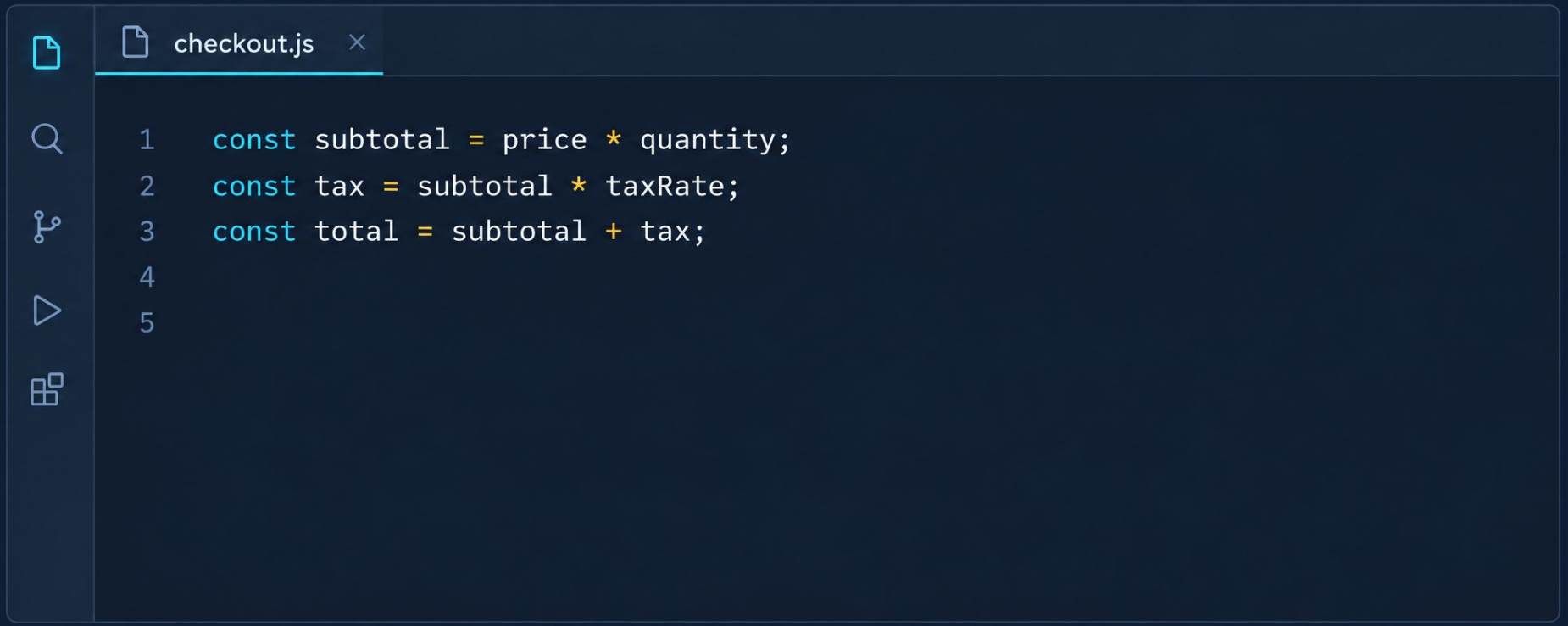
UFMG

Translating Compilers Research Into Hardware Emulation

Compilers Laboratory

Rafael Alvarenga de Azevedo
arafael@cadence.com


checkout.js
1 const subtotal = price * quantity;
2 const tax = subtotal * taxRate;
3 const total = subtotal + tax;
4
5

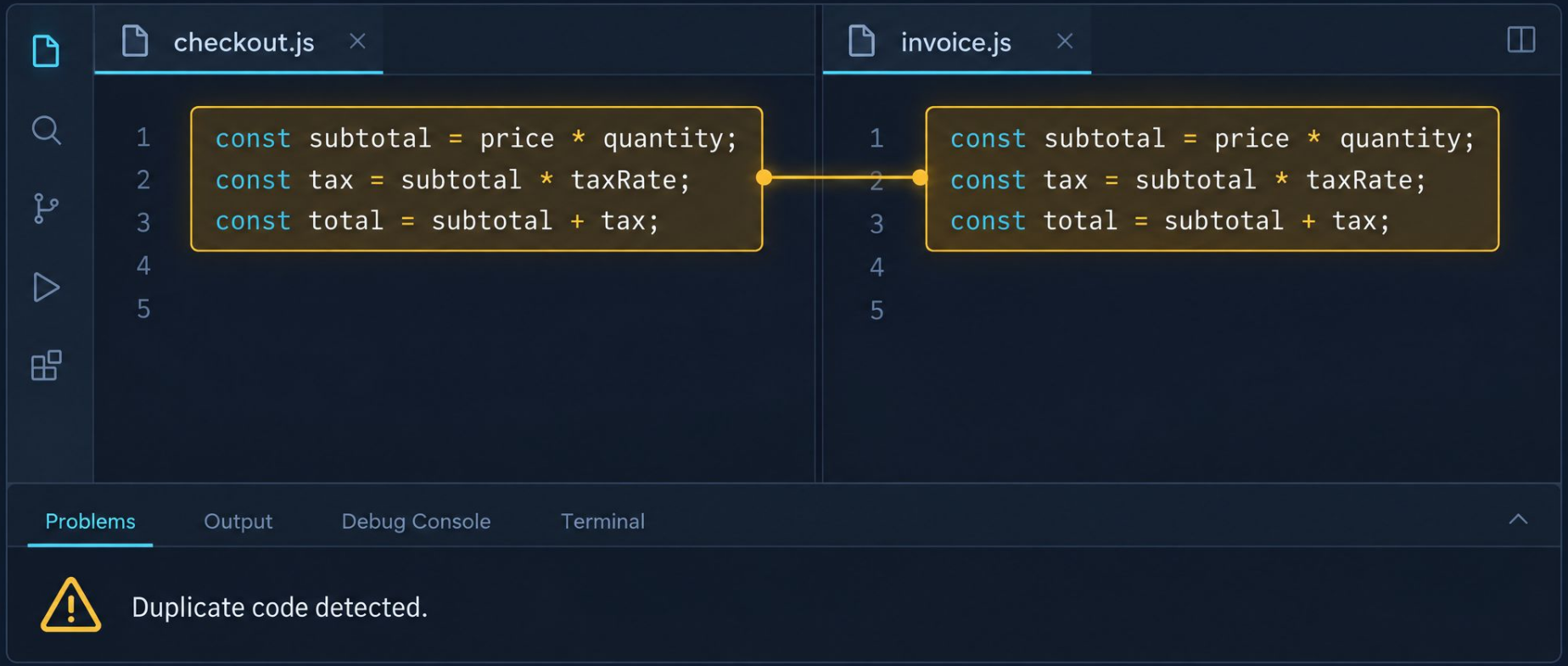
checkout.js

```
1  const subtotal = price * quantity;
2  const tax = subtotal * taxRate;
3  const total = subtotal + tax;
4
5
```

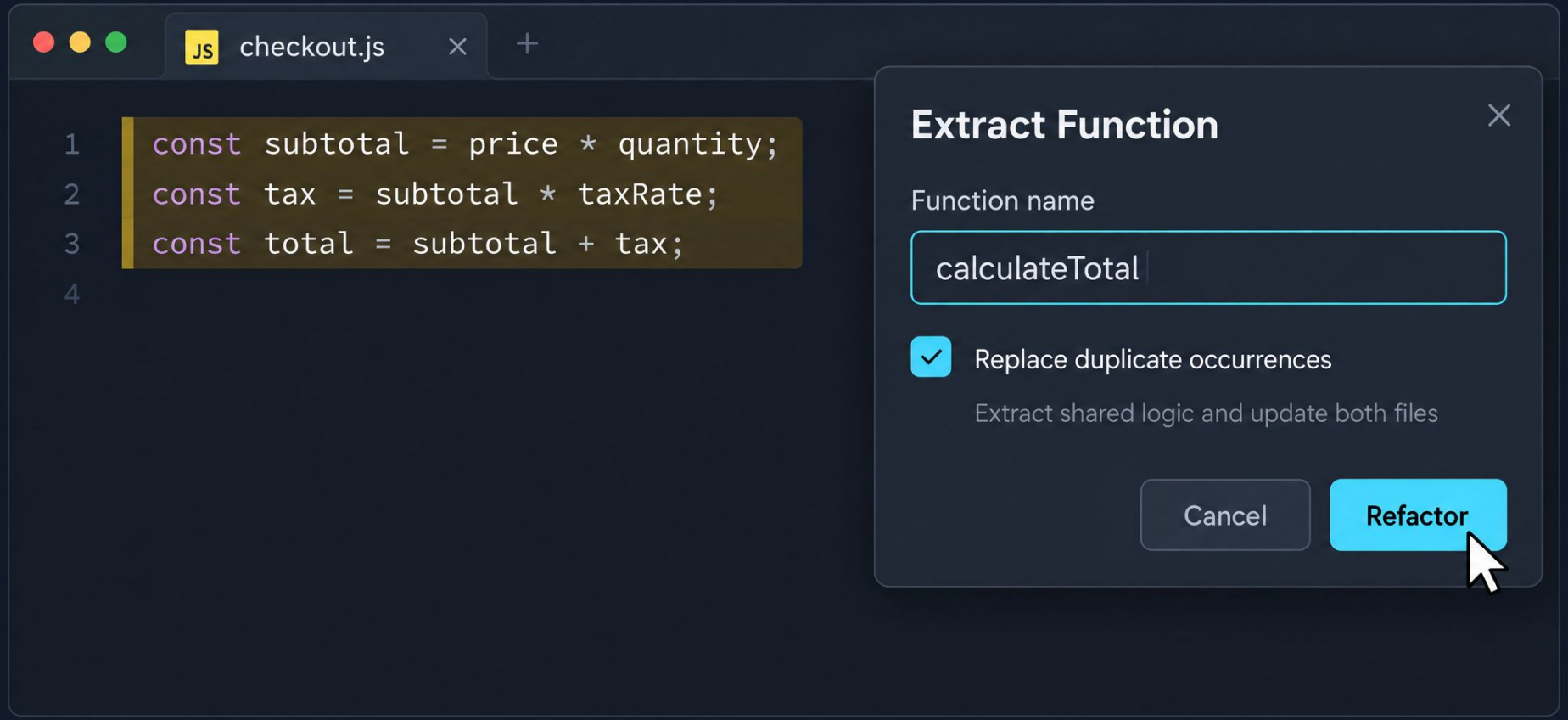
invoice.js

```
1  const subtotal = price * quantity;
2  const tax = subtotal * taxRate;
3  const total = subtotal + tax;
4
5
```

1. Detect duplicate code



2. Extract a reusable function



3. Reuse shared logic

totals.js

```
1  export function calculateTotal(price, quantity, taxRate) {
2    const subtotal = price * quantity;
3    const tax = subtotal * taxRate;
4    return subtotal + tax;
5  }
```

checkout.js

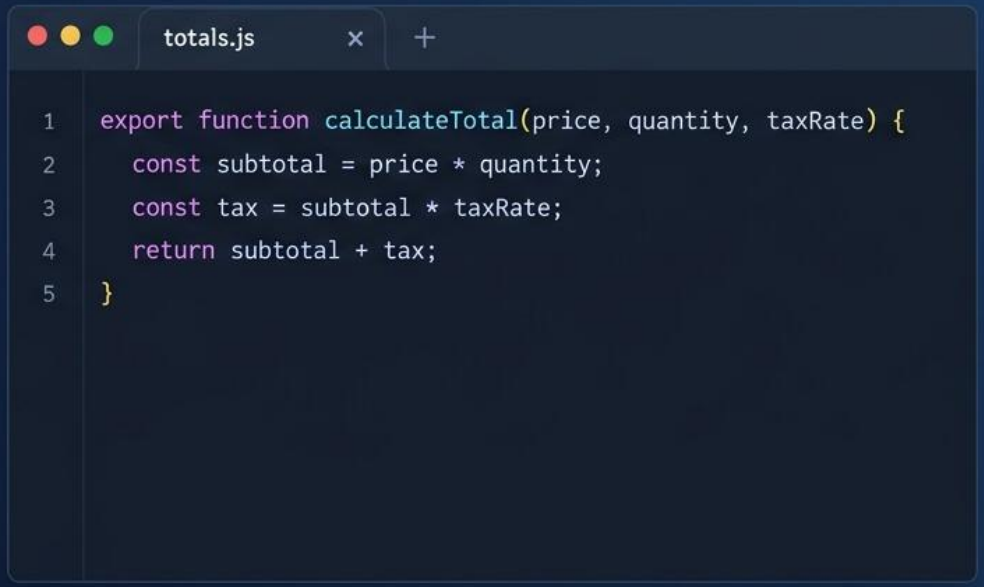
```
1  import { calculateTotal } from "./totals.js";
2  const total = calculateTotal(price, quantity, taxRate);
```

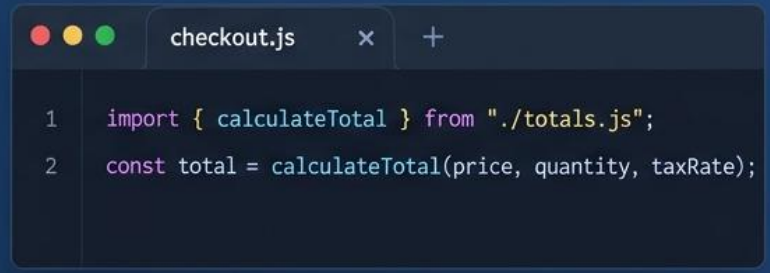
invoice.js

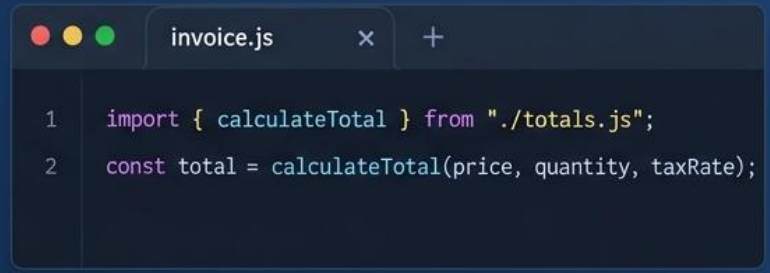
```
1  import { calculateTotal } from "./totals.js";
2  const total = calculateTotal(price, quantity, taxRate);
```



Duplicate code removed


totals.js
export function calculateTotal(price, quantity, taxRate) {
const subtotal = price * quantity;
const tax = subtotal * taxRate;
return subtotal + tax;
}


checkout.js
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);


invoice.js
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);

totals.js

```
export function calculateTotal(price, quantity, taxRate) {
  const subtotal = price * quantity;
  const tax = subtotal * taxRate;
  return subtotal + tax;
}
```

checkout.js

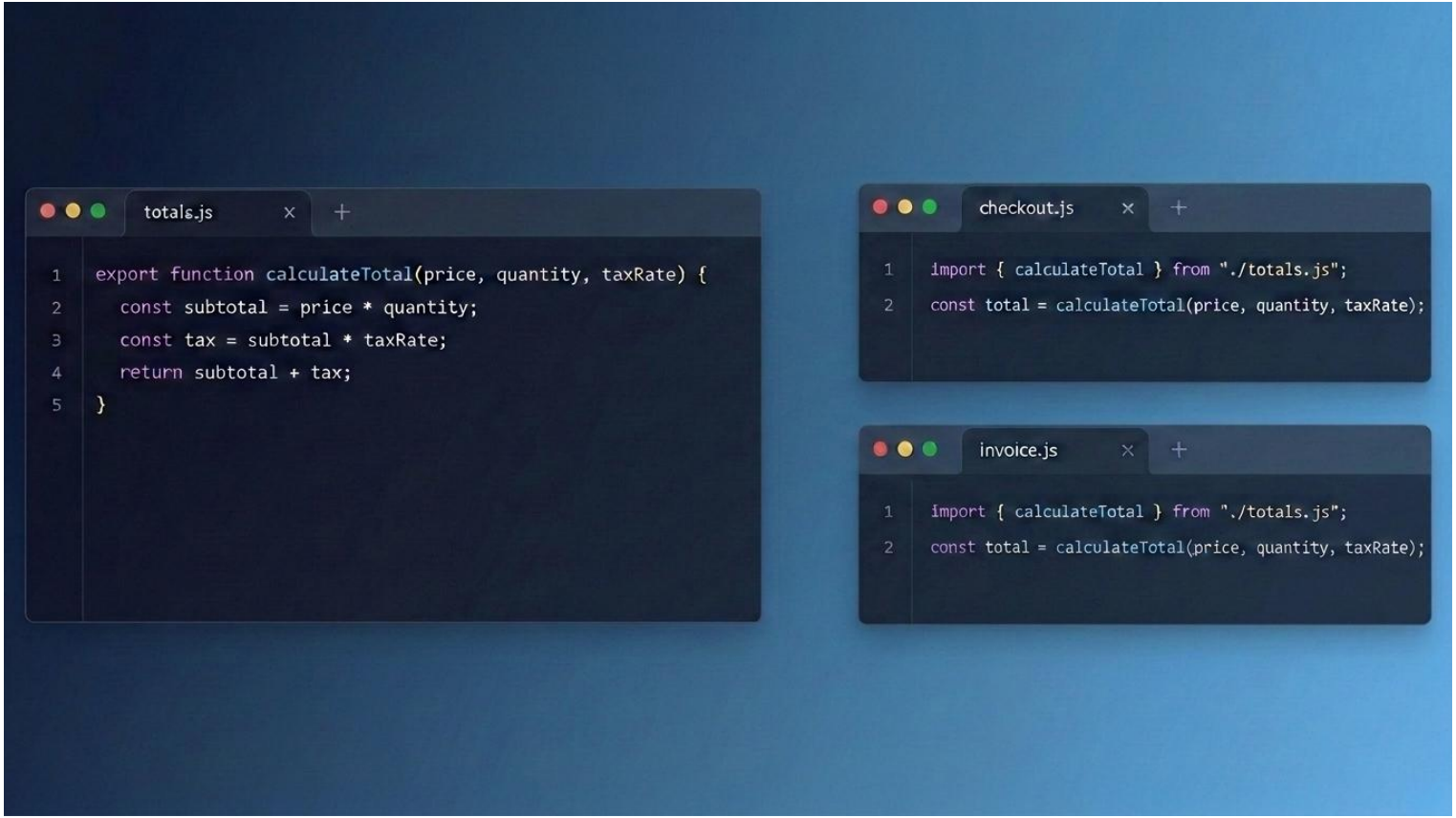
```
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
```

invoice.js

```
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
```

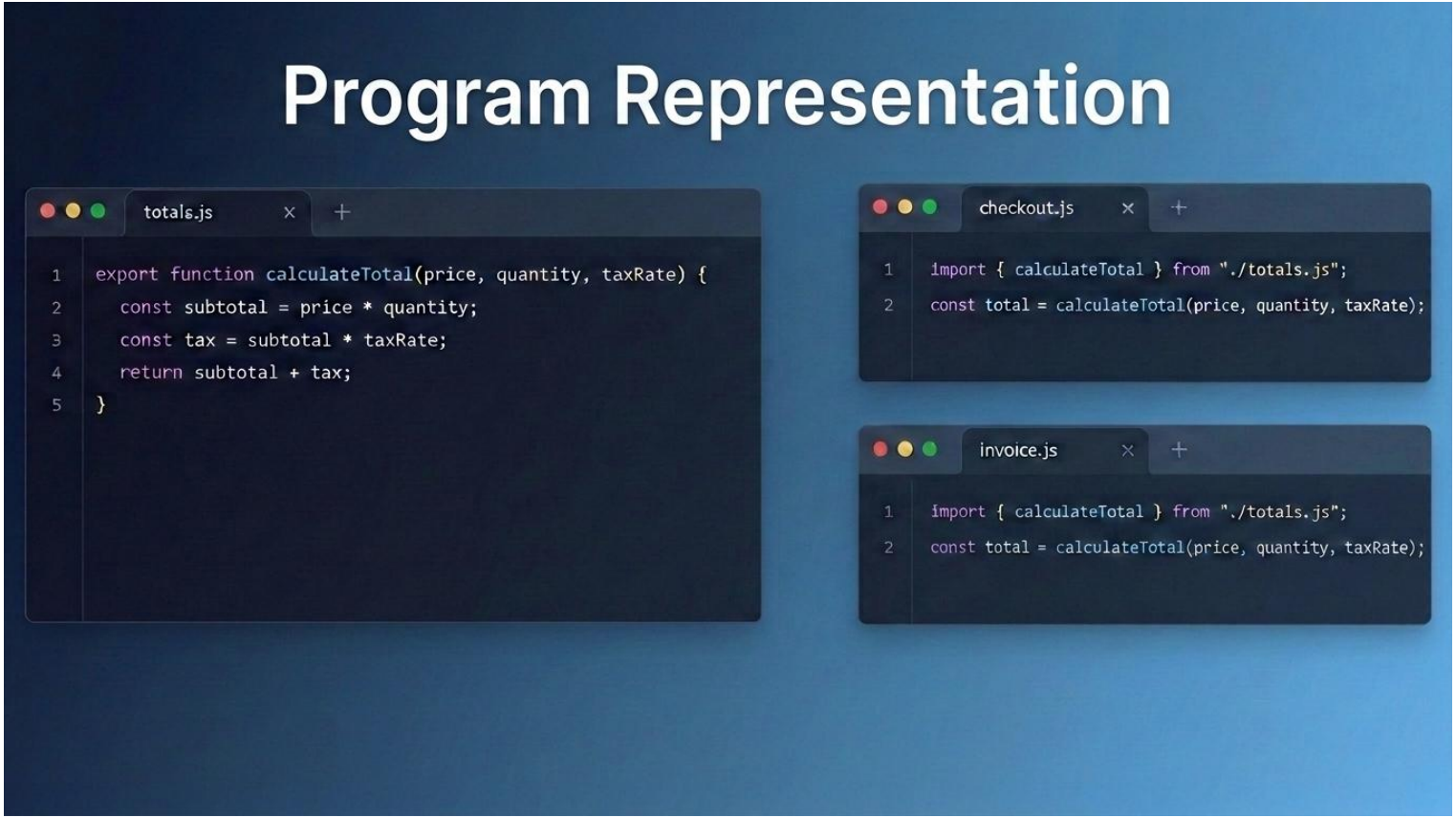
Problem: how to write a program that does this automatically?

Compiler Theory

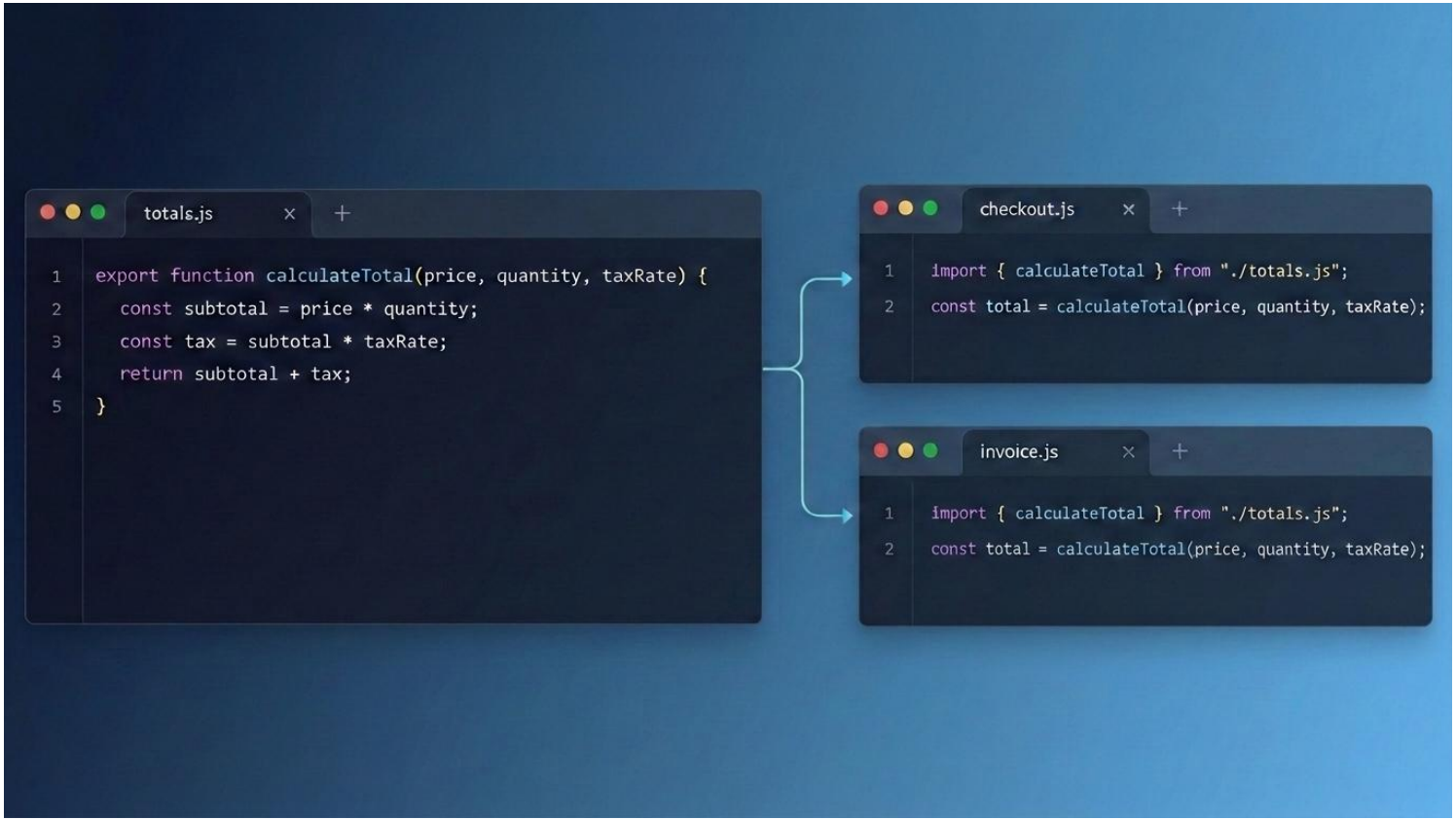


Problem: how to write a program that does this automatically?

Program Representation

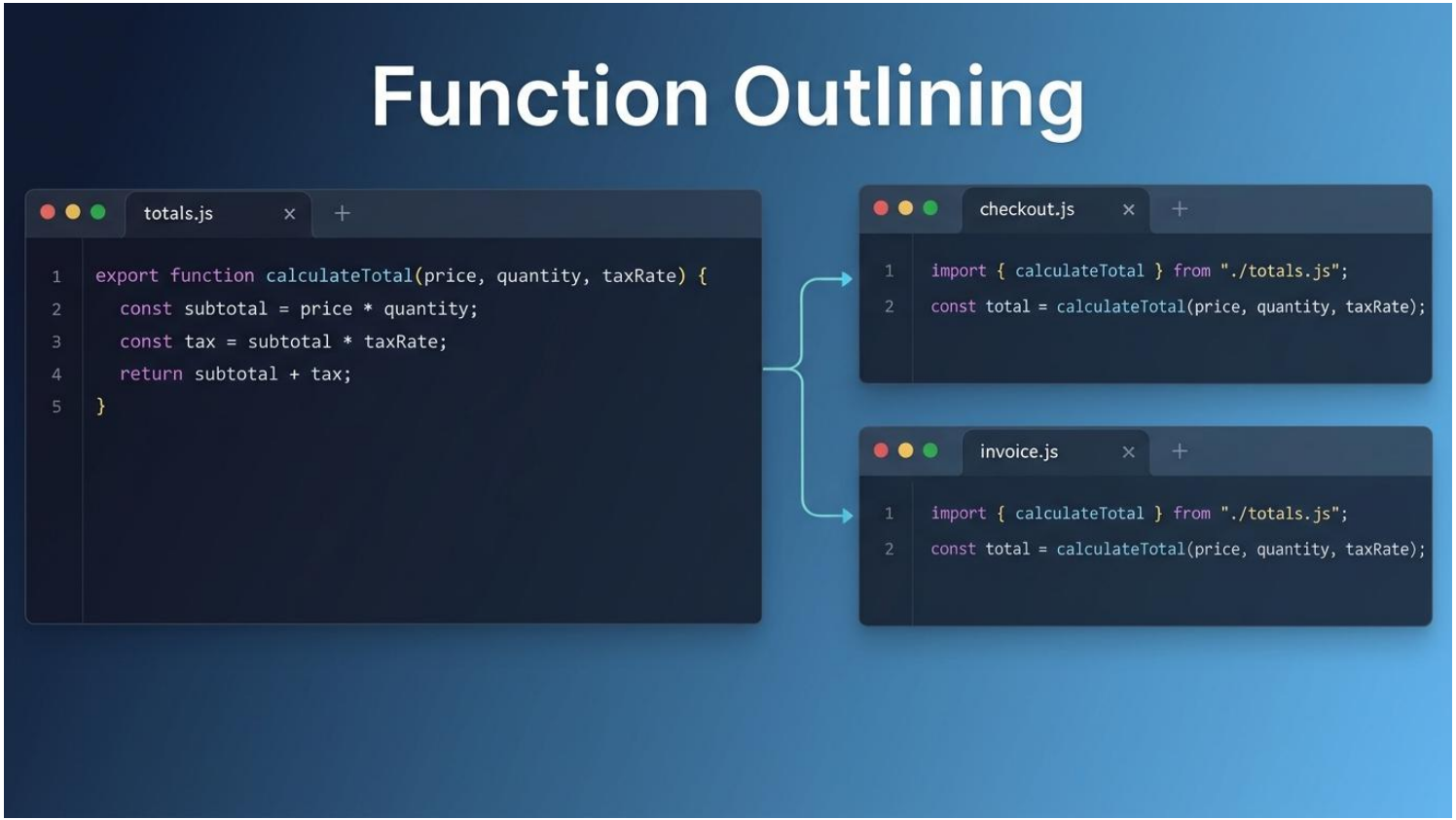


Problem: how to write a program that does this automatically?

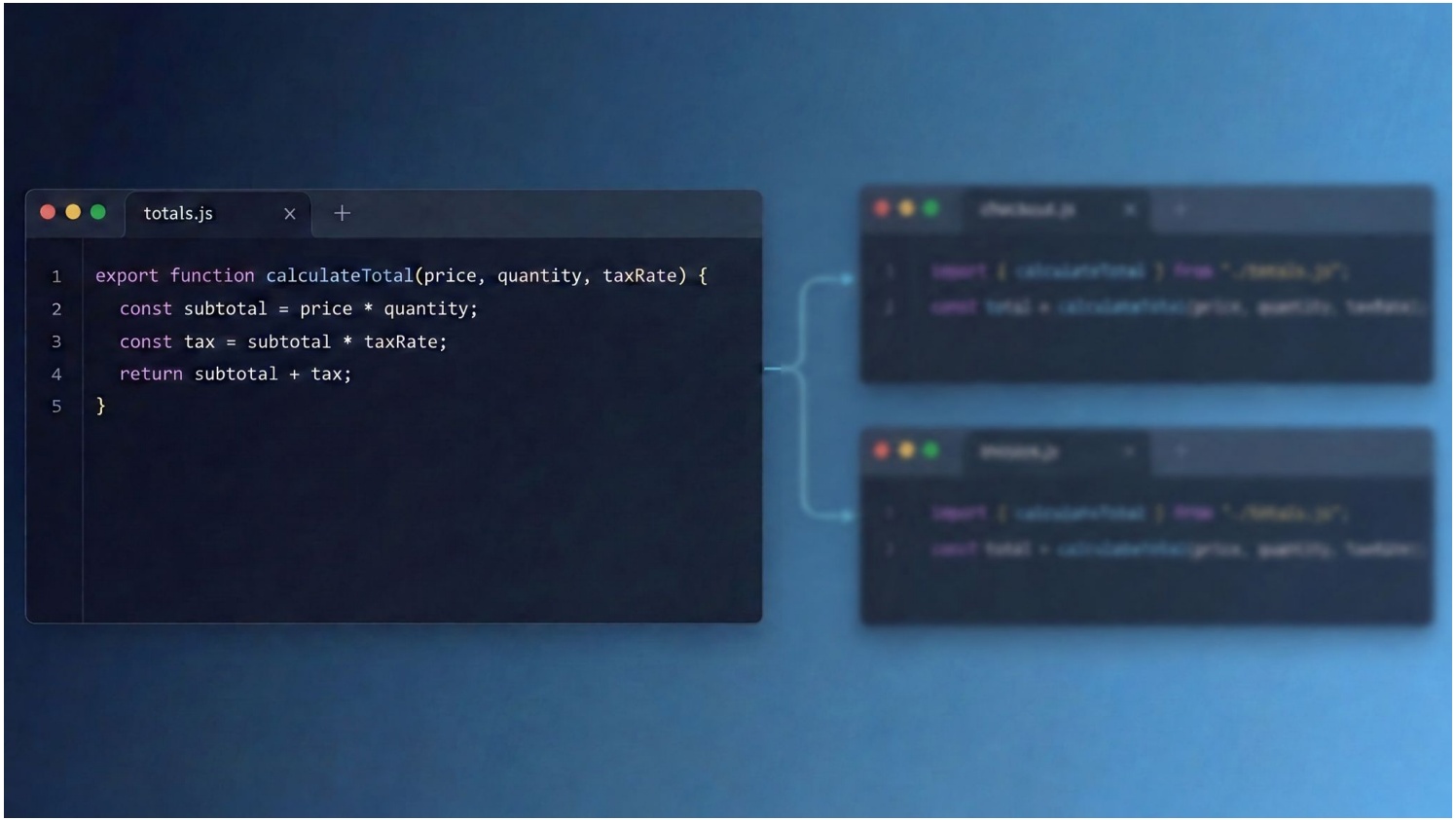


Problem: how to write a program that does this automatically?

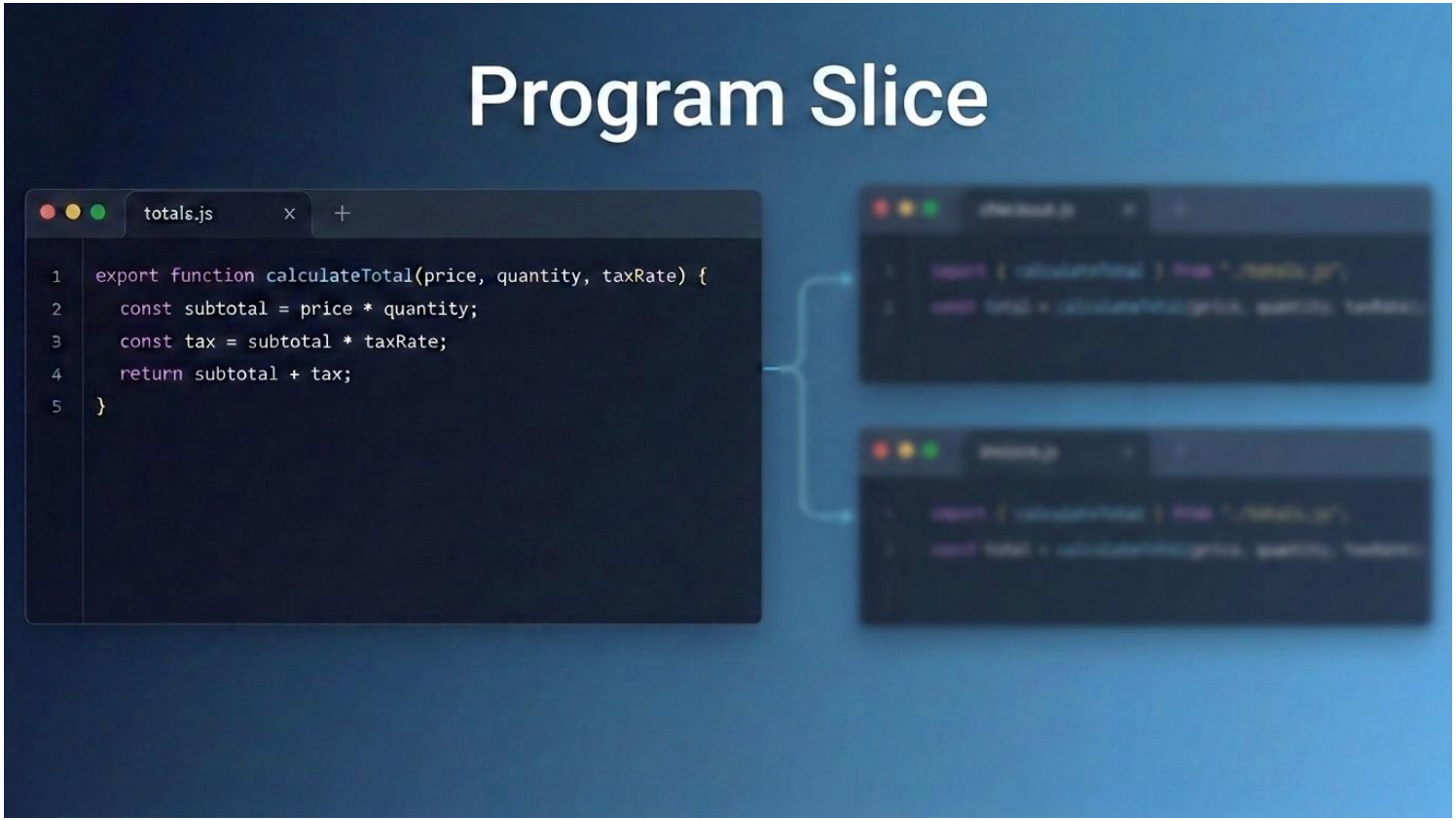
Function Outlining



Problem: how to write a program that does this automatically?

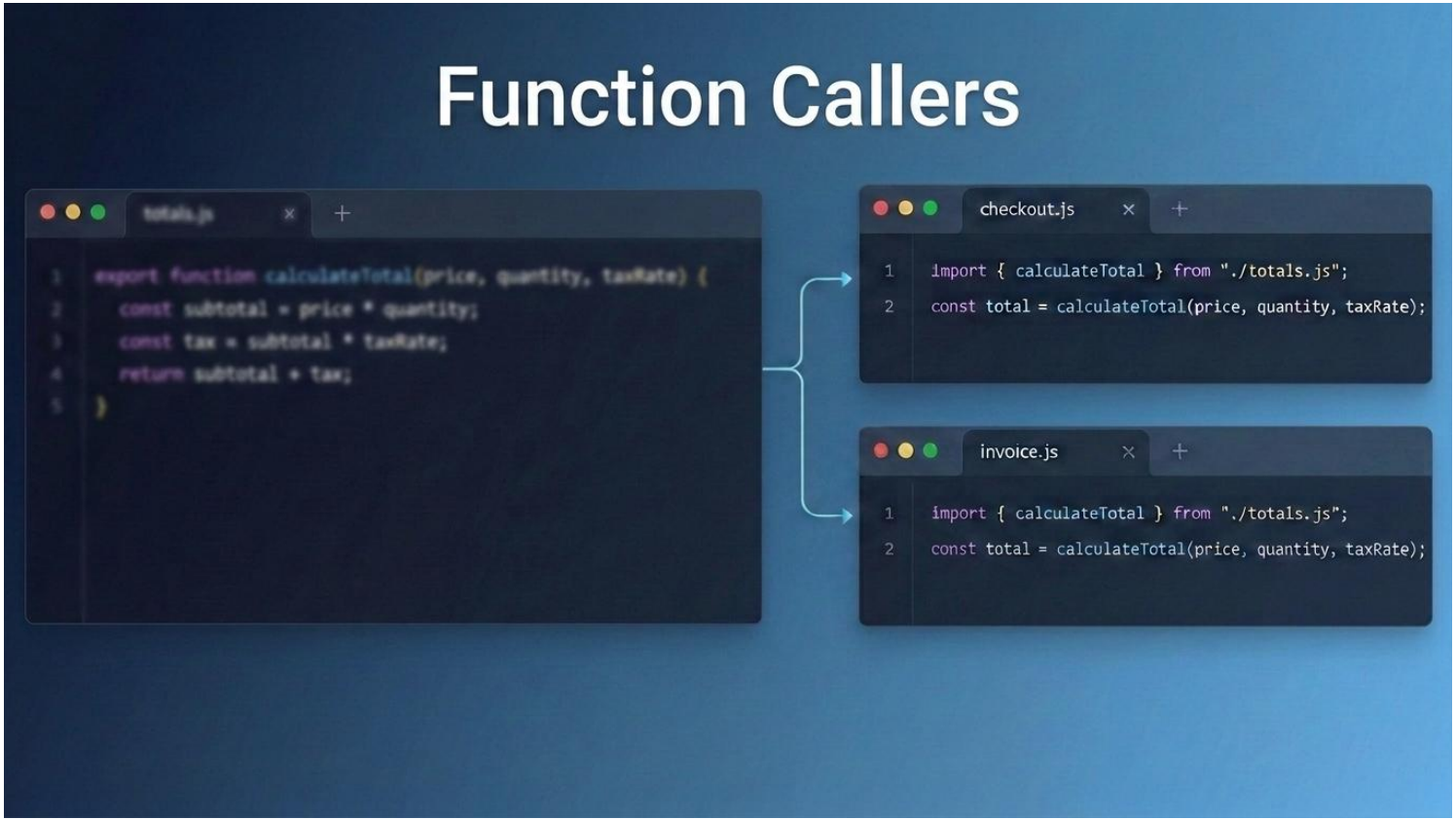


Problem: how to write a program that does this automatically?



Problem: how to write a program that does this automatically?

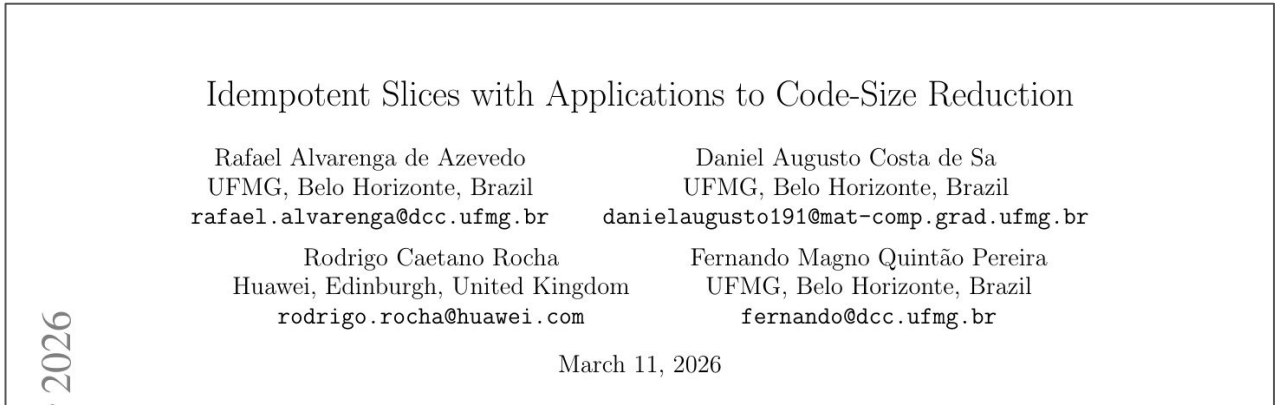
Function Callers



Problem: how to write a program that does this automatically?

Idempotent Backward Slice

Idempotent Backward Slice



Idempotent Slices with Applications to Code-Size Reduction

Rafael Alvarenga de Azevedo
UFMG, Belo Horizonte, Brazil
rafael.alvarenga@dcc.ufmg.br

Daniel Augusto Costa de Sa
UFMG, Belo Horizonte, Brazil
danielaugusto191@mat-comp.grad.ufmg.br

Rodrigo Caetano Rocha
Huawei, Edinburgh, United Kingdom
rodrigo.rocha@huawei.com

Fernando Magno Quintão Pereira
UFMG, Belo Horizonte, Brazil
fernando@dcc.ufmg.br

March 11, 2026

2026

Example

Example

DCC

```
01  int foo(int N) {
02    int x = 0;
03    int s = 1;
04    int t = 0;
05    while (x < N) {
06      x += 1;
07      s *= 2;
08      t += 3;
09    }
10    s = s + 1;
11    u = s = t;
12    return u;
13  }
```

Example

DCC

```
01  int foo(int N) {
02    int x = 0;
03    int s = 1;
04    int t = 0;
05    while (x < N) {
06      x += 1;
07      s *= 2;
08      t += 3;
09    }
10    s = s + 1;
11    u = s = t;
12    return u;
13  }
```



Example

```
01  int foo(int N) {
02    int x = 0;
03    int s = 1;
04    int t = 0;
05    while (x < N) {
06      x += 1;
07      s *= 2;
08      t += 3;
09    }
10    s = s + 1;
11    u = s = t;
12    return u;
13  }
```

slice criterion

```
01  int foo(int N) {
02    int x = 0;
03    int s = 1;
04    int t = 0;
05    while (x < N) {
06      x += 1;
07      s *= 2;
08      t += 3;
09    }
10    s = s + 1;
11    u = s = t;
12    return u;
13  }
```

Example - **Program Slice**

DCC

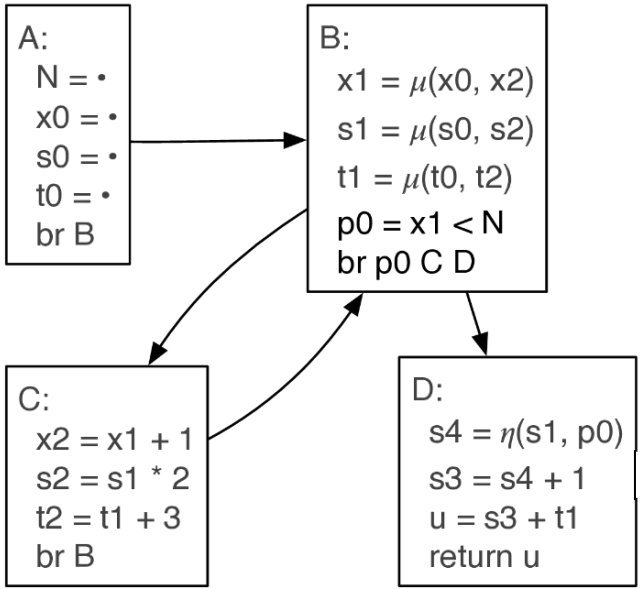
```
01  int foo(int N) {
02    int x = 0;
03    int s = 1;
04    int t = 0;
05    while (x < N) {
06      x += 1;
07      s *= 2;
08      t += 3;
09    }
10    s = s + 1;
11    u = s = t;
12    return u;
13  }
```

Example

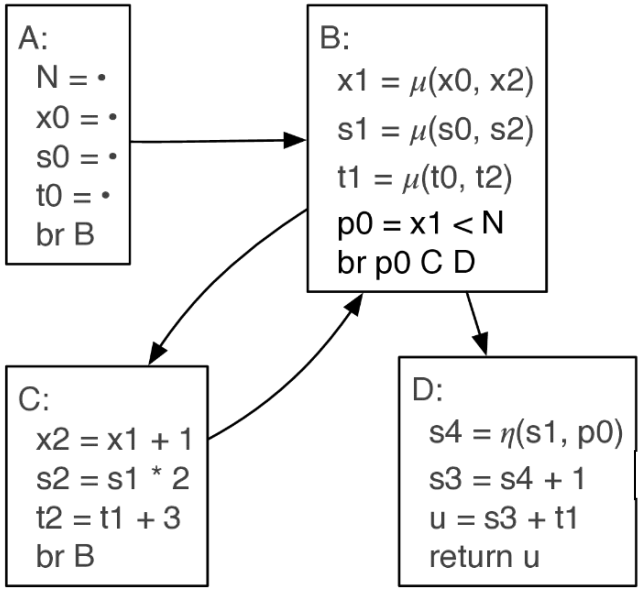
DCC

```
01  int foo(int N) {
02    int x = 0;
03    int s = 1;
04    int t = 0;
05    while (x < N) {
06      x += 1;
07      s *= 2;
08      t += 3;
09    }
10    s = s + 1;
11    u = s = t;
12    return u;
13  }
```

Example

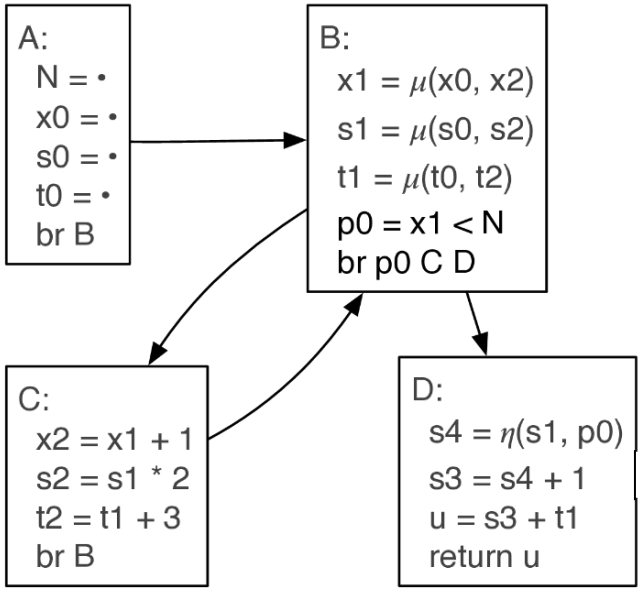

A:
N = •
x0 = •
s0 = •
t0 = •
br B
B:
x1 = μ(x0, x2)
s1 = μ(s0, s2)
t1 = μ(t0, t2)
p0 = x1 < N
br p0 C D
C:
x2 = x1 + 1
s2 = s1 * 2
t2 = t1 + 3
br B
D:
s4 = η(s1, p0)
s3 = s4 + 1
u = s3 + t1
return u

Example



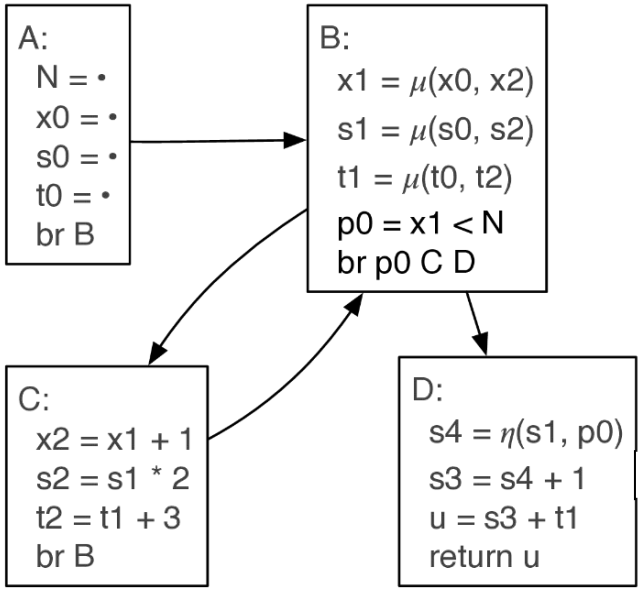
Control flow graph

Example

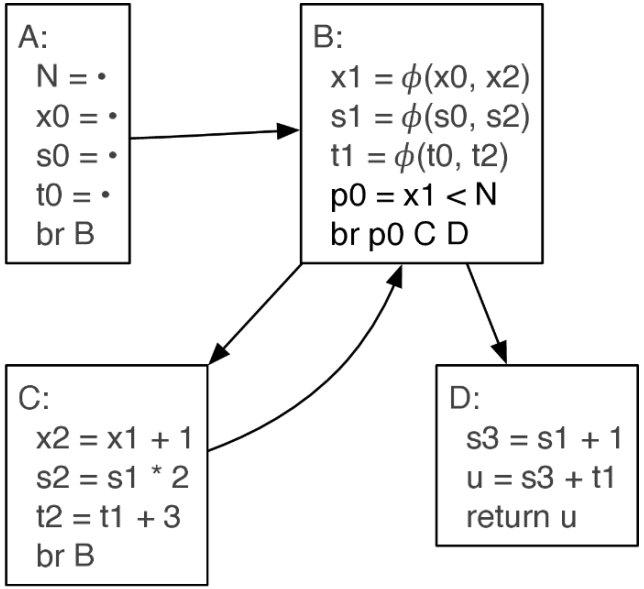


GSA form

Example



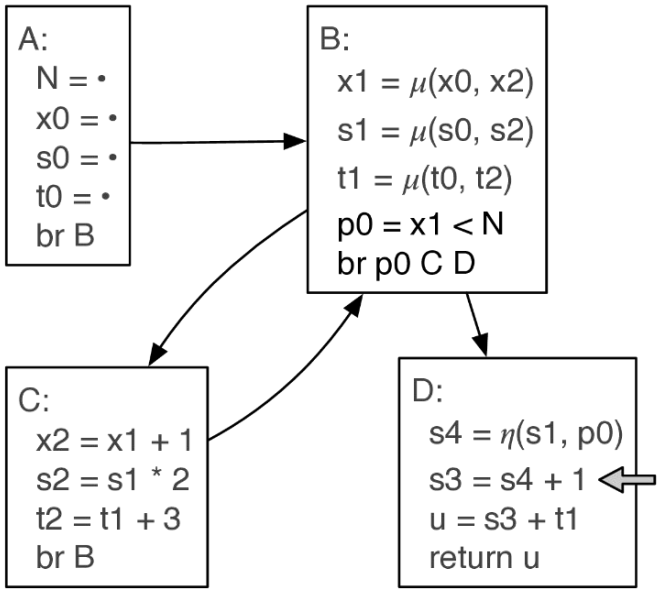
GSA form



SSA form

Example - **Program Representation**

DCC



GSA form

```
01  int foo(int N) {
02    int x = 0;
03    int s = 1;
04    int t = 0;
05    while (x < N) {
06      x += 1;
07      s *= 2;
08      t += 3;
09    }
10    s = s + 1;
11    u = s = t;
12    return u;
13  }
```

LLVM Pass

LLVM Pass

DCC

LLVM Pass

DCC



Project Daedalus

A code compression technique based on the identification and extraction of recurrent program slices in the LLVM IR

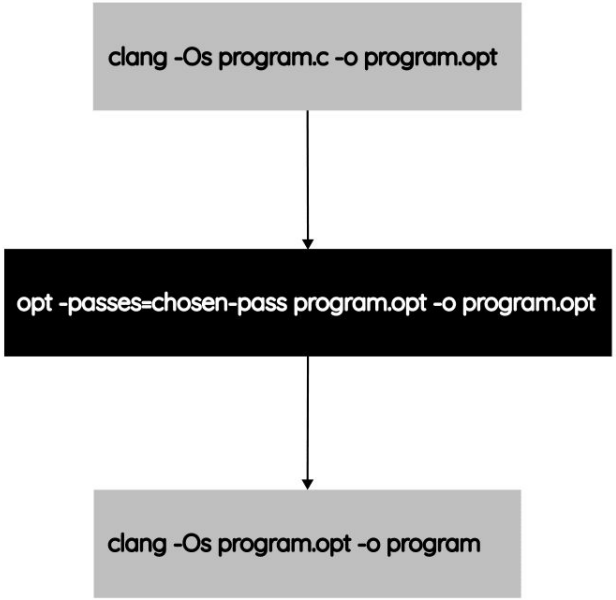
Evaluation

DCC

Evaluation

DCC

Compilation pipeline

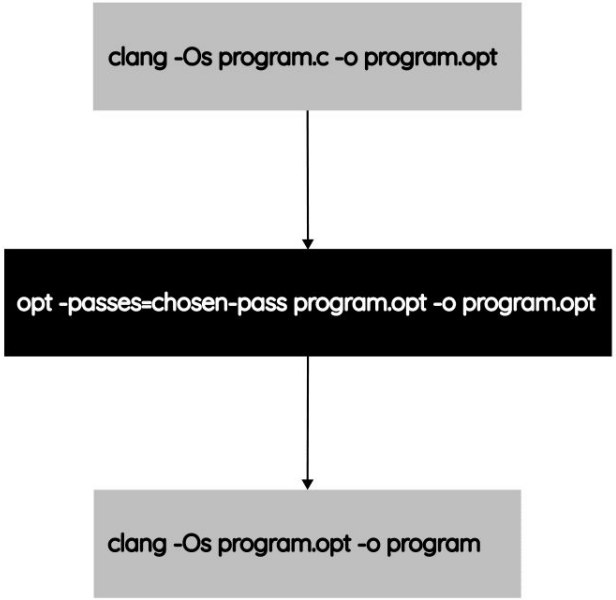


Evaluation

DCC

Compilation pipeline

LLVM Test Suite

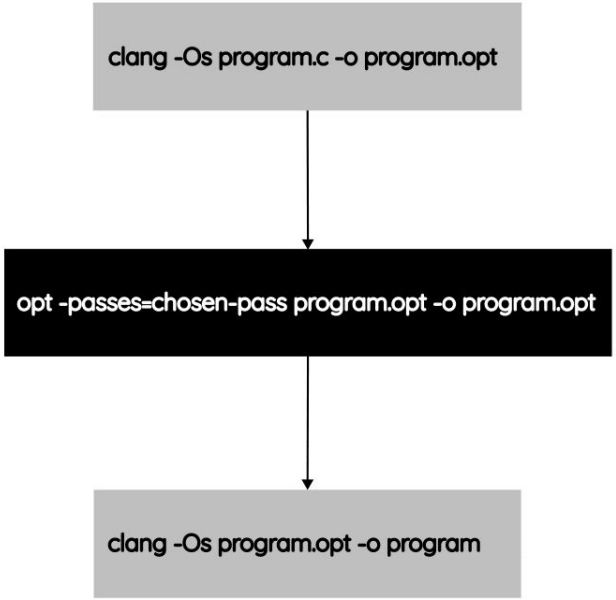


Evaluation

DCC

Compilation pipeline

- LLVM Test Suite
- 2007 programs



Evaluation

DCC

2007 programs total

Evaluation

DCC

2007 programs total

Metric
Instcount
.text size
Exec. Time
Comp. Time

Evaluation

DCC

2007 programs total

Metric
Instcount
.text size
Exec. Time
Comp. Time

Number of LLVM instructions

Evaluation

DCC

2007 programs total

Metric
Instcount
.text size
Exec. Time
Comp. Time



Evaluation

DCC

2007 programs total

Metric
Instcount
.text size
Exec. Time
Comp. Time

Test execution time

Evaluation

DCC

2007 programs total

Metric
Instcount
.text size
Exec. Time
Comp. Time

Test compilation time

Evaluation

DCC

2007 programs total

	Daedalus	
Metric	**Affected**	**Geom.**
Instcount	142	-0.24%
.text size	133	0.11%
Exec. Time	161	0.06%
Comp. Time	303	4.22%

Evaluation

DCC

2007 programs total

	Daedalus	
Metric	**Affected**	**Geom.**
Instcount	142	-0.24%
.text size	133	0.11%
Exec. Time	161	0.06%
Comp. Time	303	4.22%



Evaluation

DCC

2007 programs total

	Daedalus	
Metric	**Affected**	**Geom.**
Instcount	142	-0.24%
.text size	133	0.11%
Exec. Time	161	0.06%
Comp. Time	303	4.22%

Reduction

Growth

Evaluation

DCC

2007 programs total

	Daedalus		Func. Merging	
Metric	**Affected**	**Geom.**	**Affected**	**Geom.**
Instcount	142	-0.24%	183	-0.35%
.text size	133	0.11%	190	0.39%
Exec. Time	161	0.06%	177	0.25%
Comp. Time	303	4.22%	241	2.06%

Evaluation

DCC

2007 programs total

	Daedalus		Func. Merging		IROutliner	
Metric	**Affected**	**Geom.**	**Affected**	**Geom.**	**Affected**	**Geom.**
Instcount	142	-0.24%	183	-0.35%	207	-0.65%
.text size	133	0.11%	190	0.39%	160	-0.19%
Exec. Time	161	0.06%	177	0.25%	160	-0.09%
Comp. Time	303	4.22%	241	2.06%	258	2.48%

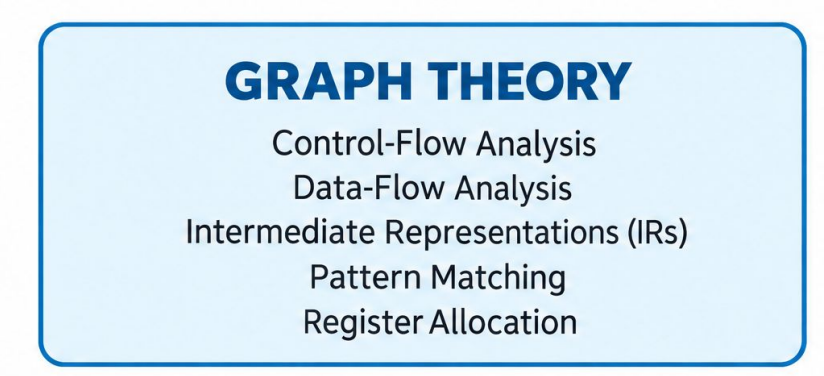
What Compilers Teach You

What Compilers Teach You

DCC

What Compilers Teach You

DCC



What Compilers Teach You

DCC

GRAPH THEORY

Control-Flow Analysis
Data-Flow Analysis
Intermediate Representations (IRs)
Pattern Matching
Register Allocation

CORRECTNESS

Formal Methods
Formal Verification
Bug Prevention
Specification Testing
Proving Compiler Logic

What Compilers Teach You

DCC

GRAPH THEORY

Control-Flow Analysis
Data-Flow Analysis
Intermediate Representations (IRs)
Pattern Matching
Register Allocation

SYSTEMS PROGRAMMING

Optimizing for Hardware
Kernel/OS Integration
Runtime Systems
Direct Hardware Access
Multi-core Processing

CORRECTNESS

Formal Methods
Formal Verification
Bug Prevention
Specification Testing
Proving Compiler Logic

What Compilers Teach You

DCC

GRAPH THEORY

Control-Flow Analysis
Data-Flow Analysis
Intermediate Representations (IRs)
Pattern Matching
Register Allocation

SYSTEMS PROGRAMMING

Optimizing for Hardware
Kernel/OS Integration
Runtime Systems
Direct Hardware Access
Multi-core Processing

CORRECTNESS

Formal Methods
Formal Verification
Bug Prevention
Specification Testing
Proving Compiler Logic

TRADE-OFFS

Balancing Performance
Code Size
Compile Time
Memory Usage
Security

From Compilers to Hardware Emulation

Hardware Emulation

DCC

Hardware Emulation - Synthesis

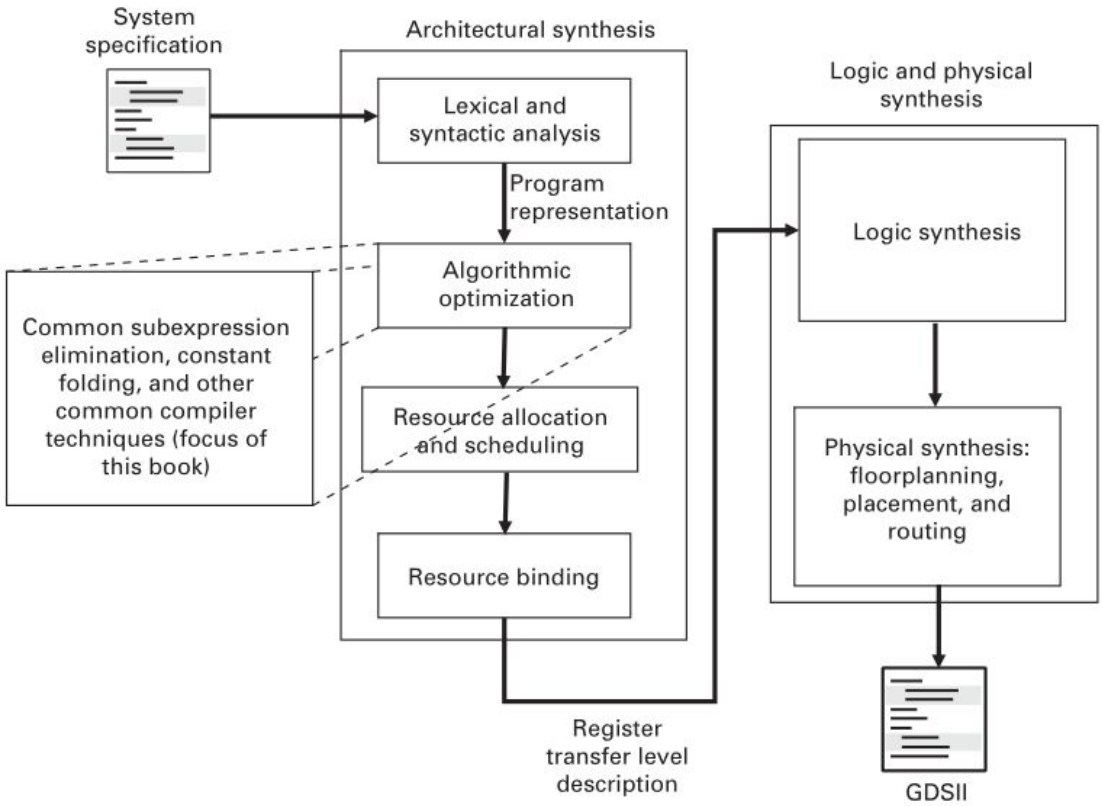


Image from [2]

Hardware Emulation

DCC

Hardware Emulation

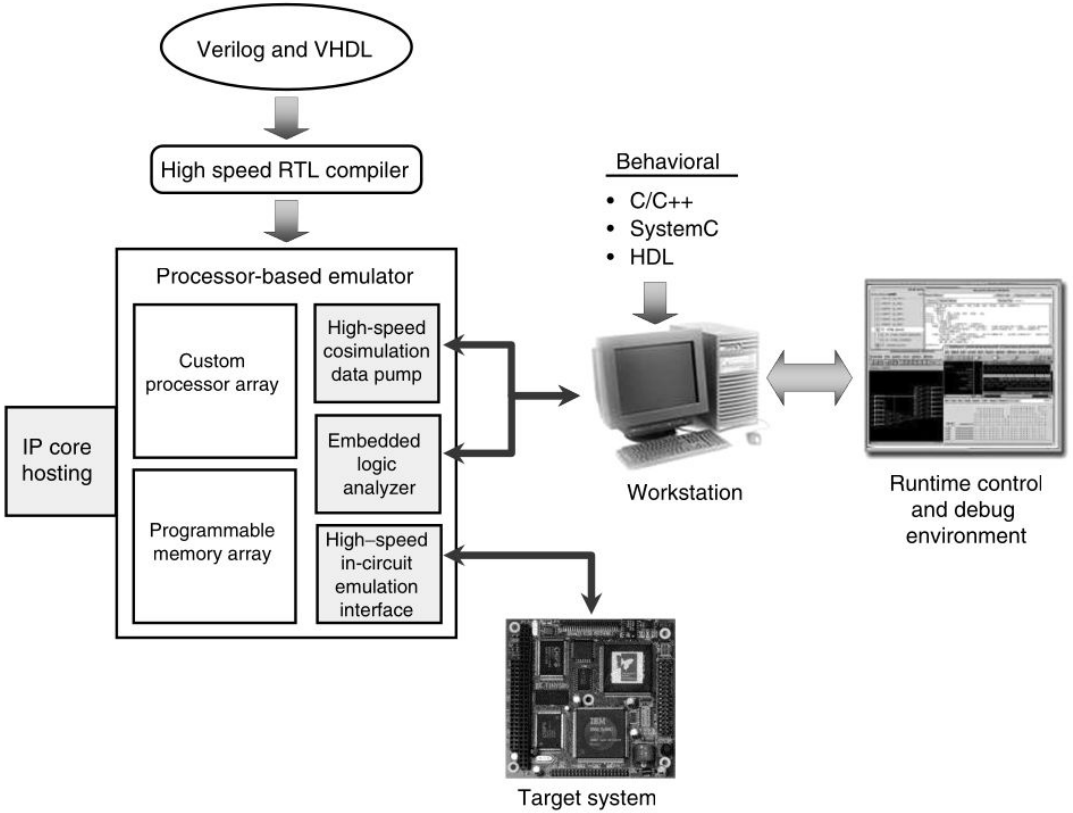
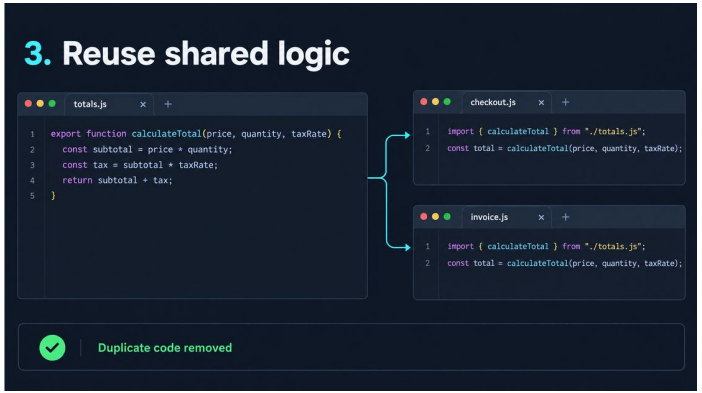
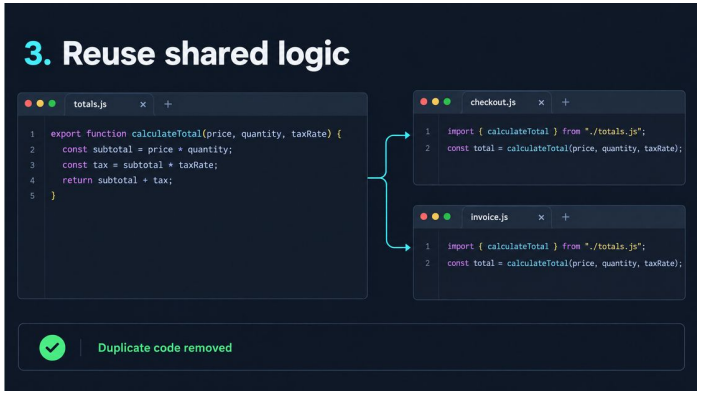


Image from [3]

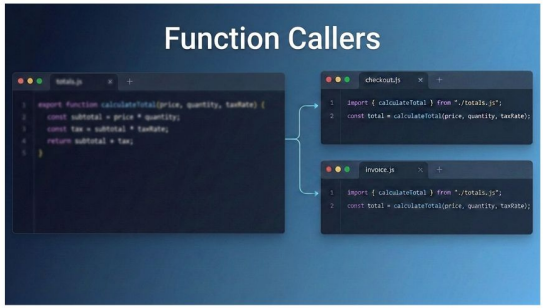
Conclusion


3. Reuse shared logic
totals.js
export function calculateTotal(price, quantity, taxRate) {
const subtotal = price * quantity;
const tax = subtotal * taxRate;
return subtotal + tax;
}
checkout.js
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
invoice.js
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
Duplicate code removed

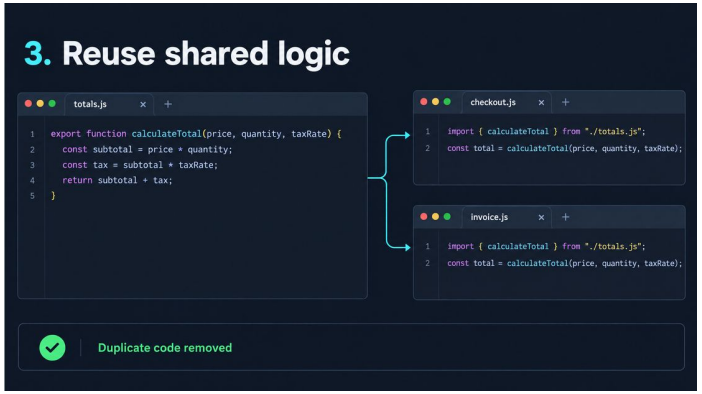
Conclusion



Conclusion

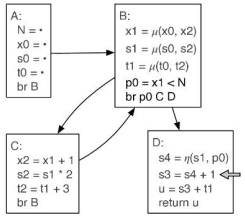


Problem: how to write a program that does this automatically?



Example - **Program Representation**

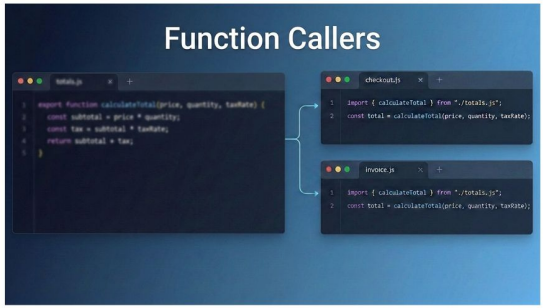
DCC



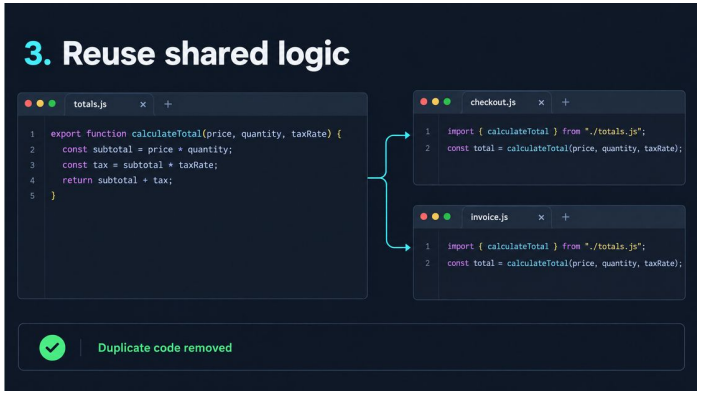
GSA form

```
01 int foo(int N) {
02   int x = 0;
03   int s = 1;
04   int t = 0;
05   while (x < N) {
06     x += 1;
07     s *= 2;
08     t += 3;
09   }
10   s = s + 1;⇦
11   u = s + t;
12   return u;
13 }
```

Conclusion

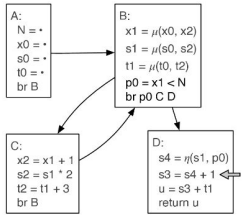


Problem: how to write a program that does this automatically?



Example - **Program Representation**

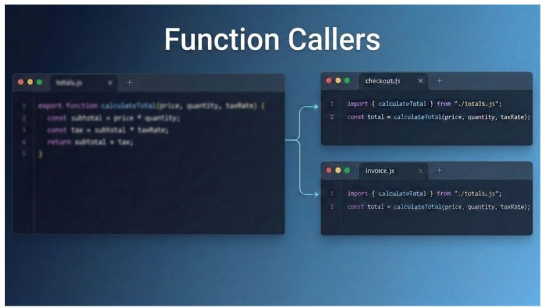
DCC



GSA form

```
01 int foo(int N) {
02   int x = 0;
03   int s = 1;
04   int t = 0;
05   while (x < N) {
06     x += 1;
07     s *= 2;
08     t += 3;
09   }
10   s = s + 1;⇦
11   u = s = t;
12   return u;
13 }
```

Conclusion



Problem: how to write a program that does this automatically?

LLVM Pass



Project Daedalus

A code compression te
the identification ar
recurrent program sli

3. Reuse shared logic

totals.js

```
export function calculateTotal(price, quantity, taxRate) {
  const subtotal = price * quantity;
  const tax = subtotal * taxRate;
  return subtotal + tax;
}
```

checkout.js

```
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
```

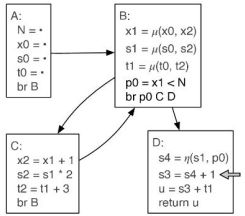
invoice.js

```
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
```

Duplicate code removed

Example - **Program Representation**

DCC

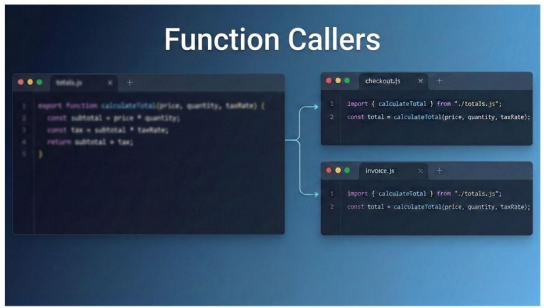


GSA form

```
01 int foo(int N) {
02   int x = 0;
03   int s = 1;
04   int t = 0;
05   while (x < N) {
06     x += 1;
07     s *= 2;
08     t += 3;
09   }
10   s = s + 1;⇦
11   u = s = t;
12   return u;
13 }
```

Conclusion

Function Callers



Problem: how to write a program that does this automatically?

LLVM Pass



Project Daedalus

A code compression t
the identification a
recurrent program sl

Evaluation

DCC

2007 programs total

	Daedalus		Func. Merging		IROutliner	
Metric	Affected	Geom.	Affected	Geom.	Affected	Geom.
Instcount	142	-0.24%	183	-0.35%	207	-0.65%
.text size	133	0.11%	190	0.39%	160	-0.19%
Exec. Time	161	0.06%	177	0.25%	160	-0.09%
Comp. Time	303	4.22%	241	2.06%	258	2.48%

3. Reuse shared logic

totals.js

```
export function calculateTotal(price, quantity, taxRate) {
  const subtotal = price * quantity;
  const tax = subtotal * taxRate;
  return subtotal + tax;
}
```

checkout.js

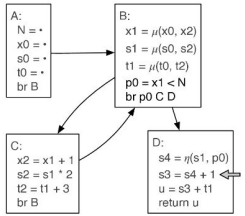
```
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
```

invoice.js

```
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
```

Duplicate code removed

Example - **Program Representation**



GSA form

```
01 int foo(int N) {
02   int x = 0;
03   int s = 1;
04   int t = 0;
05   while (x < N) {
06     x += 1;
07     s *= 2;
08     t += 3;
09   }
10   s = s + 1;
11   u = s = t;
12   return u;
13 }
```

What Compilers Teach You

DCC

GRAPH THEORY
Control-Flow Analysis
Data-Flow Analysis
Intermediate Representations (IRs)
Pattern Matching
Register Allocation

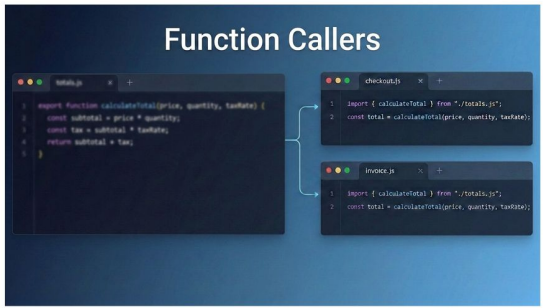
SYSTEMS PROGRAMMING
Optimizing for Hardware
Kernel/OS Integration
Runtime Systems
Direct Hardware Access
Multi-core Processing

CORRECTNESS
Formal Methods
Formal Verification
Bug Prevention
Specification Testing
Proving Compiler Logic

TRADE-OFFS
Balancing Performance
Code Size
Compile Time
Memory Usage
Security

Conclusion

Function Callers



Problem: how to write a program that does this automatically?

LLVM Pass



Project Daedalus

A code compression t
the identification a
recurrent program sl

Evaluation

DCC

2007 programs total

Metric	Daedalus		Func. Merging		IROutliner	
	Affected	Geom.	Affected	Geom.	Affected	Geom.
Instcount	142	-0.24%	183	-0.35%	207	-0.65%
.text size	133	0.11%	190	0.39%	160	-0.19%
Exec. Time	161	0.06%	177	0.25%	160	-0.09%
Comp. Time	303	4.22%	241	2.06%	258	2.48%

3. Reuse shared logic

totals.js

```
export function calculateTotal(price, quantity, taxRate) {
  const subtotal = price * quantity;
  const tax = subtotal * taxRate;
  return subtotal + tax;
}
```

checkout.js

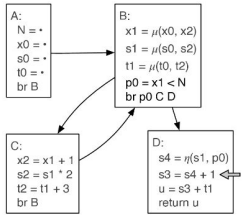
```
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
```

invoice.js

```
import { calculateTotal } from "./totals.js";
const total = calculateTotal(price, quantity, taxRate);
```

Duplicate code removed

Example - **Program Representation**



GSA form

```
01 int foo(int N) {
02   int x = 0;
03   int s = 1;
04   int t = 0;
05   while (x < N) {
06     x += 1;
07     s *= 2;
08     t += 3;
09   }
10   s = s + 1;
11   u = s = t;
12   return u;
13 }
```

What Compilers Teach You

DCC

GRAPH THEORY
Control-Flow Analysis
Data-Flow Analysis
Intermediate Representations (IRs)
Pattern Matching
Register Allocation

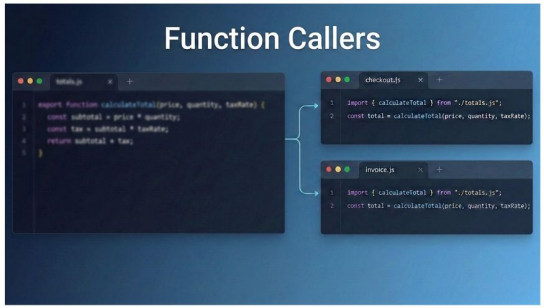
SYSTEMS PROGRAMMING
Optimizing for Hardware
Kernel/OS Integration
Runtime Systems
Direct Hardware Access
Multi-core Processing

CORRECTNESS
Formal Methods
Formal Verification
Bug Prevention
Specification Testing
Proving Compiler Logic

TRADE-OFFS
Balancing Performance
Code Size
Compile Time
Memory Usage
Security

Conclusion

Function Callers



Problem: how to write a program that does this automatically?

LLVM Pass



Project Daedalus
A code compression t
the identification a
recurrent program sl

Evaluat

2007 prog

Metric
Instcount
.text size
Exec. Time
Comp. Tim

Hardware Emulation

DCC

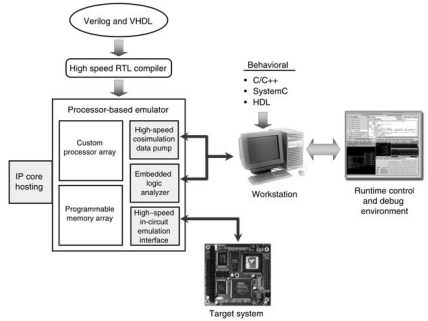


Image from [3]

References & Contact

DCC

Rafael Alvarenga de Azevedo

arafael@cadence.com

References

[1] Azevedo, Rafael Alvarenga de, et al. **Idempotent Slices with Applications to Code-Size Reduction**. arXiv:2603.09726, arXiv, 10 Mar. 2026. arXiv.org, https://doi.org/10.48550/arXiv.2603.09726.

[2] Kastner, Ryan, et al. **Arithmetic Optimization Techniques for Hardware and Software Design**. 1st ed., Cambridge University Press, 2010. DOI.org (Crossref), https://doi.org/10.1017/CBO9780511712180.

[3] Scheffer, Louis Kossuth, et al., editors. **EDA for IC System Design, Verification, and Testing**. CRC Taylor & Francis, 2006. Electronic Design Automation for Integrated Circuits Handbook.