

An extensible toolkit for refactoring

Mathieu Verbaere
Programming Tools Group
University of Oxford

MSR Cambridge
December 13, 2005

What is refactoring?

- ▶ Refactoring is the process of gradually improving the design of existing code by performing behaviour-preserving program transformations.
 - ▶ *Rename*
 - ▶ *Extract Method*
 - ▶ *Extract Interface*
 - ▶ ...
- ▶ A few refactorings only have been automated in IDEs. Their implementation is complex.

Extract Method bug in Visual Studio

```
public void F(bool b)
{
    int i;
    // from
    if (b)
    {
        i = 0;
        Console.WriteLine(i);
    }
    // to
    i = 1;
    Console.WriteLine(i);
}
```



```
public void F(bool b)
{
    int i;
    i = NewMethod(b);
    i = 1;
    Console.WriteLine(i);
}
private static int NewMethod(bool b)
{
    int i;
    if (b)
    {
        i = 0;
        Console.WriteLine(i);
    }
    return i;
}
```

Extract Method bug in Visual Studio

```
public void F(bool b)
{
    int i;
    // from
    if (b)
    {
        i = 0;
        Console.WriteLine(i);
    }
    // to
    i = 1;
    Console.WriteLine(i);
}
```



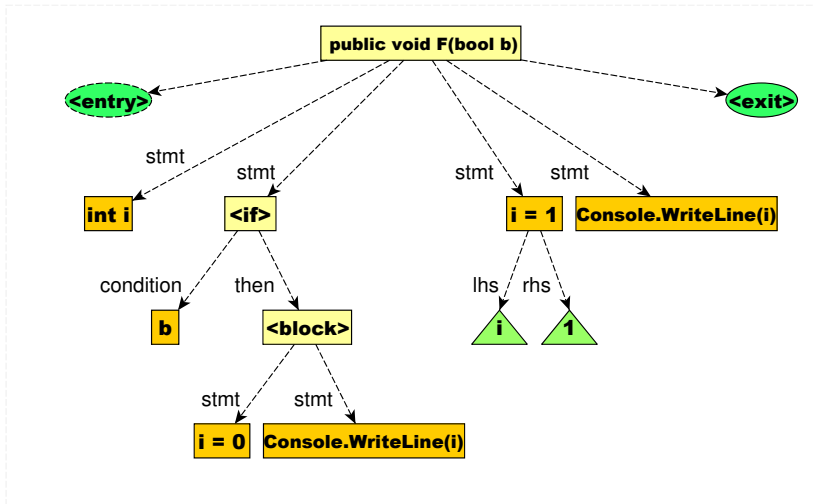
```
public void F(bool b)
{
    int i;
    i = NewMethod(b);
    i = 1;
    Console.WriteLine(i);
}
private static int NewMethod(bool b)
{
    int i;
    if (b)
    {
        i = 0;
        Console.WriteLine(i);
    }
    return i;
}
```

i in *NewMethod* is returned without necessarily being assigned.

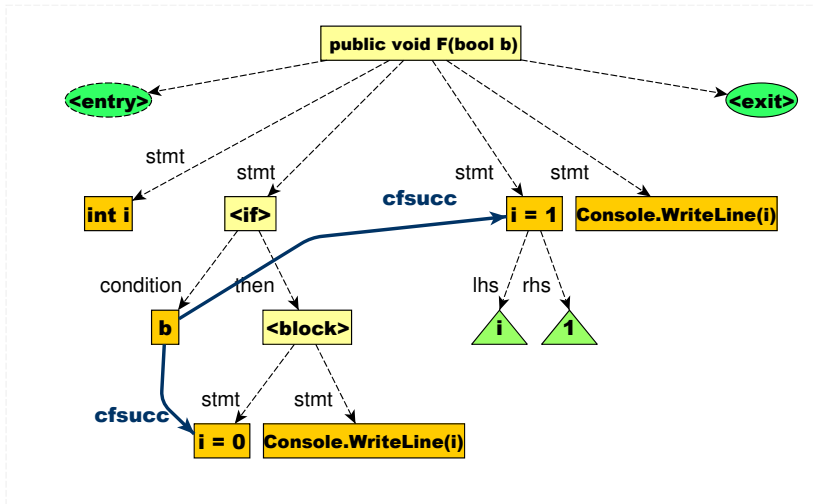
JunGL: a scripting language for refactoring

- ▶ Support to allow developers to write new refactorings is rudimentary in existing systems.
- ▶ JunGL is a hybrid of a functional language like ML and a logic query language akin to Datalog.
- ▶ The main data structure is a graph representing all information about the program to transform.

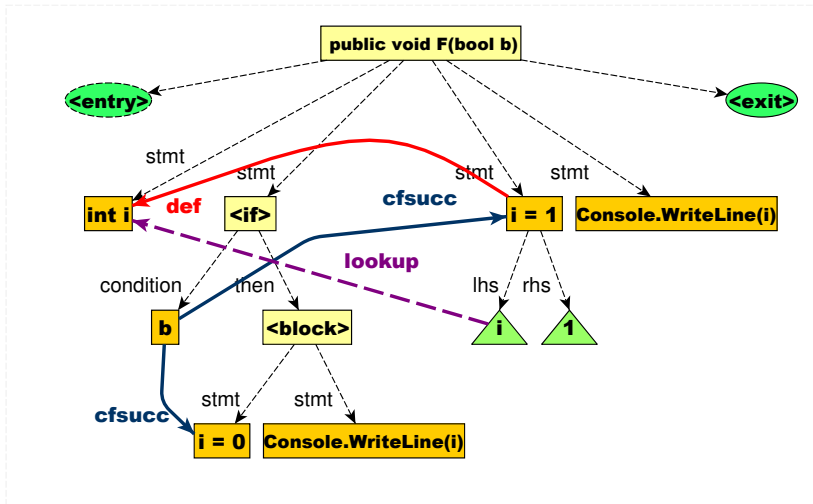
JunGL - lazy edge definitions



JunGL - lazy edge definitions



JunGL - lazy edge definitions



JunGL script example

Definition of *value* parameters for *Extract Method* mechanisation:

```
let value =  
  { ?x |  
    In(?x, variables) &  
    MayUseBeforeDefInSelection(?x) &  
    !( MayDefInSelection(?x) &  
      MayUseBeforeDefAfterSelection(?x) )  
  }
```

Predicates have an elegant definition in JunGL. For instance,

```
let predicate MayUseBeforeDefAfterSelection(?x) =  
  [endNode]  
  (local ?z: cfsucc[?z] & ![?z]def[?x])+  
  [?u]use[?x]
```

holds if there is a path from the end node to a use of x with no intervening definition of x .

Conclusion

- ▶ A prototype of JunGL is implemented on the .NET platform using both *C#* and *F#*.
- ▶ Future work: ensure some properties of the JunGL scripts (using lightweight theorem proving)
- ▶ More information available at <http://progtools.comlab.ox.ac.uk/projects/jungl>