Overview of BIR (B Intermediate Representation) v0.9

Gordon Brebner

15 December, 2015

Introduction

BIR (B Intermediate Representation, pronounced "Beer") is a proposed intermediate representation for high-level packet processing programs being compiled to target packet processing architectures. It is a product of the Open Source SDN organization's Protocol Independent Forwarding (PIF) project, which also operates as a working group within the Specifications area of the Open Networking Foundation. The two main motivations for BIR are: (a) to avoid a proliferation of custom compilers for specific high-level language and target architecture combinations; and (b) to offer a neutral packet processing model that is not tied closely to specific languages and/or architectures but does not impede the efficient compilation of programs to targets. BIR is the product of lengthy technical discussions and experiments. It is a successor to AIR (A Intermediate Representation, the early PIF strawman proposal, pronounced "Air") and was particularly influenced by existing ongoing research on intermediate representations for packet processing including NetASM (Princeton), POF (Huawei) and PX (Xilinx), by the emergent P4 high-level language, and by a broad spectrum of target architectures.

BIR represents programs using a three-layer packet processing model:

- Data Flow level (top level): a linear processing pipeline through which packets and associated metadata pass. Each pipeline stage is termed a (packet) processor. A processor either has a programmatic description as a BIR Control Flow level component, or meets the pipeline stage interface but does not have an explicit BIR description.
- Control Flow level (middle level): a state machine that operates on a packet and its associated metadata. Each state has a programmatic description as a BIR Basic Block level component that performs an unconditional sequence of instructions. Transitions between states are described using conditional jumps that are taken at the end of the Basic Block execution.
- Basic Block level (bottom level): a sequence of BIR Instructions that is executed within a particular context during the processing of a packet and its metadata. This local context optionally has two components: a header a particular offset within the packet; and a table providing lookup to contribute to the processing.

The Control Flow and Basic Block levels (and the use of that terminology) are familiar from traditional general-purpose intermediate representations used by compilers, such as LLVM. The Data Flow level, the use of header and table contexts in the Basic Block level, and the provided Instruction repertoire, are familiar from domain-specific models of packet processing, such as the OpenFlow datapath.

This overview of BIR version 0.9 is provided for review by relevant stakeholders, including high-level packet processing language designers and compiler writers; target architecture specialists; and runtime configuration and operation API designers and implementers. A summary of the various object types used in BIR representations, as introduced in the following sections, is given in Appendix 1.

Data Flow level description

A Data Flow, which constitutes a described packet processing system instance, is represented by a BIR **processor_layout** type object. This object has one attribute: an ordered list of **processor** type objects. The ordered list represents the linear packet processing pipeline from start to finish.

A BIR **processor** type object is either a **control_flow** type object or an **other_processor** type object. The former is described in the next section. The latter type object is used to include a packet processor that is not explicitly described as a BIR Control Flow, which allows BIR system instances to include other standard library or user-provided packet processing components. An **other_processor** object has one attribute, the name of a processor class. This name is used to access standard metadata associated with the processor by searching a standard or user-provided directory tree. This metadata minimally includes any metadata associated with BIR-defined Control Flow processors. These other processors may, in general, store and reorder packets, which are not capabilities of standard BIR Control Flow processors.

The pipeline may, in general, contain multiple packets being processed simultaneously, both across stages and within stages. Particular target architectures may place restrictions on the extent to which such parallel packet processing is allowed.

The familiar use case for a Data Flow is a packet forwarding datapath, where packets are classified and/or modified under the influence of tables that are read or written by a runtime control interface.

A matter for further study is whether a more general directed graph might be used rather than just a simple ordered list. For example, this could be a way of extending BIR's expressive capability towards describing conditional packet switching.

Control Flow level description

A Control Flow is represented by a BIR **control_flow** type object. This object has one attribute: an **offset_block** type object. This object is used to describe the starting state for the Control Flow. It has two attributes: an expression for computing a non-negative integer packet offset value and a conditional nest for selecting a **basic_block** type object. The computed value is used to specify the offset into the packet that applies at the beginning of the execution of the Control Flow. The object type is described in the next section. The selected object is used to specify the first Basic Block used in execution of the Control Flow. The calculations can use operands from metadata travelling with the packet, as well as constant values.

The Control Flow behaves as a finite state machine, where the states are Basic Blocks with local contexts. In each state, the Basic Block is executed in order to perform (unconditional) packet processing. The initial state is determined using the **offset_block** type object. Then state transitions occur as a Basic Block associated with the current state processes its own **offset_block** type object to determine the Basic Block associated with the next state, as discussed in the next section. The final state is determined when a Basic Block processes its own **offset_block** type object, and gets a termination result rather than a next Basic Block result. At this point, the Control Flow has completed its execution. As the state transitions occur, the offset into the packet is updated using the **offset_block** type object.

There are two familiar use cases for a Control Flow. The first is a packet parser or editor, where the states correspond to successive packet headers. The second is a match-action table flow, where the states correspond to successive tables used.

Basic Block Level description

A Basic Block is represented by a BIR **basic_block** type object. This object has four attributes. The first attribute is an optional **header** type object, which specifies the format of the local header in the packet being processed. The second attribute is an optional **table** type object, which specifies the local table. The third attribute is an ordered list of **instruction** objects, which represents the sequence of instructions executed. The fourth attribute is an **offset_block** type object, which is used to describe the next state (or termination) in the enclosing Control Flow.

The Basic Block is where packet processing work is actually done. Each one can be data plane driven, in that it operates on a localized header of the packet, and control plane driven, in that it has access to a single table with a control interface. The local header begins at the offset into the packet that is provided on entry to the Basic Block. Overall, instructions in the Basic Block can operate on the local header of the packet, all metadata carried with the packet, and the local table. Headers and metadata, tables, and instructions, are described in the next three sections respectively. Both local header and local table are optional, their inclusion or not depending on the function of the Basic Block.

After the unconditional sequence of instructions has been executed, the next state of the enclosing Control Flow is selected. The **offset_block** object is used to describe the next state for the Control Flow. It has two attributes: an expression for computing a non-negative integer packet offset value and a conditional nest for selecting a **basic_block** type object. The computed value is used to specify the offset into the packet that applies in the next state of the execution of the Control Flow, and must be greater than or equal to the offset in the current state. If backward movement in the packet is required (offset reduction), this must be done by starting a new Control Flow. The object specifies the next Basic Block, which may be null to denote Control Flow termination. The calculations can use operands from the local packet header or metadata travelling with the packet, as well as constant values.

Packets, Headers, and Metadata

Packets are the fundamental units operated upon by BIR packet processing system instances. They do not feature in the static BIR representation of a system, but their existence underpins all of the components of the BIR representation. Packets contain BIR-defined headers, which are fixed-format sections of the packet (not necessarily just at the beginning of the packet, despite the name). Packets are accompanied by additional BIR-defined metadata as they are processed by the system.

A header format is represented by a BIR **header** type object. This object has one attribute: an ordered list of fields, which specifies the format of the header. Each field has two components: an identifier string and a positive integer width in bits. Header objects are used in Basic Blocks to interpret the raw bit string that constitutes a physical packet.

A collection of metadata is represented by a BIR **metadata** type object. This object has one attribute: an ordered list of fields, which specifies the content of the metadata. Each field has three components: an identifier string, a positive integer width in bits, and an optional unsigned integer initial value. An instance of each defined metadata object type travels with each packet that is processed. These are used to hold per-packet state during processing.

Tables

Tables are the means by which packet processing is influenced by a control plane. A control interface can be used to add, modify, or delete, entries in a table. Table lookups are performed during packet processing by Basic Blocks.

A table is represented by a BIR **table** type object. This object has five attributes: a table type string indicating the style of lookup (differentiating indexed, binary CAM, longest-prefix match, and TCAM, for example), a positive integer length giving the number of entries, an ordered list of request fields, an ordered list of response fields, and a list of data plane operations supported by the table besides lookup (add entry, modify entry, delete entry, for example). Each field in the request and response lists has two components: an identifier string and a positive integer width in bits.

The request and response formats have no implied semantics beyond that fact that an ordered list of one or more values is supplied as a lookup request to the table, and an ordered list of one or more values is returned as a response from the table. A basic table will only provide lookup as its data plane operation.

Instructions

Instructions are the means by which Basic Blocks perform packet processing. There are three instruction types: F (header or metadata single Field assignment), H (packet Header or metadata complete assignment), and M (built-in or user-provided Method call). An instruction has two arguments, the exact nature of the arguments depending on the specific instruction type.

An F type instruction has two arguments. The first is the name of a header field within the local header for the Basic Block, or the name of a metadata field. This is the destination of the single field assignment. The second is an expression for computing the assigned value, with operands drawn from local header fields, metadata fields, or constants. Expressions are described in the next section.

An H type instruction has two arguments. The first is the name of the local header for the Basic Block, or the name of a metadata collection. This is the destination of the complete assignment. The second is the name of a metadata collection, which is supplied as a request to the local table for the Basic Block, the result of the lookup being used as the assigned value. Other ways of specifying the assigned value, for example using the local header or a metadata collection, are a matter for future study.

An M type instruction has two arguments. The first is the name of a method to be called. This is either the name of a built-in method, or a name which is looked up ~~dynamically~~ at compile time in a user-provided library collection. The second is a list of zero or more parameters for the method, the nature of this list depending on the method that is named. There are four built-in methods:

- Header insert, Hinsert(header_name [,length]): insert a header at the current packet offset
- Header remove, Hremove([,length]): remove a header at the current packet offset
- Entry insert, Einsert(request, result [,mask]): add an entry to the local table
- Entry remove, Eremove(request): remove an entry from the local table

More detail on these built-in method calls in given in Appendix 2.

Expressions

Expressions are used in various contexts, as mentioned in preceding sections: for computing initial and next packet offset values, for evaluating conditions in a condition nest for selecting an initial or next Basic Block, and for computing values assigned by F group instructions.

BIR expressions are atomic, in the sense that the BIR representation does not explicitly decompose them further into individual operations to be carried out in some order. This matter is left to back-end compilation for particular targets. The operators usable within BIR expressions are the standard arithmetic, bitwise, and logical, operators found in languages such as C or Python. The operands depend on the context of the expression. These may be metadata fields, or constants, in all contexts. These may be packet header fields in Basic Blocks that have a local packet header specified. Parentheses may be used in expressions to override standard operator precedence.

The exact representation, and semantics, of expressions will be defined precisely in a future version of this specification.

Annotations

The danger of a common intermediate representation is that useful information in a higher-level language description may be lost and/or obfuscated from back-end compilation to target architectures. To avoid this, and to assist with debugging, BIR representations can contain annotations, falling into three broad categories: (i) source description information, such as filenames, line numbers, and column numbers; (ii) source structure information, such as hierarchical and modular naming and scoping; and (iii) function information, such as description of the packet processing capabilities of particular modules. The first two categories will have standard formats. The third category will have an evolving list of known capabilities, one example being packet parsing. The exact representation of annotations will be defined precisely in a future version of this specification.

Runtime APIs

This specification is concerned with the BIR representation of packet processing systems. An accompanying specification will concern the use of this representation when operating such systems. This requires a BIR-level API that can sit between a particular high-level language runtime API, and a particular target architecture runtime API. Notably, this must contain primitives for accessing Tables within the BIR representation, to supply the control interface to the packet processing. Runtime access of any other BIR components is a matter for future study.

Status and Open Issues

This document contains the first outline draft of the BIR specification. One topic not addressed explicitly is the handling of stateful information between packets, particularly when there are multiple access points. These situations may arise through "other" processors in the data flow, callable methods in the control flow, and Tables. A possible rule is to allow multiple read points, but only single write points.

APPENDIX 1: Summary of BIR object types

The following object types are used to form BIR representations of packet processing systems.

Name	Description	Attributes
processor_layout	Data flow pipeline	Ordered list of processors
processor	Pipeline processor	
control_flow	Control flow processor	Starting state (offset_block)
other_processor	Non control flow processor	Name of processor class
offset_block	Offset and Basic Block evaluation	Packet offset calculation; Basic Block selection
basic_block	Basic Block	Local header format (optional); Local table (optional); Ordered list of instructions; Next state (offset_block)
header	Packet header format	Ordered list of fields
metadata	Metadata collection format	Ordered list of fields
table	Table	Table type; Table depth; Ordered list of request fields; Ordered list of response fields; List of data plane operations
instruction	Basic Block instruction	Instruction type: F, H, or M Argument 1 Argument 2

APPENDIX 2: Summary of BIR built-in methods

The following built-in methods can be called from M type instructions in Basic Blocks.

Name	Description	Parameters
hInsert	Insert header at current offset, and make it the local header for the enclosing Basic Block	Header format; Overriding size in bits (optional)
hRemove	Remove header at current offset, and stop it being the local header for the enclosing Basic Block	Overriding size in bits (optional)
eInsert	Insert entry into the local Table for the enclosing Basic Block	Request field value set; Response field value set; Mask field value set (optional)
eRemove	Remove entry from the local Table for the enclosing Basic Block	Request field value set