P4 Language Cheat Sheet

Basic Data Types

```
// typedef: introduces alternate type name
typedef bit<48> macAddr_t;
typedef bit<32> ip4Addr_t;

// headers: ordered collection of members
// operations test and set validity bits:
// isValid(), setValid(), setInvalid()
header ethernet_t {
  macAddr_t dstAddr;
  macAddr_t srcAddr;
  bit<16> type;
}

// variable declaration and member access
ethernet_t ethernet;
macAddr_t src = ethernet.srcAddr;

// struct: unordered collection of members
struct headers_t {
  ethernet_t ethernet;
}
```

Parsing

```
// packet_in: extern for input packet
extern packet_in {
  void extract<T>(out T hdr);
  void extract<T>(out T hdr,in bit<32> n);
  T lookahead<T>();
  void advance(in bit<32> n);
  bit<32> length();
}

// parser: begins in special "start" state
state start {
  transition parse_ethernet;
}

// User-defined parser state
state parse_ethernet {
  packet.extract(hdr.ethernet);
  transition select(hdr.ethernet.type) {
    0x800: parse_ipv4;
    default: accept;
  }
}
```

Statements & Expressions

```
// Local metadata declaration, assignment
bit<16> tmp1; bit<16> tmp2;
tmp1 = hdr.ethernet.type;

// bit slicing, concatenation
tmp2 = tmp1[7:0] ++ tmp1[15:8];

// addition, subtraction, casts
tmp2 = tmp1 + tmp1 - (bit<16>)tmp1[7:0];

// bitwise operators
tmp2 = (~tmp1 & tmp1) | (tmp1 ^ tmp1);
tmp2 = tmp1 << 3;
```

Actions

```
// Inputs provided by control-plane
action set_next_hop(bit<32> next_hop) {
  if (next_hop == 0) {
    metadata.next_hop = hdr.ipv4.dst;
  } else {
    metadata.next_hop = next_hop;
  }
}

// Inputs provided by data-plane
action swap_mac(inout bit<48> x,
                inout bit<48> y) {
  bit<48> tmp = x;
  x = y;
  y = tmp;
}

// Inputs provided by control/data-plane
action forward(in bit<9> p, bit<48> d) {
  standard_metadata.egress_spec = p;
  headers.ethernet.dstAddr = d;
}

// Remove header from packet
action decap_ip_ip() {
    hdr.ipv4 = hdr.inner_ipv4;
    hdr.inner_ipv4.setInvalid();
}
```

Tables

```
table ipv4_lpm {
  key = {
    hdr.ipv4.dstAddr : lpm;
    // standard match kinds:
    // exact, ternary, lpm
  }
  // actions that can be invoked
  actions = {
    ipv4_forward;
    drop;
    NoAction;
  }
  // table properties
  size = 1024;
  default_action = NoAction();
}
```

Control Flow

```
apply {
  // branch on header validity
  if (hdr.ipv4.isValid()) {
    ipv4_lpm.apply();
  }
  // branch on table hit result
  if (local_ip_table.apply().hit) {
    send_to_cpu();
  }
  // branch on table action invocation
  switch (table1.apply().action_run) {
    action1: { table2.apply(); }
    action2: { table3.apply(); }
  }
}
```

Deparsing

```
// packet_out: extern for output packet
extern packet_out {
  void emit<T>(in T hdr);
}

apply {
  // insert headers into pkt if valid
  packet.emit(hdr.ethernet);
}
```

Header Stacks

```
// header stack declaration
header label_t {
  bit<20> label;
  bit bos;
}
struct header_t {
  label_t[10] labels;
}
header_t hdr;

// remove from header stack
action pop_label() {
  hdr.labels.pop_front(1);
}

// add to header stack
action push_label(in bit<20> label) {
  hdr.labels.push_front(1);
  hdr.labels[0].setValid();
  hdr.labels[0] = { label, 0};
}
```

Advanced Parsing

```
// common defns for IPv4 and IPv6
header ip46_t {
  bit<4> version;
  bit<4> reserved;
}

// header stack parsing
state parse_labels {
  packet.extract(hdr.labels.next);
  transition select(hdr.labels.last.bos) {
    0: parse_labels; // create loop
    1: guess_labels_payload;
  }
}

// lookahead parsing
state guess_labels_payload {
  transition select(packet.lookahead<
    ip46_t>().version) {
    4 : parse_inner_ipv4;
    6 : parse_inner_ipv6;
    default : parse_inner_ethernet;
  }
}
```

V1Model - Architecture

```
// common externs
extern void truncate(in bit<32> length);
extern void resubmit<T>(in T x);
extern void recirculate<T>(in T x);
enum CloneType { I2E, E2I }
extern void clone(in CloneType type,
                  in bit<32> session);

// v1model pipeline elements
parser Parser<H, M>(
  packet_in pkt,
  out H hdr,
  inout M meta,
  inout standard_metadata_t std_meta
);
control VerifyChecksum<H, M>(
  inout H hdr,
  inout M meta
);
control Ingress<H, M>(
  inout H hdr,
  inout M meta,
  inout standard_metadata_t std_meta
);
control Egress<H, M>(
  inout H hdr,
  inout M meta,
  inout standard_metadata_t std_meta
);
control ComputeChecksum<H, M>(
  inout H hdr,
  inout M meta
);
control Deparser<H>(
  packet_out b, in H hdr
);

// v1model switch
package V1Switch<H, M>(
  Parser<H, M> p,
  VerifyChecksum<H, M> vr,
  Ingress<H, M> ig,
  Egress<H, M> eg,
  ComputeChecksum<H, M> ck,
  Deparser<H> d
);
```

V1Model - Standard Metadata

```
struct standard_metadata_t {
    bit<9>  ingress_port;
    bit<9>  egress_spec;
    bit<9>  egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1>  drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
    bit<19> deq_qdepth;
    bit<48> ingress_global_timestamp;
    bit<48> egress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<32> resubmit_flag;
    bit<16> egress_rid;
    bit<1>  checksum_error;
    bit<32> recirculate_flag;
}
```

V1Model - Counters & Registers

```
// counters
counter(8192, CounterType.packets) c;

action count(bit<32> index) {
    //increment counter at index
    c.count(index);
}

// registers
register<bit<48>>(16384) r;

action ipg(out bit<48> ival, bit<32> x) {
  bit<48> last;
  bit<48> now;
  r.read(last, x);
  now = std_meta.ingress_global_timestamp;
  ival = now - last;
  r.write(x, now);
}
```