(\*

The BNF ( Backus Naur Format ) contained in this document conforms (roughly) to the EBNF format documented at: <http://en.wikipedia.org/wiki/Extended_Backus–Naur_Form>

Note: This grammar lacks syntactic separators. This is an attempt to make the grammar independent of any specific representation, that is, not specific to JSON, XML, YAML etc. As a result the grammar is syntactically incomplete. The comma separator (normally used to represent concatenation) and white space (normally irrelevant) are used here to separate items that require some (unspecified) syntactic separation.

\*)

BIR\_file = bir\_element {bir\_element} processor\_layout ;

bir\_element = processor

| basic\_block

| table

| metadata

(\* Should we add "header" as an ordered collection? \*)

| struct

;

processor = control\_flow

| other\_processor

;

control\_flow = cfp\_id

"type" "control\_flow"

"start\_control\_state" control\_state

;

other\_processor = op\_id

"type" " other\_processor"

"inputs" {data\_id}

"outputs" {data\_id} ;

control\_state = offset\_expression

{next\_bb} default\_bb; (\* order is meaningful \*)

offset\_expression = ?expression yielding a non-negative integer? ;

next\_bb = guard ( bb\_id | "$done$" ) ;

default\_bb = bb\_id | "$done$" ;

guard = ?expression yielding TRUE or FALSE? ;

basic\_block = bb\_id

"type" "basic\_block"

["local\_header" struct\_id]

["local\_table" table\_id]

instructions

next\_control\_state ;

instructions = "instructions" {instruction} ;

instruction = V\_inst | O\_inst | M\_inst ;

V\_inst = "V" ( data\_id "." field\_id | field\_id ) bits\_expr ;

bits\_expr = ?expression yielding a value to set in a data field? ;

(\* pif\_ir/bir/utils/bir\_parser.py \*)

O\_inst = "O" operation\_id operands ; (\* need to expand details \*)

M\_inst = "M" method\_id M\_results M\_parameters ;

M\_results = data\_id ;

M\_parameters = data\_id ;

next\_control\_state = "next\_control\_state" control\_state ;

table = table\_id

"type" "table"

"match\_type" match\_type (\* need to change to per field \*)

"depth" ?positive integer?

"request" data\_id

"reponse" data\_id

"operations" {operation}

;

match\_type = "indexed" | "exact" | "binary CAM" | "LPM" | "TCAM" ;

(\* Should match\_type be associated with the table or with the fields of the request? \*)

operation = " add\_entry" | "modify\_entry" | "delete\_entry" ;

metadata = data\_id

"type" "metadata"

"values" struct\_id

"visibility" visibility (\* could be supplanted by input/output on processor\_layout \*)

"value\_inits" {VALUE} (\* ordered list of values corresponding to fields in struct\_id \*)

;

visibility = "in" | "out" | "inout" | "none" ; (\* wrt/ processor\_layout \*)

struct = struct\_id

"fields" field\_id field\_width {field\_id field\_width} ;

field\_width = ?positive integer width in bits? ;

field\_id = identifier ;

data\_id = identifier ;

table\_id = identifier ;

bb\_id = identifier ;

op\_id = identifier ;

cfp\_id = identifier ;

pl\_id = identifier ;

identifier = ? starts with a letter A to Z or a to z or an underscore (\_), followed by zero or more letters, underscores and digits (0 to 9)? ;

processor\_layout = pl\_id

"type" "processor\_layout"

"format" format\_type

"implementation" processor {processor}

"inputs" {data\_id}

"outputs" {data\_id}

;

format\_type = "list" ;