Programs, Security and Games

Nikos Tzevelekos

Queen Mary, University of London

What this talk is about

Program security is a focal aspect of Security

Analyses boil down to producing rules on observable program behaviour

We will see how game semantics can be used to obtain accurate analyses

Program security: Why?

- Security very often concerns software
- Analyses can be very expressive
- Analyses supplemented with proofs/tools

Program security: What?

- Reachability
  - Safety, liveness
- Access control
  - resource access respects given policies
- Integrity
  - valuable data not changed undetectably
- Secrecy
  - secret information not revealed to some environment

Program security: What?

- Reachability
  - Safety, liveness
- Access control
  - resource access respects given policies
- Integrity
  - valuable data not changed undetectably
- Secrecy
  - secret information not revealed to some environment

Secrecy example

```
procedure sec{
   int h = HIGH;
   int l = 0;


   ...


   return l;
}
```

`sec` is **secure** if it returns the same value (`l`) for all possible values of `HIGH`

Higher-order example

```
int f(int x);

procedure sec{
   int h = HIGH;
   ...
   int g(int y){
    ...
    }
   return g;
}
```

`sec` is secure if ... ?

Higher-order example

```
int f(int x);

procedure sec{
   int h = HIGH;

   ...

   int g(int y){
     ...
     }
   return g;
}
```

`sec` is secure if ... ?

General case

P : a program depending on secret variable h.

P is secure for h if, for all v,v',

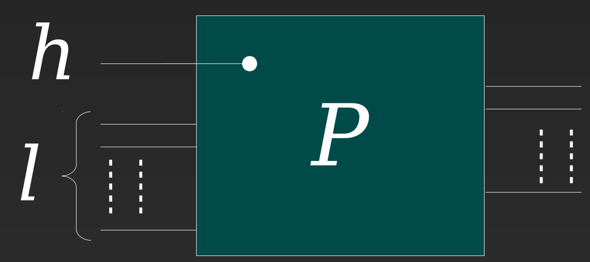
$$P(h=v) \simeq P(h=v')$$

General case

P : a program depending on secret variable h.

P is **secure** for h if, for all v,v',

$$P(h=v) \simeq P(h=v')$$



non-interference

Program semantics

What does $P(h=v) \simeq P(h=v')$ mean?

Program semantics

What does $P(h{=}v) \simeq P(h{=}v')$ mean?

- related to program meaning:
  - Syntax
  - Machine code
  - Input/output function
  - Abstract procedure
  - Logic rules
  - ...
  - Observable behaviour

Observational equivalence

$$P \simeq P'$$

- if P and P' have the same observable behaviour
- i.e. if, for any computational context $C(-)$,

$C(P)$ terminates $\Leftrightarrow$ $C(P')$ terminates

Capturing equivalence

How do we establish $P \simeq P'$?

Capturing equivalence

How do we establish $P \simeq P'$?

- Externally: look at all possible contexts $C(-)$
  - too many contexts – *infeasible*
- Internally: look at P
  - too many 'behaviours' – *incomplete*

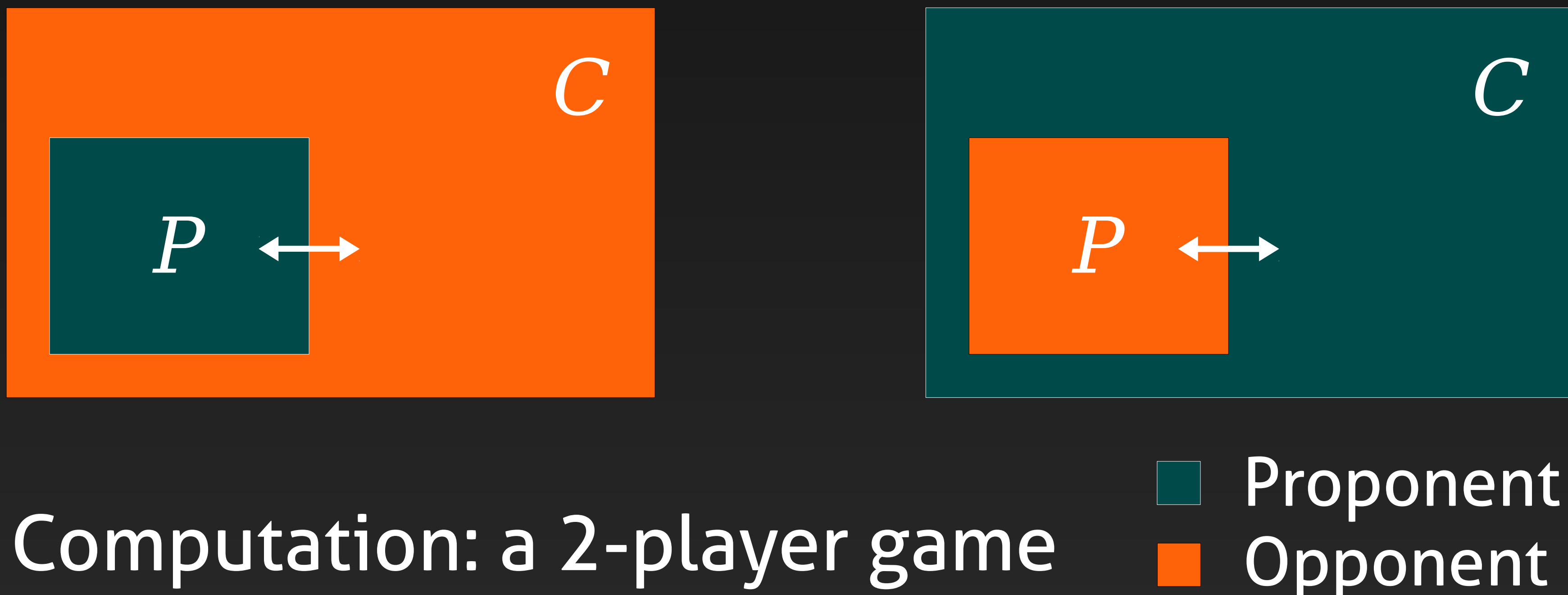
Computation as a game

Computation: a 2-player game

Proponent

Opponent

Computation as a game



Computation: a 2-player game

- Combines external with internal view
- into a single description

Full Abstraction

Full abstraction

Define a semantics function:

$$\mu : \text{Syntax} \longrightarrow U$$

such that:

$$P \simeq P' \iff \mu(P) = \mu(P')$$

Full abstraction

Define a semantics function:

$$\mu : \text{Syntax} \longrightarrow U$$

such that:

$$P \simeq P' \Leftrightarrow \mu(P) = \mu(P')$$

Game semantics is fully abstract

Game Semantics

- Computation is modelled as a 2-player game between:
  - *Opponent* (the environment)
  - *Proponent* (the program)

Example games

```
int f(int x){

   return x+1;
}
```

Example games

```
int f(int x){

  return x+1;
}
```

O	`call(f,5)`

Example games

```
int f(int x){

  return x+1;
}
```

O	`call(f,5)`
P	`ret(6)`

Example games

```
int f(int x){

  return x+1;
}
```

O	call(f,5)
P	ret(6)

O	call(f,6)
P	ret(7)

Example games

```
int f(int x){

  return x+1;
}
```

O	call(f,5)
P	ret(6)

O	call(f,6)
P	ret(7)

⋮

Example games

```
int f(int y);

int add1(int x){

   return f(x)+1;
}
```

Example games

```
int f(int y);

int add1(int x){

    return f(x)+1;
}
```

O	`call(add1,5)`

Example games

```
int f(int y);

int add1(int x){

   return f(x)+1;
}
```

O	call(add1,5)
P	call(f,5)

Example games

```
int f(int y);

int add1(int x){

   return f(x)+1;
}
```

O	`call(add1,5)`
P	`call(f,5)`
O	`ret(3)`

Example games

```
int f(int y);

int add1(int x){

   return f(x)+1;
}
```

O	`call(add1,5)`
P	`call(f,5)`
O	`ret(3)`
P	`ret(4)`

Example games

```
int f(int y);

int add1(int x){

   return f(x)+1;
}
```

O	call(add1,5)
P	call(f,5)
O	ret(3)
P	ret(4)

O	call(add1,6)
P	call(f,6)
O	ret(1)
P	ret(2)

⋮

Example games

```
int f(int y);

int add1(int x){

   return f(x)+1;
}
```

$c(i)$	O
$c_f(i)$	P
$r_f(j)$	O
$r(j+1)$	P
...	

O	`call(add1,5)`
P	`call(f,5)`
O	`ret(3)`
P	`ret(4)`

O	`call(add1,6)`
P	`call(f,6)`
O	`ret(1)`
P	`ret(2)`

⋮

Example games

```
int f(int y);

int add1a(int x){

   return ???;
}
```

O	`call(add1a,5)`
P	`call(f,5)`
O	`ret(3)`
P	`call(f,3)`
O	`ret(13)`
P	`ret(14)`

⋮

Example games

```
int f(int y);

int add1a(int x){

   return f(f(x))+1;
}
```

O	`call(add1a,5)`
P	`call(f,5)`
O	`ret(3)`
P	`call(f,3)`
O	`ret(13)`
P	`ret(14)`

$c(i)$	$c_f(i)$	$r_f(j)$	$c_f(j)$	$r_f(k)$	$r(k+1)$	...
O	P	O	P	O	P	

⋮

Composition

```
int f(int y);

int add1(int x){

   return f(x)+1;
}
```

Composition

```
int f(int x){
  return x+1;
}
int add1(int x){
  return f(x)+1;
}
```

Composition

f

O	call(f,5)
P	ret(6)

O	call(f,6)
P	ret(7)

add1

O	call(add1,5)
P	call(f,5)
O	ret(3)
P	ret(4)

O	call(add1,6)
P	call(f,6)
O	ret(1)
P	ret(2)

Composition

add1

	O	call(add1,5)
	P	
	O	
	P	

f

O	call(f,5)
P	ret(6)

O	call(f,6)
P	ret(7)

O	call(add1,6)
P	call(f,6)
O	ret(1)
P	ret(2)

Composition

add1

	O	call(add1,5)
call(f,5)	*P*	
	O	
	P	

f

O	call(f,5)
P	ret(6)

O	call(f,6)
P	ret(7)

O	call(add1,6)
P	call(f,6)
O	ret(1)
P	ret(2)

Composition

f

O	call(f,5)
P	ret(6)

add1

	O	call(add1,5)
call(f,5)	*P*	
	O	
	P	

O	call(f,6)
P	ret(7)

O	call(add1,6)
P	call(f,6)
O	ret(1)
P	ret(2)

Composition

f

O	call(f,5)
P	ret(6)

add1

	O	call(add1,5)
call(f,5)	*P*	
ret(6)	*O*	
	P	

O	call(add1,6)
P	call(f,6)
O	ret(1)
P	ret(2)

O	call(f,6)
P	ret(7)

Composition

f

O	call(f,5)
P	ret(6)

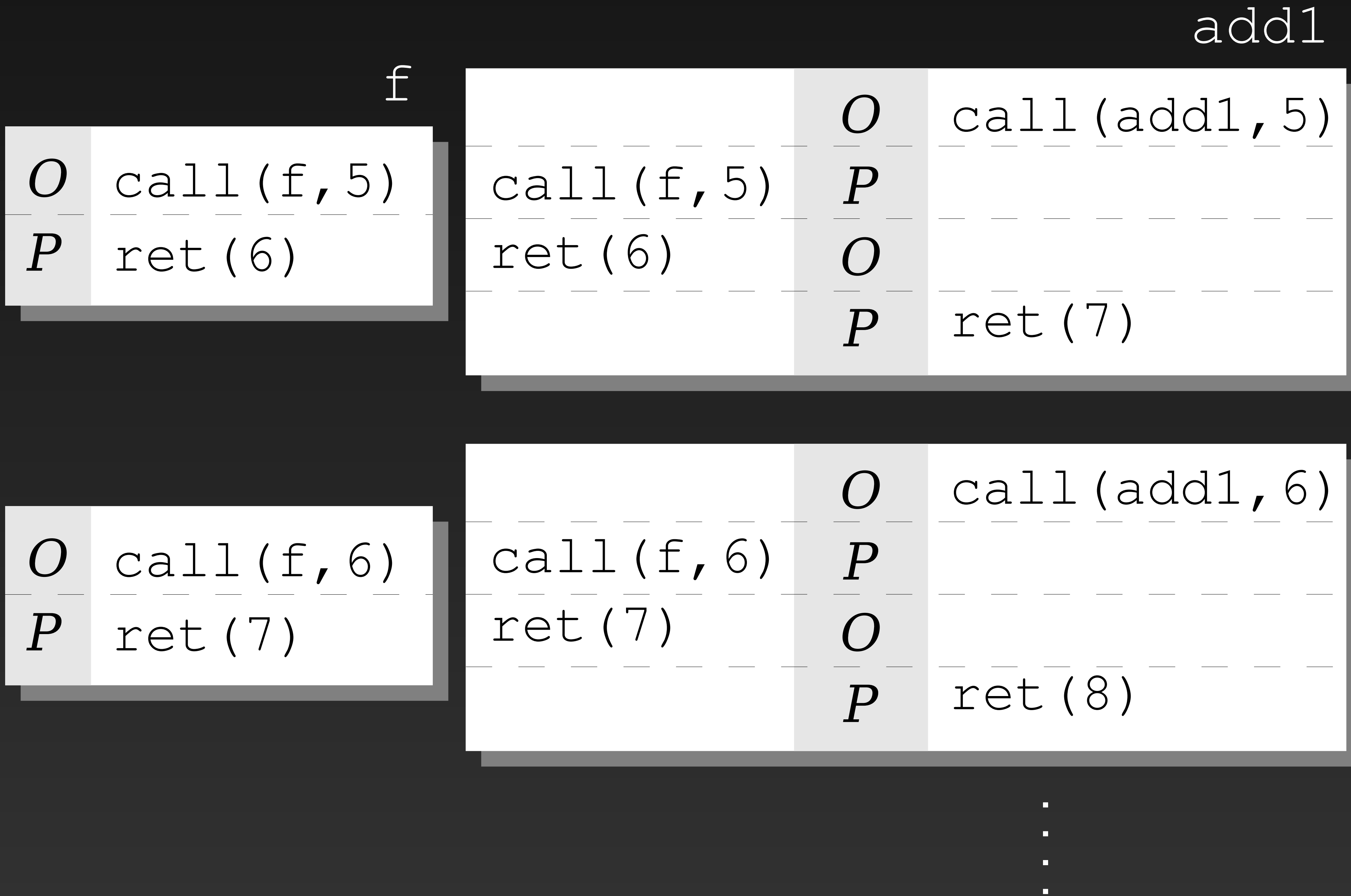
add1

	O	call(add1,5)
call(f,5)	*P*	
ret(6)	*O*	
	P	ret(7)

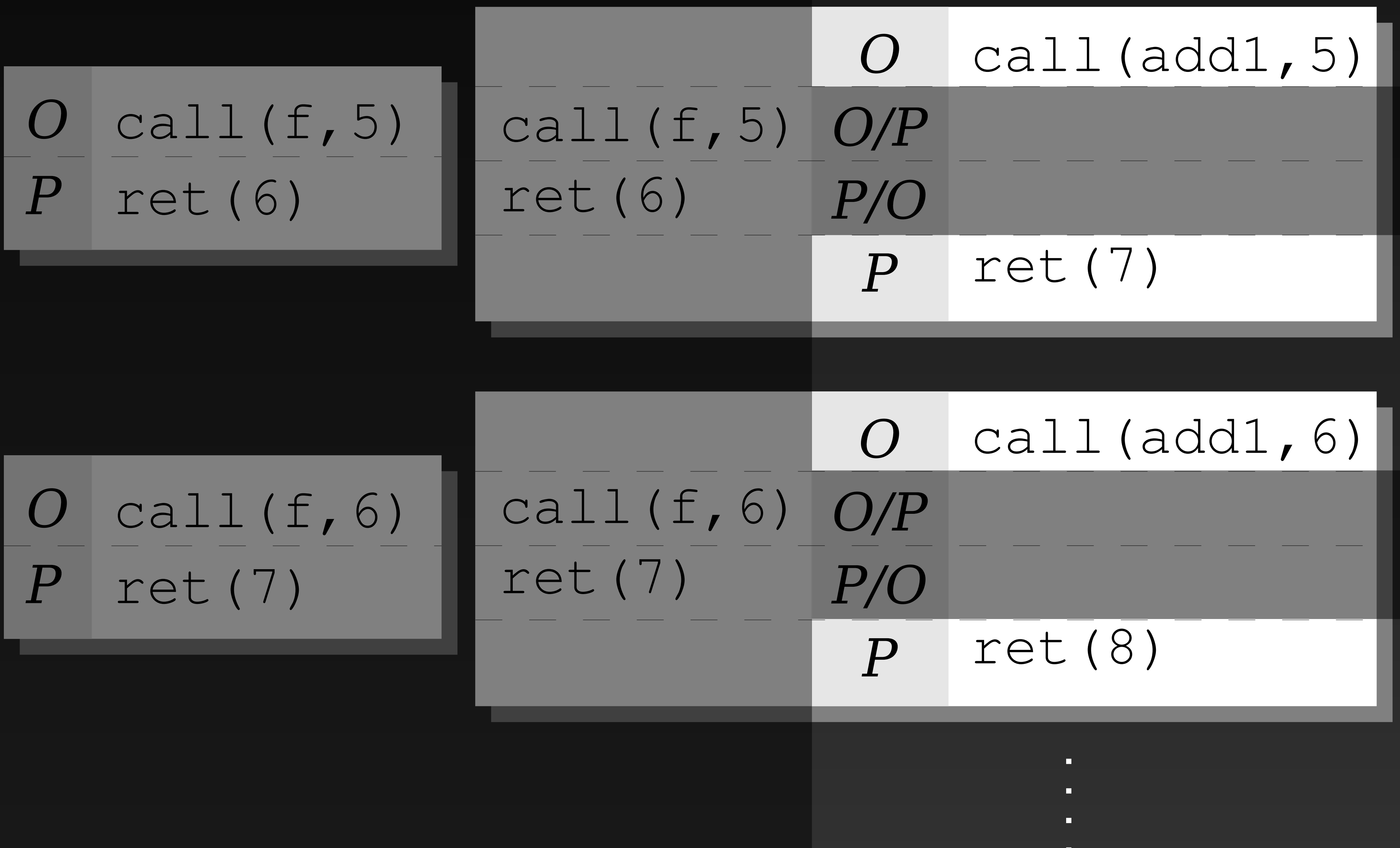
O	call(f,6)
P	ret(7)

O	call(add1,6)
P	call(f,6)
O	ret(1)
P	ret(2)

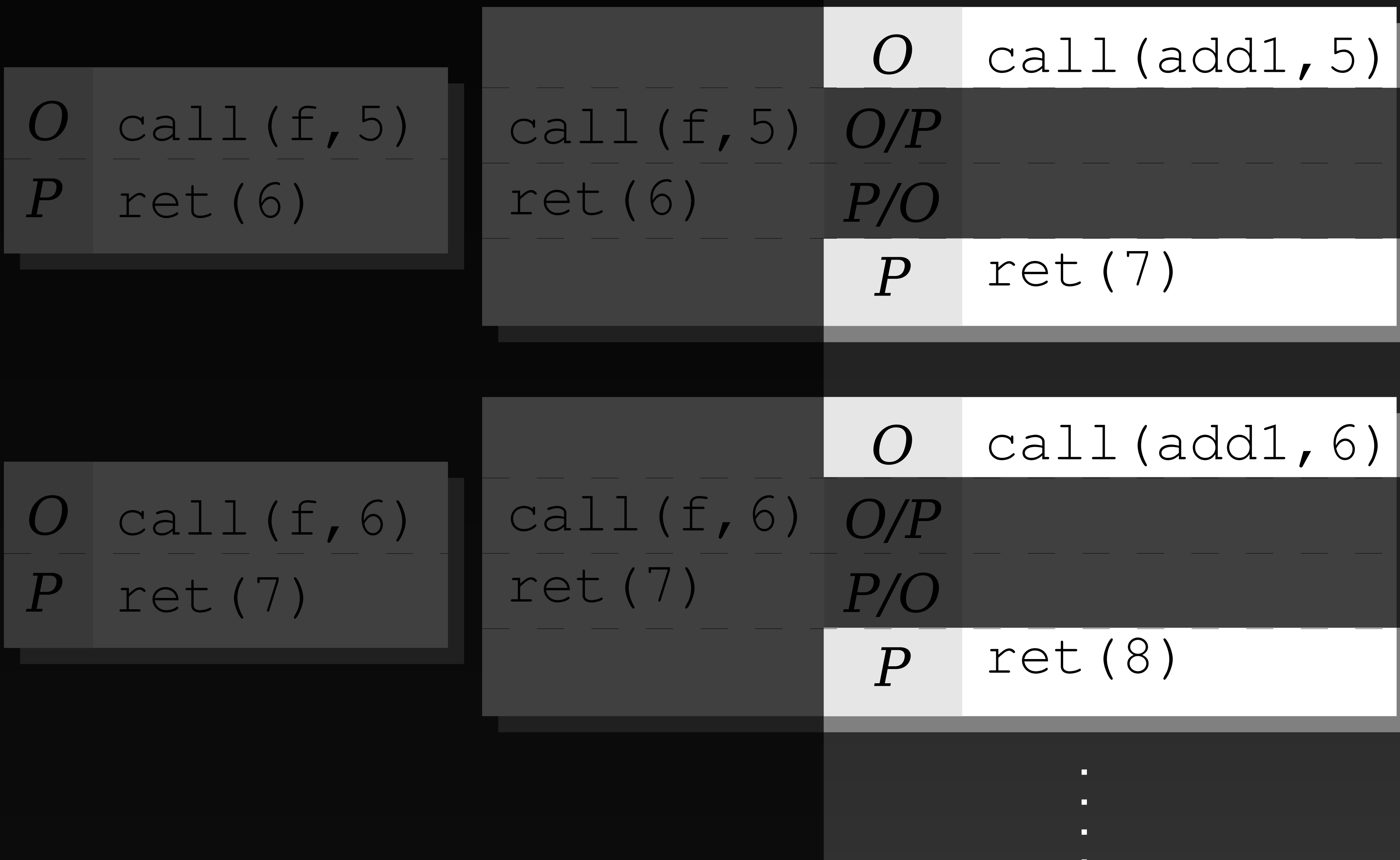
Composition



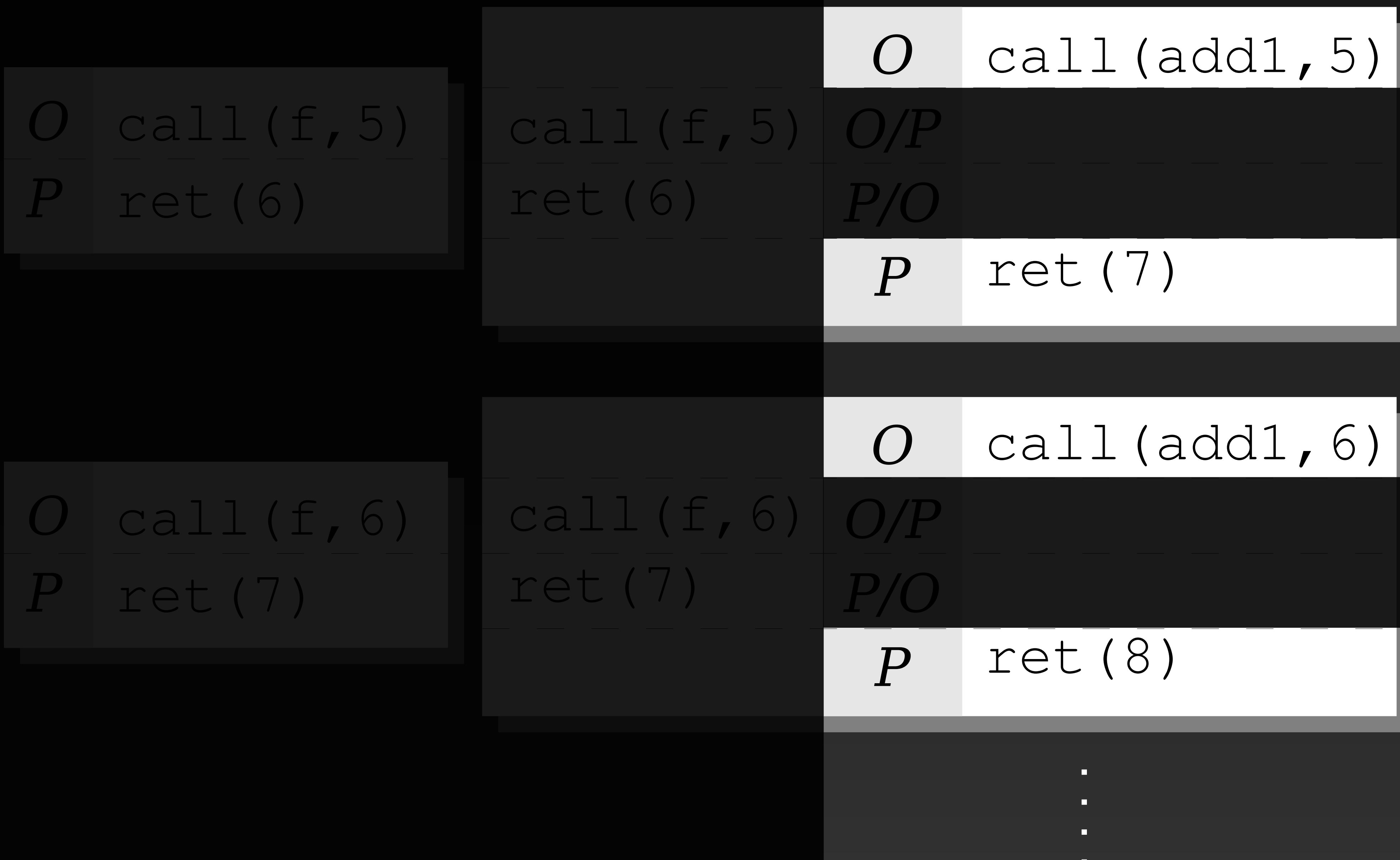
Composition



Composition



Composition



Composition

O	`call(add1,5)`
P	`ret(7)`

O	`call(add1,6)`
P	`ret(8)`

⋮

Composition

```
int g(int x){
  return x+2;
}
```

O	call(add1,5)
P	ret(7)

O	call(add1,6)
P	ret(8)

⋮

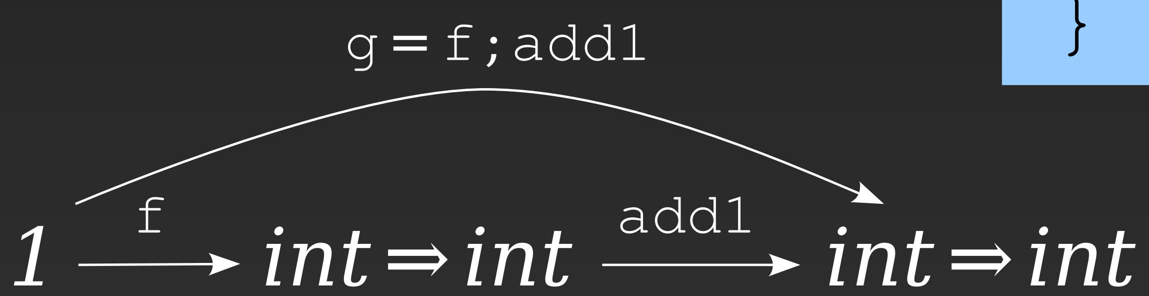
c(i) r(i+2) ...
O P

Composition

```
int g(int x){

  return x+2;
}
```

```
int f(int x){

  return x+1;
}
```

```
int add1(int x){

   return f(x)+1;
}
```



Game Semantics

- Computation is modelled as a 2-player game between:
  - *Opponent* (the environment)
  - *Proponent* (the program)
- Qualitative games
- Programs $\longmapsto$ *strategies* for Proponent
- Families (i.e. *categories*) of games

Story so far

- Pure functions
- Integer/ HO state
- Non-det./ probability
- Exceptions/ control
- Recursive types
- Polymorphism
- Names

- Algorithmic games
- Abstract interpretation
- Control-flow analysis

- Access control
- Name flow as IF (Information Flow)

- full abstraction
- program analysis
- security

Access control

```
int f(int y);

int sec(){
    int h=HIGH;

    return f(h)+1;
}
```

Access control

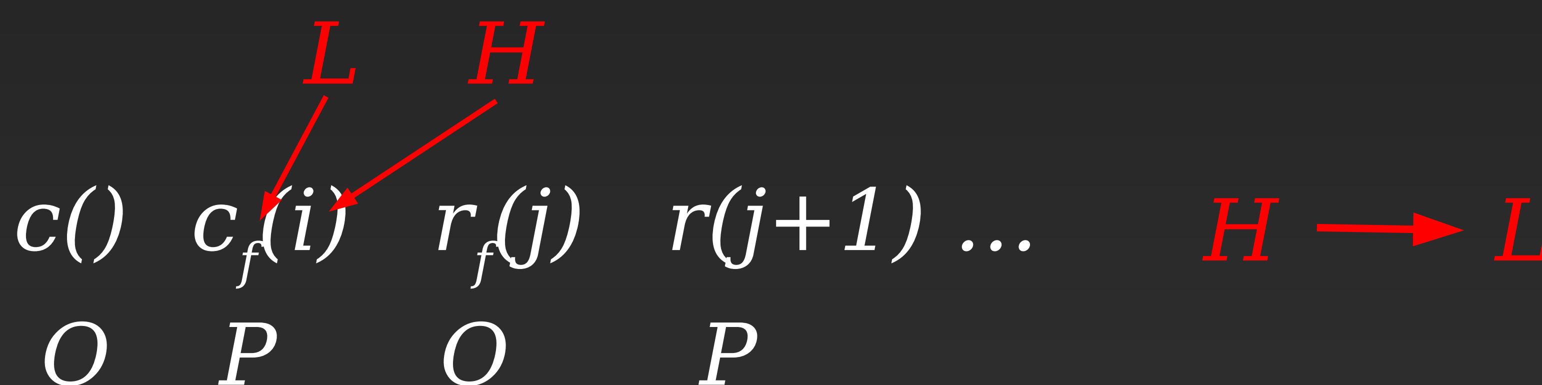
```
int f(int y);

int sec(){
    int h=HIGH;

    return f(h)+1;
}
```

$c()\quad c_f(i)\quad r_f(j)\quad r(j+1)\ \ldots$

$O\quad P\quad O\quad P$

Access control

```
int f(int y);

int sec(){
    int h=HIGH;

    return f(h)+1;
}
```



Name flow

```
int sec( name => int f ){
    name h1=HIGH1;
    name h2=HIGH2;

    return (f(h1)==f(h2))
}
```

Name flow

```
int sec( name=>int f ){
    name h1=HIGH1;
    name h2=HIGH2;

    return (f(h1)==f(h2))
}
```

- like int's
- only "=="

Name flow

```
int sec( name=>int f ){
    name h1=HIGH1;
    name h2=HIGH2;

    return (f(h1)==f(h2))
}
```

- the secret is: `HIGH1=HIGH2`
- can it be guessed without storing names?

Name flow

```
int sec( name=>int f ){
   name h1=HIGH1;
   name h2=HIGH2;

   return (f(h1)==f(h2))
}
```

$c(f)$ $c_f(n_1)$ $r_f(b_1)$ $c_f(n_2)$ $r_f(b_2)$ $r(b)$

O P O P O P

Name flow

```
int sec( name=>int f ){
   name h1=HIGH1;
   name h2=HIGH2;

   return (f(h1)==f(h2))
}
```

$b = (b_1 = b_2)$

$c(f)$ $c_f(n_1)$ $r_f(b_1)$ $c_f(n_2)$ $r_f(b_2)$ $r(b)$

O P O P O P

Name flow

```
int sec( name=>int f ){
    name h1=HIGH1;
    name h2=HIGH2;

    return (f(h1)==f(h2))
}
```

$b=(b_1=b_2)$

$c(f)$ $c_f(n_1)$ $r_f(b_1)$ $c_f(n_2)$ $r_f(b_2)$ $r(b)$

O P O P O P

Name flow

```
int sec( name=>int f ){
    name h1=HIGH1;
    name h2=HIGH2;

    return (f(h1)==f(h2))
}
```

$b=(b_1=b_2)$

$c(f)\ \ c_f(n_1)\ \ r_f(b_1)\ \ c_f(n_2)\ \ r_f(b_2)\ \ r(b)$

$c(f)\ \ c_f(n_1)\ \ c(f_1)\ \ c_{f_1}(n_1)\ \ \ldots$

$O\ \ P\ \ O\ \ P\ \ O\ \ P$

Name flow

```
int sec( name=>int f ){
    name h1=HIGH1;
    name h2=HIGH2;

    return (f(h1)==f(h2))
}
```

```
int i=0;

int f(name x){
    int f1(name y){
        if (x==y) return 0;
        i=1; return 1;
    }
    return sec(f1);
}
```

$c(f)$ $c_f(n_1$

$c(f)$ $c_f(n_1$

O P

What's next

Pure functions
Integer/ HO state
Non-det./ probability
Exceptions/ control
Recursive types
Polymorphism
Names

Algorithmic games

Abstract interpretation

Control-flow analysis

Access control

Name flow as IF

Quantitative IF

- full abstraction
- program analysis
- security

Further reading

- Samson Abramsky, Radha Jagadeesan: *Game Semantics for Access Control*. MFPS 2009: 135-156.
- Dan R. Ghica: *Applications of Game Semantics: From Program Analysis to Hardware Synthesis.* LICS 2009: 17-26.
- Pasquale Malacaria, Chris Hankin: *Non-Deterministic Games and Program Analysis: An Application to Security.* LICS 1999: 443-452.
- Nikos Tzevelekos: *Program equivalence in a simple language with state.* Computer Languages, Systems & Structures, 38(2): 181–198, 2012.