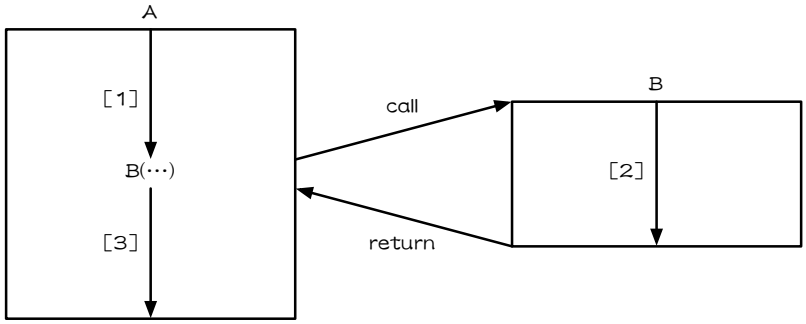
京都産業大学　コンピュータ理工学部　CS・NM・IS学科　特別研究
2012年2月22日(水)　第4・5時限　65実験室

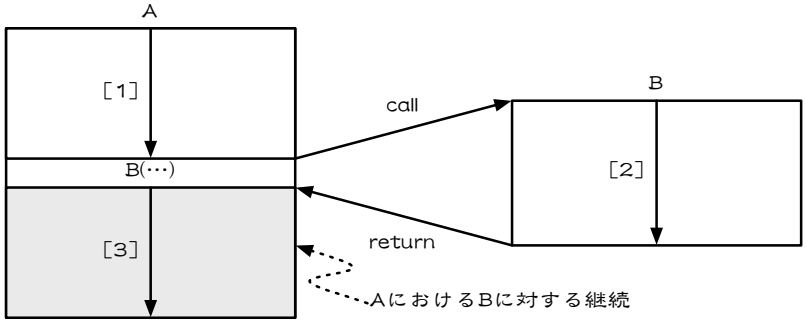
継続

(コンティニュエーション：処理過程の未来像)

青木 淳
aokisan@cb3.so-net.ne.jp
http://www.cc.kyoto-su.ac.jp/~atsushi/
研究室： 第2実験室棟 3階 73研究室

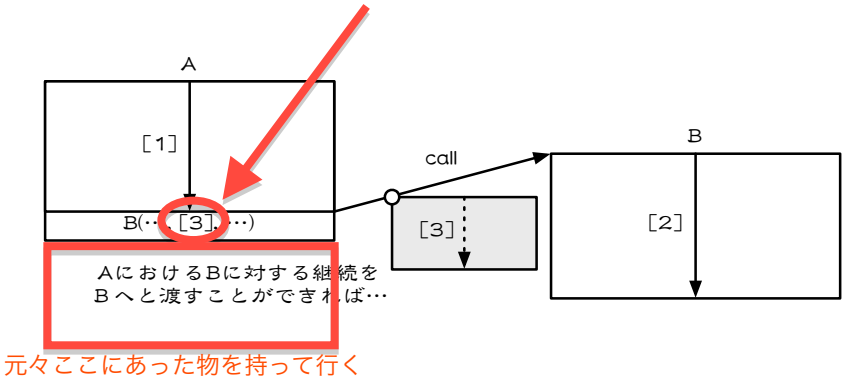
Call & Return



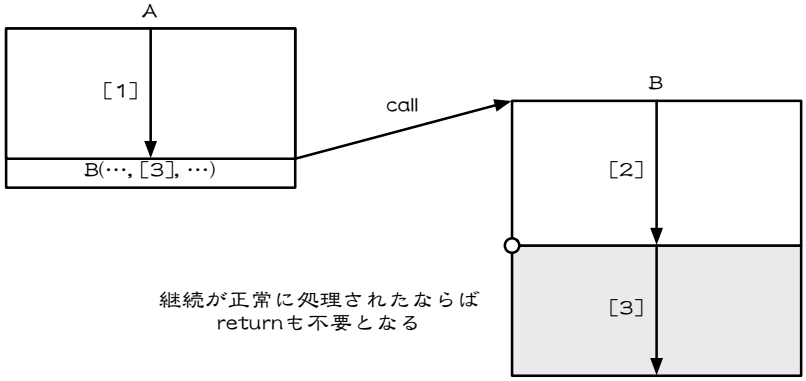


Continuation

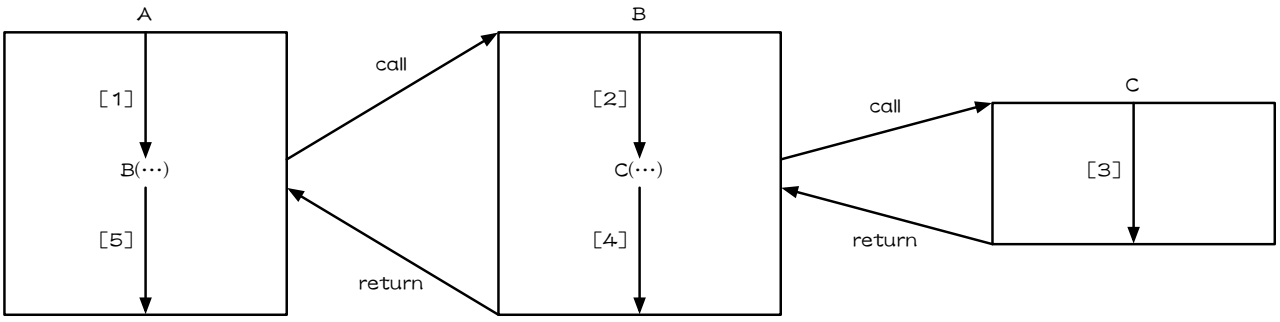
Smalltalk ではブロッククロージャとして渡す

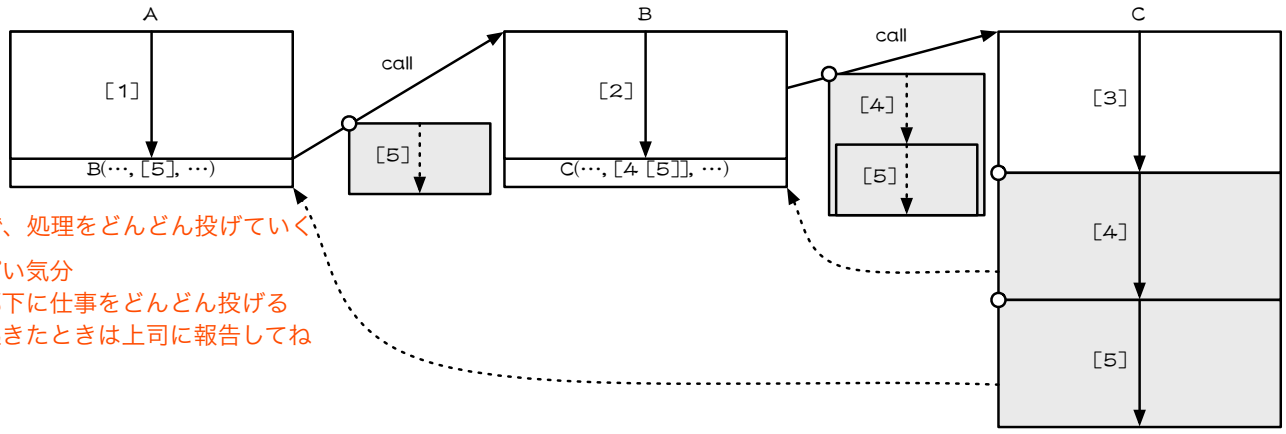


元々ここにあった物を持って行く



継続を次々に渡す





こんな感じで、処理をどんどん投げていく

会社っぽい気分
上司が部下に仕事をどんどん投げる
問題が起きたときは上司に報告してね

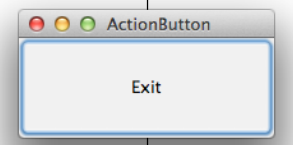
イベントリスナー

たとえば、Javaのプログラムにおいて、
ウィンドウを開く際に、ウェジェットにリスナーを登録してから、
ウィンドウを開き、そして、イベントループに入って、
ウェジェットの特定イベントが起きると、登録したリスナーの特定メソッドが起動される。

```
import java.awt.Button;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ActionButton extends Frame implements ActionListener
{
    ActionButton()
    {
        super("ActionButton");
        Button aButton = new Button("Exit");
        aButton.addActionListener(this);
        this.add(aButton);
        this.setSize(200, 100);
        this.setLocation(100, 100);
        this.setVisible(true);
    }

    public void actionPerformed(ActionEvent e)
    {
        System.exit(0);
    }

    public static void main(String[] arguments)
    {
        new ActionButton();
    }
}
```

インターフェースを実装して...

クロージャにしてコンティニュエーションとして渡す



ActionListener インタフェースを取り込むと actionPerfomed (何かが起きたら呼び出されるメソッド) を実装してねと言われる

だが、この二つが離れているため、どれが呼ばれるか分からなくて、こういう構造は嫌だよねと言い出す人がいるなので、下のような書き方をするように

匿名クラス

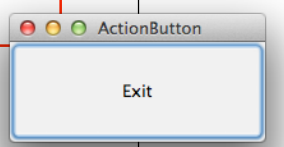
継続を渡すほうがエレガントではないのか... 匿名クラスなどを記述するのではなくて...
Javaにおいて、クロージャ(閉包)がファーストクラスになることを期待する理由が、ここにある

```
import java.awt.Button;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ActionButton extends Frame
{
    ActionButton()
    {
        super("ActionButton");
        Button aButton = new Button("Exit");
        aButton.addActionListener(
            new ActionListener()
            {
                public void actionPerformed(ActionEvent e)
                {
                    System.exit(0);
                }
            }
        );
        this.add(aButton);
        this.setSize(200, 100);
        this.setLocation(100, 100);
        this.setVisible(true);
    }

    public static void main(String[] arguments)
    {
        new ActionButton();
    }
}
```

ここはなくなった

クロージャにしてコンティニュエーションとして渡したほうが...

どれが呼ばれるか分かるようになったのでさっきよりはマシ

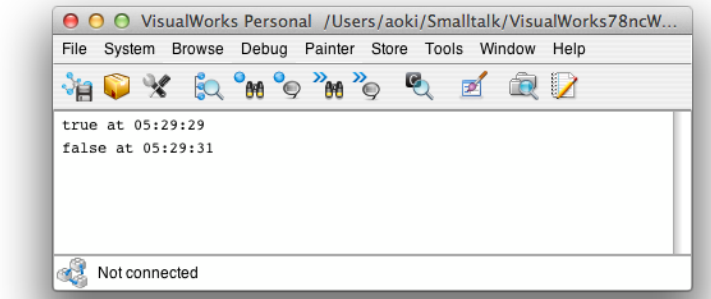


値を有するモノ

SmalltalkのValueHolderを見てみよう、エレガントだよ...
あなたの値が変化したときに、この処理(閉包)を実行してね、と予約しておく、これはまさに継続！

クロージャにしてコンティニュエーションとして渡している

```
| aButton |
Transcript clear.
aButton := ValueHolder with: false.
aButton compute:
        [Transcript
            nextPutAll: aButton value printString;
            nextPutAll: ' at ';
            nextPutAll: Time now printString;
            cr;
            flush].
2 seconds wait.
aButton value: true.
2 seconds wait.
aButton value: false
```

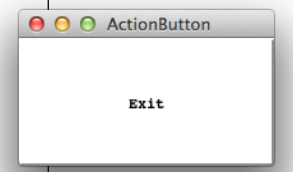


継続としての閉包

Smalltalkにおいては、クロージャ(閉包)がファーストクラスである
グラフィカルユーザインターフェースのプログラムにも、継続としての閉包が多用されている

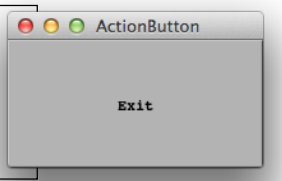
Smalltalk on VisualWorks

```
| aWindow aButton aView |
aWindow := ScheduledWindow new.
aWindow label: 'ActionButton'.
aButton := ValueHolder with: false.
aButton compute: [aWindow controller close].
aView := ActionButtonView model: aButton.
aView label: 'Exit' asText allBold asComposedText.
aWindow component: aView.
aWindow openIn: (100 @ 100 extent: 200 @ 100)
```



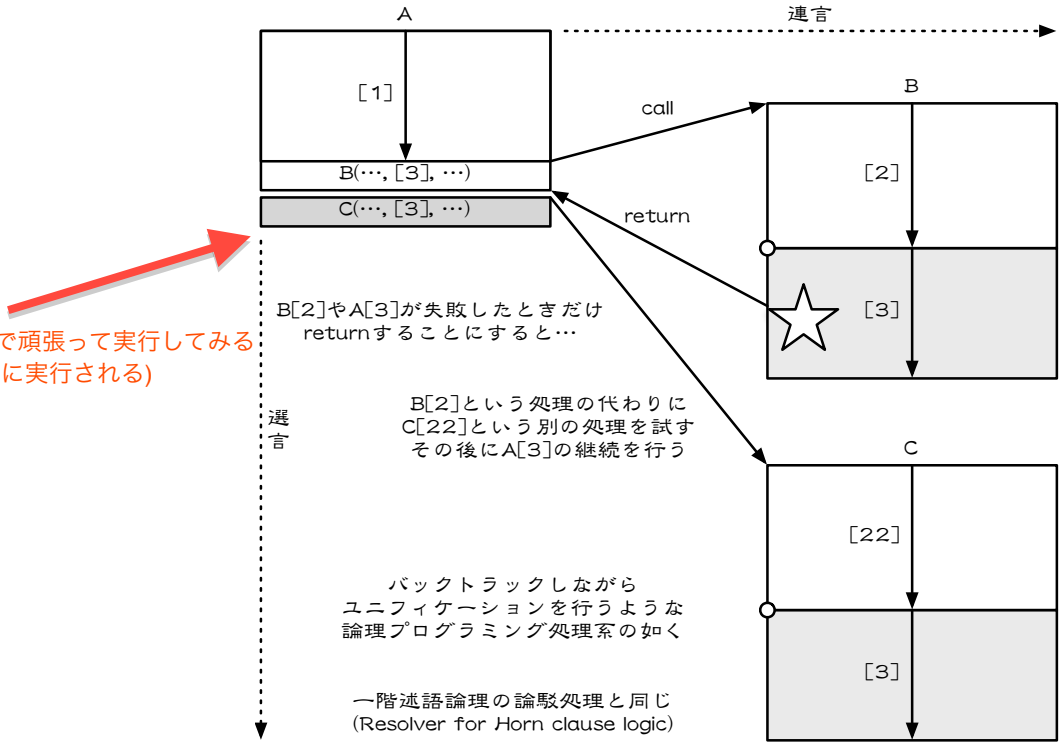
Smalltalk with Jun

```
| aWindow aButton |
aWindow := ScheduledWindow new.
aWindow label: 'ActionButton'.
aButton := JunButtonModel value: false visual: 'Exit' asText allBold.
aButton action: [aWindow controller close].
aWindow component: aButton createButtonView.
aWindow openIn: (100 @ 100 extent: 200 @ 100)
```



連言と選言

Prolog のような論理プログラミング風に示すと



私はここまでやったのですが、あとこれだけやらなければならない、
それをお願いできるかしらん、あなたのやることが済んだら、やってちょうだい。
もしも何か不都合なことが生じたならば、すぐに知らせてください。

継続(1)

実行結果は全て一緒

Continuation

```
| aTime |
aTime := Time now.
Transcript
	nextPutAll: aTime printString;
	cr;
	flush
```

```
| aClosure aTime |
aClosure :=
		[Transcript
			nextPutAll: aTime printString;
			cr;
			flush].	"外部スコープ変数を用いる閉包を生成する。"
aTime := Time now.
aClosure value	"閉包を実行する。"
```

```
| aClosure aTime |
aClosure :=
		[:time |
		Transcript
			nextPutAll: time printString;
			cr;
			flush].	"外部スコープ変数を用いない閉包を生成する。"
aTime := Time now.
aClosure value: aTime	"値を渡して閉包を実行する。"
```

```
| aClosure aContinuation aTime |
aClosure := [:time :continuation | continuation value: time printString].
aContinuation :=
		[:string |
		Transcript
			nextPutAll: string;
			cr;
			flush].
aTime := Time now.
aClosure value: aTime value: aContinuation	"値と継続を渡して閉包を実行する。"
```

継続(2)

Continuation

```
| aValue aValueHolder |
aValue := 15 factorial.
aValueHolder := ValueHolder new. "値を保持するオブジェクトを作る。"
aValueHolder value: aValue.      "値を設定する。"
aValue := aValueHolder value.    "値を取り出す。"
Transcript
       nextPutAll: aValue printString;
       nextPutAll: ' at ';
       nextPutAll: Time now printString;
       cr;
       flush
```

```
| aValue aClosure aValueHolder |
aValue := 15 factorial.
aClosure :=
             [:value |
             Transcript
                    nextPutAll: value printString;
                    nextPutAll: ' at ';
                    nextPutAll: Time now printString;
                    cr;
                    flush].
aValueHolder := ValueHolder new.
aValueHolder compute: aClosure.  "あなたの値が変化したとき、この閉包を実行してね。"
aValueHolder value: aValue
```

```
| aValue aContinuation aClosure aValueHolder |
aValue := 15 factorial.
aContinuation :=
             [
             [:string |
             Transcript
                    nextPutAll: string;
                    nextPutAll: ' at ';
                    nextPutAll: Time now printString;
                    cr;
                    flush]
                          yourself].
aClosure := [:value :continuation | continuation value: value printString].
aValueHolder := ValueHolder new.
aValueHolder with: aContinuation compute: aClosure.  "あなたの値が変化したとき、この継続と一緒にこの閉包を実行してね。"
aValueHolder value: aValue
```


継続(3)

Continuation

```
| aClosure aValue |
aClosure :=
             [:n |
             | a |
c            (n isInteger not or: [n negative]) ifTrue: [^self error: 'boo!'].
             a := 1.
             (1 to: n) do: [:i | a := a * i]. "反復している。"
             a yourself].
a   aValue := aClosure value: 10.    "階乗を計算する。"
Transcript
       nextPutAll: aValue printString;
       nextPutAll: ' at ';
b      nextPutAll: Time now printString;
       cr;
       flush. "値を時刻と共にトランスクリプトに書き出す。"
^aValue yourself    "値を応答する。"
```

```
| aContinuation aClosure |
aContinuation :=
             [:value |
             Transcript
c                   nextPutAll: value printString;
                    nextPutAll: ' at ';
                    nextPutAll: Time now printString;
                    cr;
                    flush. "値を時刻と共にトランスクリプトに書き出す。"
             ^value yourself     "値を応答する。".
aClosure :=                                  これが実行されるので、最後の self halt は実行されない
             [:n :continuation |                 (別のメソッドではないので、呼び出し元のメソッドに処理が返る)
             | a |
b            (n isInteger not or: [n negative]) ifTrue: [^self error: 'boo!'].
             a := 1.
             (1 to: n) do: [:i | a := a * i]. "反復している。"
             continuation value: a].
aClosure value: 10 value: aContinuation.      "階乗を計算する閉包を継続譲渡で実行する。"
a   self halt    "ここは実行されることはない！"
```

継続譲渡(１)

CPS : Continuation Passing Style

```
| aClosure aValue |
aClosure :=
            [:n |
            | a |
            (n isInteger not or: [n negative]) ifTrue: [^self error: 'boo!'].
            a := n isZero ifTrue: [1] ifFalse: [(aClosure value: n - 1) * n]. "再帰している。"
            a yourself].
aValue := aClosure value: 10.
Transcript
      nextPutAll: aValue printString;
      nextPutAll: ' at ';
      nextPutAll: Time now printString;
      cr;
      flush.
^aValue yourself
```

```
| aContinuation aClosure |
aContinuation :=
            [:value |
            Transcript
                  nextPutAll: value printString;
                  nextPutAll: ' at ';
                  nextPutAll: Time now printString;
                  cr;
                  flush.
            ^value yourself].
aClosure :=
            [:n :continuation |
            (n isInteger not or: [n negative]) ifTrue: [^self error: 'boo!'].
            n isZero
                  ifTrue: [continuation value: 1]
                  ifFalse: [aClosure value: n - 1 value: [:a | continuation value: a * n]].      "再帰で継続譲渡している。"
            self halt    "ここが実行されることはない！"].
aClosure value: 10 value: aContinuation.      "階乗を計算する閉包を継続譲渡で実行する。"
self halt    "ここが実行されることはない！"
```

これを継続で渡しているので、どんどんと溜まっている状態

詳しくは、 KSU, KSU-High, High, Class page4, page43 を参照

実行過程はこの様になっている

継続譲渡(２)

```
"factorial by CPS (Continuation Passing Style)"
| con0 |
con0 :=
        [:a0 |
        | con1 |
        con1 :=
                [:a1 |
                | con2 |
                con2 :=
                        [:a2 |
                        | con3 |
                        con3 :=
                                [:a3 |
                                | con4 |
                                con4 :=
                                        [:a4 |
                                        | con5 |
                                        con5 :=
                                                [:a5 |
                                                | con6 |
                                                con6 :=
                                                        [:a6 |
                                                        | con7 |
                                                        con7 :=
                                                                [:a7 |
                                                                | con8 |
                                                                con8 :=
                                                                        [:a8 |
                                                                        | con9 |
                                                                        con9 :=
                                                                                [:a9 |
                                                                                | con10 |
                                                                                con10 :=
                                                                                        [:value |
                                                                                        Transcript
                                                                                            nextPutAll: value printString;
                                                                                            nextPutAll: ' at ';
                                                                                            nextPutAll: Time now printString;
                                                                                            cr;
                                                                                            flush.
                                                                                        ^value yourself].
                                                                                con10 value: a9 * 10].
                                                                        con9 value: a8 * 9].
                                                                con8 value: a7 * 8].
                                                        con7 value: a6 * 7].
                                                con6 value: a5 * 6].
                                        con5 value: a4 * 5].
                                con4 value: a3 * 4].
                        con3 value: a2 * 3].
                con2 value: a1 * 2].
        con1 value: a0 * 1].
con0 value: 1
```