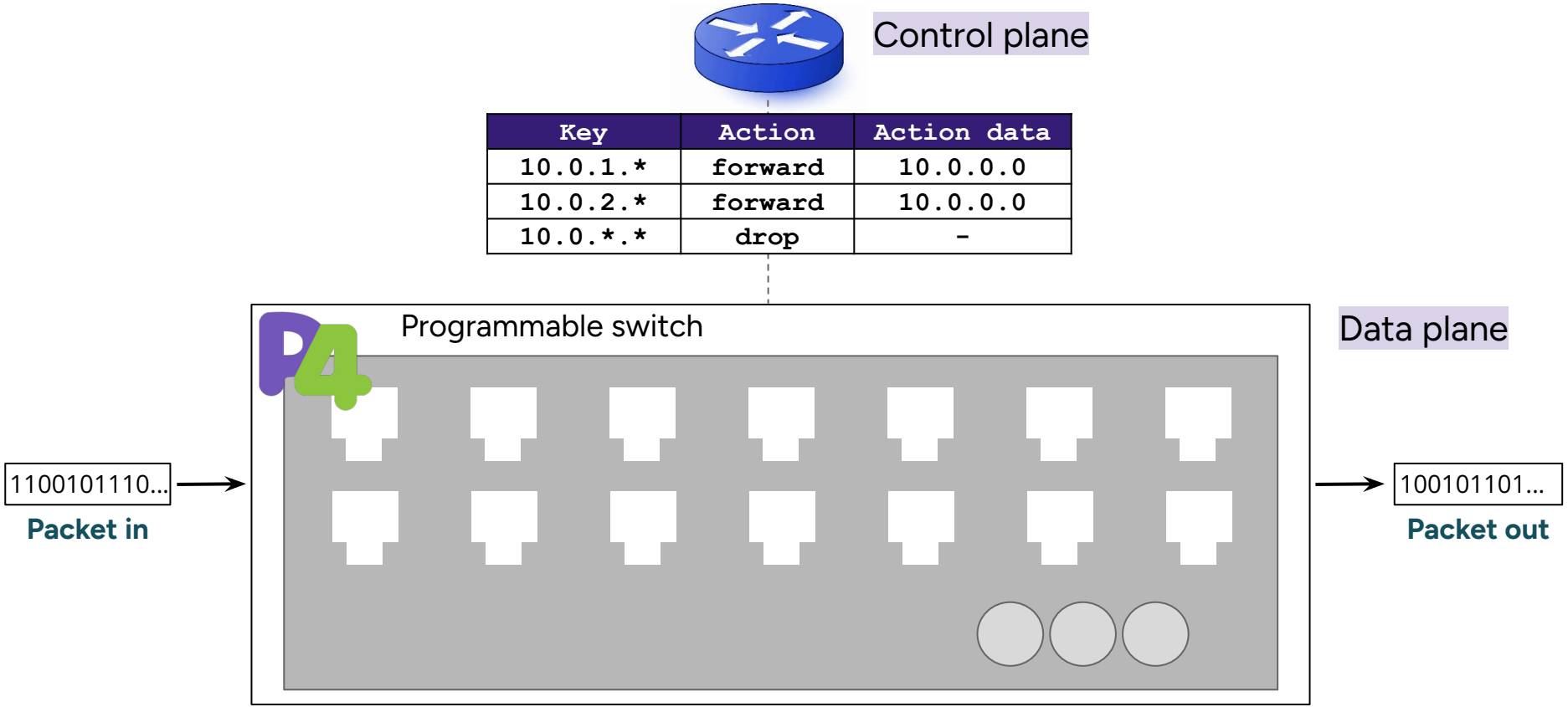
HOL4P4: Semantics and Type System for Data Planes

Anoud Alshnakat
KTH Royal Institute of Technology
anoud@kth.se

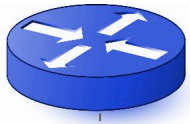
Software Defined Networking (SDN)



P4 Features

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ... //src and newAddr will be
}         swapped here as well

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

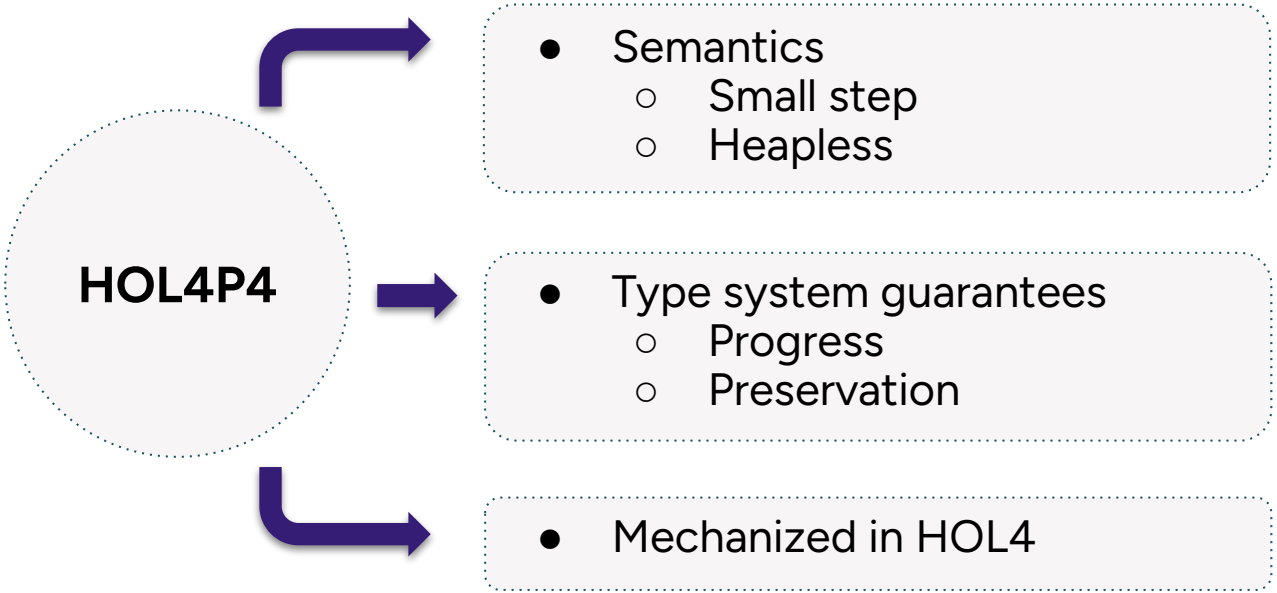
- Interaction with tables
- No loops, recursion, or call-by-reference
- Calling convention: copy-in/copy-out



Control plane

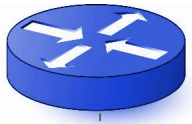
Key	Action	Action data
10.0.1.*	forward	10.0.0.0
10.0.2.*	forward	10.0.0.0
10.0.*.*	drop	-

P4 Formalization



Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}


table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```



Control plane

Key	Action	Action data
10.0.1.*	forward	10.0.0.0
10.0.2.*	forward	10.0.0.0
10.0.*.*	drop	-

Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

MyCtrl	forward(10.0.0.0);	

h.ip.src ↦ 10.0.0.2

Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}

action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

forward	swap(h.ip.src, newAddr); ...	newAddr ↦ 10.0.0.0
MyCtrl	* ;	

h.ip.src ↦ 10.0.0.2

Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}

action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

swap	bit<32> tmp = x;
	x = y;
	y = tmp;
		x ↦ (10.0.0.2, h.ip.src)
		y ↦ (10.0.0.0, newAddr)

forward	*; ...	newAddr ↦ 10.0.0.0
MyCtrl	*;	

h.ip.src ↦ 10.0.0.2

Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

swap	x = y; y = tmp;	x ↦ (10.0.0.2, h.ip.src) y ↦ (10.0.0.0, newAddr) tmp ↦ 10.0.0.2
forward	*; ...	newAddr ↦ 10.0.0.0
MyCtrl	*;	

h.ip.src ↦ 10.0.0.2

Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

swap	`y = tmp;`	`x ↦ (10.0.0.0, h.ip.src)`
		`y ↦ (10.0.0.0, newAddr )`
		`tmp ↦ 10.0.0.2`

forward	`*;` `...`	`newAddr ↦ 10.0.0.0`
MyCtrl	`*;`	

`h.ip.src ↦ 10.0.0.2`

Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

swap		x ↦ (10.0.0.0, h.ip.src) y ↦ (10.0.0.2, newAddr) tmp ↦ 10.0.0.2
forward	*; ...	newAddr ↦ 10.0.0.0
MyCtrl	*;	

h.ip.src ↦ 10.0.0.2

Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

swap		x ↦ (10.0.0.0, h.ip.src) y ↦ (10.0.0.2, newAddr) tmp ↦ 10.0.0.2
forward	*; ...	newAddr ↦ 10.0.0.0
MyCtrl	*;	

h.ip.src ↦ 10.0.0.2

Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

swap		x ↦ (10.0.0.0, h.ip.src) y ↦ (10.0.0.2, newAddr) tmp ↦ 10.0.0.2
forward	*; ...	newAddr ↦ 10.0.0.0
MyCtrl	*;	

h.ip.src ↦ 10.0.0.2

Example: function call

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}

action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

forward	*; ...	newAddr ↦ 10.0.0.2
MyCtrl	*;	

h.ip.src ↦ 10.0.0.0

Typing Variable Maps

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

swap	`bit<32> tmp = x;`
	`x = y;`
	`y = tmp;`
		$x \mapsto (10.0.0.2, h.ip.src)$
		$y \mapsto (10.0.0.0, newAddr)$
		$x \mapsto (bv^{32}, h.ip.src)$
		$y \mapsto (bv^{32}, newAddr)$

forward	`*;` `...`	$newAddr \mapsto 10.0.0.0$ $newAddr \mapsto bv^{32}$
MyCtrl	`*;`	

$h.ip.src \mapsto 10.0.0.2$

ψ_G $h.ip.src \mapsto bv^{32}$

Typing Copy-in

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

$F(\text{swap}) = ((\text{inout}, \text{bv}^{32}, x), (\text{inout}, \text{bv}^{32}, y)) \rightarrow \text{void}$

$F(\text{forward}) = ((\text{none}, \text{bv}^{32}, \text{newAddr})) \rightarrow \text{void}$

swap	bit<32> tmp = x;
	x = y;
	y = tmp;
		x ↦ (10.0.0.2, h.ip.src)
		y ↦ (10.0.0.0, newAddr)
		x ↦ (bv^{32}, h.ip.src)
		y ↦ (bv^{32}, newAddr)

forward	*; ...	newAddr ↦ 10.0.0.0 newAddr ↦ bv^{32}
MyCtrl	*;	

h.ip.src ↦ 10.0.0.2

ψ_G h.ip.src ↦ bv^{32}

Typing Copy-in

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

$F(\text{swap}) = ((\text{inout}, bv^{32}, x), (\text{inout}, bv^{32}, y)) \rightarrow \text{void}$

$F(\text{forward}) = ((\text{none}, bv^{32}, \text{newAddr})) \rightarrow \text{void}$

swap	bit<32> tmp = x;
	x = y;
	y = tmp;
		x ↦ (10.0.0.2, h.ip.src)
		y ↦ (10.0.0.0, newAddr)
		x ↦ (bv^{32}, h.ip.src)
		y ↦ (bv^{32}, newAddr)

forward	*; ...	newAddr ↦ 10.0.0.0 newAddr ↦ bv^{32}
MyCtrl	*;	

ψ_G

h.ip.src ↦ 10.0.0.2

h.ip.src ↦ bv^{32}

Typing Copy-out

```
control MyCtrl (inout headers h, … ) {
...

void swap(inout bit<32> x, inout bit<32> y){
      bit<32> tmp = x;
      x = y;
      y = tmp
}


action forward(bit<32> newAddr){
      swap (h.ip.src, newAddr);
      ...
}

table ip_match {
      key = { h.ip.dst: lpm; }
      actions = { forward; drop; }
}

apply { ip_match.apply(); }
}
```

Ensure type and l-value consistency between caller's arguments and callee's parameters

$F(\text{swap}) = ((\text{inout}, bv^{32}, x), (\text{inout}, bv^{32}, y)) \rightarrow \text{void}$

$F(\text{forward}) = ((\text{none}, bv^{32}, \text{newAddr})) \rightarrow \text{void}$

swap	bit<32> tmp = x;
	x = y;
	y = tmp;
		x ↦ (10.0.0.2, h.ip.src)
		y ↦ (10.0.0.0, newAddr)
		x ↦ (bv^{32} , h.ip.src)
		y ↦ (bv^{32} , newAddr)

forward	*; ...	newAddr ↦ 10.0.0.0 newAddr ↦ bv^{32}
MyCtrl	*;	

ψ_G

h.ip.src ↦ 10.0.0.2

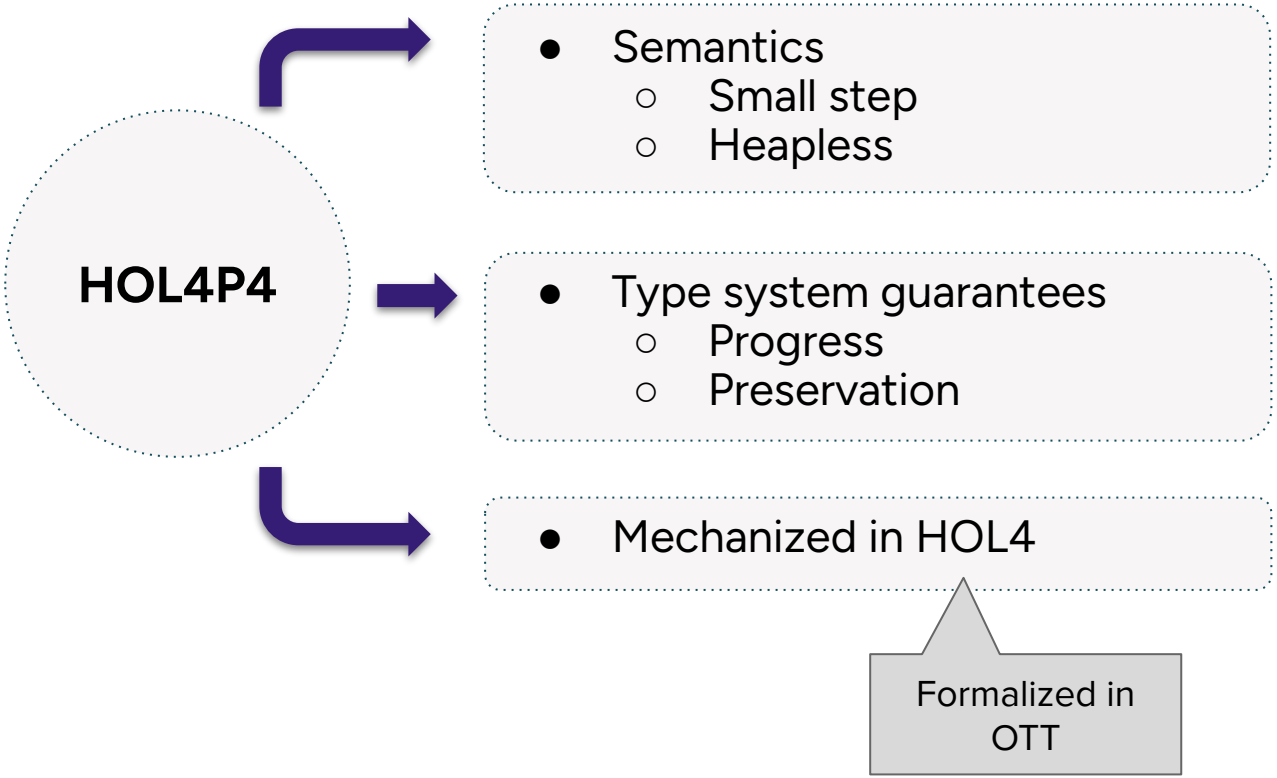
h.ip.src ↦ bv^{32}

Progress and Preservation

If $\Gamma, \overline{\psi} \vdash s$ and $\Gamma \vdash F, T$ then

- if $\mathrm{NonTerminal}(s)$, then exists s' such that $F, T \vdash s \to s'$
- if $F, T \vdash s \to s'$, then there exists $\overline{\psi'}$ such that $\Gamma, \overline{\psi'} \vdash s'$

P4 Formalization



OTT

```
[ tau1 x1 d1 , .. , taun xn dn ] tau = t_lookup_funn ( funn' , delta )
t_scopes_tup ( order , funn , delta ) |-  ( e1 , .. , en ) : ( [ tau1 , .. , taun ] , [ b1 , .. , bn ] )
out_is_lval  [ d1 , .. , dn ]  [ b1 , .. , bn ]
----------------------------------- :: call
t_scopes_tup ( order , funn , delta ) |- call funn' ( e1 , .. , en ) : ( tau , false  )
```

```
/\ ( (* e_typ_call *) ! (e_tau_x_d_b_list:(e#tau#x#d#b) list) (t_scopes_tup:t_scopes_tup) (order:order) (delta:delta) (funn':funn) (tau:tau)

(clause_name "e_typ_call") /\

(( SOME ( ((MAP (\(e_,tau_,x_,d_,b_) . (tau_,x_,d_)) e_tau_x_d_b_list)) , tau ) = t_lookup_funn  funn'   delta ) /\

( !i. (i< LENGTH  ((MAP (\(e_,tau_,x_,d_,b_) . e_) e_tau_x_d_b_list)) ) ==>
 ( e_typ  t_scopes_tup    (funn,delta)
    (EL i  ((MAP (\(e_,tau_,x_,d_,b_) . e_) e_tau_x_d_b_list)) )
    (t_tau (EL i  ((MAP (\(e_,tau_,x_,d_,b_) . tau_) e_tau_x_d_b_list)) ))
    (EL i  ((MAP (\(e_,tau_,x_,d_,b_) . b_) e_tau_x_d_b_list)) ) ) ) /\

( out_is_lval  ((MAP (\(e_,tau_,x_,d_,b_) . d_) e_tau_x_d_b_list)) ((MAP (\(e_,tau_,x_,d_,b_) . b_) e_tau_x_d_b_list)))

 ==>
( ( e_typ t_scopes_tup  (funn,delta)  (e_call funn' ((MAP (\(e_,tau_,x_,d_,b_) . e_) e_tau_x_d_b_list))) (t_tau tau)  F  ))))
```

OTT → HOL4

```
[ tau1 x1 d1 , .. , taun xn dn ] tau = t_lookup_funn ( funn' , delta )
t_scopes_tup ( order , funn , delta ) |-  ( e1 , .. , en ) : ( [ tau1 , .. , taun ] , [ b1 , .. , bn ] )
out_is_lval  [ d1 , .. , dn ]  [ b1 , .. , bn ]
---------------------------------- :: call
t_scopes_tup ( order , funn , delta ) |- call funn' ( e1 , .. , en ) : ( tau , false  )
```

```
/\ ( (* e_typ_call *) ! (e_tau_x_d_b_list:(e#tau#x#d#b) list) (t_scopes_tup:t_scopes_tup) (order:order) (delta:delta) (funn':funn) (tau:tau)

(clause_name "e_typ_call") /\

(( SOME ( ((MAP (\(e_,tau_,x_,d_,b_) . (tau_,x_,d_)) e_tau_x_d_b_list)) , tau ) = t_lookup_funn  funn'   delta ) /\

( !i. (i< LENGTH  ((MAP (\(e_,tau_,x_,d_,b_) . e_) e_tau_x_d_b_list)) ) ==>
 ( e_typ  t_scopes_tup    (funn,delta)
    (EL i  ((MAP (\(e_,tau_,x_,d_,b_) . e_) e_tau_x_d_b_list)) )
    (t_tau (EL i  ((MAP (\(e_,tau_,x_,d_,b_) . tau_) e_tau_x_d_b_list)) ))
    (EL i  ((MAP (\(e_,tau_,x_,d_,b_) . b_) e_tau_x_d_b_list)) ) ) /\

( out_is_lval  ( MAP (\(e_,tau_,x_,d_,b_) . d_) e_tau_x_d_b_list)) ( MAP (\(e_,tau_,x_,d_,b_) . b_) e_tau_x_d_b_list)))

 ==>
( ( e_typ t_scopes_tup  (funn,delta)  (e_call funn' ( MAP (\(e_,tau_,x_,d_,b_) . e_) e_tau_x_d_b_list)) (t_tau tau)  F  ))))
```

Lemmas regarding MAP, ZIP, UNZIP, Pairs … etc

Most useful tactics in proofs

```
REPEAT (BasicProvers.FULL_CASE_TAC >> gvs[])
```

```
fun OPEN_EXP_TYP_TAC exp_term =

(Q.PAT_X_ASSUM ` e_typ (t1,t2) t exp_term typ bl` (fn thm =>
ASSUME_TAC (SIMP_RULE (srw_ss()) [Once e_typ_cases] thm)))
```

Conclusion

- Small-step heapless semantics for P4.
- Type system that guarantees progress and preservation.
- Meta theory is mechanized in HOL4.

github.com/kth-step/HOL4P4