isValid(), setValid(), setInvalid()。当一个结构体（local variable or header type）初始化时，其为invalid。

注：不允许header嵌套

成功执行execute指令就会将valid bit置为true

Push\_front：右移，负责插入

Pop\_front：左移，负责删除

1. 基本结构
2. stream

stream用来标识在pipeline中流动的变量，例如stream packet\_seg=packet[100]，表示从packet中取出前100bytes流向pipeline进行一系列操作；

**stream packet\_seg = packet[100]**

1. headers

与P4分开定义header不同，rP4将header的定义集中在headers结构内；在该结构内，使用hdr定义每个header；

**headers { }**

1. hdr

在每个hdr内，field的定义方式与rP4相同，但是跟随每个protocol的还有它的解析方式以及针对该protocol的一系列方法，这些方法只能操作该protocol的字段；

**hdr ethernet\_t {**

**bit<48> dstAddr;**

**bit<48> srcAddr;**

**bit<16> etherType;**

**implicit parser(etherType) {**

**0x0800: ipv4\_t;**

**0x0812: probe\_t;**

**default:;**

**}**

**}**

1. implicit

该关键字在hdr内部进行定义，从而表征隐式parser（implicit parser）及隐式处理函数（implicit func）；

**implicit func update\_checksum() {**

**this.hdrChecksum = HashAlgorithm.csum16(this.version,**

**this.ihl,**

**this.diffserv,**

**this.totalLen,**

**this.identification,**

**this.flags,**

**this.fragOffset,**

**this.ttl,**

**this.protocol,**

**this.srcAddr,**

**this.dstAddr);**

**}**

1. atom关键字

该关键字同样出现在hdr定义内，表示在进行protocol解析时必须进行的动作。

**hdr probe\_t {**

**bit<8> hop\_cnt;**

**implicit parser(hop\_cnt) {**

**0: probe\_fwd\_t;**

**default: probe\_data\_t;**

**}**

**atom {**

**metadata.remaining = hop\_cnt + 1;**

**}**

**}**

1. header\_sequence关键字

该关键字的定义与P4中struct headers的定义是相同的，描述header中protocol出现的顺序（与是否出现无关）。需要注意的是，hs内的所有变量均为指针，因为rP4是不进行实际的extract，只是记录在100bytes中protocol的offset及field的offset和length。

**header\_sequence {**

**ethernet\_t ethernet;**

**ipv4\_t ipv4;**

**probe\_t probe;**

**probe\_data\_t[10] probe\_data;**

**probe\_fwd\_t[10] probe\_fwd;**

**}hs;**

1. actions

该关键字与headers的定义类似，是所有action的集合。在每个action前面会有ingress/egress关键字，表示该action应部署在ingress pipeline还是egress pipeline。

**actions { }**

1. action

action内部与P4相似，由一系列的primitive组成。action的参数为para，其在action内部可以根据table返回的参数进行调用，在编译rP4时会将其翻译成参数索引的形式。

**action ipv4\_forward(para) {**

**copy\_field(para.port, standard\_metadata.egress\_spec);**

**copy\_field(hs.ethernet.dstAddr, hs.ethernet.srcAddr);**

**copy\_field(para.dstAddr, hs.ethernet.dstAddr);**

**decrement(hs.ipv4.ttl);**

**}**

1. structs

与actions、headers类似。其中最经常出现的是metadata、ingress\_variable和egress\_variable，表示随packet的元数据、处理packet时ingress pipeline可配置的变量、处理packet时egress pipeline可配置的变量。

**structs {**

**struct metadata {**

**bit<8> egress\_spec;**

**bit<8> remaining;**

**}meta;**

**struct ingress\_variable {**

**}iv;**

**struct egress\_variable {**

**register<bit<32>>(8) byte\_cnt\_reg;**

**register<bit<48>>(8) last\_time\_reg;**

**bit<32> byte\_cnt;**

**bit<32> new\_byte\_cnt;**

**bit<48> last\_time;**

**bit<48> cur\_time;**

**}ev;**

**}**

1. control

最重要的是控制流程。其中rP4\_Ingress和rP4\_Egress表示两个pipeline内需要处理的逻辑。stage为配置processor需要的逻辑。control之间可以进行调用。

**control rP4\_Ingress {**

**stage ipv4\_lpm\_s {**

**hs.ipv4 = packet.ipv4\_t => [verify\_checksum()];**

**action, para = hs.ipv4\_lpm.lpm([ipv4.dstAddr]);**

**switch action:**

**1: ipv4\_forward(para);**

**2: drop();**

**default: drop();**

**}**

**stage probe\_s {**

**hs.probe = packet.probe\_t;**

**action, para = ;**

**switch action:**

**default: do\_probe();**

**}**

**}**

1. pipelines

存储packet所有可能走的pipeline顺序。可以看到其是使用rP4\_Ingress.xxx进行表征的。

**pipelines {**

**pipeline{**

**rP4\_Ingress.ipv4\_lpm\_s**

**}**

**pipeline{**

**rP4\_Ingress.probe\_s,**

**rP4\_Egress.reg\_update\_s,**

**rP4\_Egress.probe\_data\_s**

**}**

**}**

1. 顺序符号：-> => <=

三种顺序符号有不同的意义。其中“->”表示执行完某个pipeline之后应执行的操作，比如rP4\_Egress->[update\_checksum()]表示egress pipeline之后更新IPv4的checksum。“=>”表示阻塞式执行，在该语句不满足时执行“=>”之后的操作，并退出该stage；“<=”表示非阻塞式执行，在该语句不满足时不进行操作，继续执行后面的语句。这种表示方法是为了避免在stage中出现if else的判断，因为parser matcher executer已经对操作进行解耦，如果加上判断语句则又将这些操作耦合起来。

**control rP4\_Ingress {**

**stage src\_route\_s {**

**hs.srcRoutes = packet.extract(srcRoute\_t) => [drop()];**

**hs.ipv4 <= packet.ipv4\_t;**

**action, para = (hs.srcRoutes[0].bos == 1) << 1 + (hs.ipv4.isValid());**

**switch action:**

**11: src\_route\_1();**

**10: src\_route\_0();**

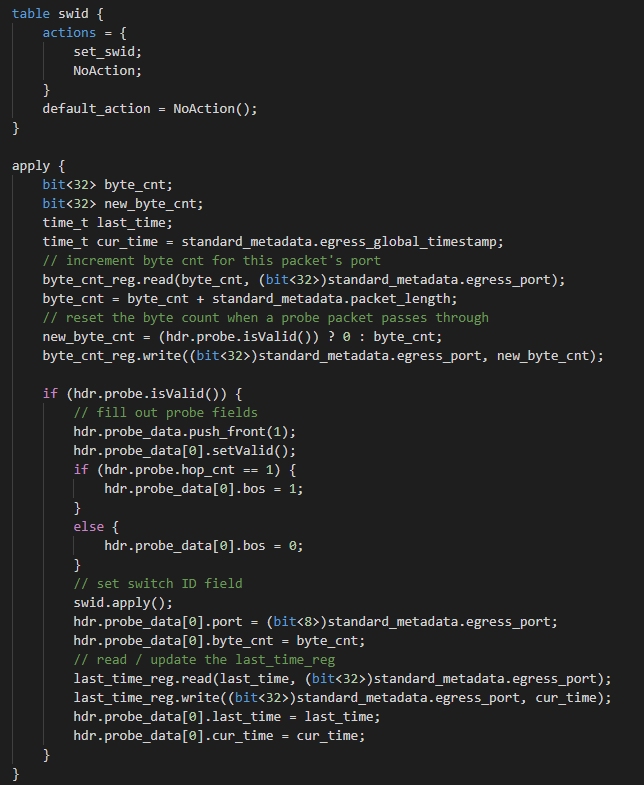
**default: drop();**

**}**

**}**

1. P4内逻辑转换

由于P4主函数内的逻辑不是解耦的，如下图所示，apply内有一系列的逻辑操作，并且有if else等分支判断语句，rP4要做的就是将耦合的逻辑解耦成使用stage最少、pipeline速度最快的独立模块。



首先，apply内if语句之前的语句可以分解成一个独立的stage。对于if else语句，rP4对此进行编码处理。