Honey Potion: an eBPF Backend for Elixir

Kael Soares kaelaugusto@dcc.ufmg.br

Vinícius Pacheco pacheco@cadence.com

Marcos Vieira mmvieira@dcc.ufmg.br

Rodrigo Ribeiro rodrigo.ribeiro@ufop.edu.br

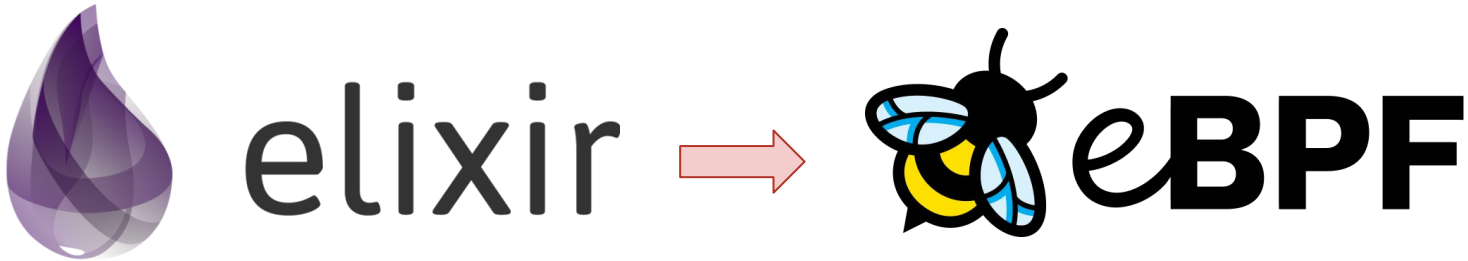
Fernando Pereira fernando@dcc.ufmg.br

Compilers Laboratory

Goal

DCC

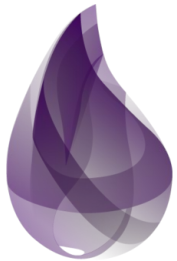
Honey Potion: an eBPF Backend for Elixir



Goal

DCC

- Functional
- Dynamic
- Concurrent



elixir → eBPF

Goal

DCC

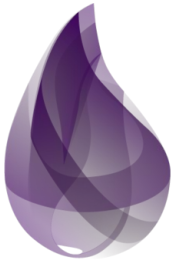
- Functional
- Dynamic
- Concurrent

Creator: José Valim

First appeared: 2012

Stable release: 1.18.1

Platform: Erlang (BEAM)



elixir → eBPF

Goal

DCC

- Functional
- Dynamic
- Concurrent

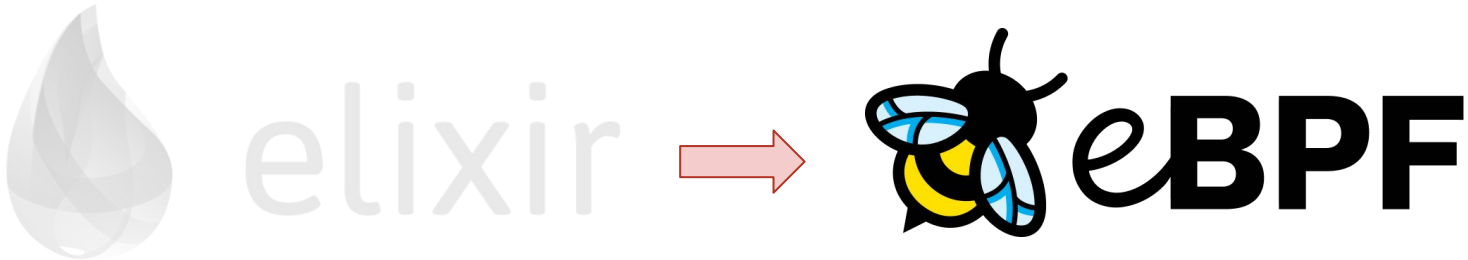
- Imperative
- Low-Level
- Restricted

Goal

DCC

Eg.: a signal handler

- Imperative
- Low-Level
- Restricted



Goal

DCC

- Restricted

Verified in the Linux Kernel

DCC

- ~~Restricted~~ Verifier

Verified in the Linux Kernel

DCC

- Ensured Termination
- In-Bound Memory Accesses

- ~~Restricted~~ Verifier

Verified in the Linux Kernel

DCC

- Ensured Termination
- In-Bound Memory Accesses

- Recursion-Rich
- Turing Complete

- No backward branch
- 1 million instructions

Verified in the Linux Kernel

- Ensured Termination
- In-Bound Memory Accesses

- Recursion-Rich
- Dynamically typed

- 512 bytes of stack
- 11 registers

Verified in the Linux Kernel

DCC

- Ensured Termination
- In-Bound Memory Accesses

- Every call is inlined
- Local variables
- Spills

- Recursion-Rich
- Dynamically typed

- 512 bytes of stack
- 11 registers

Verified in the Linux Kernel

DCC

- Ensured Termination
- In-Bound Memory Accesses

- Every call is inlined
- Local variables
- Spills
- Heap (eBPF map)

- Recursion-Rich
- Dynamically typed

- 512 bytes of stack
- 11 registers

Example

DCC

```
01  defmodule Counter do
02    @counter %{count: 0}
03
04    def increment() do
05      count = Map.get(@counter, :count, 0)
06      new_count = count + 1
07      put_in(@counter[:count], new_count)
08      new_count
09    end
10  end
```

Example

DCC

```
01  defmodule Counter do
02    @counter %{count: 0}
03
04    def increment() do
05      count = Map.get(@counter, :count, 0)
06      new_count = count + 1
07      put_in(@counter[:count], new_count)
08      new_count
09    end
10  end
```

```
    r1 = 0
    r2 = map_fd
    call bpf_map_lookup_elem
    r3 = r0
    if r3 == 0 goto init

    r4 = *(r3)
    r4 += 1
    *(r3) = r4
    call bpf_map_update_elem
    exit

init:
    r4 = 0
    call bpf_map_update_elem
    exit
```

Example

DCC

```
defmodule Counter do
  @counter %{count: 0}

  def increment() do
    count = Map.get(@counter, :count, 0)
    new_count = count + 1
    put_in(@counter[:count], new_count)
    new_count
  end
end
```

Load key (0) into r1
Load map file descriptor into r2
Lookup value in the map
Store result in r3
If NULL, initialize counter

```
r1 = 0
r2 = map_fd
call bpf_map_lookup_elem
r3 = r0
if r3 == 0 goto init

r4 = *(r3)
r4 += 1
*(r3) = r4
call bpf_map_update_elem
exit

init:
```

Initialize counter

```
r4 = 0
call bpf_map_update_elem
exit
```

Example

DCC

```
defmodule Counter do
  @counter %{count: 0}

  def increment() do
    count = Map.get(@counter, :count, 0)
    new_count = count + 1
    put_in(@counter[:count], new_count)
    new_count
  end
end
```

Load counter value

```
r1 = 0
r2 = map_fd
call bpf_map_lookup_elem
r3 = r0
if r3 == 0 goto init

r4 = *(r3)
r4 += 1
*(r3) = r4
call bpf_map_update_elem
exit

init:
r4 = 0
call bpf_map_update_elem
exit
```

Example

DCC

```
defmodule Counter do
  @counter %{count: 0}

  def increment() do
    count = Map.get(@counter, :count, 0)
    new_count = count + 1
    put_in(@counter[:count], new_count)
    new_count
  end
end
```

Increment it

```
  r1 = 0
  r2 = map_fd
  call bpf_map_lookup_elem
  r3 = r0
  if r3 == 0 goto init

  r4 = *(r3)
  r4 += 1
  *(r3) = r4
  call bpf_map_update_elem
  exit

init:
  r4 = 0
  call bpf_map_update_elem
  exit
```

Example

DCC

```
defmodule Counter do
  @counter %{count: 0}

  def increment() do
    count = Map.get(@counter, :count, 0)
    new_count = count + 1
    put_in(@counter[:count], new_count)
    new_count
  end
end
```

Update map

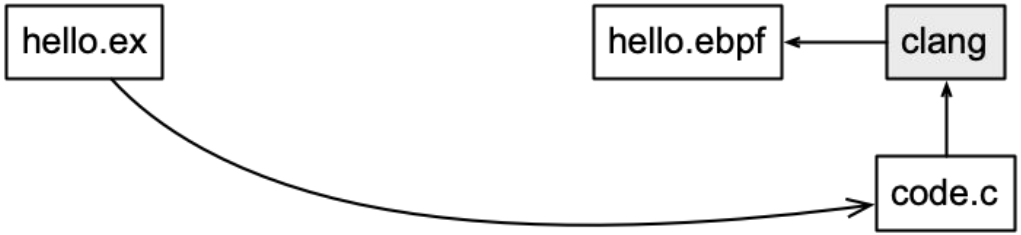
```
  r1 = 0
  r2 = map_fd
  call bpf_map_lookup_elem
  r3 = r0
  if r3 == 0 goto init

  r4 = *(r3)
  r4 += 1
  *(r3) = r4
  call bpf_map_update_elem
  exit

init:
  r4 = 0
  call bpf_map_update_elem
  exit
```

How To?

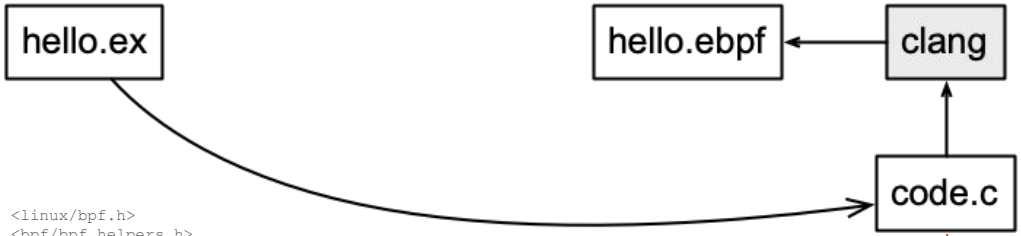

```
defmodule Counter do
  @counter %{count: 0}

  def increment() do
    count = Map.get(@counter, :count, 0)
    new_count = count + 1
    put_in(@counter[:count], new_count)
    new_count
  end
end
```

```
  r1 = 0
  r2 = map_fd
  call bpf_map_lookup_elem
  r3 = r0
  if r3 == 0 goto init

  r4 = *(r3)
  r4 += 1
  *(r3) = r4
  call bpf_map_update_elem
  exit

init:
  r4 = 0
  call bpf_map_update_elem
  exit
```

How To?



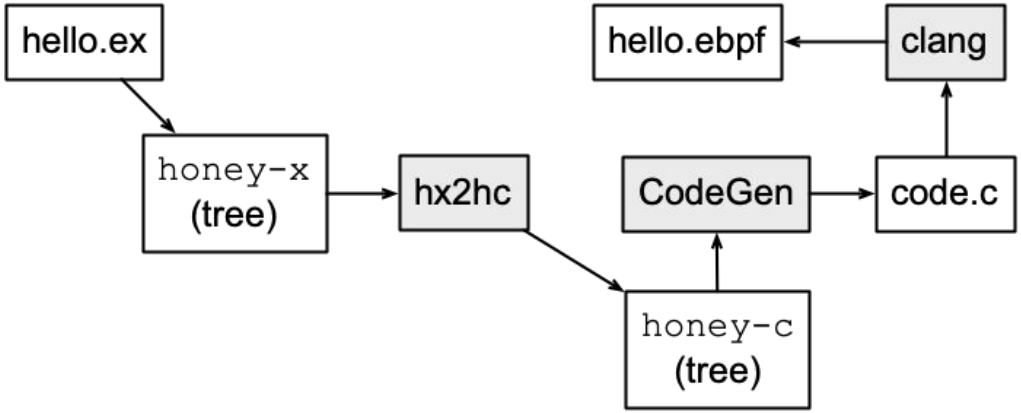
How To?



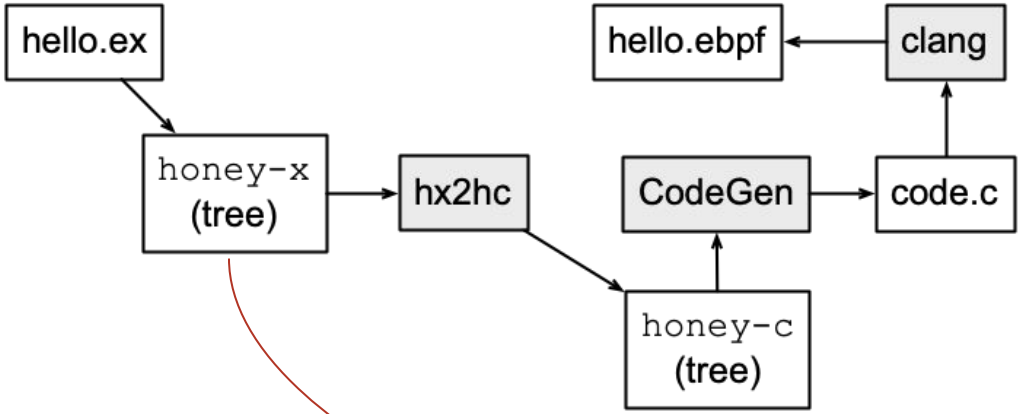

```
#include <linux/bpf.h>
#include <bpf/bpf_helpers.h>

struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, 1);
    __type(key, int);
    __type(value, int);
} counter_map SEC(".maps");

SEC("xdp")
int increment_counter(struct __sk_buff *skb) {
    int key = 0;
    int zero = 0;
    int *value;

    value = bpf_map_lookup_elem(&counter_map, &key);
    if (!value) {
        bpf_map_update_elem(&counter_map, &key, &zero, BPF_ANY);
        value = &zero;
    }

    (*value)++;

    bpf_map_update_elem(&counter_map, &key, value, BPF_ANY);

    return XDP_PASS;
}

char _license[] SEC("license") = "GPL";
```

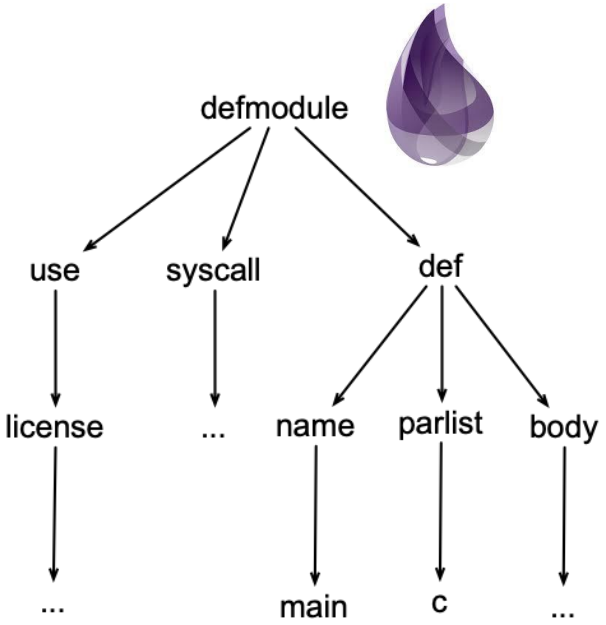
How To?

DCC


hello.ex
honey-x
(tree)
hx2hc
honey-c
(tree)
CodeGen
hello.ebpf
clang
code.c

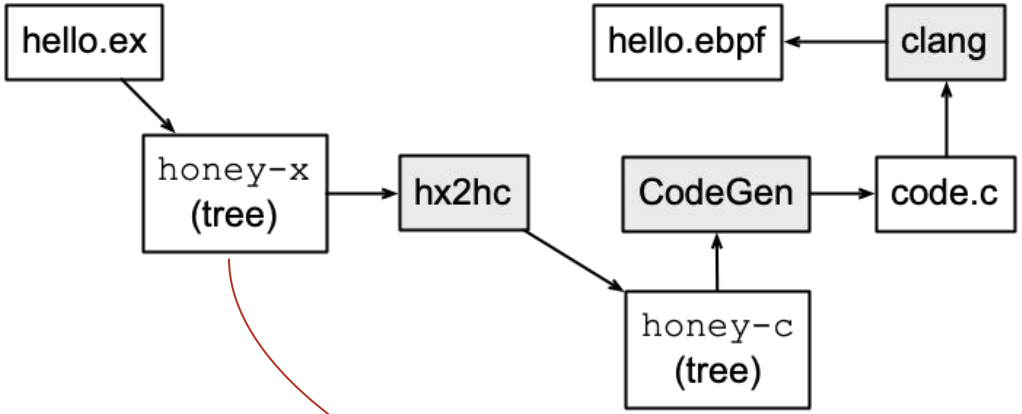
How To?

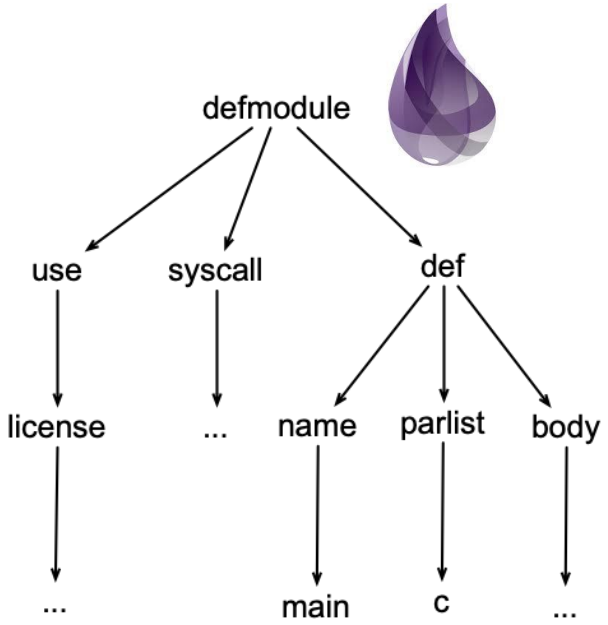




Tree in static single-assignment form

How To?

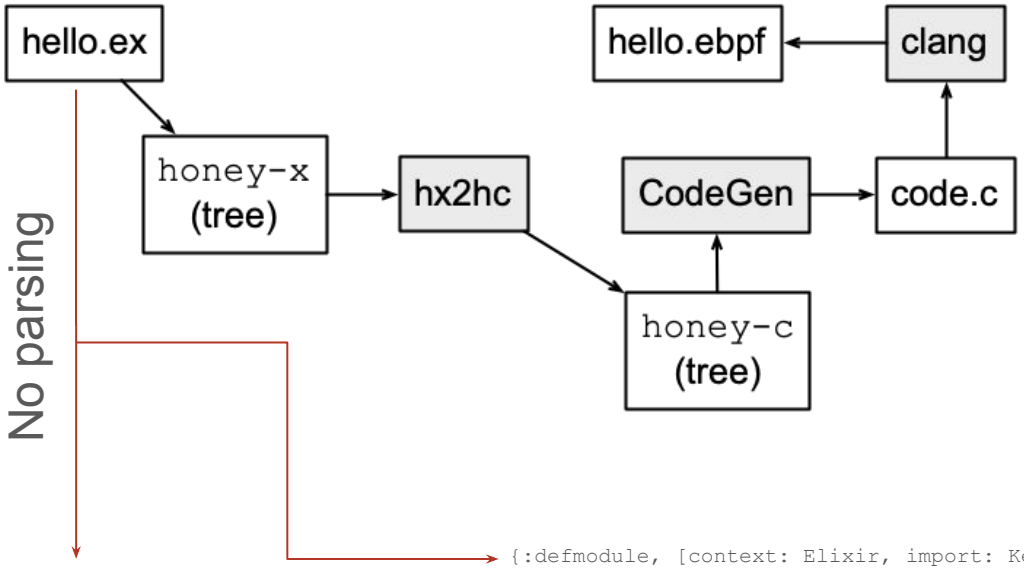




Metaprogramming!

1. **Capture Elixir's AST** using quote.
2. **Transform it into a structured IR** (Honey-X).

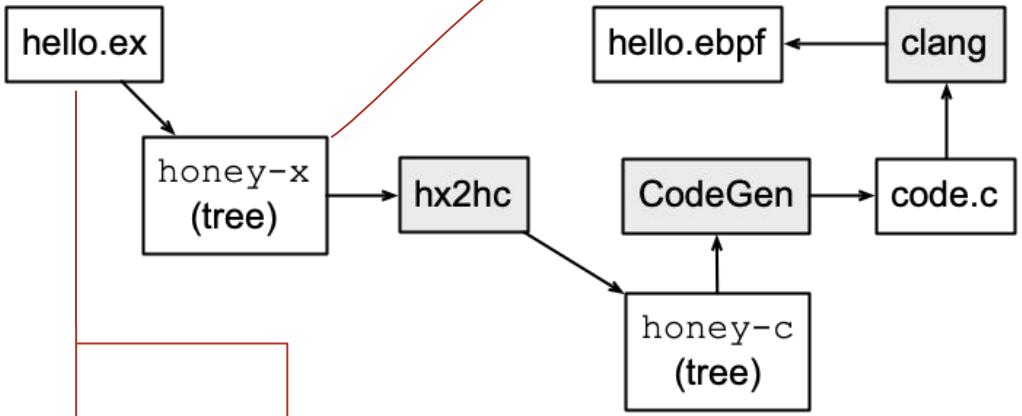
How To?




```
iex> quote do
...> def inc(x) do
...>   x + 1
...> end
...> end
```

```
{:defmodule, [context: Elixir, import: Kernel], [
  {:__aliases__, [alias: false], [:Example]},
  [
  do: {:def, [context: Elixir, import: Kernel], [
    {:inc, [context: Elixir], [{:x, [], nil}]},
    [do: {:+, [context: Elixir, import: Kernel], [{:x, [], nil}, 1]}]
  ]}
  ]
]}
```

How To?

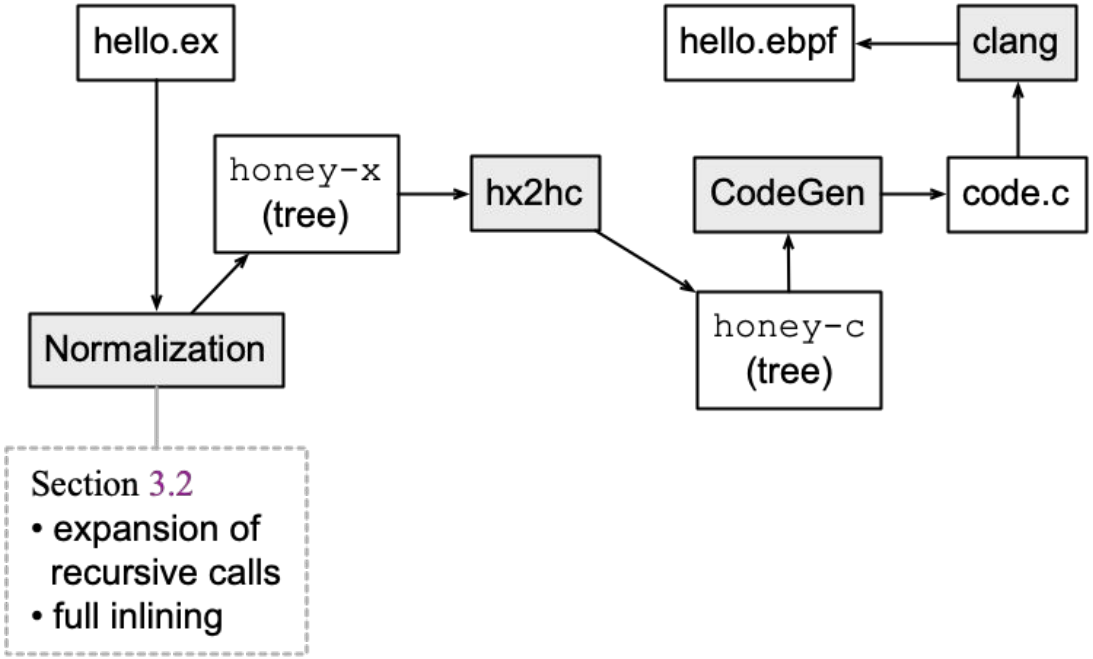



```
%HoneyX{
  type: :function,
  name: :inc,
  params: [:x],
  body: %HoneyX{
  type: :binop,
  operator: :+,
  left: %HoneyX{
    type: :variable,
    name: :x
  },
  right: %HoneyX{
    type: :literal,
    value: 1
  }
  }
}
```

```
iex> quote do
...> def inc(x) do
...>   x + 1
...> end
...> end
```

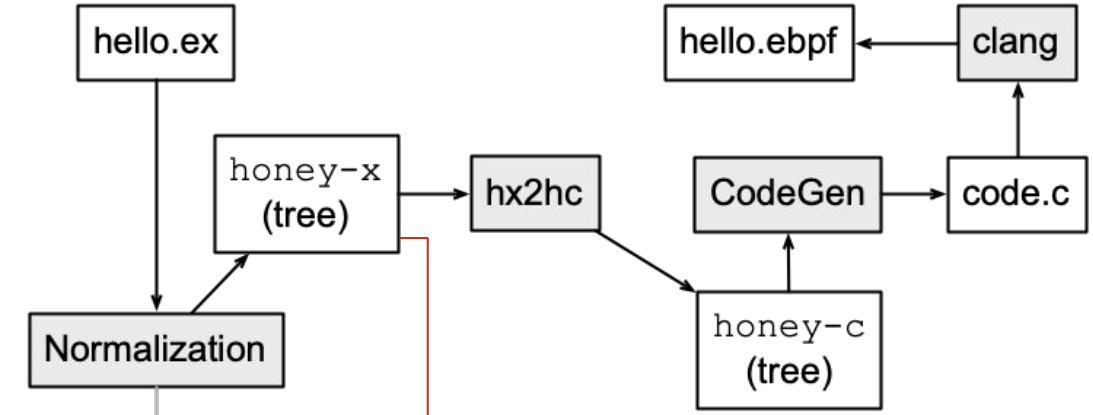
```
{:defmodule, [context: Elixir, import: Kernel], [
  {:__aliases__, [alias: false], [:Example]},
  [
  do: {:def, [context: Elixir, import: Kernel], [
    {:inc, [context: Elixir], [{:x, [], nil}]},
    [do: {:+, [context: Elixir, import: Kernel], [{:x, [], nil}, 1]}]
  ]}
  ]
]}
```

How To?


hello.ex
hello.ebpf
clang
honey-x
(tree)
hx2hc
CodeGen
code.c
Normalization
honey-c
(tree)
Section 3.2
• expansion of recursive calls
• full inlining

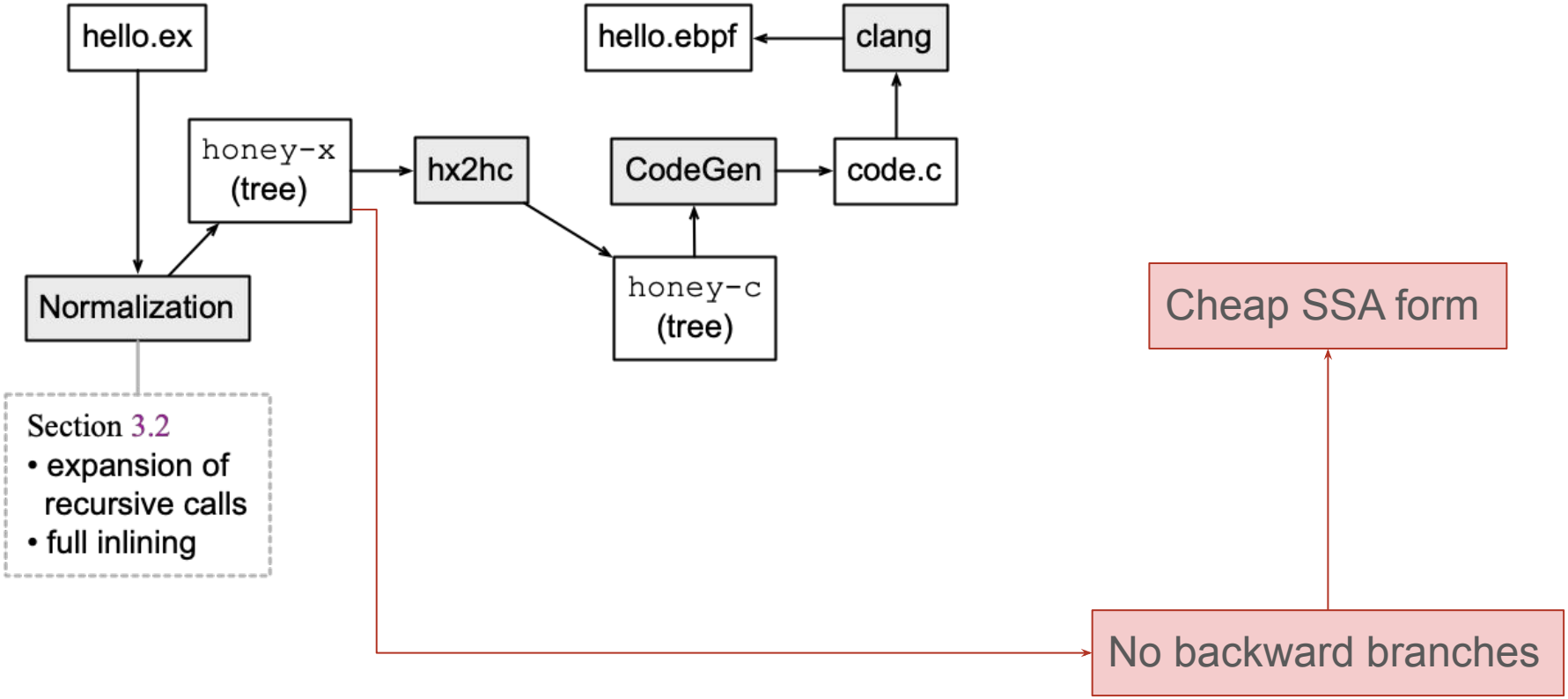
How To?

DCC

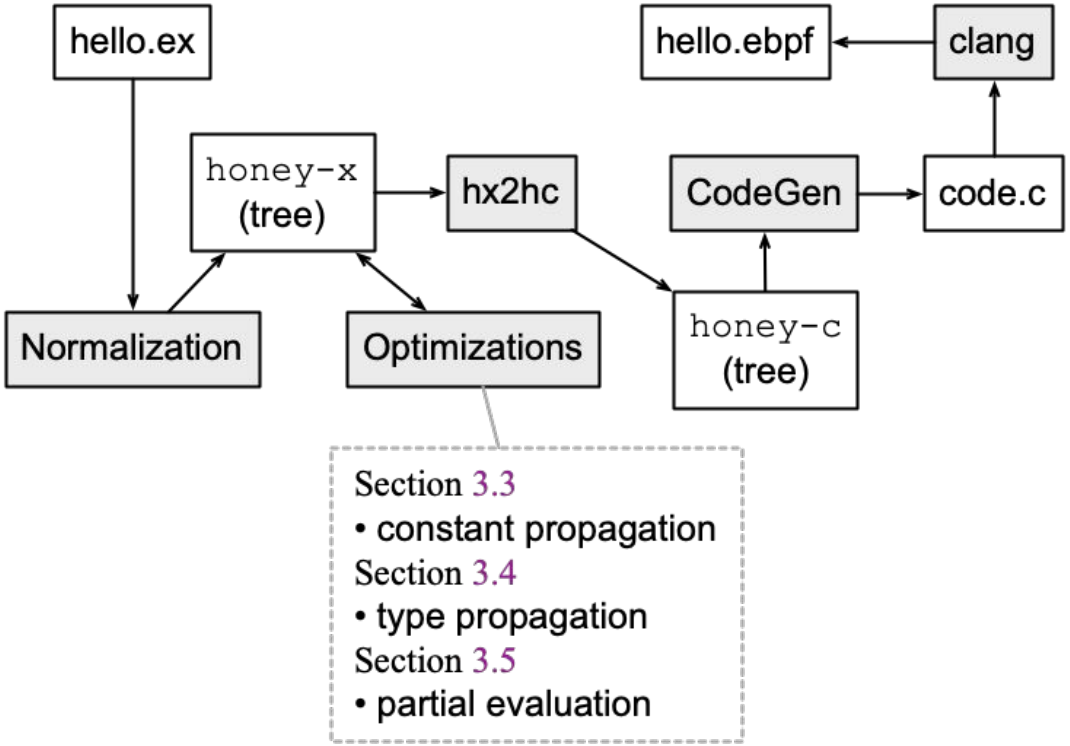


No backward branches

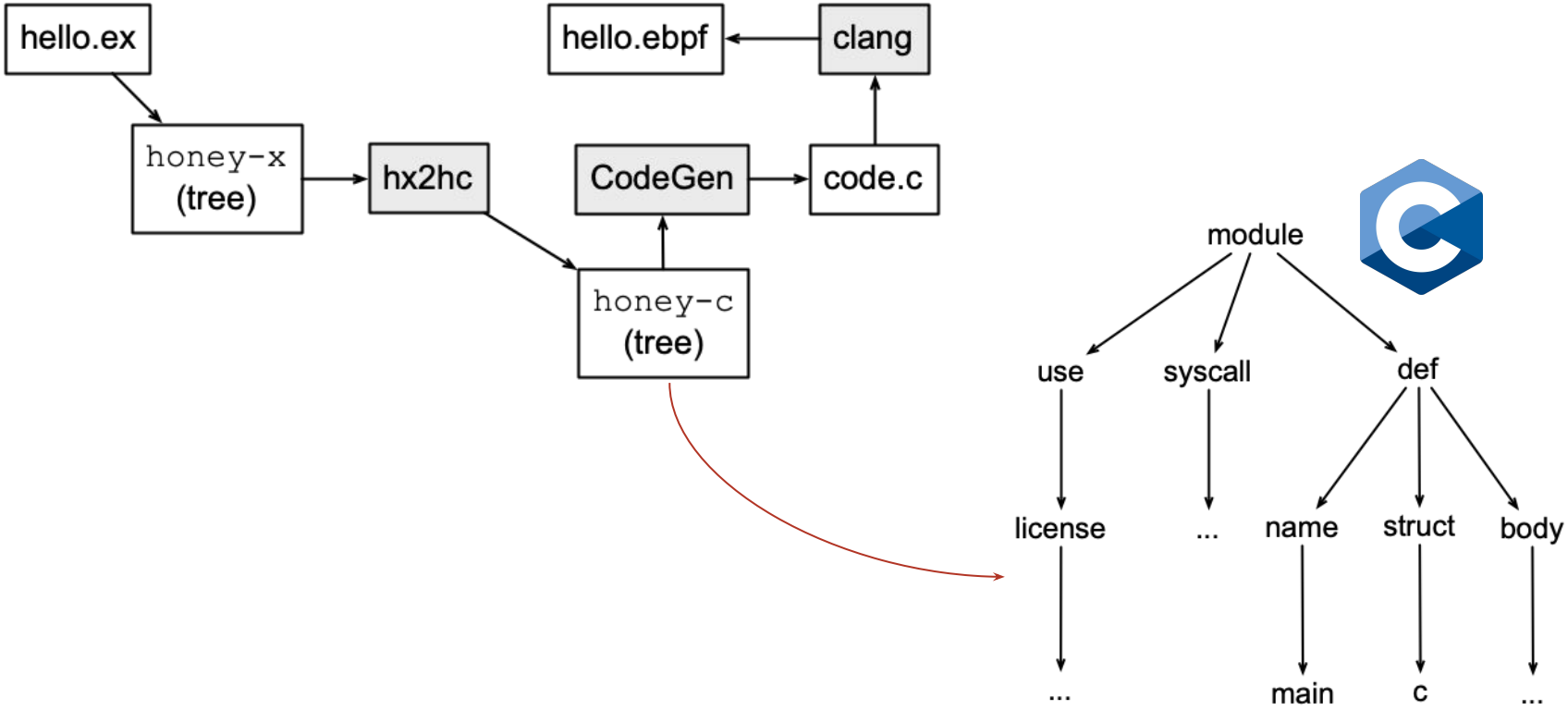
How To?


hello.ex
hello.ebpf
clang
honey-x
(tree)
hx2hc
CodeGen
code.c
Normalization
honey-c
(tree)
Section 3.2
• expansion of recursive calls
• full inlining
Cheap SSA form
No backward branches

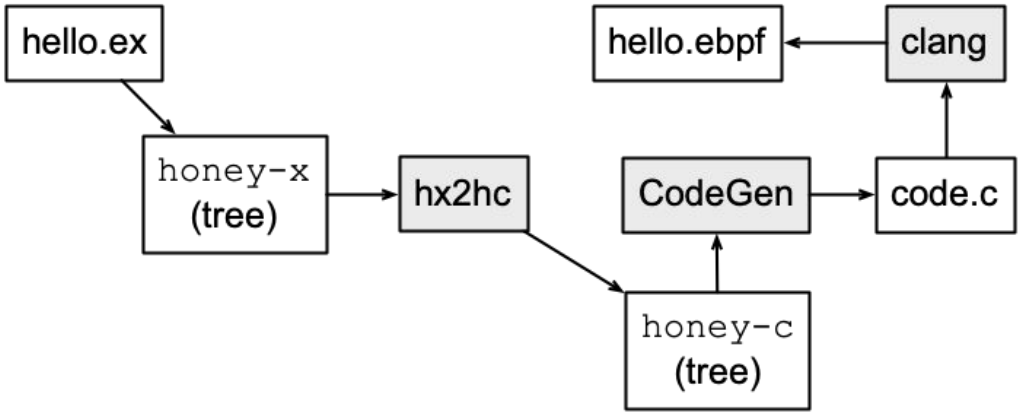
How To?

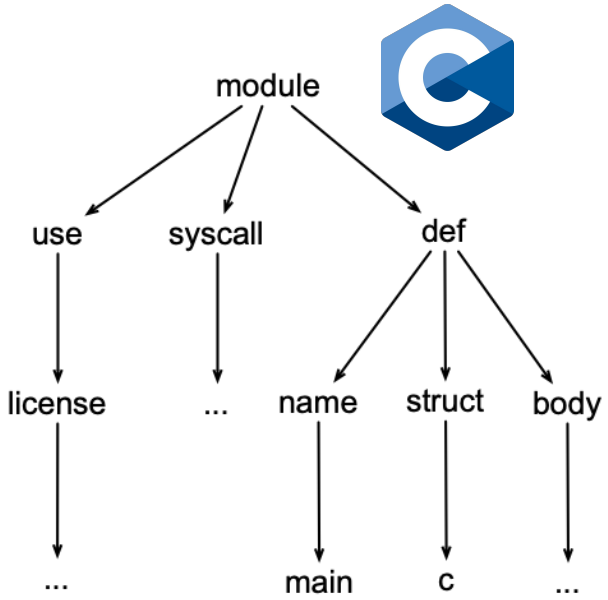

hello.ex
hello.ebpf
clang
honey-x
(tree)
hx2hc
CodeGen
code.c
Normalization
Optimizations
honey-c
(tree)
Section 3.3
• constant propagation
Section 3.4
• type propagation
Section 3.5
• partial evaluation

How To?


hello.ex
honey-x
(tree)
hx2hc
honey-c
(tree)
CodeGen
code.c
hello.ebpf
clang
module
use
syscall
def
license
...
name
struct
body
...
main
c
...

How To?

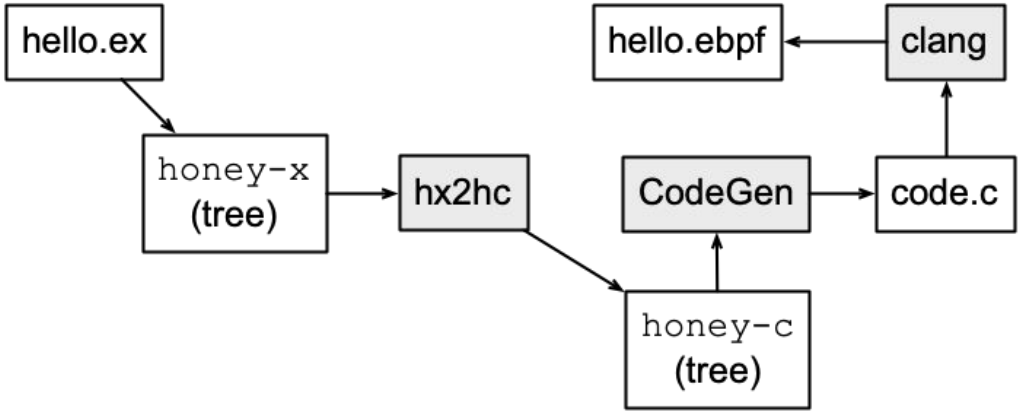


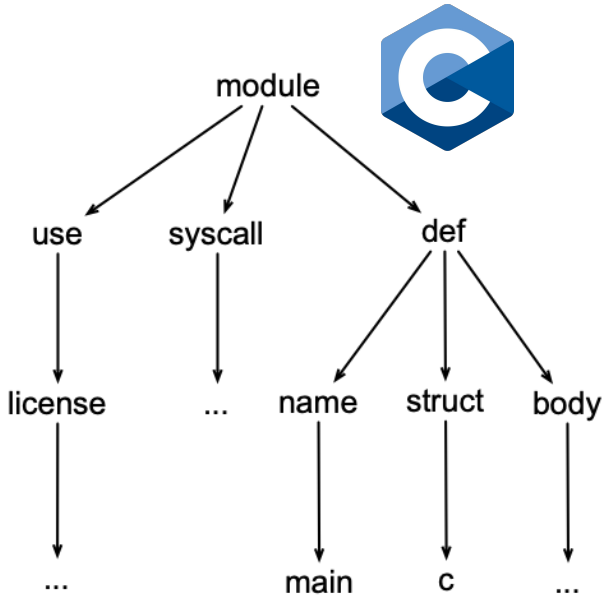


Meta Information

1. (Some) **Types**
2. (All) **Size**

How To?

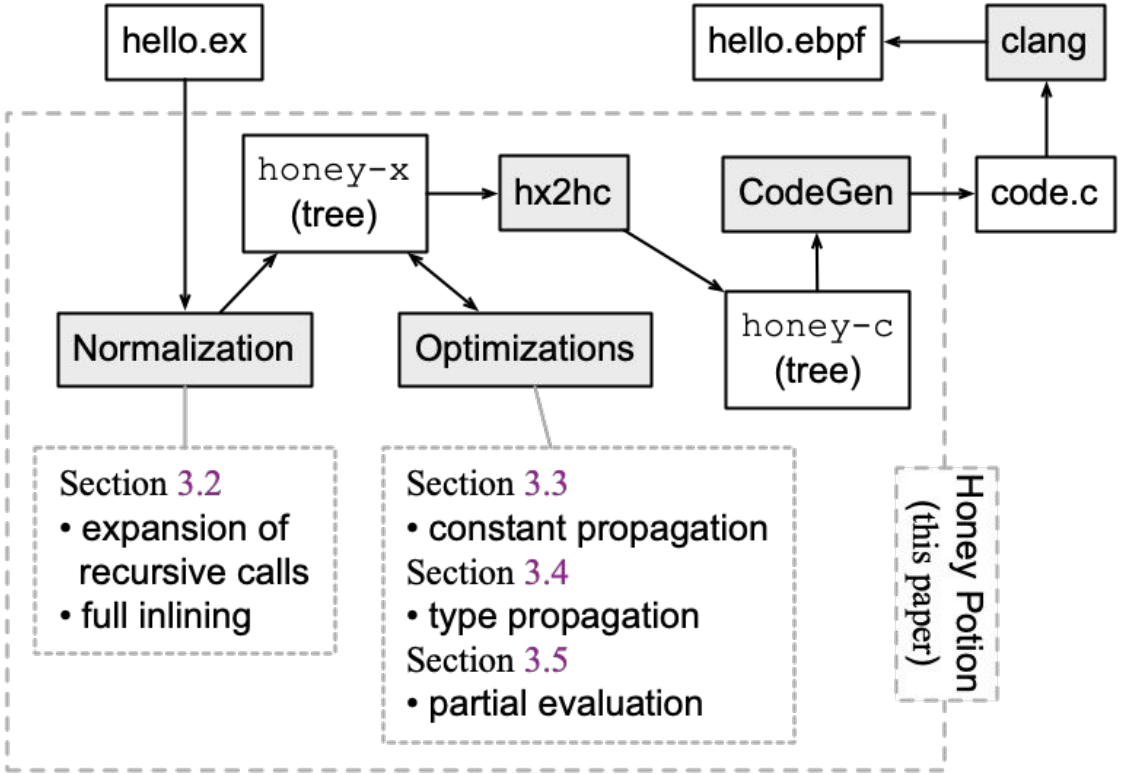




Meta Information

1. (Some) **Types**
2. (All) **Size** → Bound checks

How To?


hello.ex
hello.ebpf
clang
honey-x
(tree)
hx2hc
CodeGen
code.c
Normalization
Optimizations
honey-c
(tree)
Section 3.2
• expansion of recursive calls
• full inlining
Section 3.3
• constant propagation
Section 3.4
• type propagation
Section 3.5
• partial evaluation
Honey Potion
(this paper)

What about Recursion?

DCC

```
01 def sum(a, b) do
02   if b == 0 do
03     a
04   else
05     sum(a + 1, b - 1)
06   end
07 end
```

What about Recursion?

DCC

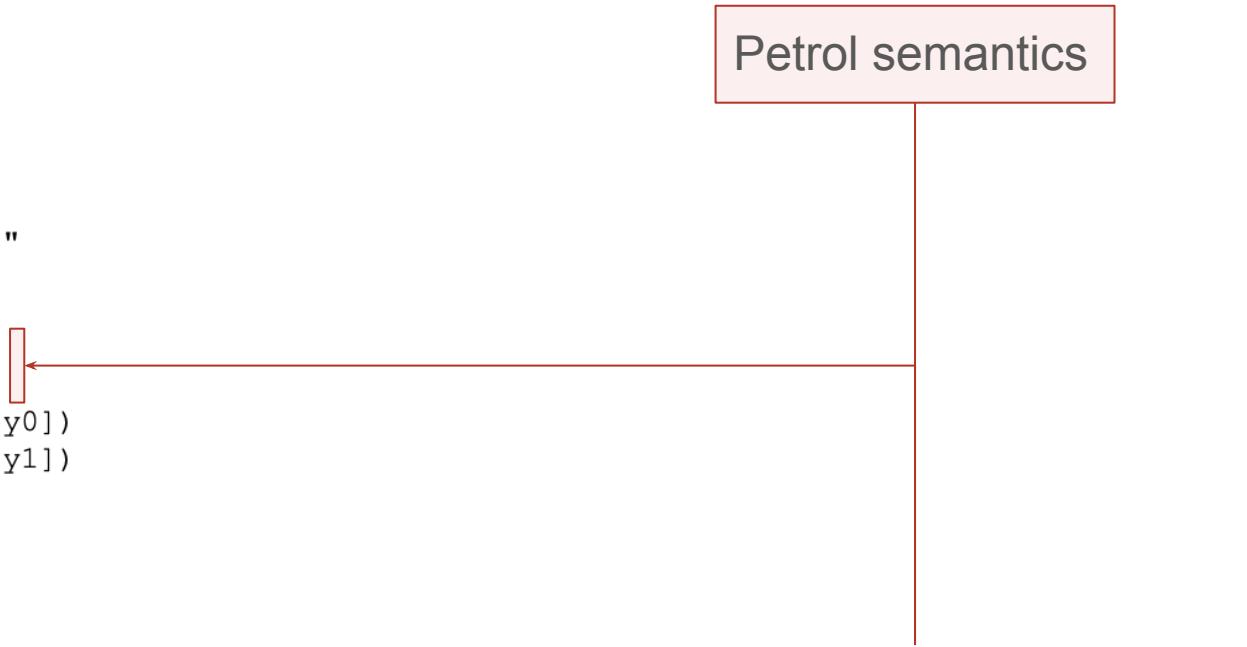
```
01 def sum(a, b) do
02   if b == 0 do
03     a
04   else
05     sum(a + 1, b - 1)
06   end
07 end
08
09 @sec ".../sys_enter_kill"
10 def main(ctx) do
11   x = 100
12   y0 = fuel 3, sum(x, 5)
13   y1 = fuel 2, sum(x, 4)
14   bpf_printk(["y0: %d", y0])
15   bpf_printk(["y1: %d", y1])
16 end
```

Programmer must specify depth

What about Recursion?

DCC

```
01 def sum(a, b) do
02   if b == 0 do
03     a
04   else
05     sum(a + 1, b - 1)
06   end
07 end
08
09 @sec ".../sys_enter_kill"
10 def main(ctx) do
11   x = 100
12   y0 = fuel 3, sum(x, 5)
13   y1 = fuel 2, sum(x, 4)
14   bpf_printk(["y0: %d", y0])
15   bpf_printk(["y1: %d", y1])
16 end
```



Full Inline Expansion

DCC

```
01 def sum(a, b) do
02   if b == 0 do
03     a
04   else
05     sum(a + 1, b - 1)
06   end
07 end
08
09 @sec ".../sys_enter_kill"
10 def main(ctx) do
11   x = 100
12   y0 = fuel 3, sum(x, 5)
13   y1 = fuel 2, sum(x, 4)
14   bpf_printk(["y0: %d", y0])
15   bpf_printk(["y1: %d", y1])
16 end
```

```
01 def sum(a0, b0) do
02   if b0 == 0 do
03     a0
04   else
05     a1 = a0 + 1, b1 = b0 - 1
06     if b1 == 0 do
07       a1
08     else
09       a2 = a1 + 1, b2 = b1 - 1
10       if b2 == 0 do
11         a2
12       else
13         a3 = a2 + 1, b3 = b2 - 1
14         if b3 == 0 do
15           a3
16         else
17           raise FORCE_END
18         end
19       end
20     end
21   end
22 end
```

Constant Propagation

DCC

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = const
06     if (x ==gen 1) do 1
07     else
08       x1 = x -gen 1
09       x * if (x1 ==gen 1) do 1
10       else
11         x2 = x1 -gen 1
12         x1 *gen if (x2==gen 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==gen 1) do 1
19     else
20       y1 = y -gen 1
21       y *gen if (y1 ==gen 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +gen b
26 end
```

Constant Propagation

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = const
06     if (x ==gen 1) do 1
07     else
08       x1 = x -gen 1
09       x * if (x1 ==gen 1) do 1
10       else
11         x2 = x1 -gen 1
12         x1 *gen if (x2==gen 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==gen 1) do 1
19     else
20       y1 = y -gen 1
21       y *gen if (y1 ==gen 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +gen b
26 end
```

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==gen 1) do 1
07     else
08       2 = 3 -gen 1
09       3 * if (2 ==gen 1) do 1
10       else
11         x2 = 2 -gen 1
12         2 *gen if (1==gen 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==gen 1) do 1
19     else
20       y1 = y -gen 1
21       y * if (y1 ==gen 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +gen b
26 end
```

Type Propagation

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = const
06     if (x ==gen 1) do 1
07     else
08       x1 = x -gen 1
09       x * if (x1 ==gen 1) do 1
10       else
11         x2 = x1 -gen 1
12         x1 *gen if (x2==gen 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==gen 1) do 1
19     else
20       y1 = y -gen 1
21       y *gen if (y1 ==gen 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +gen b
26 end
```

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==gen 1) do 1
07     else
08       2 = 3 -gen 1
09       3 * if (2 ==gen 1) do 1
10       else
11         x2 = 2 -gen 1
12         2 *gen if (1==gen 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==gen 1) do 1
19     else
20       y1 = y -gen 1
21       y * if (y1 ==gen 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +gen b
26 end
```

```
struct __sk_buff {
   __u32 len;                // Length of the packet
   __u32 pkt_type;           // Packet type
   __u32 mark;               // Generic packet mark
   __u32 queue_mapping;      // Queue mapping
   __u32 protocol;           // EtherType (IPv4,
IPv6)
   __u32 pid;                // Process ID
   ...
   __u32 remote_ip6[4];      // Source/destination
IPv6
   __u32 local_ip6[4];       // Destination/source
IPv6
   __u32 remote_port;        // Transport-layer port
   __u32 data_meta;          // Pointer to metadata
   __u64 flow_keys;          // Pointer to flow keys
};
```

Type Propagation

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = const
06     if (x ==gen 1) do 1
07     else
08       x1 = x -gen 1
09       x * if (x1 ==gen 1) do 1
10       else
11         x2 = x1 -gen 1
12         x1 *gen if (x2==gen 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==gen 1) do 1
19     else
20       y1 = y -gen 1
21       y *gen if (y1 ==gen 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +gen b
26 end
```

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==gen 1) do 1
07     else
08       2 = 3 -gen 1
09       3 * if (2 ==gen 1) do 1
10       else
11         x2 = 2 -gen 1
12         2 *gen if (1==gen 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==gen 1) do 1
19     else
20       y1 = y -gen 1
21       y * if (y1 ==gen 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +gen b
26 end
```

__u32 pid

```
struct __sk_buff {
    __u32 len;                  // Length of the packet
    __u32 pkt_type;             // Packet type
    __u32 mark;                 // Generic packet mark
    __u32 queue_mapping;        // Queue mapping
    __u32 protocol;             // EtherType (IPv4,
IPv6)
    __u32 pid;                  // Process ID
    ...
    __u32 remote_ip6[4];        // Source/destination
IPv6
    __u32 local_ip6[4];         // Destination/source
IPv6
    __u32 remote_port;          // Transport-layer port
    __u32 data_meta;            // Pointer to metadata
    __u64 flow_keys;            // Pointer to flow keys
};
```

Type Propagation

DCC

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = const
06     if (x ==gen 1) do 1
07     else
08       x1 = x -gen 1
09       x * if (x1 ==gen 1) do 1
10       else
11         x2 = x1 -gen 1
12         x1 *gen if (x2==gen 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==gen 1) do 1
19     else
20       y1 = y -gen 1
21       y *gen if (y1 ==gen 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +gen b
26 end
```

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==gen 1) do 1
07     else
08       2 = 3 -gen 1
09       3 * if (2 ==gen 1) do 1
10       else
11         x2 = 2 -gen 1
12         2 *gen if (1==gen 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==gen 1) do 1
19     else
20       y1 = y -gen 1
21       y * if (y1 ==gen 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +gen b
26 end
```

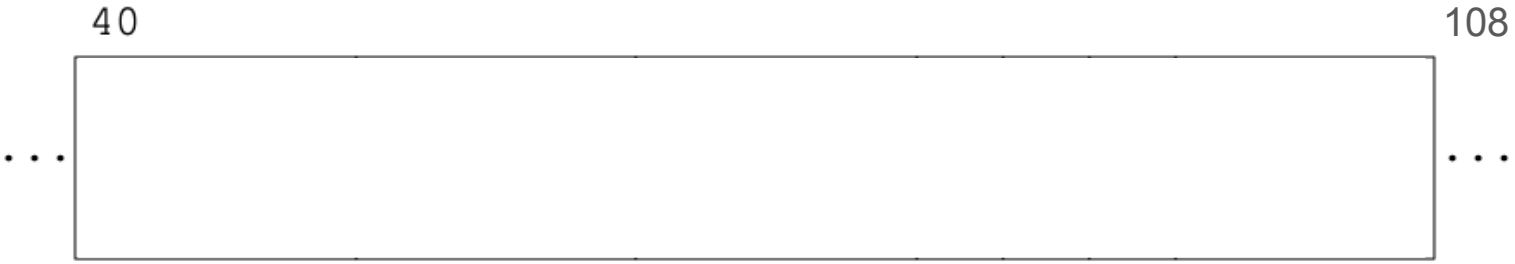
```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==int 1) do 1
07     else
08       2 = 3 -int 1
09       3 * if (2 ==int 1) do 1
10       else
11         x2 = 2 -int 1
12         2*int if (1==int 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==int 1) do 1
19     else
20       y1 = y -int 1
21       y *int if (y1 ==int 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +int b
26 end
```

Type Propagation, e.g., t = {1, 2, 3}

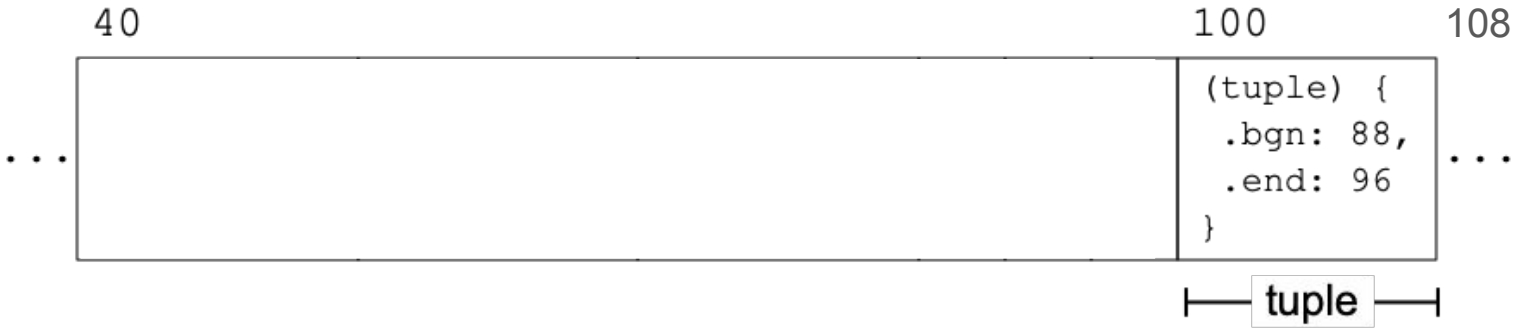
DCC

Type Propagation, e.g., t = {1, 2, 3}

DCC

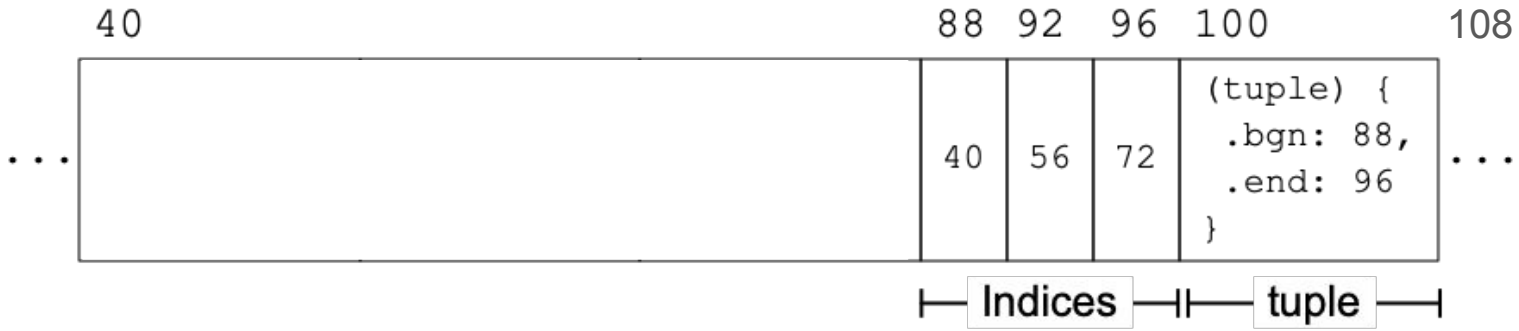


Type Propagation, e.g., t = {1, 2, 3}

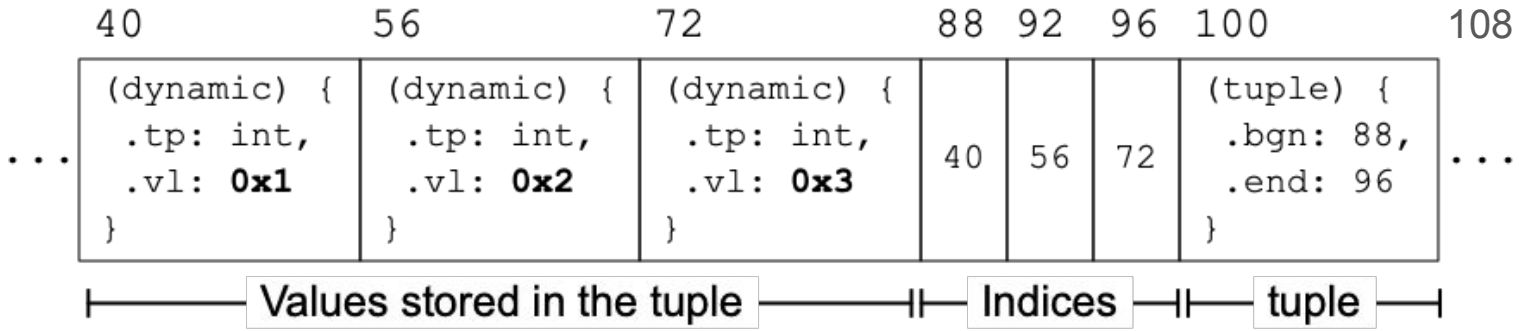


Every memory access
requires bound information

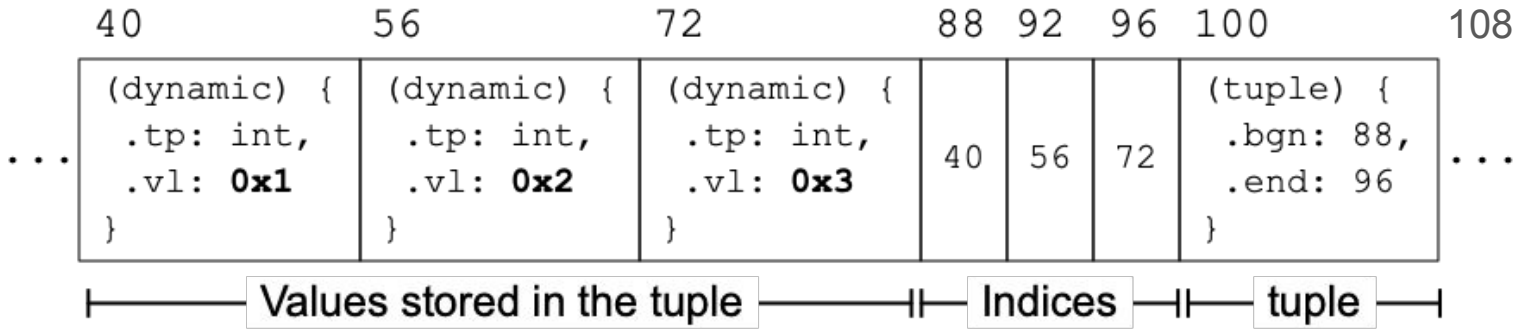
Type Propagation, e.g., t = {1, 2, 3}



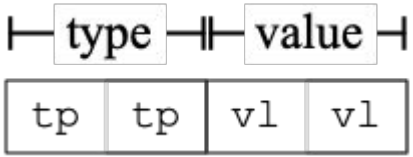
Type Propagation, e.g., t = {1, 2, 3}



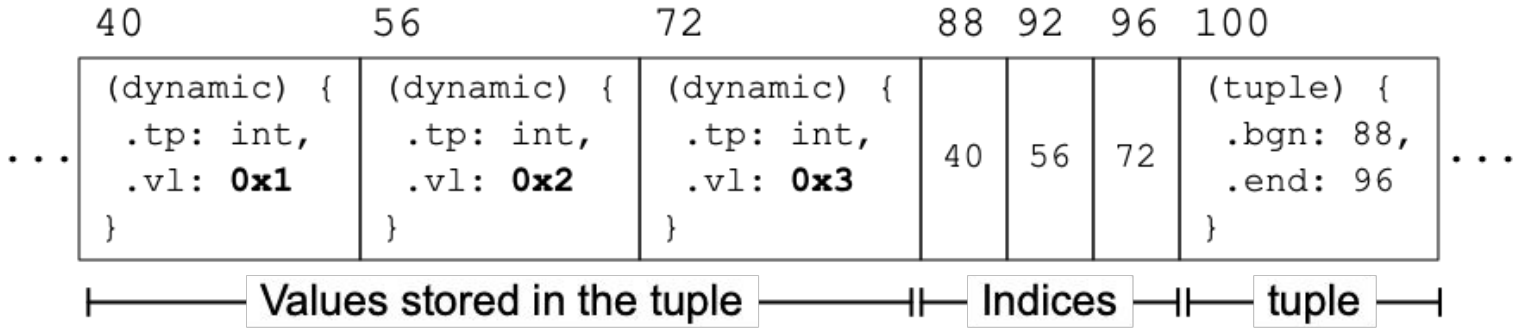
Type Propagation, e.g., t = {1, 2, 3}

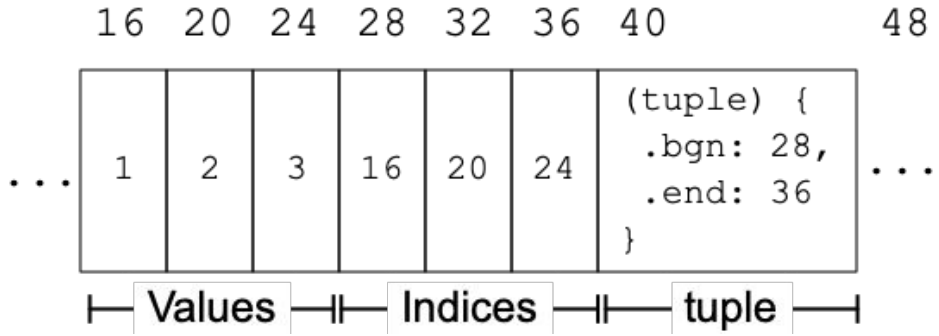


Generic values occupy
16 bytes: type + value



Type Propagation, e.g., t = {1, 2, 3}





Dead-Code Elimination

DCC

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==int 1) do 1
07     else
08       2 = 3 -int 1
09       3 * if (2 ==int 1) do 1
10       else
11         x2 = 2 -int 1
12         2*int if (1==int 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==int 1) do 1
19     else
20       y1 = y -int 1
21       y *int if (y1 ==int 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +int b
26 end
```

Dead-Code Elimination

DCC

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==int 1) do 1
07     else
08       2 = 3 -int 1
09       3 * if (2 ==int 1) do 1
10       else
11         x2 = 2 -int 1
12         2*int if (1==int 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==int 1) do 1
19     else
20       y1 = y -int 1
21       y *int if (y1==int 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   6 +int b
26 end
```

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==int 1) do 1
07     else
08       2 = 3 -int 1
09       3 * if (2 ==int 1) do 1
10       else
11         x2 = 2 -int 1
12         2*int if (1==int 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==int 1) do 1
19     else
20       y1 = y -int 1
21       y *int if (y1 ==int 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +int b
26 end
```

Dead-Code Elimination

No backward branches
+
Full inlining
+
Constant propagation
+
Dead-code elimination
==
Partial evaluation

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==int 1) do 1
07     else
08       2 = 3 -int 1
09       3 * if (2 ==int 1) do 1
10       else
11         x2 = 2 -int 1
12         2*int if (1==int 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==int 1) do 1
19     else
20       y1 = y -int 1
21       y *int if (y1==int 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   6 +int b
26 end
```

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==int 1) do 1
07     else
08       2 = 3 -int 1
09       3 * if (2 ==int 1) do 1
10       else
11         x2 = 2 -int 1
12         2*int if (1==int 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==int 1) do 1
19     else
20       y1 = y -int 1
21       y *int if (y1 ==int 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +int b
26 end
```

Dead-Code Elimination

DCC

No backward branches
+
Full inlining
+
Constant propagation
+
Dead-code elimination
+
Type propagation
==
Full specialization

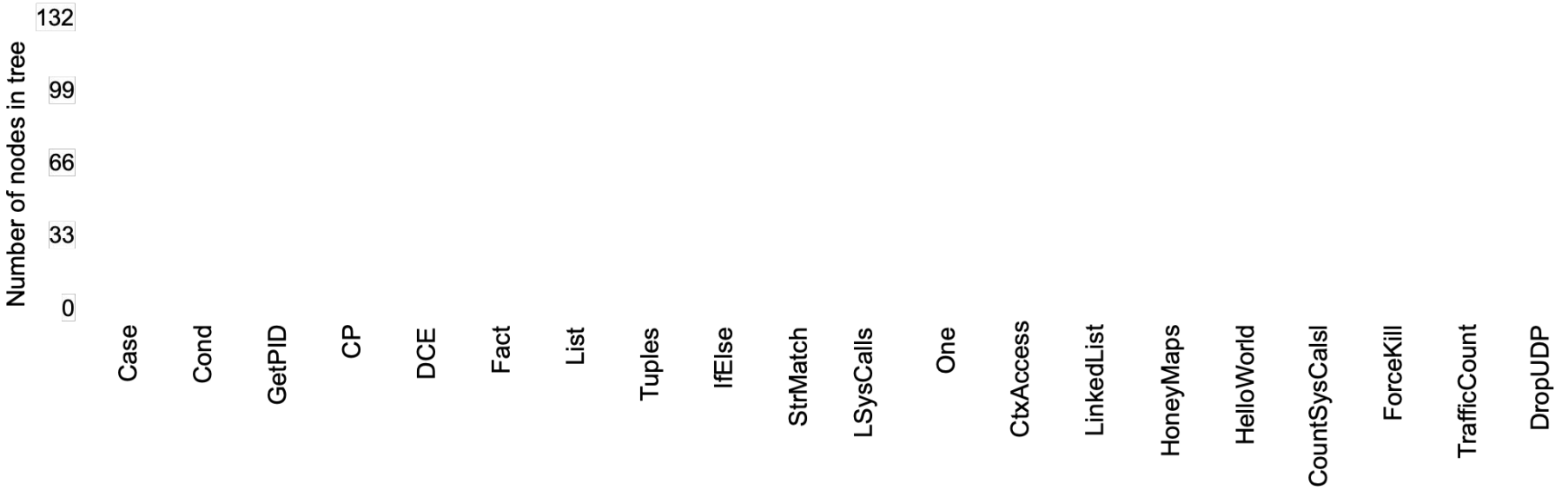
```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==int 1) do 1
07     else
08       2 = 3 -int 1
09       3 * if (2 ==int 1) do 1
10       else
11         x2 = 2 -int 1
12         2*int if (1==int 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==int 1) do 1
19     else
20       y1 = y -int 1
21       y *int if (y1==int 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   6 +int b
26 end
```

```
01 def main(ctx) do
02   const = 3
03   id = ctx.pid
04   a = (
05     x = 3
06     if (3 ==int 1) do 1
07     else
08       2 = 3 -int 1
09       3 * if (2 ==int 1) do 1
10       else
11         x2 = 2 -int 1
12         2*int if (1==int 1) do 1
13         else
14           raise "No more fuel"
15     end end end )
16   b = (
17     y = id
18     if (y ==int 1) do 1
19     else
20       y1 = y -int 1
21       y *int if (y1 ==int 1) do 1
22       else
23         raise "No more fuel."
24     end end )
25   a +int b
26 end
```

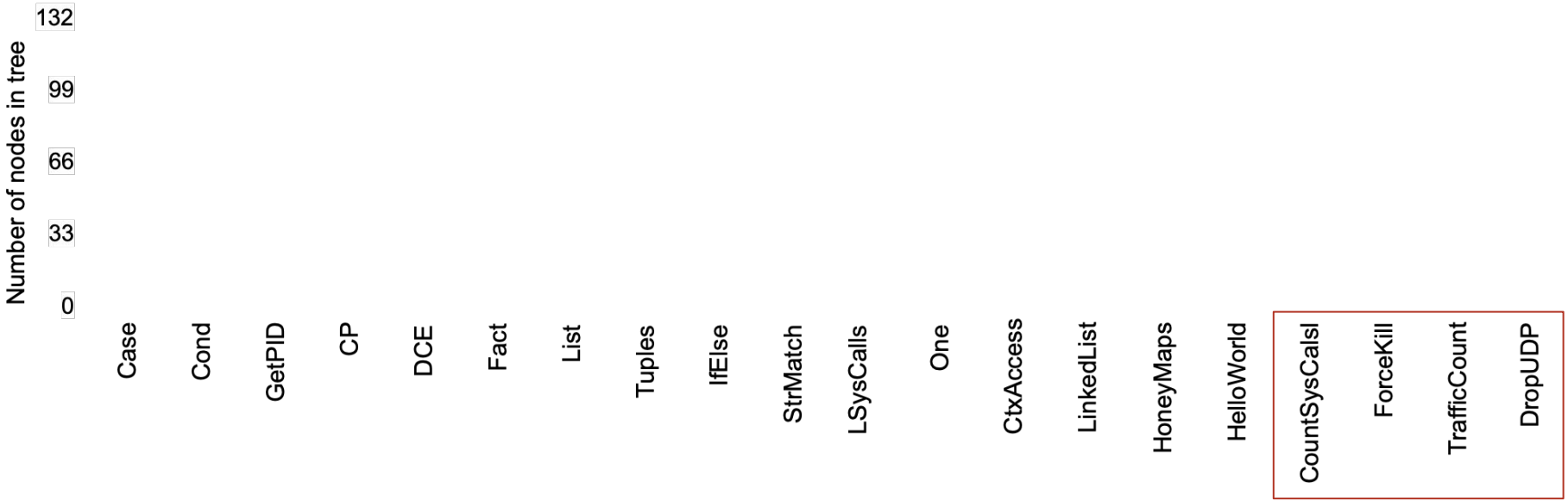
The Impact of Optimizations

The Impact of Optimizations

DCC



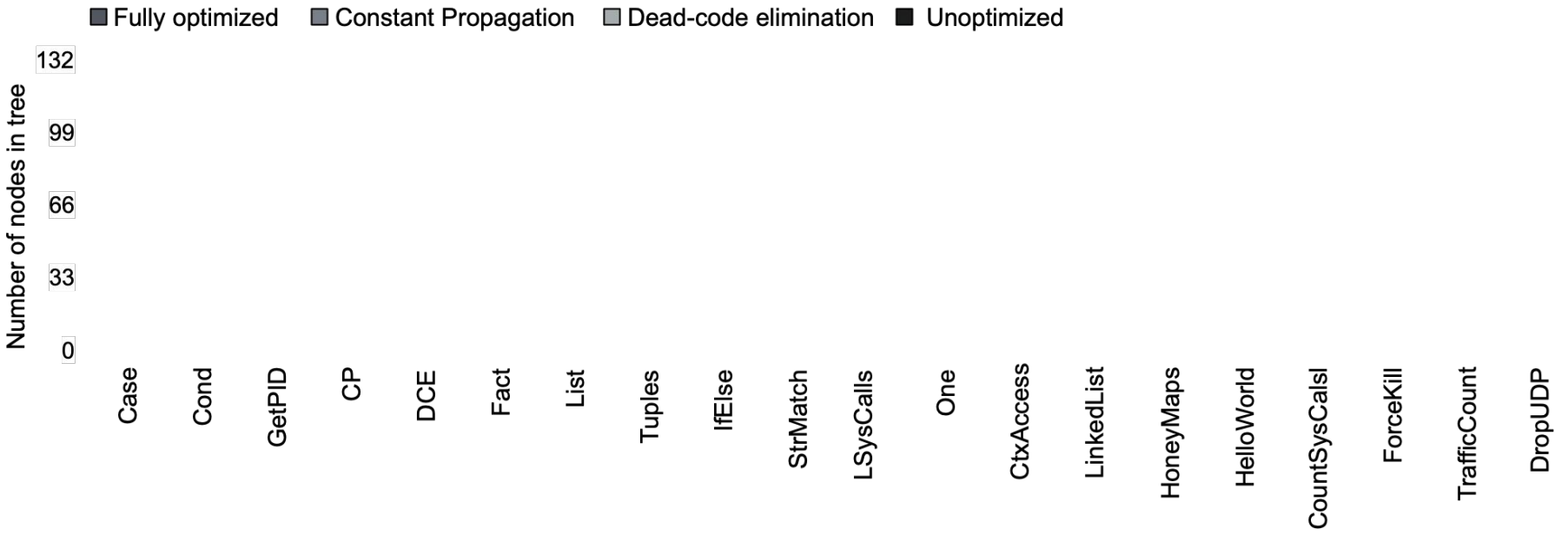
The Impact of Optimizations



https://ebpf.io/applications

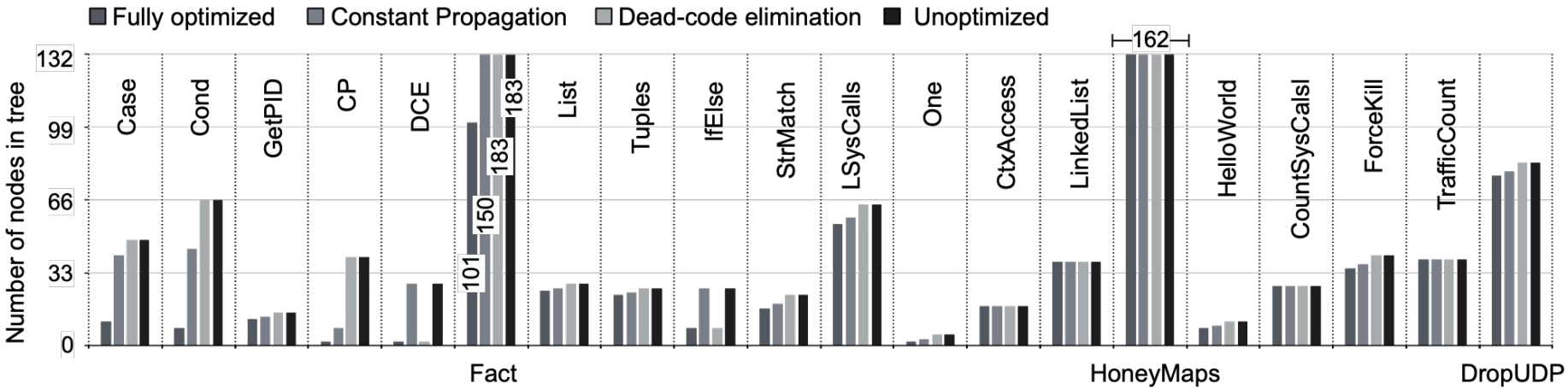
The Impact of Optimizations

DCC

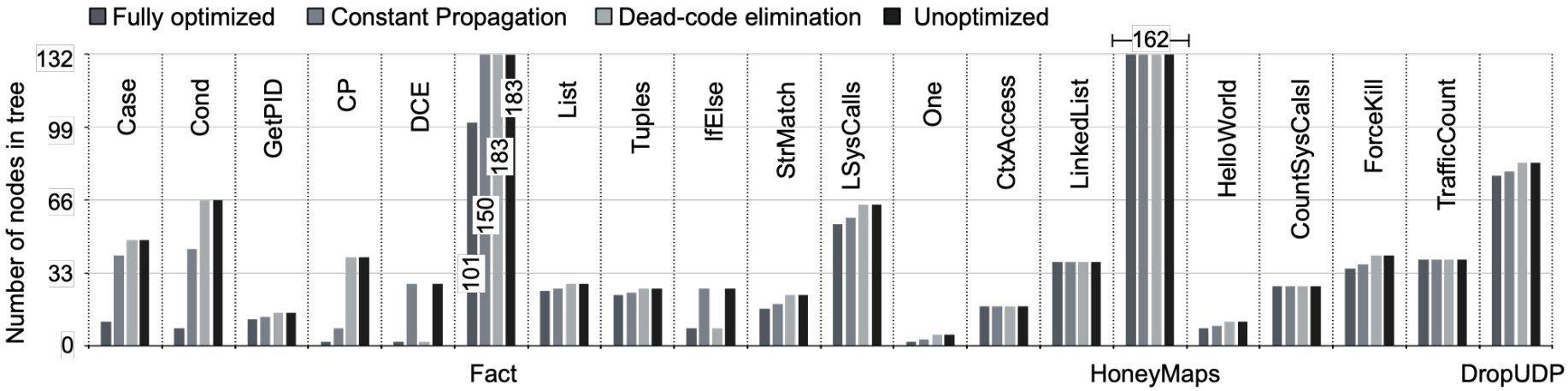


The Impact of Optimizations

DCC

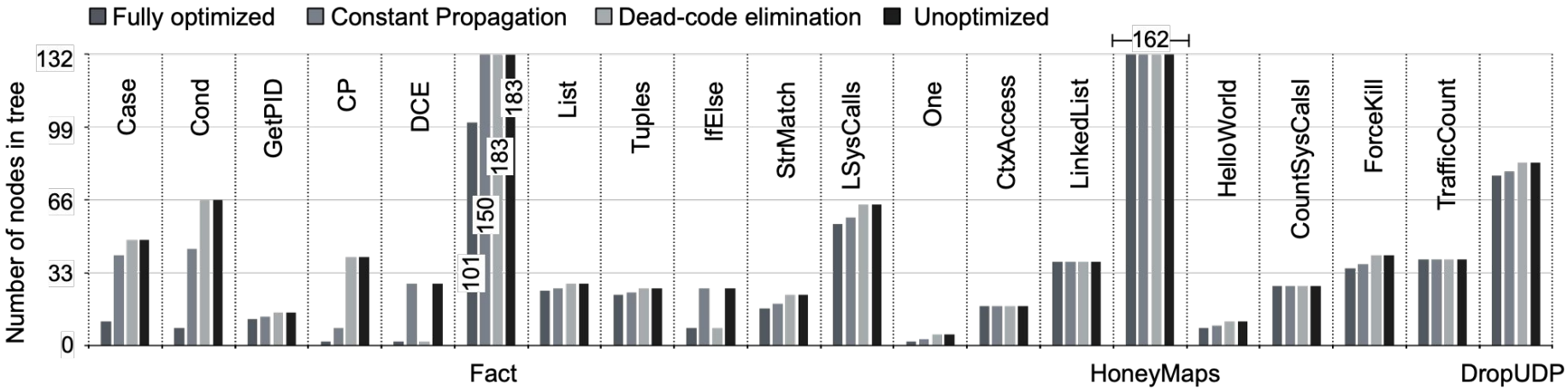


The Impact of Optimizations



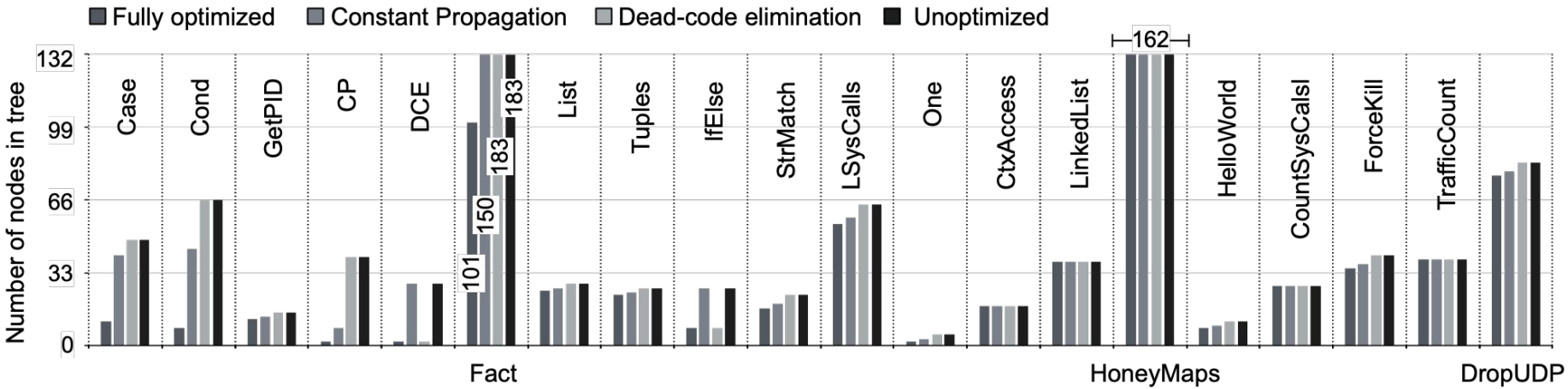
- Unoptimized: 971 nodes

The Impact of Optimizations



- Unoptimized: 971 nodes
- Dead-code elimination (DCE): 927

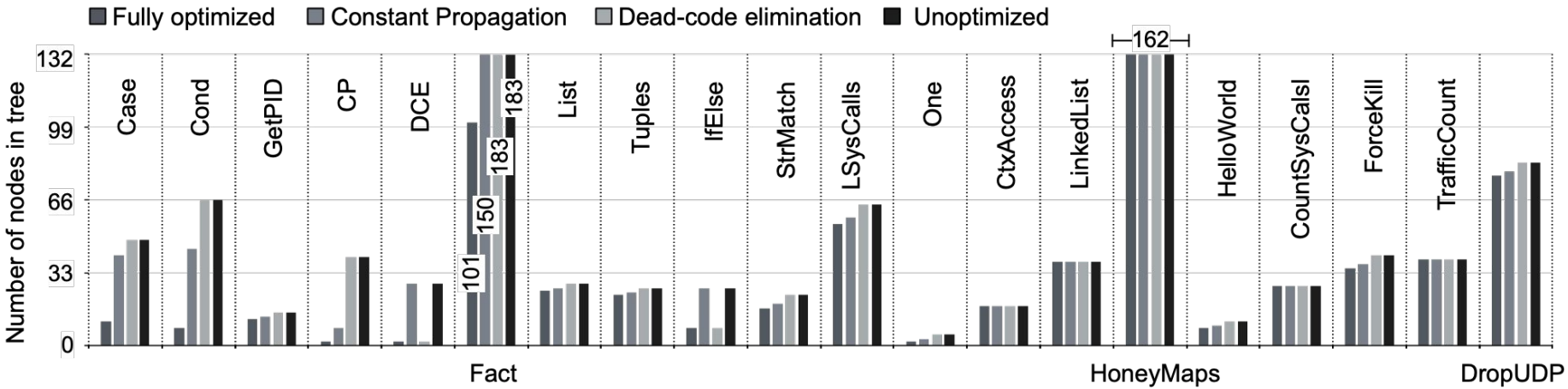
The Impact of Optimizations



- Unoptimized: 971 nodes
- Dead-code elimination (DCE): 927
- Constant-propagation (CP): 849

The Impact of Optimizations

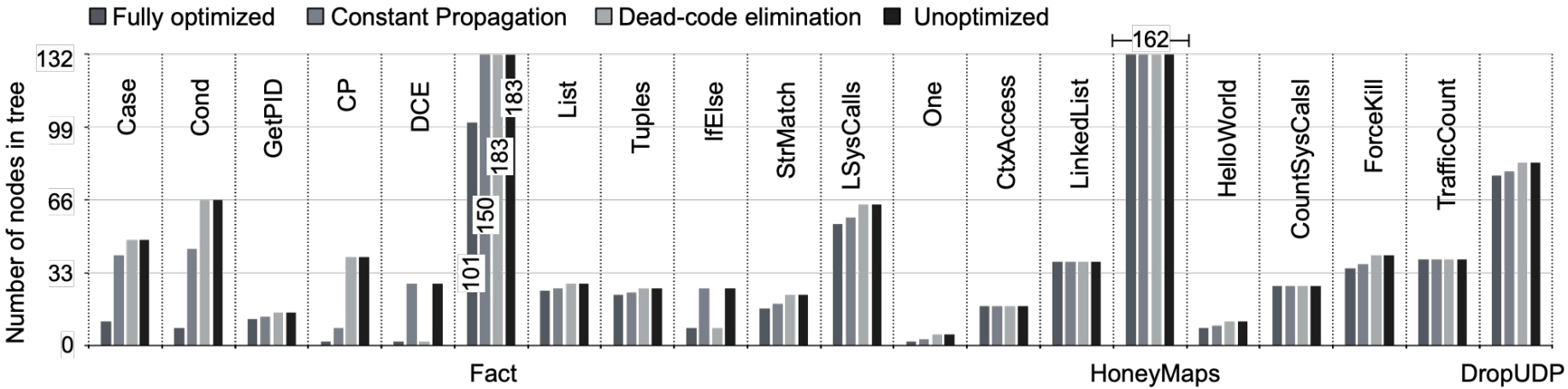
DCC



- Unoptimized: 971 nodes
- Dead-code elimination (DCE): 927
- Constant-propagation (CP): 849
- DCE + CP: 670

The Impact of Optimizations

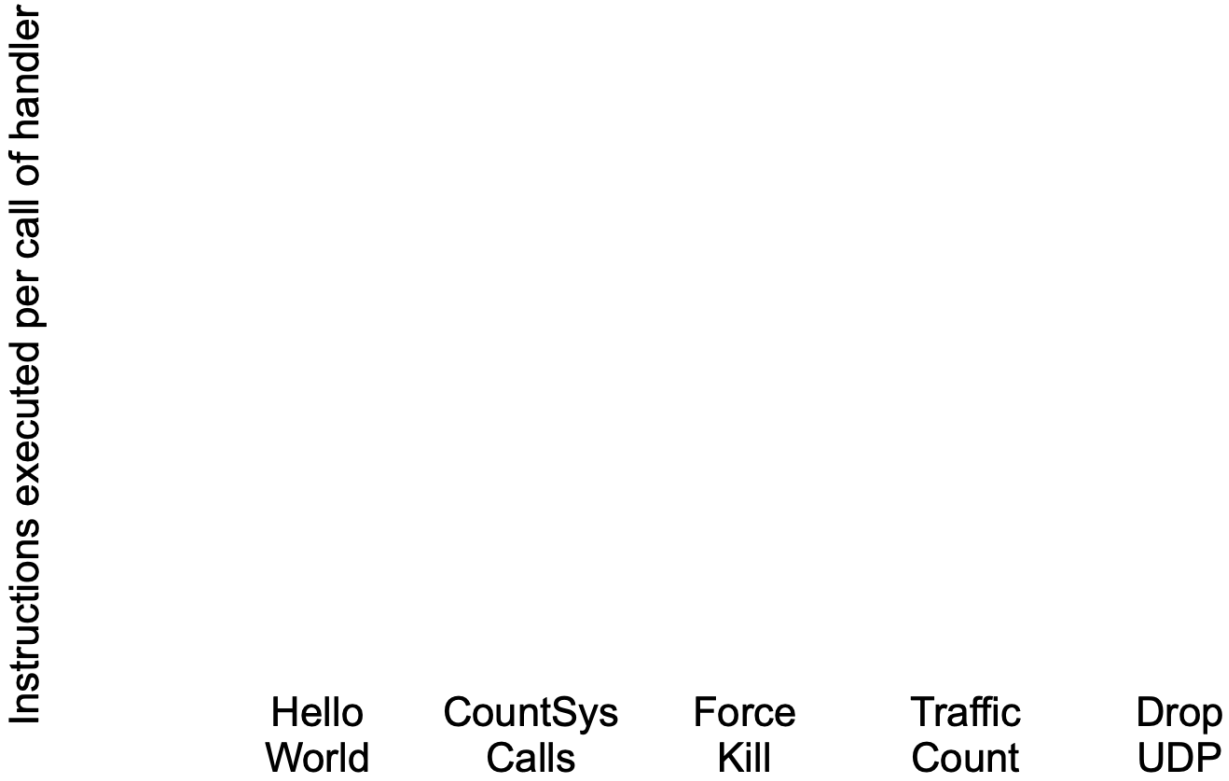
DCC



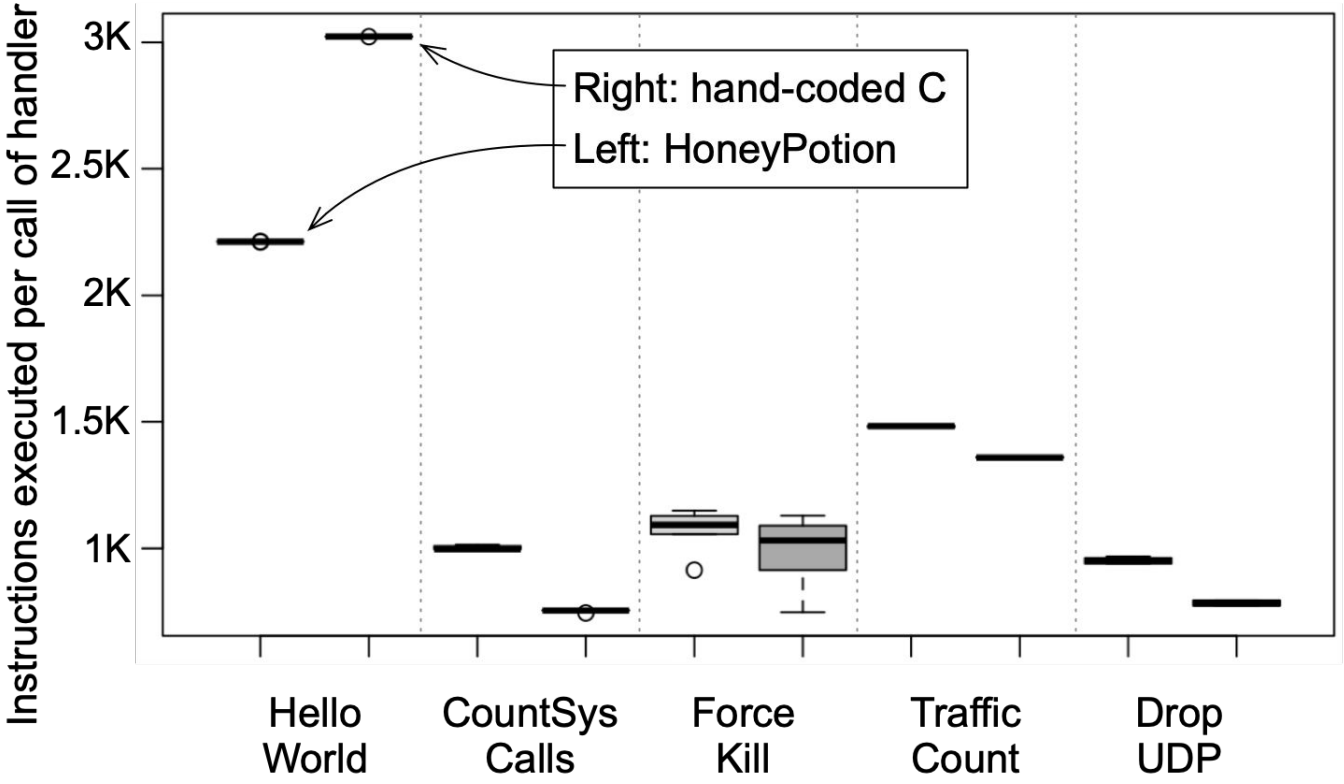
- Unoptimized: 971 nodes
- Dead-code elimination (DCE): 927
- Constant-propagation (CP): 849
- DCE + CP: 670

Type propagation cannot be disabled (stack size = 512 bytes)

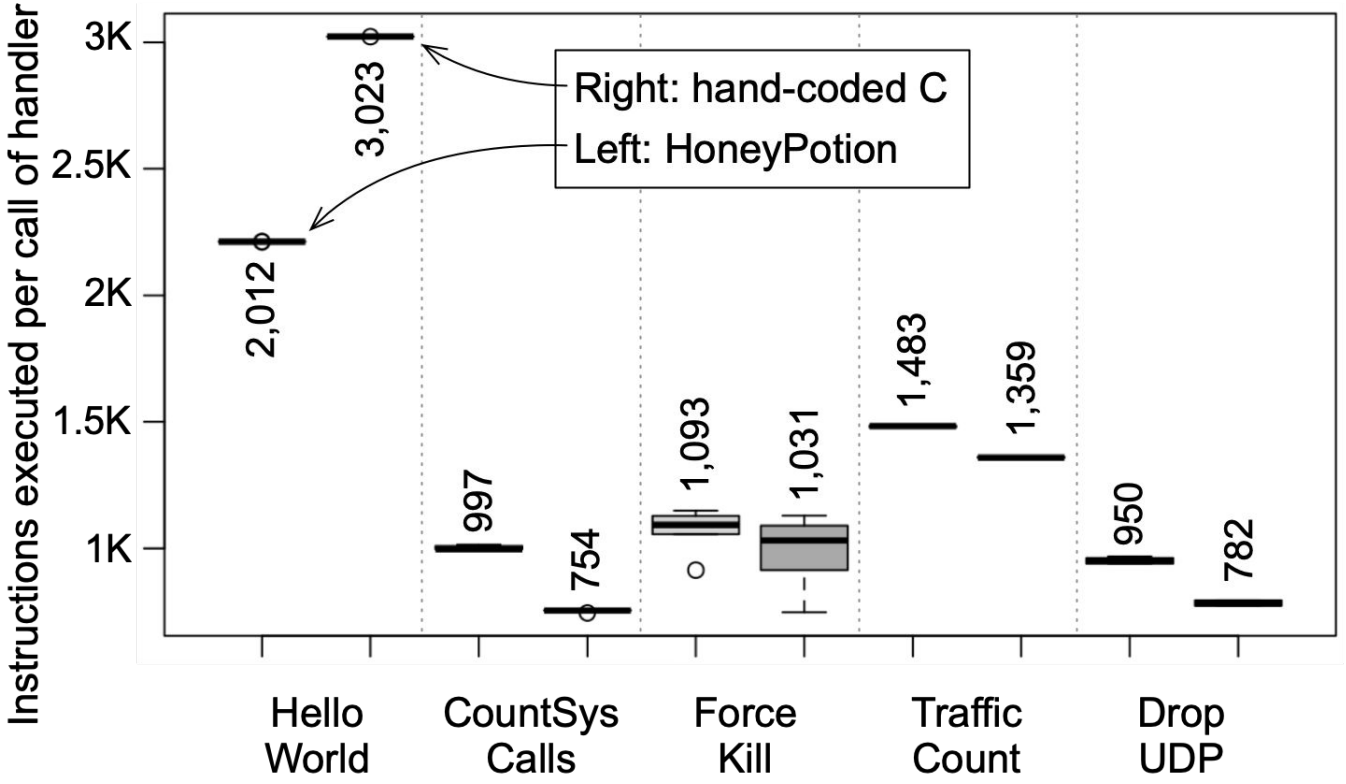
Are these programs ok?



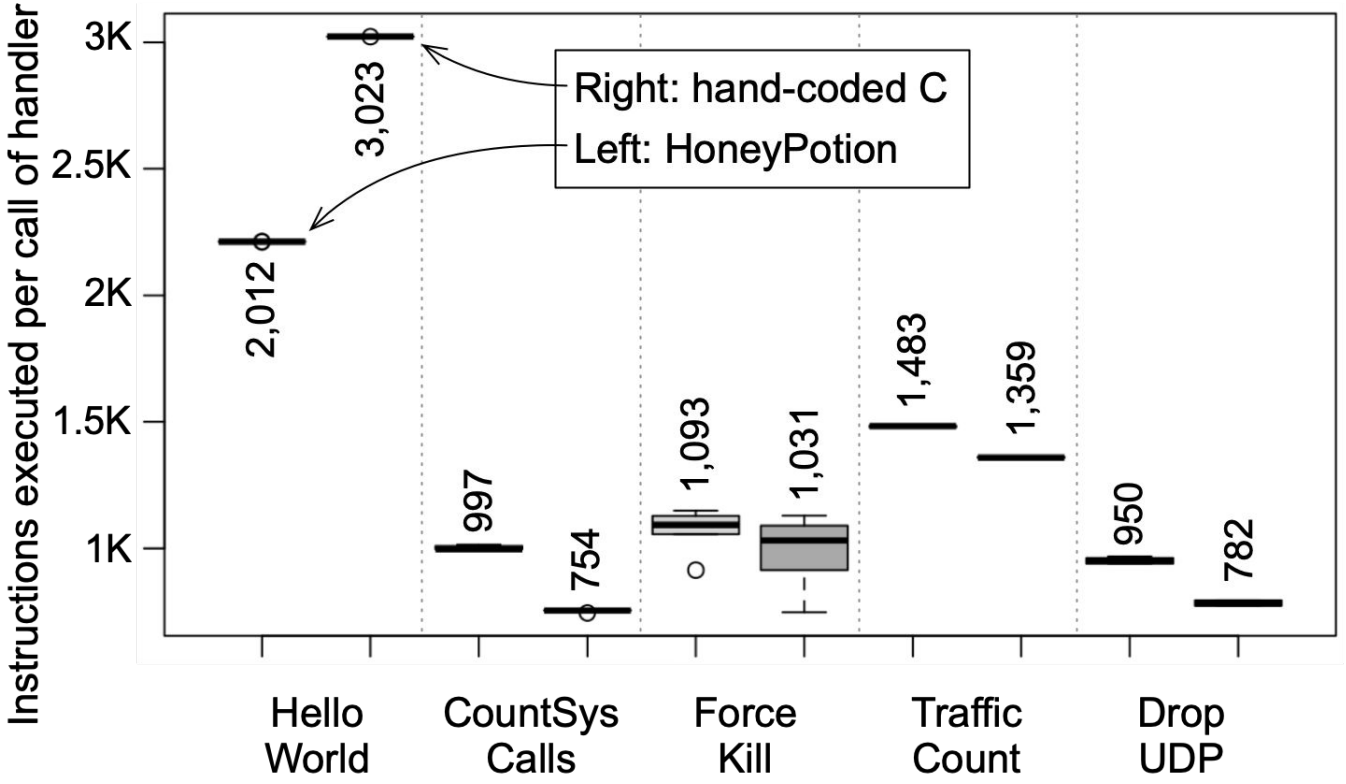
Are these programs ok?



Are these programs ok?



Are these programs ok?



1 million instructions
+
No backward branch
==
Fast execution

Are these programs ok?

DCC

- Performance not too far from C

Are these programs ok?

DCC

- Performance not too far from C
- Programs twice as short

Are these programs ok?

DCC

- Performance not too far from C
- Programs twice as short
- But binaries twice as larger...

Are these programs ok?

DCC

- Performance not too far from C
- Programs twice as short
- But binaries twice as larger...
- Verified code by default

Are these programs ok?

DCC

- Performance not too far from C
- Programs twice as short
- But binaries twice as larger...
- Verified code by default
- Very nice compilation scenario!

Are these programs ok?

- Performance not too far from C
- Programs twice as short
- But binaries twice as larger...
- Verified code by default
- Very nice compilation scenario!

If you are optimizing a Turing Incomplete language, enjoy it!

References & Contact

DCC

Fernando Pereira

fernando@dcc.ufmg.br

http://lac.dcc.ufmg.br

Kael Soares Augusto, Vinícius Pacheco, Marcos A. Vieira, Rodrigo Geraldo Ribeiro and Fernando Magno Quintão Pereira. **Honey Potion: an eBPF Backend for Elixir**. CGO 2025

References & Contact

DCC

Honey Potion - Writing eBPF with Elixir

Contribute on github

https://github.com/lac-dcc/honey-potion

Kael Soares Augusto, Vinícius Pacheco, Marcos A. Vieira, Rodrigo Geraldo Ribeiro and Fernando Magno Quintão Pereira. **Honey Potion: an eBPF Backend for Elixir**. CGO 2025