StkTokens: Enforcing Well-Bracketed Control Flow and Stack Encapsulation Using Linear Capabilities

Lau Skorstengaard[1] Dominique Devriese[2] Lars Birkedal[1]

[1]Aarhus University

[2]Vrije Universiteit Brussel

POPL, January 19, 2019

Overview

StkTokens-paper in the big picture

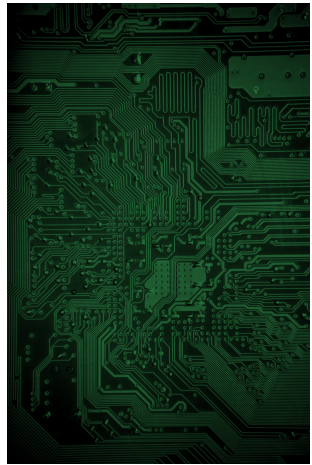
Defining well-bracketed control flow and local state encapsulation

Overview

StkTokens-paper in the big picture

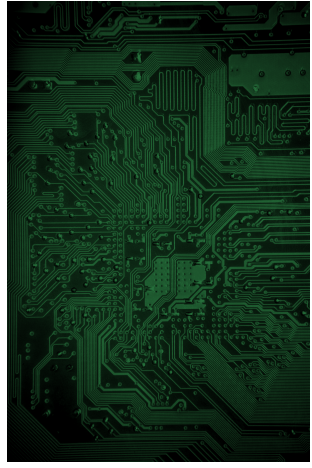
Defining well-bracketed control flow and local state encapsulation

Abstractions all the way down



Abstractions all the way down

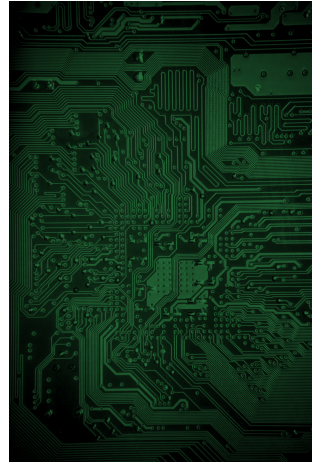
```
main:
  .cfi_startproc
# BB#0:
  pushq %rbp
.Ltmp0:
  .cfi\_def\_cfa\_offset 16
.Ltmp1:
  .cfi\_offset %rbp, -16
  movq  %rsp, %rbp
.Ltmp2:
  .cfi\_def\_cfa\_register %rbp
  subq  $16, %rsp
  movabsq $.L.str, %rdi
  movl  $0, -4(%rbp)
  movb  $0, %al
  callq printf
  xorl  %ecx, %ecx
  movl  %eax, -8(%rbp)
  movl  %ecx, %eax
  addq  $16, %rsp
  popq  %rbp
  retq
.Lfunc\_end0:
  .size main, .Lfunc\_end0-main
  .cfi\_endproc
```



Abstractions all the way down

```c
#include <stdio.h>
int main()
{
  printf("Hello, World!");
  return 0;
}
```

```
main:
  .cfi_startproc
# BB#0:
  pushq %rbp
.Ltmp0:
  .cfi\_def\_cfa\_offset 16
.Ltmp1:
  .cfi\_offset %rbp, -16
  movq  %rsp, %rbp
.Ltmp2:
  .cfi\_def\_cfa\_register %rbp
  subq  $16, %rsp
  movabsq $.L.str, %rdi
  movl  $0, -4(%rbp)
  movb  $0, %al
  callq printf
  xorl  %ecx, %ecx
  movl  %eax, -8(%rbp)
  movl  %ecx, %eax
  addq  $16, %rsp
  popq  %rbp
  retq
.Lfunc\_end0:
  .size main, .Lfunc\_end0-main
  .cfi\_endproc
```

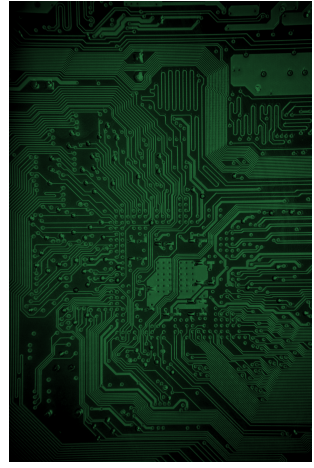


Abstractions all the way down

compilation

```c
#include <stdio.h>
int main()
{
  printf("Hello, World!");
  return 0;
}
```

```
main:
  .cfi_startproc
# BB#0:
  pushq %rbp
.Ltmp0:
  .cfi\_def\_cfa\_offset 16
.Ltmp1:
  .cfi\_offset %rbp, -16
  movq  %rsp, %rbp
.Ltmp2:
  .cfi\_def\_cfa\_register %rbp
  subq  $16, %rsp
  movabsq $.L.str, %rdi
  movl  $0, -4(%rbp)
  movb  $0, %al
  callq printf
  xorl  %ecx, %ecx
  movl  %eax, -8(%rbp)
  movl  %ecx, %eax
  addq  $16, %rsp
  popq  %rbp
  retq
.Lfunc\_end0:
  .size main, .Lfunc\_end0-main
  .cfi\_endproc
```

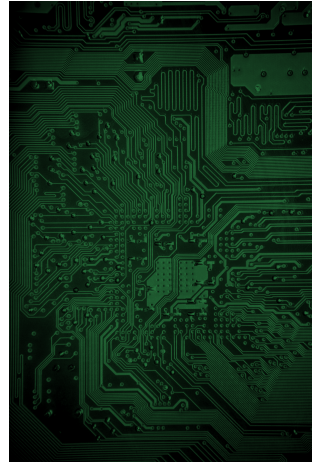


Abstractions all the way down

secure
compilation

```c
#include <stdio.h>
int main()
{
  printf("Hello, World!");
  return 0;
}
```

```
main:
  .cfi_startproc
# BB#0:
  pushq %rbp
.Ltmp0:
  .cfi\_def\_cfa\_offset 16
.Ltmp1:
  .cfi\_offset %rbp, -16
  movq  %rsp, %rbp
.Ltmp2:
  .cfi\_def\_cfa\_register %rbp
  subq  $16, %rsp
  movabsq $.L.str, %rdi
  movl  $0, -4(%rbp)
  movb  $0, %al
  callq printf
  xorl  %ecx, %ecx
  movl  %eax, -8(%rbp)
  movl  %ecx, %eax
  addq  $16, %rsp
  popq  %rbp
  retq
.Lfunc\_end0:
  .size main, .Lfunc\_end0-main
  .cfi\_endproc
```

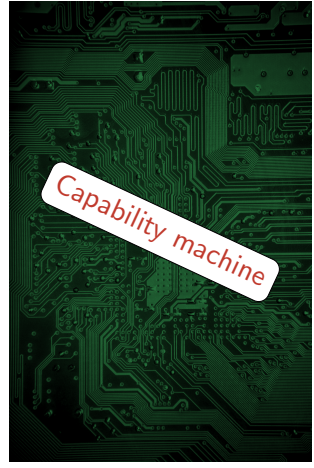


Abstractions all the way down

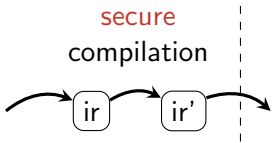
secure compilation

```c
#include <stdio.h>
int main()
{
  printf("Hello, World!");
  return 0;
}
```

```
main:
  .cfi_startproc
# BB#0:
  pushq %rbp
.Ltmp0:
  .cfi\_def\_cfa\_offset 16
.Ltmp1:
  .cfi\_offset %rbp, -16
  movq  %rsp, %rbp
.Ltmp2:
  .cfi\_def\_cfa\_register %rbp
  subq  $16, %rsp
  movabsq $.L.str, %rdi
  movl  $0, -4(%rbp)
  movb  $0, %al
  callq printf
  xorl  %ecx, %ecx
  movl  %eax, -8(%rbp)
  movl  %ecx, %eax
  addq  $16, %rsp
  popq  %rbp
  retq
.Lfunc\_end0:
  .size main, .Lfunc\_end0-main
  .cfi\_endproc
```

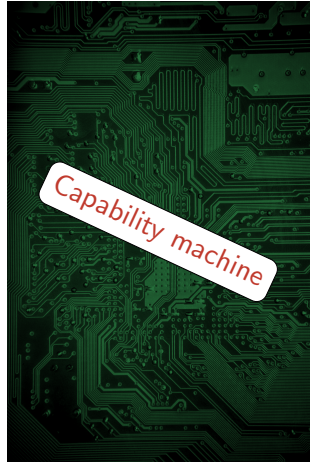


Abstractions all the way down

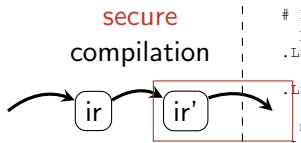



```c
#include <stdio.h>
int main()
{
  printf("Hello, World!");
  return 0;
}
```

```
main:
  .cfi_startproc
# BB#0:
  pushq %rbp
.Ltmp0:
  .cfi\_def\_cfa\_offset 16
.Ltmp1:
  .cfi\_offset %rbp, -16
  movq  %rsp, %rbp
.Ltmp2:
  .cfi\_def\_cfa\_register %rbp
  subq  $16, %rsp
  movabsq $.L.str, %rdi
  movl  $0, -4(%rbp)
  movb  $0, %al
  callq printf
  xorl  %ecx, %ecx
  movl  %eax, -8(%rbp)
  movl  %ecx, %eax
  addq  $16, %rsp
  popq  %rbp
  retq
.Lfunc\_end0:
  .size main, .Lfunc\_end0-main
  .cfi\_endproc
```

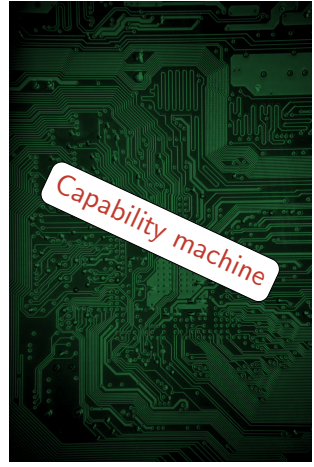


Abstractions all the way down




```c
#include <stdio.h>
int main()
{
  printf("Hello, World!");
  return 0;
}
```

```
main:
  .cfi_startproc
# BB#0:
  pushq %rbp
.Ltmp0:
  .cfi\_def\_cfa\_offset 16
.Ltmp1:
  .cfi\_offset %rbp, -16
  movq  %rsp, %rbp
.Ltmp2:
  .cfi\_def\_cfa\_register %rbp
  subq  $16, %rsp
  movabsq $.L.str, %rdi
  movl  $0, -4(%rbp)
  movb  $0, %al
  callq printf
  xorl  %ecx, %ecx
  movl  %eax, -8(%rbp)
  movl  %ecx, %eax
  addq  $16, %rsp
  popq  %rbp
  retq
.Lfunc\_end0:
  .size main, .Lfunc\_end0-main
  .cfi\_endproc
```



Paper Contents Overview

Formalization of CHERI-like capability machine with linear capabilities

StkTokens a calling convention that provably guarantees local-state encapsulation (LSE) and well-bracketed control-flow (WBCF)

Fully-abstract overlay semantics a novel way to prove LSE and WBCF claims

Overview

StkTokens-paper in the big picture

Defining well-bracketed control flow and local state encapsulation

Defining well-bracketed control flow and local state encapsulation

```
 void a()
{
    ...
    return;
}

void b()
{
    int x = 5;
    a();
    ...
    a();
    return;
}
```

Defining well-bracketed control flow and local state encapsulation

```
 void a()
{
    ...
    return;
}
```

Function `a` cannot access variable `x`

```
void b()
{
    int x = 5;
    a();
    ...
    a();
    return;
}
```

Local-state encapsulation

Defining well-bracketed control flow and local state encapsulation

```
 void a()
{
    ...
    return;
}

void b()
{
    int x = 5;
    a();
    ...
→   a();
    return;
}
```

Well-bracketed control flow

Defining well-bracketed control flow and local state encapsulation

```
→ void a()
  {
      ...
      return;
  }

  void b()
  {
      int x = 5;
      a();
      ...
      a();
      return;
  }
```

Well-bracketed control flow

Defining well-bracketed control flow and local state encapsulation

```
 void a()
{
→   ...
    return;
}

void b()
{
    int x = 5;
    a();
    ...
    a();
    return;
}
```

Well-bracketed control flow

Defining well-bracketed control flow and local state encapsulation

```
 void a()
{
    ...
→   return;
}

void b()
{
    int x = 5;
    a();
    ...
    a();
    return;
}
```

Well-bracketed control flow

Defining well-bracketed control flow and local state encapsulation

```
 void a()
{
    ...
    return;
}

void b()
{
    int x = 5;
    a();
    ...
    a();
    return;
}
```

Well-bracketed control flow

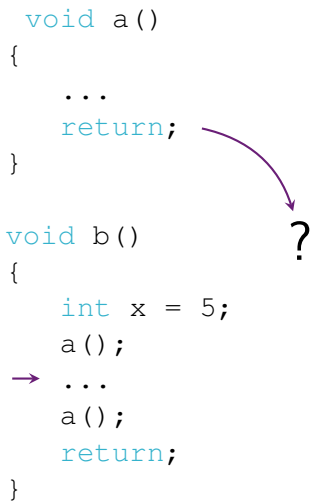
Defining well-bracketed control flow and local state encapsulation

```
 void a()
{
    ...
    return;
}

void b()
{
    int x = 5;
    a();
    ...
    a();
→   return;
}
```

Well-bracketed control flow

Defining well-bracketed control flow and local state encapsulation



Well-bracketed control flow

Desired properties of the WBCF and LSE definition

1. *Intuitive*
2. *Useful for reasoning*
3. *Reusable in secure compiler chains*
4. *Arguably "complete"*

Overlay Semantics

```
move  rtmp1 42
store rstk rtmp1
cca   rstk -1
geta  rtmp1 rstk
cca   rretc 5
move  rtmp1 pc
cca   rtmp1 -20
```

```
load  rtmp1 rtmp1
cca   rtmp1 -21
cseal rretd rtmp1
move  rretc pc
xjmp  r1 r2
cseal rretc rtmp1
move  rtmp1 0
```

Linear Capability Machine (LCM)

Overlay Semantics

```
move  rtmp1 42
store rstk rtmp1
cca   rstk -1
geta  rtmp1 rstk
cca   rretc 5
move  rtmp1 pc
cca   rtmp1 -20
```

```
load  rtmp1 rtmp1
cca   rtmp1 -21
cseal rretd rtmp1
move  rretc pc
xjmp  r1 r2
cseal rretc rtmp1
move  rtmp1 0
```

Overlay Semantics (oLCM)

```
move  rtmp1 42
store rstk rtmp1
cca   rstk -1
geta  rtmp1 rstk
cca   rretc 5
move  rtmp1 pc
cca   rtmp1 -20
```

```
load  rtmp1 rtmp1
cca   rtmp1 -21
cseal rretd rtmp1
move  rretc pc
xjmp  r1 r2
cseal rretc rtmp1
move  rtmp1 0
```

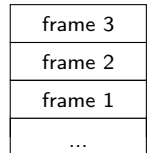
Linear Capability Machine (LCM)

Overlay Semantics

```
move  rtmp1 42
store rstk rtmp1
cca   rstk -1
geta  rtmp1 rstk
cca   rretc 5
move  rtmp1 pc
cca   rtmp1 -20
```

```
load  rtmp1 rtmp1
cca   rtmp1 -21
cseal rretd rtmp1
move  rretc pc
xjmp  r1 r2
cseal rretc rtmp1
move  rtmp1 0
```

Builtin call stack



Overlay Semantics (oLCM)

```
move  rtmp1 42
store rstk rtmp1
cca   rstk -1
geta  rtmp1 rstk
cca   rretc 5
move  rtmp1 pc
cca   rtmp1 -20
```

```
load  rtmp1 rtmp1
cca   rtmp1 -21
cseal rretd rtmp1
move  rretc pc
xjmp  r1 r2
cseal rretc rtmp1
move  rtmp1 0
```

Linear Capability Machine (LCM)

Overlay Semantics

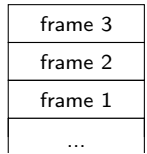
```
move  rtmp1 42
store rstk rtmp1
cca   rstk
geta  rtmp1 rstk
cca   rretc 5
move  rtmp1 pc
cca   rtmp1 -20
```

call

```
load  rtmp1 rtmp1
cca   rtmp1 -21
cseal rretd rtmp1
move  rretc pc
xjmp  r1 r2
cseal rretc rtmp1
move  rtmp1 0
```

return

Builtin call stack



Overlay Semantics (oLCM)

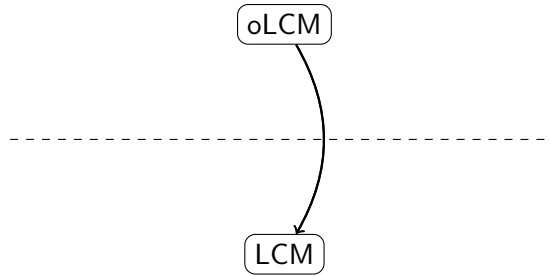
```
move  rtmp1 42
store rstk rtmp1
cca   rstk -1
geta  rtmp1 rstk
cca   rretc 5
move  rtmp1 pc
cca   rtmp1 -20
```

```
load  rtmp1 rtmp1
cca   rtmp1 -21
cseal rretd rtmp1
move  rretc pc
xjmp  r1 r2
cseal rretc rtmp1
move  rtmp1 0
```

Linear Capability Machine (LCM)

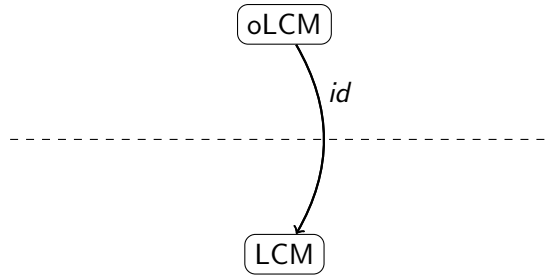
Fully-abstract overlay semantics

- Compile from oLCM to LCM



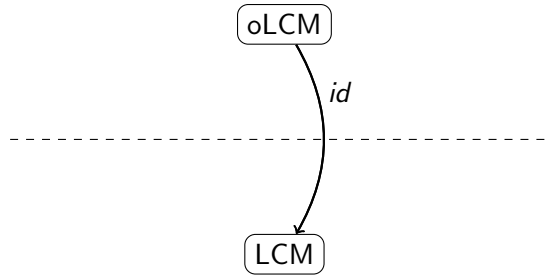
Fully-abstract overlay semantics

- Compile from oLCM to LCM
- Compilation is the identity



Fully-abstract overlay semantics

- Compile from oLCM to LCM
- Compilation is the identity
- Builtin call stack abstraction in oLCM preserved after compilation



Evaluating the definition

1. *Intuitive*
2. *Useful for reasoning*
3. *Reusable in secure compiler chains*
4. *Arguably "complete"*

Evaluating the definition

1. *Intuitive*
   Yes, call stack corresponds exactly to our intuition.
2. *Useful for reasoning*
3. *Reusable in secure compiler chains*
4. *Arguably "complete"*

Evaluating the definition

1. *Intuitive*
   Yes, call stack corresponds exactly to our intuition.
2. *Useful for reasoning*
   Yes, ...
3. *Reusable in secure compiler chains*

4. *Arguably "complete"*

Evaluating the definition

1. *Intuitive*
   Yes, call stack corresponds exactly to our intuition.
2. *Useful for reasoning*
   Yes, ...
3. *Reusable in secure compiler chains*
   Yes, fully-abstract compilations compose vertically, so oLCM can be used as a target for other compilation phases.
4. *Arguably "complete"*

Evaluating the definition

1. *Intuitive*
   Yes, call stack corresponds exactly to our intuition.
2. *Useful for reasoning*
   Yes, ...
3. *Reusable in secure compiler chains*
   Yes, fully-abstract compilations compose vertically, so oLCM can be used as a target for other compilation phases.
4. *Arguably "complete"*
   Yes, ...

The StkTokens calling convention

The StkTokens calling convention

... is explained in the paper.

Thank you!

Full-abstraction proof sketch