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

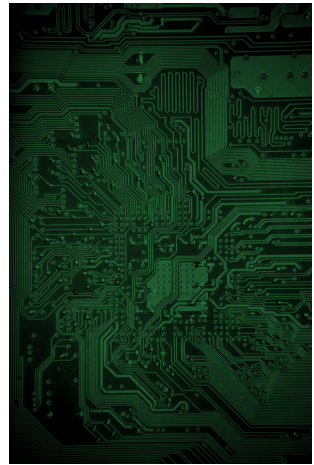
Defining well-bracketed control flow and local state encapsulation

Overview

StkTokens-paper in the big picture

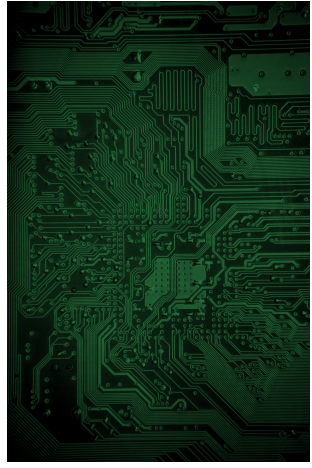
Defining well-bracketed control flow and local state encapsulation

Abstractions all the way down



Abstractions all the way down

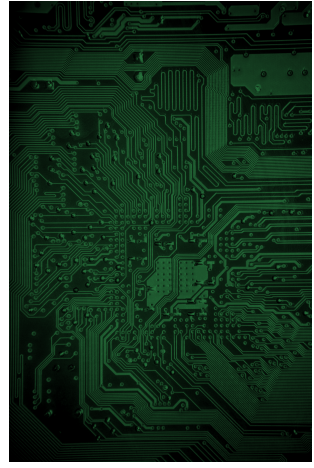
```
main:
  .cfi_startproc
# BB#0:
  pushq %rbp
.Ltmp0:
  .cfi\_def\_cfa\_offset 16
.Ltmp1:
  .cfi\_offset %rbp, -16
  movq  %rsp, %rbp
.Ltmp2:
  .cfi\_def\_cfa\_register %rbp
  subq  $16, %rsp
  movabsq $.L.str, %rdi
  movl  $0, -4(%rbp)
  movb  $0, %al
  callq printf
  xorl  %ecx, %ecx
  movl  %eax,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

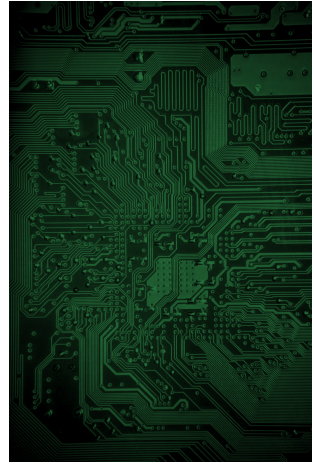


Abstractions all the way down

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
```

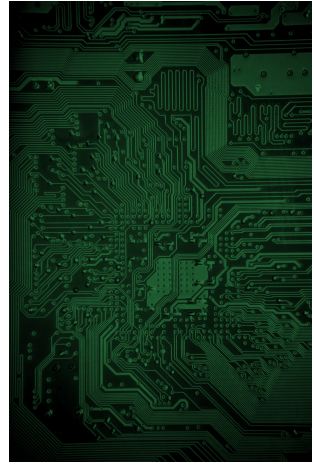


Abstractions all the way down

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
```

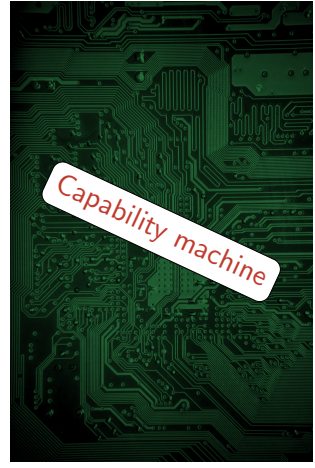


Abstractions all the way down

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
```



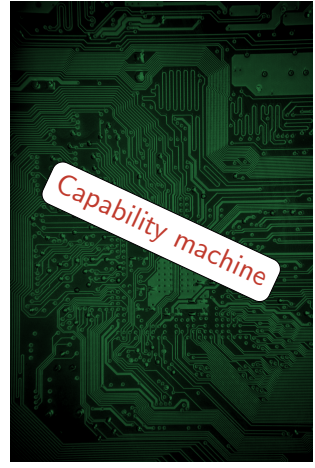
Abstractions all the way down

secure compilation

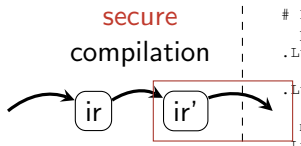
ir → ir'

```
#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
```

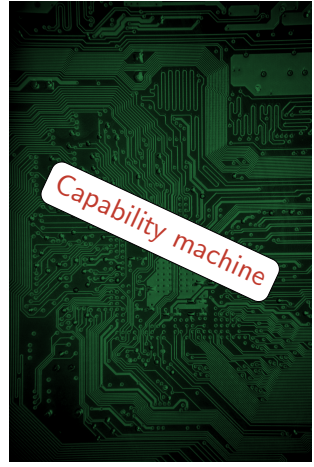


Abstractions all the way down




```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()
{
    int y = 40+2;
    return;
}

void b()
{
    int x = 5;
    a();
    x = 2;
    a();
    return;
}
```

Defining well-bracketed control flow and local state encapsulation

```
void a()
{
    int y = 40+2;
    return;
}
```

Function `a` cannot access variable `x`

```
void b()
{
    int x = 5;
    a();
    x = 2;
    a();
    return;
}
```

Local-state encapsulation

Defining well-bracketed control flow and local state encapsulation

```
void a()
{
    int y = 40+2;
    return;
}

void b()
{
→   int x = 5;
    a();
    x = 2;
    a();
    return;
}
```

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"*

Fully-Abstract Overlay Semantics

Given

- Target semantics
- Calling convention

Define

- New semantics for the same syntax
- Properties by inspection

In this case:
Given

- Capability machine with linear capabilities
- StkTokens

Define

- Same machine, but with built-in call stack
  - Series of instructions corresponding to call and return are interpreted as call and return, respectively, using the call stack

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, ...

Thank you!