中国科学技术大学
University of Science and Technology of China

# More on Lambda Calculus

## 《程序语言设计和程序分析》


张昱

0551-63603804，yuzhang@ustc.edu.cn

中国科学技术大学

计算机科学与技术学院

# Non-terminating reduction



$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$

$\rightarrow (\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$

$\rightarrow \ldots$

$(\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x))$

$\rightarrow f\ ((\lambda x.\ f\ (x\ x))\ (\lambda x.\ f\ (x\ x)))$

$\rightarrow \ldots$

$(\lambda x.\ x\ x\ y)\ (\lambda x.\ x\ x\ y)$

$\rightarrow (\lambda x.\ x\ x\ y)\ (\lambda x.\ x\ x\ y)\ y$

$\rightarrow \ldots$

# Terminating & non-terminating

**Term may have both terminating and non-terminating reduction sequences**

$(\lambda u.\ \lambda v.\ v)\ ((\lambda x.\ x\ x)(\lambda x.\ x\ x))$

$\rightarrow \lambda v.\ v$

$(\lambda u.\ \lambda v.\ v)\ ((\lambda x.\ x\ x)(\lambda x.\ x\ x))$

$\rightarrow (\lambda u.\ \lambda v.\ v)\ ((\lambda x.\ x\ x)(\lambda x.\ x\ x))$

$\rightarrow \ldots$

# Reduction strategies



- **Normal-order** reduction: choose the *left-most, outer-most* redex first

  $(\lambda u.\ \lambda v.\ v)\ ((\lambda x.\ x\ x)(\lambda x.\ x\ x))$
  $\rightarrow \lambda v.\ v$

  *Normal-order reduction will find normal form if exists*

- **Applicative-order** reduction: choose the *left-most, inner-most* redex first

  $(\lambda u.\ \lambda v.\ v)\ ((\lambda x.\ x\ x)(\lambda x.\ x\ x))$
  $\rightarrow (\lambda u.\ \lambda v.\ v)\ ((\lambda x.\ x\ x)(\lambda x.\ x\ x))$
  $\rightarrow \ldots$

# Reduction strategies

中国科学技术大学
University of Science and Technology of China

## Examples

***Normal-order***

$(\lambda x.\ x\ x)\ ((\lambda y.\ y)\ (\lambda z.\ z))$

$\rightarrow ((\lambda y.\ y)\ (\lambda z.\ z))\ ((\lambda y.\ y)\ (\lambda z.\ z))$

$\rightarrow (\lambda z.\ z)\ ((\lambda y.\ y)\ (\lambda z.\ z))$

$\rightarrow (\lambda y.\ y)\ (\lambda z.\ z)$

$\rightarrow \lambda z.\ z$

***Applicative-order***

$(\lambda x.\ x\ x)\ ((\lambda y.\ y)\ (\lambda z.\ z))$

$\rightarrow (\lambda x.\ x\ x)\ (\lambda z.\ z)$

$\rightarrow (\lambda z.\ z)\ (\lambda z.\ z)$

$\rightarrow \lambda z.\ z$

# Reduction strategies

## Examples

Applicative-order may ***not*** be as efficient as normal-order when the argument is not used.

***Normal-order***

$(\lambda x.\ p)\ ((\lambda y.\ y)\ (\lambda z.\ z))$

$\rightarrow p$

***Applicative-order***

$(\lambda x.\ p)\ ((\lambda y.\ y)\ (\lambda z.\ z))$

$\rightarrow (\lambda x.\ p)\ (\lambda z.\ z)$

$\rightarrow p$

# Reduction strategies



- Similar to (but subtly different from) *evaluation strategies* in language theories
  - Call-by-name (like normal-order)
    - ALGOL 60
  - Call-by-need ("memorized version" of call-by-name)
    - Haskell, R, …
  - Call-by-value (like applicative-order)
    - C, …
  - …

arguments are not evaluated, but directly substituted into function body

*called "lazy evaluation"*

*called "eager evaluation"*

# Main points till now



- **Syntax**: notation for defining functions
  - "Pure": without adding any additional syntax

    (Terms) M, N ::= x | λx. M | M N

- **Semantics** (reduction rules)

  $(\lambda x.\ M)\ N \rightarrow [N/x]M \quad (\beta)$

- Next: programming in "pure" λ-calculus
  - Encoding data and operators

# Programming in λ-calculus



- **Encoding Boolean values and operators**
  - **True ≡ λx. λy. x**
  - **False ≡ λx. λy. y**

# Programming in λ-calculus

## Encoding Boolean values and operators

- **True ≡ λx. λy. x**
- **False ≡ λx. λy. y**
- **not ≡ λb. b False True**

not True
→ True False True
→ False

not False
→ False False True
→ True

# Programming in λ-calculus

## Encoding Boolean values and operators

- True ≡ λx. λy. x
- False ≡ λx. λy. y
- not ≡ λb. b False True
- and ≡ λb. λb'. b b' False

and True b
→* True b False
→ b

and False b
→* False b False
→ False

# Programming in λ-calculus

## Encoding Boolean values and operators

- True ≡ λx. λy. x
- False ≡ λx. λy. y
- not ≡ λb. b False True
- and ≡ λb. λb'. b b' False
- or ≡ λb. λb'. b True b'

or True b
→* True True b
→ True

or False b
→* False True b
→ b

# Programming in λ-calculus

## Encoding Boolean values and operators

- True ≡ λx. λy. x
- False ≡ λx. λy. y
- not ≡ λb. b False True
- and ≡ λb. λb'. b b' False
- or ≡ λb. λb'. b True b'
- if b then M else N ≡ b M N

*Not unique encoding*

# Programming in λ-calculus

## Encoding Boolean values and operators

- True ≡ λx. λy. x
- False ≡ λx. λy. y
- not ≡ λb. b False True
- and ≡ λb. λb'. b b' False
- or ≡ λb. λb'. b True b'
- if b then M else N ≡ b M N
- not' ≡ λb. λx. λy. b y x

not' True
→ λx. λy. True y x
→ λx. λy. y = False

not' False
→ λx. λy. False y x
→ λx. λy. x = True

# Programming in λ-calculus

## Church numerals

- $\underline{0} \equiv \lambda f.\ \lambda x.\ x$ *(the same as False!)*
- $\underline{1} \equiv \lambda f.\ \lambda x.\ f\ x$
- $\underline{2} \equiv \lambda f.\ \lambda x.\ f\ (f\ x)$
- $\underline{n} \equiv \lambda f.\ \lambda x.\ f^n\ x$

# Programming in λ-calculus

## Church numerals

- $\underline{0} \equiv \lambda f.\ \lambda x.\ x$ *(the same as False!)*
- $\underline{1} \equiv \lambda f.\ \lambda x.\ f\ x$
- $\underline{2} \equiv \lambda f.\ \lambda x.\ f\ (f\ x)$
- $\underline{n} \equiv \lambda f.\ \lambda x.\ f^n\ x$
- $\mathbf{succ} \equiv \lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x)$

succ $\underline{n}$
$\rightarrow \lambda f.\ \lambda x.\ f\ (\underline{n}\ f\ x)$
$= \lambda f.\ \lambda x.\ f\ ((\lambda f.\ \lambda x.\ f^n\ x)\ f\ x)$
$\rightarrow \lambda f.\ \lambda x.\ f\ (f^n\ x)$
$= \lambda f.\ \lambda x.\ f^{n+1}\ x$
$= \underline{n+1}$

# Programming in λ-calculus

## Church numerals

- $\underline{0} \equiv \lambda f.\ \lambda x.\ x$ *(the same as False!)*
- $\underline{1} \equiv \lambda f.\ \lambda x.\ f\ x$
- $\underline{2} \equiv \lambda f.\ \lambda x.\ f\ (f\ x)$
- $\underline{n} \equiv \lambda f.\ \lambda x.\ f^n\ x$
- $succ \equiv \lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x)$
- $succ' \equiv \lambda n.\ \lambda f.\ \lambda x.\ n\ f\ (f\ x)$

# Programming in λ-calculus

## Church numerals

- $\underline{0} \equiv \lambda f.\ \lambda x.\ x$
- $\underline{1} \equiv \lambda f.\ \lambda x.\ f\ x$
- $\underline{2} \equiv \lambda f.\ \lambda x.\ f\ (f\ x)$
- $\underline{n} \equiv \lambda f.\ \lambda x.\ f^n\ x$
- $\text{succ} \equiv \lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x)$
- $\text{iszero} \equiv \lambda n.\ \lambda x.\ \lambda y.\ n\ (\lambda z.\ y)\ x$

iszero $\underline{0}$
$\rightarrow \lambda x.\ \lambda y.\ \underline{0}\ (\lambda z.\ y)\ x$
$= \lambda x.\ \lambda y.\ (\lambda f.\ \lambda x.\ x)\ (\lambda z.\ y)\ x$
$\rightarrow \lambda x.\ \lambda y.\ (\lambda x.\ x)\ x$
$\rightarrow \lambda x.\ \lambda y.\ x$ = True

iszero $\underline{1}$
$\rightarrow \lambda x.\ \lambda y.\ \underline{1}\ (\lambda z.\ y)\ x$
$= \lambda x.\ \lambda y.\ (\lambda f.\ \lambda x.\ f\ x)\ (\lambda z.\ y)\ x$
$\rightarrow \lambda x.\ \lambda y.\ (\lambda x.\ (\lambda z.\ y)\ x)\ x$
$\rightarrow \lambda x.\ \lambda y.\ ((\lambda z.\ y)\ x)$
$\rightarrow \lambda x.\ \lambda y.\ y$ = False

iszero (succ $\underline{n}$) $\rightarrow^*$ False

# Programming in λ-calculus

## Church numerals

- $\underline{0} \equiv \lambda f.\ \lambda x.\ x$
- $\underline{1} \equiv \lambda f.\ \lambda x.\ f\ x$
- $\underline{2} \equiv \lambda f.\ \lambda x.\ f\ (f\ x)$
- $\underline{n} \equiv \lambda f.\ \lambda x.\ f^n\ x$
- $succ \equiv \lambda n.\ \lambda f.\ \lambda x.\ f\ (n\ f\ x)$
- $iszero \equiv \lambda n.\ \lambda x.\ \lambda y.\ n\ (\lambda z.\ y)\ x$
- $add \equiv \lambda n.\ \lambda m.\ \lambda f.\ \lambda x.\ n\ f\ (m\ f\ x)$
- $mult \equiv \lambda n.\ \lambda m.\ \lambda f.\ n\ m\ f$

# Programming in λ-calculus

- **Booleans**
- **Natural numbers**
- **Pairs**
- **Lists**
- **Trees**
- **Recursive functions**
- **…**

***Read supplementary materials: A***

中国科学技术大学
University of Science and Technology of China

# Lambda Expression in C++

Syntax Changes from C++11 to C++20

Lambdas: From C++11 to C++20, Part1 Part2

https://www.open-std.org/jtc1/sc22/wg21/

# C++ Lambda

## Since C++ 2011-N3242, N3337

**5.1 Primary expressions , 5.1.2 Lambda expressions** [expr.prim.lambda]

多个可选项

```
[capture list] (params list) mutable exception-> return type
{ function body }
```

可以修改按值传入的变量的副本（不会将改变传递到lambda表达式之外）

可以抛出指定类型的异常

指示定义的匿名函数的返回值类型

```
#include <algorithm>
#include <cmath>
#include <array>
#include <iostream>
#include <string_view>
void abssort(int *x, unsigned N) {
  std::sort(x, x + N,
    [](int a, int b) {
      return std::abs(a) < std::abs(b);
    });
}
```

**捕获值列表**
能捕获的是在当前作用域可访问的值，允许这些值在lambda体中直接使用

# C++ Lambda

## Since C++ 2011-N3242, N3337

### 5.1 Primary expressions , 5.1.2 Lambda expressions

```
[capture list] (params list) mutable exception-> return type
{ function body }
```

```
#include <algorithm>
#include <cmath>
#include <array>
#include <iostream>
#include <string_view>
void abssort(int *x, unsigned N) {
  std::sort(x, x + N,
    [](int a, int b) {
      return std::abs(a) < std::abs(b);
    });
}
```

https://www.open-std.org/jtc1/sc22/wg21/

```
int main()
{
    std::array<int, 10> s =
      {5, 7, -4, 2, 8, -6, 1, -9, 0, 3};
    auto print = [&s]
      (std::string_view const rem) {
        for (auto a : s) {
            std::cout << a << ' ';
        }
        std::cout << ": " << rem << '\n';
    };
    stdabssort(s.begin(), 10);
}
```

捕获值列表

# Lambda Capture List



## C++2011: Lambda-capture: & | = | id | & id | this

- Captured by value
- Captured by reference
- Captured implicitly
  - [=] captured by value
  - [&] captured by reference
- Mixed
  - [&, x]captured by reference implicitly, but x by value
  - [=, &y] captured by value implicitly, but y by reference

```
int x = 0; int y = 42;
auto qqq = [x, &y] {
  std::cout << "x: " << x << std::endl;
  std::cout << "y: " << y << std::endl;
  ++y; ++x;
};
qqq();  // x==0, y==43
```

```
int x = 0; int y = 42;
auto qqq = [x, &y]() mutable {
++y; ++x;
};
qqq();  // x==0, y==43
```

# Realization of Lambda

中国科学技术大学
University of Science and Technology of China

https://godbolt.org/

https://cppinsights.io/

## The compiler: lambda➔ real class

```
#include <algorithm>
#include <cmath>
#include <array>
#include <iostream>
#include <string_view>
void abssort(int *x, unsigned N) {
  std::sort(x, x + N,
    [](int a, int b) {
      return std::abs(a) < std::abs(b);
    });
}
```

```
void abssort(int * x, unsigned int N)
{

  class __lambda_8_6
  {
    public:
    inline bool operator()(int a, int b) const
    {
      return abs(a) < abs(b);
    }

    using retType_8_6 = bool (*)(int, int);
    inline operator retType_8_6 () const noexcept
    {
      return __invoke;
    };

    private:
    static inline bool __invoke(int a, int b)
    {
      return __lambda_8_6{}.operator()(a, b);
    }

    public:
    // inline /*constexpr */ __lambda_8_6(const __lambda_8_6 &) noexcept = default;
    // inline /*constexpr */ __lambda_8_6(__lambda_8_6 &&) noexcept = default;
    // inline __lambda_8_6 & operator=(const __lambda_8_6 &) /* noexcept */ = delete;

  };

  std::sort(x, x + N, __lambda_8_6(__lambda_8_6{}));
}
```

# C++ Lambda Evolution

- **C++ 2014 – N3797, N4140** [expr.prim.lambda]
  - **Capture: & | = | id [*init*] | & id [*init*] | this**
    - Introduce capture initializer
  - **Parameter list can take auto arguments**
    - Introduce generic lambda

```
int x = 4;
auto y = [&r = x, x = x+1]()->int {
    r += 2;
    return x+2;
  }(); // Updates ::x to 6, and initializes y to 7.
auto foo = [](auto x, auto y) {/* … */}
```

# C++ Lambda Evolution

- C++ 2017 – N4659
  - Capture：add *this P0018r3 (2016.3.4)

**Issue: Lambda expressions cannot capture `*this` by value**

Lambda expressions declared within a non-static member function explicilty or implicitly captures the `this` pointer to access to member variables of `this`. Both capture-by-reference `[&]` and capture-by-value `[=]` *capture-defaults* implicitly capture the `this` pointer, therefore member variables are always accessed by reference via `this`. Thus the capture-default has no effect on the capture of `this`.

如何捕获this对象的成员变量？

```
struct S {
        int x ;
        void f() {
                // The following lambda captures are currently identical
                auto a = [&]() { x = 42 ; } // OK: transformed to (*this).x
                auto b = [=]() { x = 43 ; } // OK: transformed to (*this).x
                a();
                assert( x == 42 );
                b();
                assert( x == 43 );
        }
};
```

# C++ Lambda Evolution



## C++ 2017 – N4659

### Capture：add *this P0018r3 (2016.3.4)

Asynchronous dispatch of closures is a cornerstone of parallelism and concurrency. When a lambda is asynchronously dispatched from within a non-static member function, via `std::async` or other concurrency / parallelism dispatch mechanism the `*this` object *cannot* be captured by value Thus when the `std::future` (or other handle) to the dispatched lambda outlives the originating class the lambda's captured `this` pointer is invalid.

```
class Work {
private:
        int value ;
public:
        Work() : value(42) {}
        std::future spawn()
        { return std::async( [=]()->int{ return value ; }); }
};

std::future foo()
{
        Work tmp ;
        return tmp.spawn();
        // The closure associated with the returned future
        // has an implicit this pointer that is invalid.
}

int main()
{
        std::future f = foo();
        f.wait();
        // The following fails due to the
        // originating class having been destroyed
        assert( 42 == f.get() );
        return 0 ;
}
```

返回一个异步future (异步分派的lambda，其中使用了this.value)

foo()返回上面定义的异步分派lambda (但这里this指针失效)

由于Work实例已不存在，这里试图获取this.value失败

# C++ Lambda Evolution

## C++ 2017 – N4659

### Capture：add *this P0018r3 (2016.3.4)

Current and future hardware architectures specifically targeting parallelism and concurrency have heterogeneous memory systems. For example, NUMA regions, attached accelerator memory, and processing-in-memory (PIM) stacks. In these architectures it will often result in significantly improved performance if the closure is copied to the data upon which it operates, as opposed to moving the data to and from the closure.

在并行架构上，可通过将闭包复制到要操作的数据上来改善性能，
而不是将数据从单个闭包移入或移出

For example, parallel execution of a closure on large data spanning NUMA regions will be more performant if a copy of that closure residing in the same NUMA region acts upon that data. If a full (self-contained) capture-by-value lambda closure were given to a parallel dispatch, such as in the parallelism technical specification, then the library could create copies of that closure within each NUMA region to improve data locality for the parallel computation. For another example, a closure dispatched to an attached accelerator with separate memory must be copied to the accelerator's memory before execution can occur. Thus current and future architectures *require* the capability to copy closures to data.

并行架构需要编程语言提供复制闭包给数据的能力

# C++ Lambda Evolution



## C++ 2017 – N4659

### Capture: add *this P0018r3 (2016.3.4)

**Error-prone and onerous work-around:** `[=, tmp=*this]`

A potential work-around for this deficiency is to explicitly capture a copy the originating class.

```
class Work {
private:
        int value ;
public:
        Work() : value(42) {}
        std::future spawn()
        {
                return std::async( [=,tmp=*this]()->int{ return tmp.value ; });
        }
};
```

按值捕获并且初始化为*this，本质上是复制原始的类实例

➔低效

```
class Work {
public:
        void do_something() const {
                // for ( int i = 0 ; i < N ; ++i )
                foreach( Parallel , 0 , N , [=,tmp=*this]( int i )
                {
                        // A modestly long loop body where
                        // every reference to a member must be modified
                        // for qualification with 'tmp.'
                        // Any mistaken omissions will silently fail
                        // as references via 'this->'.
                }
                );
        }
};
```

繁重的

必须修改代码中成员的每个引用以添加 tmp.限定条件

# Capture *this



## 解决对策：引入*this

```
class Task {
private:
  double a; double b;
public:
  Task() : a(0.0), b(0.0) {}
  ~Task() { a = std::nan(""); b = std::nan(""); }
  void set_a(double a) {this.a = a; }
  void set_b(double b) {this.b = b; }
  std::future<double> run() {
    return std::async(std::launch::async, [=, *this]() {
      return std::sqrt(a* a + b * b);
    });
  }
};
```

```
int main() {
  Task t;
  t.set_a(5.0);
  t.set_b(10.0);
  std::cout <<t.run().get();
}
```

异步任务：可能在Task对象销毁后被执行

Captured the object pointed by this by value

# C++ Core Guidelines



- lambda expression: when to use
  - F.50: Use a lambda when a function won't do (to capture local variables, or to write a local function)
  - F.52: Prefer capturing by reference in lambdas that will be used locally, including passed to algorithms
  - F.53: Avoid capturing by reference in lambdas that will be used non-locally, including returned, stored on the heap, or passed to another thread
  - ES.28: Use lambdas for complex initialization, especially of const variables

- **C.170: If you feel like overloading a lambda, use a generic lambda**

```
void f(int);
void f(double);
auto f = [](char); // error: cannot overload variable and function

auto g = [](int) { /* ... */ };
auto g = [](double) { /* ... */ }; // error: cannot overload variables

auto h = [](auto) { /* ... */ }; // OK
```


中国科学技术大学
University of Science and Technology of China

# Lambda Expression in Python

## 4.7.6. Lambda Expressions

- Small anonymous functions can be created with the lambda keyword.

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)  42
>>> f(1)  43
```

```
>>> pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
>>> pairs.sort(key=lambda pair: pair[1])
>>> pairs
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
```


中国科学技术大学
University of Science and Technology of China

# Others

# LAMBDA- Excel

## LAMBDA: The ultimate Excel worksheet function

- =LAMBDA( X, Y, SQRT( X*X+Y*Y ) )

- =LAMBDA( X, Y, LET( XS, X*X, YS, Y*Y, SQRT( XS+YS ) ) )

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | HEAD | =LAMBDA(str, IF(str="","error: HEAD of empty string", LEFT(str, 1))) | | | |
| 2 | TAIL | =LAMBDA(str, IF(str="","error: TAIL of empty string", RIGHT(str, LEN(str)-1))) | | | |
| 3 | REVERSE | =LAMBDA(str, IF(str="", "", REVERSE(TAIL(str)) & HEAD(str)))) | | | |
| 4 | | | | | |
| 5 | 12345 | 1 | 2345 | 54321 | 12345 |
| 6 | | =HEAD(A5) | =TAIL(A5) | =REVERSE(A5) | =REVERSE(D5) |