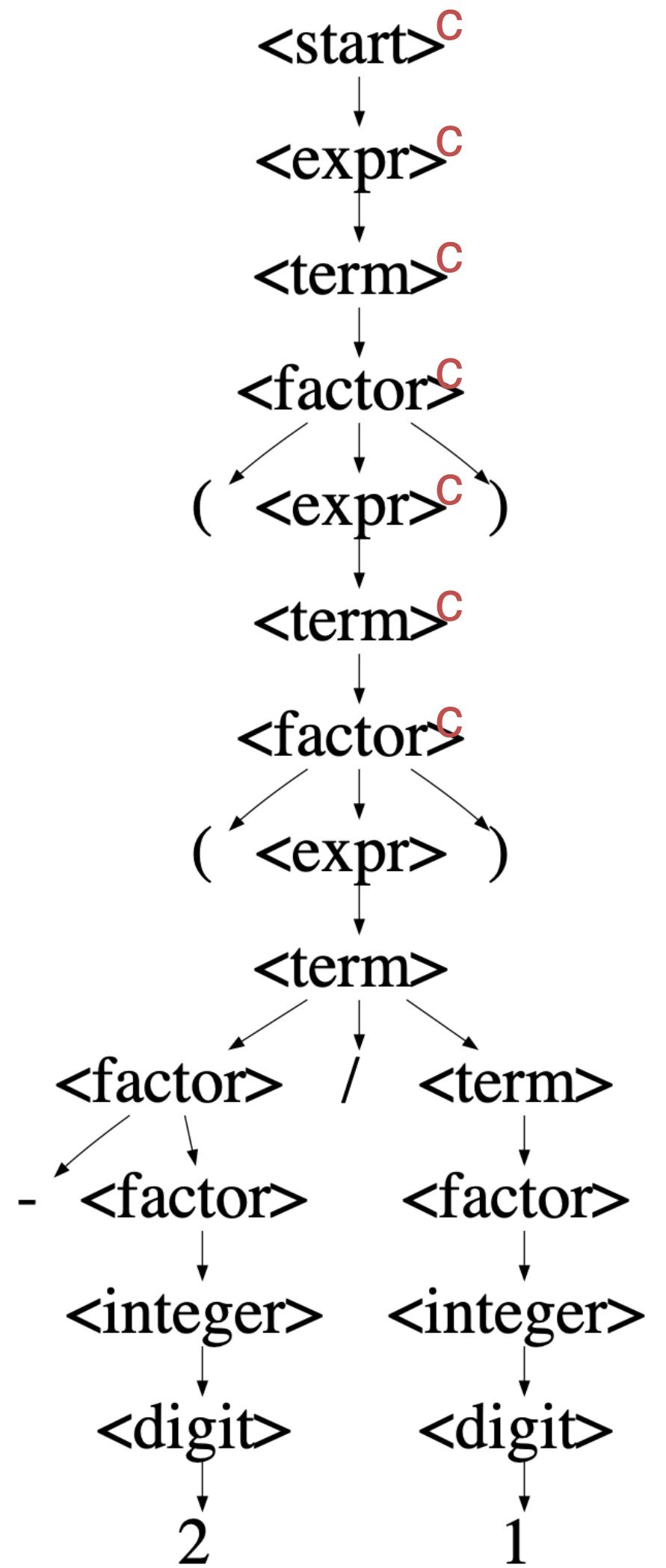
((1 - 2))

X

((2 * 3 + 4))

X

((1 - 2))



((1 - 2))

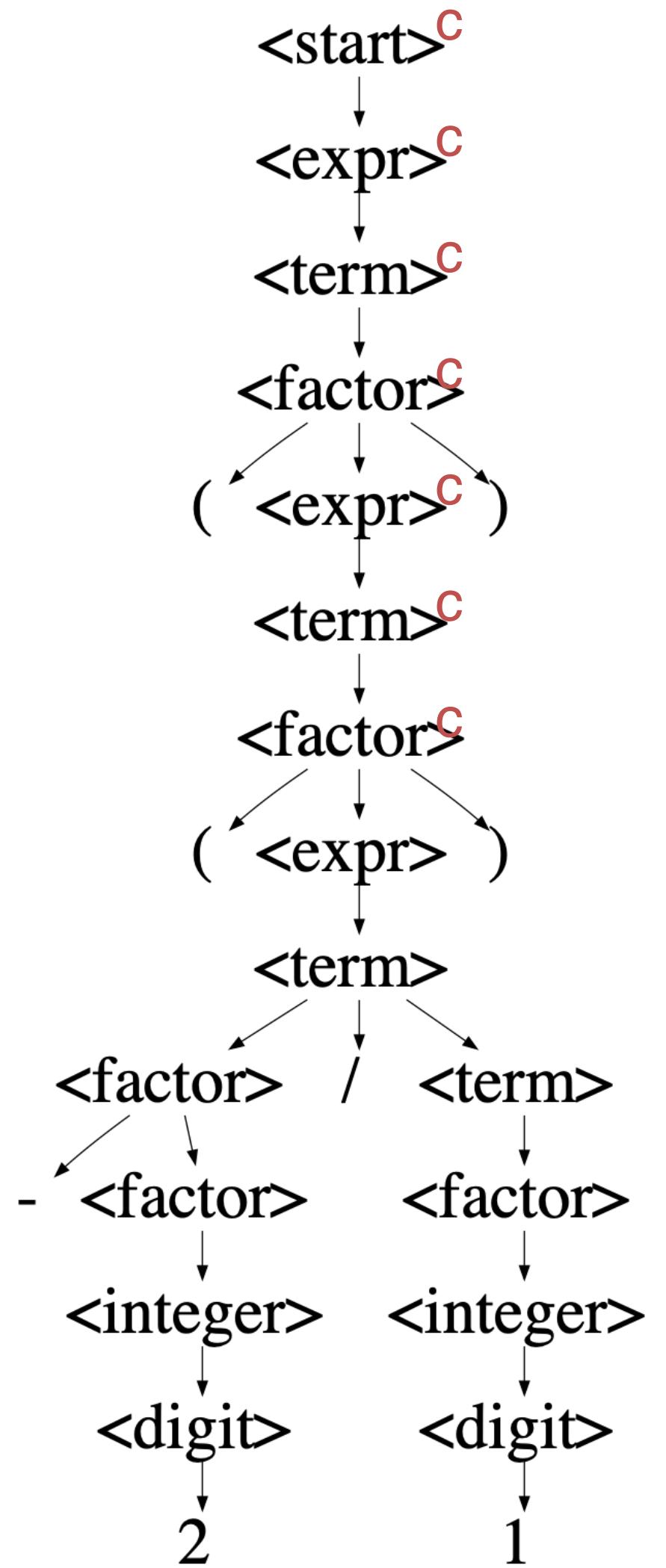
X

((2 * 3 + 4))

X

((- 2 / 1))

((1 - 2))



((1 - 2))

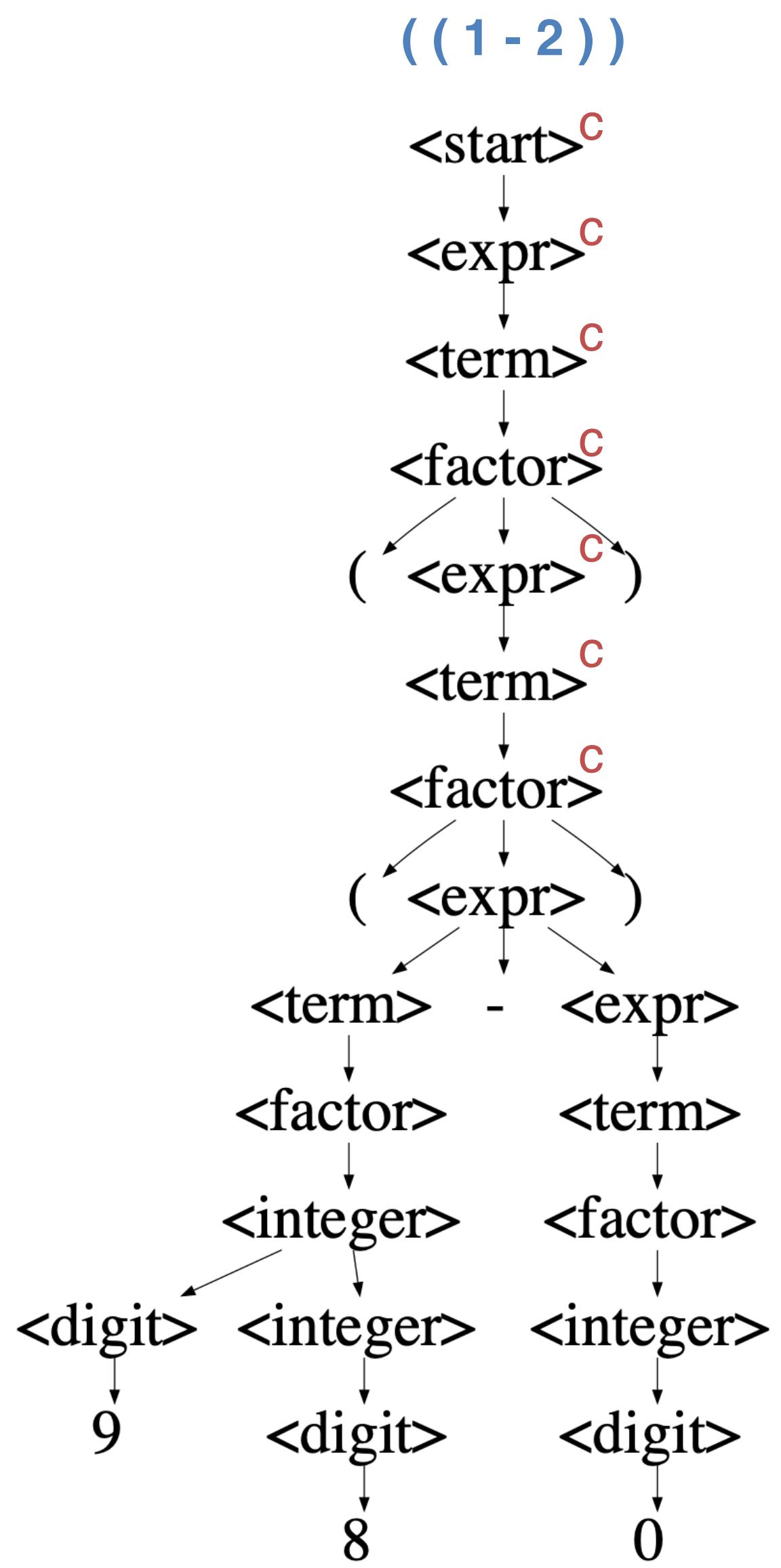
X

((2 * 3 + 4))

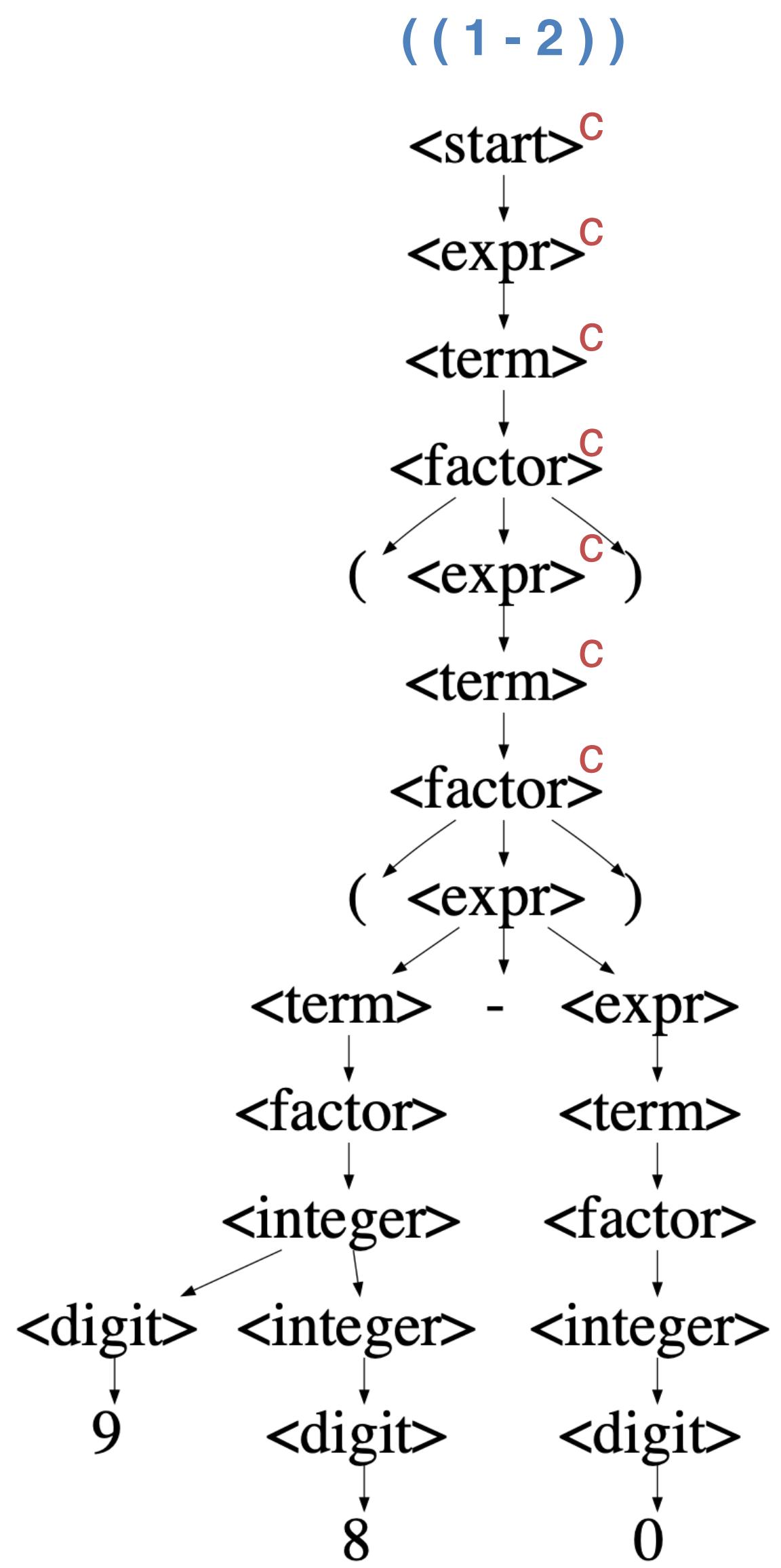
X

((- 2 / 1))

X

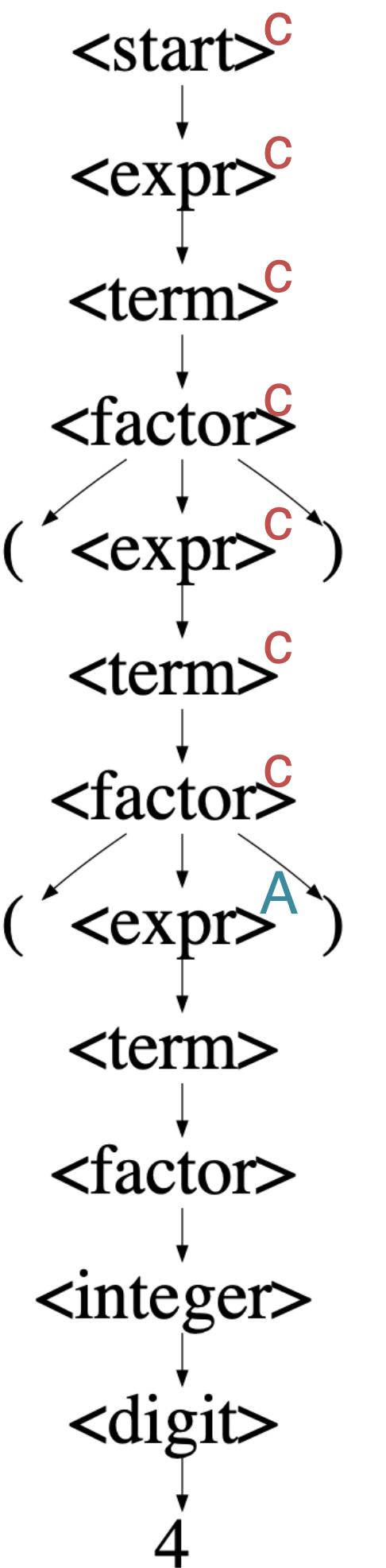


((1 - 2)) X
 ((2 * 3 + 4)) X
 ((- 2 / 1)) X
 ((98 - 0))

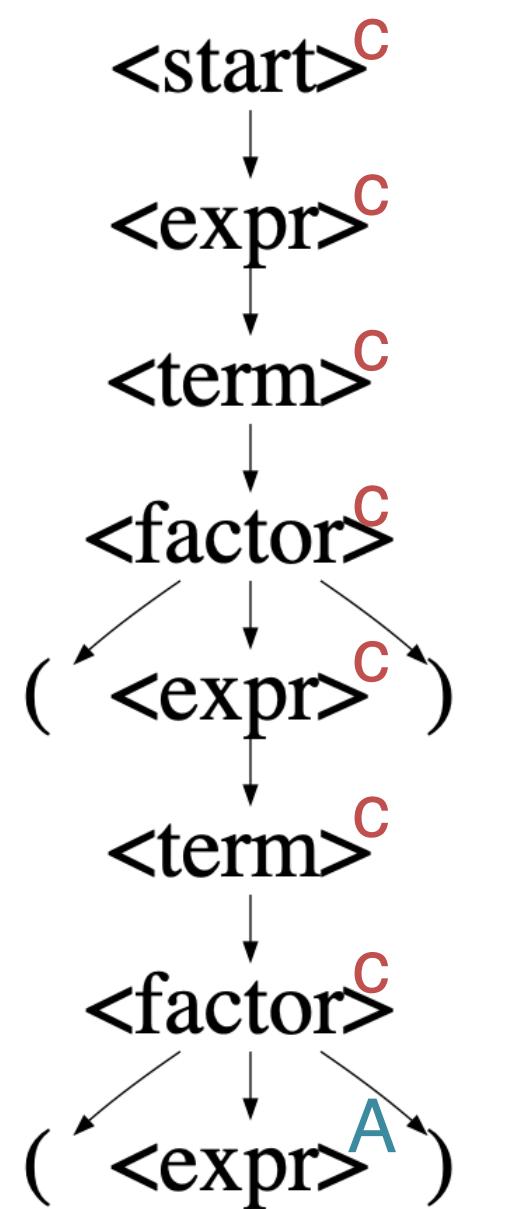


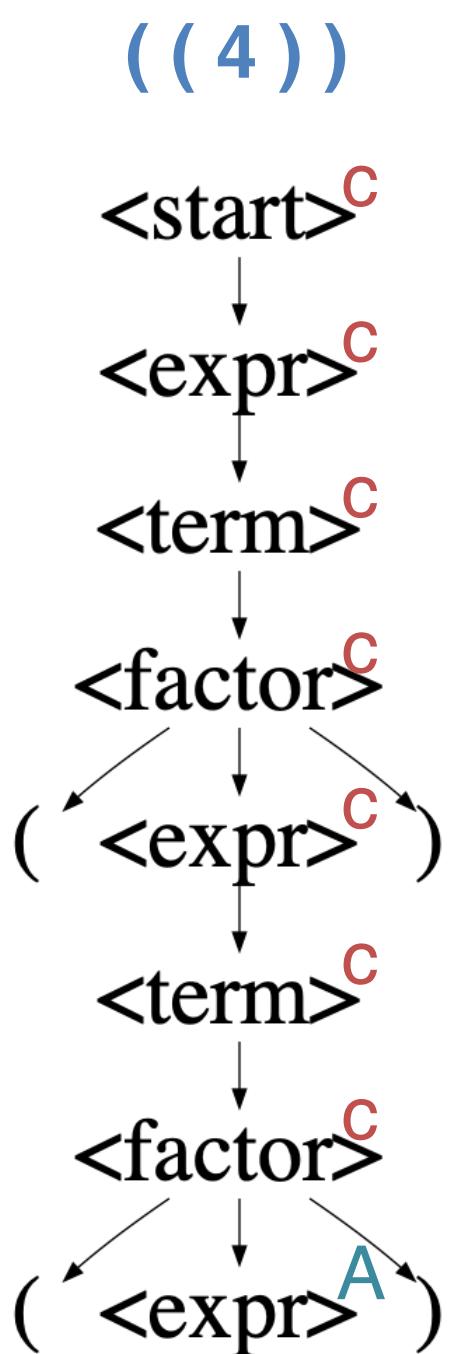
((1 - 2)) X
 ((2 * 3 + 4)) X
 ((- 2 / 1)) X
 ((98 - 0)) X

((4))



((4))





```

def check(parsed):
    if parsed.is_nested() and parsed.child.is_nested():
        raise Exception()
    return input
  
```

Minimized Input

((4))

Abstract Failure Inducing Input

((<expr>))

Specialized Grammar

```

<start F>   := <expr F>
<expr F>    := <term F> ' + ' <expr>
                | <term> ' + ' <expr F>
                | <term F> ' - ' <expr>
                | <term> ' - ' <expr F>
                | <term F>
<term F>     := <factor F> ' * ' <term>
                | <factor> ' * ' <term F>
                | <factor F> ' / ' <term>
                | <factor> ' / ' <term F>
                | <factor F>
<factor F>   := '+' <factor F>
                | '-' <factor F>
                | '(' <expr F> ')'
                | '(' <expr F1> ')'
<expr F1>    := <term F2>
<term F2>    := <factor F3>
<factor F3>  := '(' <expr> ')'

```

<factor F> is ((<expr>))

Specialized Grammar

```

<start F>      ::= <expr F>
<expr F>       ::= <term F> ' + ' <expr>
                  | <term> ' + ' <expr F>
                  | <term F> ' - ' <expr>
                  | <term> ' - ' <expr F>
                  | <term F>
<term F>        ::= <factor F> ' * ' <term>
                  | <factor> ' * ' <term F>
                  | <factor F> ' / ' <term>
                  | <factor> ' / ' <term F>
                  | <factor F>
<factor F>      ::= '+' <factor F>
                  | '-' <factor F>
                  | '(' <expr F> ')'
                  | '(' <expr F1> ')'
<expr F1>        ::= <term F2>
<term F2>        ::= <factor F3>
<factor F3>      ::= '(' <expr> ')' .778 - (((1) - 3
349 + (((1) - 3 * 3 + (-22
8 + ((8)) + --1 + / 1 - 39 + (1) - 456 +
74 + 3 + ((178 - 88 / (3393-1) * 1002 / 3 + 1+ 3439))
24343433 +23343 - ((74 + 334 + ((178 - 88 / (3393-1) * 1002 / 3 + 1+ 3439))

```

<factor F> is ((<expr>))

Input Algebras

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** localhost:8888/notebooks/x4_0_InputAlgebras.ipynb
- Header:** jupyter x4_0_InputAlgebras Last Checkpoint: 11 hours ago (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Navigate, Widgets, Help, Trusted, Python 3 (ipykernel)
- Contents Sidebar:** A tree view of the notebook's structure:
 - 1 Input Algebras
 - 1.1 Imports
 - 1.2 Expression Grammar
 - 1.3 And Grammars
 - 1.3.1 Combining tokens
 - 1.4 Combining rules
 - 1.4.1 Combining rulesets
 - 1.4.2 definition conjunction
 - 1.4.3 grammar conjunction
 - 1.5 Producing inputs with at least one of the
 - 1.5.1 Nonterminals
 - 1.6 rules
 - 1.7 rulesets
 - 1.7.1 definition disjunction
 - 1.7.2 grammar disjunction
 - 2 A grammar with no fault inducing fragment
 - 2.1 Unreachable Grammar
 - 2.2 Negated pattern grammar.
 - 2.3 Negation of full evocative expressions
 - 3 Done
- Main Content Area:** The '1 Input Algebras' section is expanded, showing:

1 Input Algebras

More complex and fully worked out examples including abstraction of real world bugs from Clojure, Rhino, Lua and a few Unix utilities [here](#).

1.1 Imports

```
In [ ]: import src.utils as utils
```

```
In [ ]: import ipynb.fs.full.x0_4_HierarchicalReducer as hdd
```

```
In [ ]: import ipynb.fs.full.x0_2_GrammarFuzzer as fuzzer
```

```
In [ ]: import ipynb.fs.full.x0_3_Parser as parser
```

```
In [ ]: from ipynb.fs.full.x0_4_HierarchicalReducer import PRes
```

```
In [ ]: import ipynb.fs.full.x3_0_AbstractingInputs as abstractinginputs
```

```
In [ ]: import sympy
```

```
In [ ]: import random
```

```
In [ ]: random.seed(0)
```

```
In [ ]: import itertools as I
```
- ### 1.2 Expression Grammar

```
In [ ]: import sympy
```

```
 $\langle json \rangle ::= \langle elt \rangle$ 
 $\langle elt \rangle ::= \langle object \rangle \mid \langle array \rangle \mid \langle string \rangle \mid \langle number \rangle$ 
\mid 'true' \mid 'false' \mid 'null'
 $\langle object \rangle ::= \{ \langle items \rangle \} \mid \{ \}$ 
 $\langle items \rangle ::= \langle item \rangle \mid \langle item \rangle , \langle items \rangle$ 
 $\langle item \rangle ::= \langle string \rangle : \langle elt \rangle$ 
 $\langle array \rangle ::= [ \langle elts \rangle ] \mid []$ 
 $\langle elts \rangle ::= \langle elt \rangle \mid \langle elt \rangle , \langle elts \rangle$ 
 $\langle string \rangle ::= " \langle chars \rangle "$ 
 $\langle chars \rangle ::= \langle char \rangle \langle chars \rangle \mid \epsilon$ 
 $\langle char \rangle ::= [A-Za-z0-9]$ 
 $\langle number \rangle ::= \langle digits \rangle$ 
 $\langle digits \rangle ::= \langle digit \rangle \langle digits \rangle \mid \langle digit \rangle$ 
 $\langle digit \rangle ::= [0-9]$ 
```

```

⟨json⟩ ::= ⟨elt⟩
⟨elt⟩ ::= ⟨object⟩ | ⟨array⟩ | ⟨string⟩ | ⟨number⟩
| ‘true’ | ‘false’ | ‘null’
⟨object⟩ ::= ‘{’ ⟨items⟩ ‘}’ | ‘{}’
⟨items⟩ ::= ⟨item⟩ | ⟨item⟩ ‘,’ ⟨items⟩
⟨item⟩ ::= ⟨string⟩ ‘:’ ⟨elt⟩
⟨array⟩ ::= ‘[’ ⟨elts⟩ ‘]’ | ‘[]’
⟨elts⟩ ::= ⟨elt⟩ | ⟨elt⟩ ‘,’ ⟨elts⟩
⟨string⟩ ::= “” ⟨chars⟩ “”
⟨chars⟩ ::= ⟨char⟩ ⟨chars⟩ | ε
⟨char⟩ ::= [A-Za-z0-9]
⟨number⟩ ::= ⟨digits⟩
⟨digits⟩ ::= ⟨digit⟩ ⟨digits⟩ | ⟨digit⟩
⟨digit⟩ ::= [0-9]

```

```

def jsoncheck(json):
    if any_key_has_null_value(json):
        fail('key value must not be null')
process(json)

```

{"abc": null}



```

⟨json⟩ ::= ⟨elt⟩
⟨elt⟩ ::= ⟨object⟩ | ⟨array⟩ | ⟨string⟩ | ⟨number⟩
| ‘true’ | ‘false’ | ‘null’
⟨object⟩ ::= ‘{’ ⟨items⟩ ‘}’ | ‘{}’
⟨items⟩ ::= ⟨item⟩ | ⟨item⟩ ‘,’ ⟨items⟩
⟨item⟩ ::= ⟨string⟩ ‘:’ ⟨elt⟩
⟨array⟩ ::= ‘[’ ⟨elts⟩ ‘]’ | ‘[]’
⟨elts⟩ ::= ⟨elt⟩ | ⟨elt⟩ ‘,’ ⟨elts⟩
⟨string⟩ ::= “” ⟨chars⟩ “”
⟨chars⟩ ::= ⟨char⟩ ⟨chars⟩ | ε
⟨char⟩ ::= ‘[A-Za-z0-9]’
⟨number⟩ ::= ⟨digits⟩
⟨digits⟩ ::= ⟨digit⟩ ⟨digits⟩ | ⟨digit⟩
⟨digit⟩ ::= ‘[0-9]’

```

```

def jsoncheck(json):
    if any_key_has_null_value(json):
        fail('key value must not be null')
process(json)

```

<item N> is <string> : null

{"abc": null}



```

⟨json⟩ ::= ⟨elt⟩
⟨elt⟩ ::= ⟨object⟩ | ⟨array⟩ | ⟨string⟩ | ⟨number⟩
| ‘true’ | ‘false’ | ‘null’
⟨object⟩ ::= ‘{’ ⟨items⟩ ‘}’ | ‘{}’
⟨items⟩ ::= ⟨item⟩ | ⟨item⟩ ‘,’ ⟨items⟩
⟨item⟩ ::= ⟨string⟩ ‘:’ ⟨elt⟩
⟨array⟩ ::= ‘[’ ⟨elts⟩ ‘]’ | ‘[]’
⟨elts⟩ ::= ⟨elt⟩ | ⟨elt⟩ ‘,’ ⟨elts⟩
⟨string⟩ ::= “” ⟨chars⟩ “”
⟨chars⟩ ::= ⟨char⟩ ⟨chars⟩ | ε
⟨char⟩ ::= ‘[A-Za-z0-9]’
⟨number⟩ ::= ⟨digits⟩
⟨digits⟩ ::= ⟨digit⟩ ⟨digits⟩ | ⟨digit⟩
⟨digit⟩ ::= ‘[0-9]’

```

```

def jsoncheck(json):
    if any_key_has_null_value(json):
        fail('key value must not be null')
process(json)

```

{"abc": []}



```

⟨json⟩ ::= ⟨elt⟩
⟨elt⟩ ::= ⟨object⟩ | ⟨array⟩ | ⟨string⟩ | ⟨number⟩
| ‘true’ | ‘false’ | ‘null’
⟨object⟩ ::= ‘{’ ⟨items⟩ ‘}’ | ‘{}’
⟨items⟩ ::= ⟨item⟩ | ⟨item⟩ ‘,’ ⟨items⟩
⟨item⟩ ::= ⟨string⟩ ‘:’ ⟨elt⟩
⟨array⟩ ::= ‘[’ ⟨elts⟩ ‘]’ | ‘[]’
⟨elts⟩ ::= ⟨elt⟩ | ⟨elt⟩ ‘,’ ⟨elts⟩
⟨string⟩ ::= “” ⟨chars⟩ “”
⟨chars⟩ ::= ⟨char⟩ ⟨chars⟩ | ε
⟨char⟩ ::= ‘[A-Za-z0-9]’
⟨number⟩ ::= ⟨digits⟩
⟨digits⟩ ::= ⟨digit⟩ ⟨digits⟩ | ⟨digit⟩
⟨digit⟩ ::= ‘[0-9]’

```

```

def jsoncheck(json):
    if any_key_has_null_value(json):
        fail('key value must not be null')
process(json)

```

no ⟨item N⟩ is ⟨string⟩ : null

{"abc": []}



```

⟨json⟩ ::= ⟨elt⟩
⟨elt⟩ ::= ⟨object⟩ | ⟨array⟩ | ⟨string⟩ | ⟨number⟩
| ‘true’ | ‘false’ | ‘null’
⟨object⟩ ::= ‘{’ ⟨items⟩ ‘}’ | ‘{}’
⟨items⟩ ::= ⟨item⟩ | ⟨item⟩ ‘,’ ⟨items⟩
⟨item⟩ ::= ⟨string⟩ ‘:’ ⟨elt⟩
⟨array⟩ ::= ‘[’ ⟨elts⟩ ‘]’ | ‘[]’
⟨elts⟩ ::= ⟨elt⟩ | ⟨elt⟩ ‘,’ ⟨elts⟩
⟨string⟩ ::= “” ⟨chars⟩ “”
⟨chars⟩ ::= ⟨char⟩ ⟨chars⟩ | ε
⟨char⟩ ::= ‘[A-Za-z0-9]’
⟨number⟩ ::= ⟨digits⟩
⟨digits⟩ ::= ⟨digit⟩ ⟨digits⟩ | ⟨digit⟩
⟨digit⟩ ::= ‘[0-9]’

```

```

def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
process(json)

```

no ⟨item E⟩ is “” : ⟨elt⟩

{"abc": 124}



```

⟨json⟩ ::= ⟨elt⟩
⟨elt⟩ ::= ⟨object⟩ | ⟨array⟩ | ⟨string⟩ | ⟨number⟩
| ‘true’ | ‘false’ | ‘null’
⟨object⟩ ::= ‘{’ ⟨items⟩ ‘}’ | ‘{}’
⟨items⟩ ::= ⟨item⟩ | ⟨item⟩ ‘,’ ⟨items⟩
⟨item⟩ ::= ⟨string⟩ ‘:’ ⟨elt⟩
⟨array⟩ ::= ‘[’ ⟨elts⟩ ‘]’ | ‘[]’
⟨elts⟩ ::= ⟨elt⟩ | ⟨elt⟩ ‘,’ ⟨elts⟩
⟨string⟩ ::= “” ⟨chars⟩ “”
⟨chars⟩ ::= ⟨char⟩ ⟨chars⟩ | ε
⟨char⟩ ::= ‘[A-Za-z0-9]’
⟨number⟩ ::= ⟨digits⟩
⟨digits⟩ ::= ⟨digit⟩ ⟨digits⟩ | ⟨digit⟩
⟨digit⟩ ::= ‘[0-9]’

```

```

def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
process(json)

```

<item E> is "": <elt>

{"": 124}



```

⟨json⟩ ::= ⟨elt⟩
⟨elt⟩ ::= ⟨object⟩ | ⟨array⟩ | ⟨string⟩ | ⟨number⟩
| ‘true’ | ‘false’ | ‘null’
⟨object⟩ ::= ‘{’ ⟨items⟩ ‘}’ | ‘{}’
⟨items⟩ ::= ⟨item⟩ | ⟨item⟩ ‘,’ ⟨items⟩
⟨item⟩ ::= ⟨string⟩ ‘:’ ⟨elt⟩
⟨array⟩ ::= ‘[’ ⟨elts⟩ ‘]’ | ‘[]’
⟨elts⟩ ::= ⟨elt⟩ | ⟨elt⟩ ‘,’ ⟨elts⟩
⟨string⟩ ::= “” ⟨chars⟩ “”
⟨chars⟩ ::= ⟨char⟩ ⟨chars⟩ | ε
⟨char⟩ ::= ‘[A-Za-z0-9]’
⟨number⟩ ::= ⟨digits⟩
⟨digits⟩ ::= ⟨digit⟩ ⟨digits⟩ | ⟨digit⟩
⟨digit⟩ ::= ‘[0-9]’

```

```

def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
    if any_key_has_null_value(json):
        fail('key value must not be null')
process(json)

```

```

⟨json⟩ ::= ⟨elt⟩
⟨elt⟩ ::= ⟨object⟩ | ⟨array⟩ | ⟨string⟩ | ⟨number⟩
| ‘true’ | ‘false’ | ‘null’
⟨object⟩ ::= ‘{’ ⟨items⟩ ‘}’ | ‘{}’
⟨items⟩ ::= ⟨item⟩ | ⟨item⟩ ‘,’ ⟨items⟩
⟨item⟩ ::= ⟨string⟩ ‘:’ ⟨elt⟩
⟨array⟩ ::= ‘[’ ⟨elts⟩ ‘]’ | ‘[]’
⟨elts⟩ ::= ⟨elt⟩ | ⟨elt⟩ ‘,’ ⟨elts⟩
⟨string⟩ ::= “” ⟨chars⟩ “”
⟨chars⟩ ::= ⟨char⟩ ⟨chars⟩ | ε
⟨char⟩ ::= [A-Za-z0-9]
⟨number⟩ ::= ⟨digits⟩
⟨digits⟩ ::= ⟨digit⟩ ⟨digits⟩ | ⟨digit⟩
⟨digit⟩ ::= [0-9]

```

```

def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
    if any_key_has_null_value(json):
        fail('key value must not be null')
    process(json)

```

$\langle item\ E \rangle \text{ is } "": \langle elt \rangle$
&
no $\langle item\ N \rangle \text{ is } \langle string \rangle : \text{null}$ {": 124}



$\langle json_N \rangle ::= \langle elt_N \rangle$
 $\langle elt_N \rangle ::= \langle object_N \rangle \mid \langle array_N \rangle$
 $\langle array_N \rangle ::= '[' \langle elts_N \rangle ']'$
 $\langle object_N \rangle ::= '{' \langle items_N \rangle '}'$
 $\langle elts_N \rangle ::= \langle elt_N \rangle ',' \langle elts \rangle \mid \langle elt \rangle ',' \langle elts_N \rangle \mid \langle elt_N \rangle$
 $\langle items_N \rangle ::= \langle item_N \rangle ',' \langle items \rangle \mid \langle item \rangle ',' \langle items_N \rangle$
 $\mid \langle item \rangle N$
 $\langle item_N \rangle ::= \langle string \rangle ':' \langle elt_N \rangle \mid \langle string \rangle ':' \langle elt_{N_1} \rangle$
 $\langle elt_{N_1} \rangle ::= 'null'$

Start Symbol

```

def jsoncheck(json):
    if any_key_has_null_value(json):
        fail('key value must not be null')
    process(json)

```

$\langle item \rangle N$ is $\langle string \rangle : null$

{"abc": null}



$$\begin{aligned}
 \langle \text{json}_{\bar{N}} \rangle &::= \langle \text{elt}_{\bar{N}} \rangle \\
 \langle \text{elt}_{\bar{N}} \rangle &::= \text{'false'} \mid \text{'null'} \mid \text{'true'} \\
 &\mid \langle \text{number} \rangle \mid \langle \text{string} \rangle \mid \langle \text{array}_{\bar{N}} \rangle \mid \langle \text{object}_{\bar{N}} \rangle \\
 \langle \text{array}_{\bar{N}} \rangle &::= \text{[]}' \mid [\langle \text{elts}_{\bar{N}} \rangle] \\
 \langle \text{object}_{\bar{N}} \rangle &::= \{\} \mid \{ \langle \text{items}_{\bar{N}} \rangle \} \\
 \langle \text{elts}_{\bar{N}} \rangle &::= \langle \text{elt}_{\bar{N}} \rangle \mid \langle \text{elt}_{\bar{N}} \rangle , \langle \text{elts}_{\bar{N}} \rangle \\
 \langle \text{items}_{\bar{N}} \rangle &::= \langle \text{item}_{\bar{N}} \rangle \mid \langle \text{item}_{\bar{N}} \rangle , \langle \text{items}_{\bar{N}} \rangle \\
 \langle \text{item}_{\bar{N}} \rangle &::= \langle \text{string} \rangle : \langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle \\
 \langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle &::= \text{'false'} \mid \text{'true'} \\
 &\mid \langle \text{number} \rangle \mid \langle \text{string} \rangle \mid \langle \text{array}_{\bar{N}} \rangle \mid \langle \text{object}_{\bar{N}} \rangle
 \end{aligned}$$

Start Symbol

```

def jsoncheck(json):
    if any_key_has_null_value(json):
        fail('key value must not be null')
process(json)

```

$$\begin{aligned}
 \langle \text{json}_N \rangle &::= \langle \text{elt}_N \rangle \\
 \langle \text{elt}_N \rangle &::= \langle \text{object}_N \rangle \mid \langle \text{array}_N \rangle \\
 \langle \text{array}_N \rangle &::= [\langle \text{elts}_N \rangle] \\
 \langle \text{object}_N \rangle &::= \{ \langle \text{items}_N \rangle \} \\
 \langle \text{elts}_N \rangle &::= \langle \text{elt}_N \rangle , \langle \text{elts} \rangle \mid \langle \text{elt} \rangle , \langle \text{elts}_N \rangle \mid \langle \text{elt}_N \rangle \\
 \langle \text{items}_N \rangle &::= \langle \text{item}_N \rangle , \langle \text{items} \rangle \mid \langle \text{item} \rangle , \langle \text{items}_N \rangle \\
 &\mid \langle \text{item } N \rangle \\
 \langle \text{item}_N \rangle &::= \langle \text{string} \rangle : \langle \text{elt}_N \rangle \mid \langle \text{string} \rangle : \langle \text{elt}_{N_1} \rangle \\
 \langle \text{elt}_{N_1} \rangle &::= \text{'null'}
 \end{aligned}$$

no $\langle \text{item } N \rangle$ is $\langle \text{string} \rangle : \text{null}$

{"abc": []}



Start Symbol

```

<jsonE> ::= <eltEEEEESE> '}'
<itemSE> ::= <itemE> ',' <items> | <item> ',' <itemSE>
          | <itemE>
<itemE> ::= <string> ':' <eltE> | <stringE1> ':' <elt>
<arrayE> ::= '[' <eltsE> ']'
<eltsE> ::= <eltE> ',' <elts> | <elt> ',' <eltsE> | <eltE>
<stringE1> ::= '"' <charsE2> '"'
<charsE2> ::= ε

```

```

def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
process(json)

```

<item E> is "" : <elt>

{"abc": 124} 

Start Symbol

```

<jsonE $\wedge$ N> ::= <eltE $\wedge$ NE $\wedge$ N> ::= <arrayE $\wedge$ N> | <objectE $\wedge$ N>
<arrayE $\wedge$ N> ::= '[' <eltsE $\wedge$ N> ']'
<objectE $\wedge$ N> ::= '{' <itemsE $\wedge$ N> '}'
<eltsE $\wedge$ N> ::= <eltE $\wedge$ N> | <eltE $\wedge$ N> ',' <eltsN>
  | <eltN> ',' <eltsE $\wedge$ N>
<eltN $\wedge$ N1> ::= 'false' | 'true'
  | <number> | <string> | <objectN> | <arrayN>
<itemsE $\wedge$ N> ::= <itemE $\wedge$ N> | <itemE $\wedge$ N> ',' <itemsN>
  | <itemN> ',' <itemsE $\wedge$ N>
<itemE $\wedge$ N> ::= <stringE1> ':' <eltN $\wedge$ N1>
  | <string> ':' <eltE $\wedge$ N $\wedge$ N1>
<eltE $\wedge$ N $\wedge$ N1> ::= <arrayE $\wedge$ N> | <objectE $\wedge$ N>

```

```

def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
    if any_key_has_null_value(json):
        fail('key value must not be null')
    process(json)

```

<item E> is "" : <elt>

&

no <item N> is <string> : null {"": 124}



```
def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
    if any_key_has_null_value(json):
        fail('key value must not be null')
    process(json)
```

Evogram

```
def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
    if any_key_has_null_value(json):
        fail('key value must not be null')
    process(json)
```

Evogram

$\langle \text{json}_{E \wedge \overline{N}} \rangle$

where $\langle \text{item}_E \rangle$ **is** `"":<elt>`
 $\langle \text{item}_N \rangle$ **is** $\langle \text{string} \rangle$:`null`

```

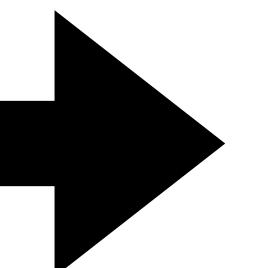
def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
    if any_key_has_null_value(json):
        fail('key value must not be null')
process(json)

```

$\langle \text{json}_{E \wedge \bar{N}} \rangle$

where $\langle \text{item}_E \rangle$ is " $:$ " : $\langle \text{elt} \rangle$
 $\langle \text{item}_N \rangle$ is $\langle \text{string} \rangle$: null

Evogram



Automatically Derived

```

 $\langle \text{json}_{E \wedge \bar{N}} \rangle ::= \langle \text{elt}_{E \wedge \bar{N}} \rangle$ 
 $\langle \text{elt}_{E \wedge \bar{N}} \rangle ::= \langle \text{array}_{E \wedge \bar{N}} \rangle \mid \langle \text{object}_{E \wedge \bar{N}} \rangle$ 
 $\langle \text{array}_{E \wedge \bar{N}} \rangle ::= '[' \langle \text{elts}_{E \wedge \bar{N}} \rangle ']'$ 
 $\langle \text{object}_{E \wedge \bar{N}} \rangle ::= '{}' \langle \text{items}_{E \wedge \bar{N}} \rangle '{}'$ 
 $\langle \text{elts}_{E \wedge \bar{N}} \rangle ::= \langle \text{elt}_{E \wedge \bar{N}} \rangle \mid \langle \text{elt}_{E \wedge \bar{N}} \rangle ',' \langle \text{elts}_{\bar{N}} \rangle$ 
 $\quad \mid \langle \text{elt}_{\bar{N}} \rangle ',' \langle \text{elts}_{E \wedge \bar{N}} \rangle$ 
 $\langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle ::= \text{'false'} \mid \text{'true'}$ 
 $\quad \mid \langle \text{number} \rangle \mid \langle \text{string} \rangle \mid \langle \text{object}_{\bar{N}} \rangle \mid \langle \text{array}_{\bar{N}} \rangle$ 
 $\langle \text{items}_{E \wedge \bar{N}} \rangle ::= \langle \text{item}_{E \wedge \bar{N}} \rangle \mid \langle \text{item}_{E \wedge \bar{N}} \rangle ',' \langle \text{items}_{\bar{N}} \rangle$ 
 $\quad \mid \langle \text{item}_{\bar{N}} \rangle ',' \langle \text{items}_{E \wedge \bar{N}} \rangle$ 
 $\langle \text{item}_{E \wedge \bar{N}} \rangle ::= \langle \text{string}_{E_1} \rangle ':' \langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle$ 
 $\quad \mid \langle \text{string} \rangle ':' \langle \text{elt}_{E \wedge \bar{N} \wedge \bar{N}_1} \rangle$ 
 $\langle \text{elt}_{E \wedge \bar{N} \wedge \bar{N}_1} \rangle ::= \langle \text{array}_{E \wedge \bar{N}} \rangle \mid \langle \text{object}_{E \wedge \bar{N}} \rangle$ 
 $\langle \text{json}_{\bar{N}} \rangle ::= \langle \text{elt}_{\bar{N}} \rangle$ 
 $\langle \text{elt}_{\bar{N}} \rangle ::= \text{'false'} \mid \text{'null'} \mid \text{'true'}$ 
 $\quad \mid \langle \text{number} \rangle \mid \langle \text{string} \rangle \mid \langle \text{array}_{\bar{N}} \rangle \mid \langle \text{object}_{\bar{N}} \rangle$ 
 $\langle \text{array}_{\bar{N}} \rangle ::= '[' \mid '[' \langle \text{elts}_{\bar{N}} \rangle ']'$ 
 $\langle \text{object}_{\bar{N}} \rangle ::= '{}' \mid '{}' \langle \text{items}_{\bar{N}} \rangle '{}'$ 
 $\langle \text{elts}_{\bar{N}} \rangle ::= \langle \text{elt}_{\bar{N}} \rangle \mid \langle \text{elt}_{\bar{N}} \rangle ',' \langle \text{elts}_{\bar{N}} \rangle$ 
 $\langle \text{items}_{\bar{N}} \rangle ::= \langle \text{item}_{\bar{N}} \rangle \mid \langle \text{item}_{\bar{N}} \rangle ',' \langle \text{items}_{\bar{N}} \rangle$ 
 $\langle \text{item}_{\bar{N}} \rangle ::= \langle \text{string} \rangle ':' \langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle$ 
 $\langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle ::= \text{'false'} \mid \text{'true'}$ 
 $\quad \mid \langle \text{number} \rangle \mid \langle \text{string} \rangle \mid \langle \text{array}_{\bar{N}} \rangle \mid \langle \text{object}_{\bar{N}} \rangle$ 
 $\langle \text{json}_{\bar{N}} \rangle ::= \langle \text{elt}_{\bar{N}} \rangle$ 
 $\langle \text{elt}_{\bar{N}} \rangle ::= \langle \text{object}_{\bar{N}} \rangle \mid \langle \text{array}_{\bar{N}} \rangle$ 
 $\langle \text{array}_{\bar{N}} \rangle ::= '[' \langle \text{elts}_{\bar{N}} \rangle ']'$ 
 $\langle \text{object}_{\bar{N}} \rangle ::= '{}' \langle \text{items}_{\bar{N}} \rangle '{}'$ 
 $\langle \text{elts}_{\bar{N}} \rangle ::= \langle \text{elt}_{\bar{N}} \rangle ',' \langle \text{elts} \rangle \mid \langle \text{elt} \rangle ',' \langle \text{elts}_{\bar{N}} \rangle \mid \langle \text{elt}_{\bar{N}} \rangle$ 
 $\langle \text{items}_{\bar{N}} \rangle ::= \langle \text{item}_{\bar{N}} \rangle ',' \langle \text{items} \rangle \mid \langle \text{item} \rangle ',' \langle \text{items}_{\bar{N}} \rangle$ 
 $\quad \mid \langle \text{item}_{\bar{N}} \rangle$ 
 $\langle \text{item}_{\bar{N}} \rangle ::= \langle \text{string} \rangle ':' \langle \text{elt}_{\bar{N}} \rangle \mid \langle \text{string} \rangle ':' \langle \text{elt}_{N_1} \rangle$ 
 $\langle \text{elt}_{N_1} \rangle ::= \text{'null'}$ 
 $\langle \text{json}_E \rangle ::= \langle \text{elt}_E \rangle$ 
 $\langle \text{elt}_E \rangle ::= \langle \text{object}_E \rangle \mid \langle \text{array}_E \rangle$ 
 $\langle \text{object}_E \rangle ::= '{}' \langle \text{items}_E \rangle '{}'$ 
 $\langle \text{items}_E \rangle ::= \langle \text{item}_E \rangle ',' \langle \text{items} \rangle \mid \langle \text{item} \rangle ',' \langle \text{items}_E \rangle$ 
 $\quad \mid \langle \text{item}_E \rangle$ 
 $\langle \text{item}_E \rangle ::= \langle \text{string} \rangle ':' \langle \text{elt}_E \rangle$ 
 $\langle \text{array}_E \rangle ::= '[' \langle \text{elts}_E \rangle ']'$ 
 $\langle \text{elts}_E \rangle ::= \langle \text{elt}_E \rangle ',' \langle \text{elts} \rangle \mid \langle \text{elt} \rangle ',' \langle \text{elts}_E \rangle \mid \langle \text{elt}_E \rangle$ 

```

```
def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
    if any_key_has_null_value(json):
        fail('key value must not be null')
    process(json)
```

```
def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
    if any_key_has_null_value(json):
        fail('key value must not be null')
    process(json)
```

$\langle \text{json}_{E \wedge \overline{N}} \rangle$

where $\langle \text{item}_E \rangle$ **is** `"":<elt>`
 $\langle \text{item}_N \rangle$ **is** $\langle \text{string} \rangle$:`null`

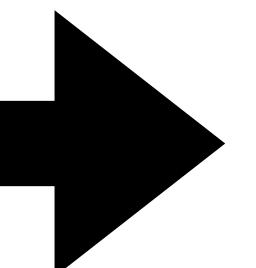
```

def jsoncheck(json):
    if no_key_is_empty_string(json):
        fail('one key must be empty')
    if any_key_has_null_value(json):
        fail('key value must not be null')
process(json)

```

$\langle \text{json}_{E \wedge \bar{N}} \rangle$

where $\langle \text{item}_E \rangle$ is " $:$ " : $\langle \text{elt} \rangle$
 $\langle \text{item}_N \rangle$ is $\langle \text{string} \rangle$: null



Automatically Derived

```

\langle \text{json}_{E \wedge \bar{N}} \rangle ::= \langle \text{elt}_{E \wedge \bar{N}} \rangle
\langle \text{elt}_{E \wedge \bar{N}} \rangle ::= \langle \text{array}_{E \wedge \bar{N}} \rangle \mid \langle \text{object}_{E \wedge \bar{N}} \rangle
\langle \text{array}_{E \wedge \bar{N}} \rangle ::= '[' \langle \text{elts}_{E \wedge \bar{N}} \rangle ']'
\langle \text{object}_{E \wedge \bar{N}} \rangle ::= '{' \langle \text{items}_{E \wedge \bar{N}} \rangle '}'
\langle \text{elts}_{E \wedge \bar{N}} \rangle ::= \langle \text{elt}_{E \wedge \bar{N}} \rangle \mid \langle \text{elt}_{E \wedge \bar{N}} \rangle ',' \langle \text{elts}_{\bar{N}} \rangle
\mid \langle \text{elt}_{\bar{N}} \rangle ',' \langle \text{elts}_{E \wedge \bar{N}} \rangle
\langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle ::= 'false' \mid 'true'
\mid \langle \text{number} \rangle \mid \langle \text{string} \rangle \mid \langle \text{object}_{\bar{N}} \rangle \mid \langle \text{array}_{\bar{N}} \rangle
\langle \text{items}_{E \wedge \bar{N}} \rangle ::= \langle \text{item}_{E \wedge \bar{N}} \rangle \mid \langle \text{item}_{E \wedge \bar{N}} \rangle ',' \langle \text{items}_{\bar{N}} \rangle
\mid \langle \text{item}_{\bar{N}} \rangle ',' \langle \text{items}_{E \wedge \bar{N}} \rangle
\langle \text{item}_{E \wedge \bar{N}} \rangle ::= \langle \text{string}_{E_1} \rangle ':' \langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle
\mid \langle \text{string} \rangle ':' \langle \text{elt}_{E \wedge \bar{N} \wedge \bar{N}_1} \rangle
\langle \text{elt}_{E \wedge \bar{N} \wedge \bar{N}_1} \rangle ::= \langle \text{array}_{E \wedge \bar{N}} \rangle \mid \langle \text{object}_{E \wedge \bar{N}} \rangle

\langle \text{json}_{\bar{N}} \rangle ::= \langle \text{elt}_{\bar{N}} \rangle
\langle \text{elt}_{\bar{N}} \rangle ::= 'false' \mid 'null' \mid 'true'
\mid \langle \text{number} \rangle \mid \langle \text{string} \rangle \mid \langle \text{array}_{\bar{N}} \rangle \mid \langle \text{object}_{\bar{N}} \rangle
\langle \text{array}_{\bar{N}} \rangle ::= '[' \mid '[' \langle \text{elts}_{\bar{N}} \rangle ']'
\langle \text{object}_{\bar{N}} \rangle ::= '{}' \mid '{}' \langle \text{items}_{\bar{N}} \rangle '{}'
\langle \text{elts}_{\bar{N}} \rangle ::= \langle \text{elt}_{\bar{N}} \rangle \mid \langle \text{elt}_{\bar{N}} \rangle ',' \langle \text{elts}_{\bar{N}} \rangle
\langle \text{items}_{\bar{N}} \rangle ::= \langle \text{item}_{\bar{N}} \rangle \mid \langle \text{item}_{\bar{N}} \rangle ',' \langle \text{items}_{\bar{N}} \rangle
\langle \text{item}_{\bar{N}} \rangle ::= \langle \text{string} \rangle ':' \langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle
\langle \text{elt}_{\bar{N} \wedge \bar{N}_1} \rangle ::= 'false' \mid 'true'
\mid \langle \text{number} \rangle \mid \langle \text{string} \rangle \mid \langle \text{array}_{\bar{N}} \rangle \mid \langle \text{object}_{\bar{N}} \rangle
\langle \text{json}_{\bar{N}} \rangle ::= \langle \text{elt}_{\bar{N}} \rangle
\langle \text{elt}_{\bar{N}} \rangle ::= \langle \text{object}_{\bar{N}} \rangle \mid \langle \text{array}_{\bar{N}} \rangle
\langle \text{array}_{\bar{N}} \rangle ::= '[' \langle \text{elts}_{\bar{N}} \rangle ']'
\langle \text{object}_{\bar{N}} \rangle ::= '{}' \langle \text{items}_{\bar{N}} \rangle '{}'
\langle \text{elts}_{\bar{N}} \rangle ::= \langle \text{elt}_{\bar{N}} \rangle \mid \langle \text{elt}_{\bar{N}} \rangle ',' \langle \text{elts}_{\bar{N}} \rangle \mid \langle \text{elt}_{\bar{N}} \rangle ',' \langle \text{elts}_{\bar{N}} \rangle \mid \langle \text{elt}_{\bar{N}} \rangle ',' \langle \text{elts}_{\bar{N}} \rangle
\langle \text{items}_{\bar{N}} \rangle ::= \langle \text{item}_{\bar{N}} \rangle \mid \langle \text{item}_{\bar{N}} \rangle ',' \langle \text{items}_{\bar{N}} \rangle \mid \langle \text{item}_{\bar{N}} \rangle ',' \langle \text{items}_{\bar{N}} \rangle \mid \langle \text{item}_{\bar{N}} \rangle ',' \langle \text{items}_{\bar{N}} \rangle
\langle \text{item}_{\bar{N}} \rangle ::= \langle \text{string} \rangle ':' \langle \text{elt}_{\bar{N}} \rangle \mid \langle \text{string} \rangle ':' \langle \text{elt}_{\bar{N}_1} \rangle
\langle \text{elt}_{\bar{N}_1} \rangle ::= 'null'
\langle \text{json}_E \rangle ::= \langle \text{elt}_E \rangle
\langle \text{elt}_E \rangle ::= \langle \text{object}_E \rangle \mid \langle \text{array}_E \rangle
\langle \text{object}_E \rangle ::= '{}' \langle \text{items}_E \rangle '{}'
\langle \text{items}_E \rangle ::= \langle \text{item}_E \rangle \mid \langle \text{item}_E \rangle ',' \langle \text{items}_E \rangle \mid \langle \text{item}_E \rangle ',' \langle \text{items}_E \rangle \mid \langle \text{item}_E \rangle ',' \langle \text{items}_E \rangle
\langle \text{item}_E \rangle ::= \langle \text{string} \rangle ':' \langle \text{elt}_E \rangle
\langle \text{array}_E \rangle ::= '[' \langle \text{elts}_E \rangle ']'
\langle \text{elts}_E \rangle ::= \langle \text{elt}_E \rangle \mid \langle \text{elt}_E \rangle ',' \langle \text{elts}_E \rangle \mid \langle \text{elt}_E \rangle ',' \langle \text{elts}_E \rangle \mid \langle \text{elt}_E \rangle ',' \langle \text{elts}_E \rangle

```

Supercharged Pattern Matchers

<calc not(D or F)>

where

<factor F> **is** ((<expr>))
<term D> **is** <factor> / 0

<json E and not(N)>

where

<item E> **is** "" : <elt>
<item N> **is** <string>: null

<C not(F) not(EW or ED or F)>

where

<forCondition F> **is** ";"
<iterationStatement EW> **is** <WHILE> "()" <statement>
<iterationStatement ED> **is** <D0> <statement> <WHILE> "()" <eos>

<ipv4addr 0 and H>

where

<quad 0> **is** "0" <num>
<quad H> **is** "0x" <num>

Alternative to Regular Expressions

jupyter x1_0_GeneratingSamples Last Checkpoint: a minute ago (autosaved)

Contents

- 1 Generating Samples
 - 1.1 Examples
 - 1.1.1 Calculator.py
 - 1.2 The error handler
 - 1.3 A random fuzzer.
 - 1.4 Adding Feedback**
 - 1.5 Building the Evolutionary Algorithm
 - 1.5.1 JSON
 - 2 Done

1 Generating Samples

In this notebook we will show how to generate valid samples for a given parser without using a grammar.

1.1 Examples

First we import the convenience utilities.

```
In [ ]: import src.utils as utils
```

1.1.1 Calculator.py

```
In [ ]: calculator = utils.load_file('subjects/calculator.py', 'calculator')
```

1.2 The error handler

We often need to interpret the error we get back. We use a simple exception class to capture the error.

```
In [ ]: with utils.ExpectError():
    calculator.main('xyz')
```

jupyter nbviewer.org/github/vrthra/mimid/blob/master/src/PymimidBook.ipynb

Mimid : Inferring Grammars

- Code for subjects [here](#)
- Evaluation starts [here](#)
 - The evaluation on specific subjects starts [here](#)
 - CGIDecode
 - Calculator
 - MathExpr
 - URLParse
 - Microjson
 - Lisp
- Results are [here](#)
- Recovering parse tree from a recognizer is [here](#)
- Recovering parse tree from parser combinator is [here](#)
- Recovering parse tree from PEG parer is [here](#)

Please note that a complete run can take an hour and a half to complete.

We start with a few Jupyter magics that let us specify examples inline, that can be turned off if needed for faster execution. Switch `TOP` to `False` if you do not want examples to complete.

```
In [1]: TOP = __name__ == '__main__'
```

The magics we use are `%%var` and `%%stop`. The `%%var` lets us specify large strings such as file contents directly without too many escapes. The `%%stop` helps with examples.

```
In [2]: from IPython.core.magic import (Magics, magics_class, cell_magic, line_magic, line_cell_magic)
class B(dict):
    def __getattr__(self, name):
        return self.__getitem__(name)
@magics_class
class MyMagics(Magics):
    def __init__(self, shell=None, **kwargs):
        super().__init__(shell=shell, **kwargs)
```

jupyter x3_0_AbstractingInputs Last Checkpoint: 11 hours ago (autosaved)

Contents

- 1 Abstracting Inputs
 - 1.1 Imports
 - 1.2 Grammars
 - 1.3 Predicate
 - 1.4 DDSets Simple**
 - 1.4.1 Example
 - 1.5 Single Fault Grammar
 - 1.5.1 Reachable Grammar
 - 1.5.2 Reachable positions.
 - 1.5.3 Insertion Grammar
 - 1.5.4 Pattern Grammar
 - 1.5.5 Insert A Fault
 - 1.6 Number the nodes.
 - 1.7 Abstract Context
 - 2 Done

1 Abstracting Inputs

1.1 Imports

```
In [ ]: import src.utils as utils
```

```
In [ ]: import ipynb.fs.full.x0_4_HierarchicalReducer as hdd
```

```
In [ ]: import ipynb.fs.full.x0_2_GrammarFuzzer as fuzzer
```

```
In [ ]: import ipynb.fs.full.x0_3_Parser as parser
```

```
In [ ]: from ipynb.fs.full.x0_4_HierarchicalReducer import PRes
```

```
In [ ]: import ast
import json
```

```
In [ ]: import random
```

```
In [ ]: if __name__ == '__main__':
    random.seed(0)
```

1.2 Grammars

```
In [ ]: import src.grammars as grammars
```

1.3 Predicate

```
In [ ]: import re
def expr_double_paren(inp):
```

jupyter x4_0_InputAlgebras Last Checkpoint: 11 hours ago (autosaved)

Contents

- 1 Input Algebras
 - 1.1 Imports
 - 1.2 Expression Grammar
 - 1.3 And Grammars
 - 1.3.1 Combining tokens
 - 1.4 Combining rules**
 - 1.4.1 Combining rulesets
 - 1.4.2 definition conjunction
 - 1.4.3 grammar conjunction
 - 1.5 Producing inputs with at least one of the
 - 1.5.1 Nonterminals
 - 1.6 rules
 - 1.7 rulesets
 - 1.7.1 definition disjunction
 - 1.7.2 grammar disjunction
 - 2 A grammar with no fault inducing fragment
 - 2.1 Unreachable Grammar
 - 2.2 Negated pattern grammar.
 - 2.3 Negation of full evocative expressions
 - 3 Done

1 Input Algebras

More complex and fully worked out examples including abstraction of real world bugs from Clojure, Rhino, Lua and a few Unix utilities [here](#).

1.1 Imports

```
In [ ]: import src.utils as utils
```

```
In [ ]: import ipynb.fs.full.x0_4_HierarchicalReducer as hdd
```

```
In [ ]: import ipynb.fs.full.x0_2_GrammarFuzzer as fuzzer
```

```
In [ ]: import ipynb.fs.full.x0_3_Parser as parser
```

```
In [ ]: from ipynb.fs.full.x0_4_HierarchicalReducer import PRes
```

```
In [ ]: import ipynb.fs.full.x3_0_AbstractingInputs as abstractinginputs
```

```
In [ ]: import sympy
```

```
In [ ]: import random
```

```
In [ ]: random.seed(0)
```

```
In [ ]: import itertools as I
```

1.2 Expression Grammar

```
In [ ]: import sympy
```