CSCD70 Compiler Optimization

Tutorial #5 Dataflow Analysis (iii)

Bojian Zheng
bojian@cs.toronto.edu

Department of Computer Science, University of Toronto

Acknowledgement: Thanks to Professor Gennady Pekhimenko, Professor Nandita Vijaykumar and students from previous offerings of CSCD70.

Iterative Framework

```
template <typename TDomainElem, typename TDomainElemRepr
          Direction TDirection, typename TMeetOp>
class Framework {
  HashSet Domain;
  // Instruction-Domain Value Mapping
  HashMap InstDomainValMap;
  bool runOnFunction(Function&);
};
```

runOnFunction

Algorithm 1: runOnFunction

Data: Domain D, Instruction-Domain Value Mapping M

```
initialize *Domain*;
initialize *Instruction-Domain Value Mapping*;
do
    traverse through the CFG update $M$ $\forall \text{inst} \in F$;
while *NOT converge*;
```

☞ `initializeDomain`, `traverseCFG`

☛ `MeetOp::top`

traverseCFG

Algorithm 2: traverseCFG

Data: Domain D, Instruction-Domain Value Mapping M
Arguments : Function F, Direction
Return : Whether M has been modified

```
for $bb \in \mathrm{TraversalOrder}(F)$ do
    if $bb$ has no meet operands then
        $\mathrm{initialVal} \leftarrow$ Boundary Condition;
    else
        $\mathrm{initialVal} \leftarrow \mathrm{MeetOp}(\mathrm{MeetOperands}(bb))$;
    $\mathrm{inputVal} \leftarrow \mathrm{initialVal}$;
    for each instruction $i \in bb$ do
        $\mathrm{TransferFunc}(i, \mathrm{inputVal}, M[i])$;
        $\mathrm{inputVal} \leftarrow M[i]$;
```

☞ `BC`, `MeetOperands` ☛ `MeetOp::operator()`, `transferFunc`

Iterative Framework

```
template <typename TDomainElem, typename TDomainElemRepr
          Direction TDirection, typename TMeetOp>
class Framework : public FunctionPass {
  HashSet Domain;
  // Instruction-Domain Value Mapping
  HashMap InstDomainValMap;
  bool runOnFunction(Function&);
  void initializeDomain(const Function&);
  bool traverseCFG(const Function&);
  DomainVal BC() const;
  DomainValVec MeetOperands(const BasicBlock&) const;
  bool transferFunc(const Instruction&, const DomainVal&,
                    DomainVal&) =0;
};
```

Available Expressions

```
class AvailExpr :
    public Framework<Expression, bool,
                     Forward, Intersect> {
  bool transferFunc(const Instruction&, const BitVector&,
                    BitVector&);
};

class Intersect {
  BitVector operator()(const BitVector&,
                       const BitVector&) const;
  BitVector top(const size_t) const;
};
```

MeetOp

```
Algorithm 3: MeetOp
Data: Domain $D$, Instruction-Domain Value Mapping $M$
Arguments : BasicBlock $bb$
Return    : Merged BitVector
return $\bigwedge_{i \in \text{MeetOperands}} (\overbrace{M[\text{back}(\text{pred}(bb))]}^{\text{MeetOperands}})$;
```

transferFunc

Algorithm 4: transferFunc

```
Data: Domain $D$, Instruction-Domain Value Mapping $M$
Arguments : Instruction $i$, BitVector $bv_i$, $bv_o$
Return    : Whether $bv_o$ has been modified
$bv'_o \leftarrow bv_i$;
for each element $e \in D$ do
    if $\text{lhs}(e) = i$ or $\text{rhs}(e) = i$ then
        $bv'_o[\text{position}(e)] = \text{false}$;
$\text{iter} \leftarrow \text{find}(D, \text{Expression}(i))$;
if $\text{iter} \neq \text{end}(D)$ then
    $bv'_o[\text{position}(e)] = \text{true}$;
$\text{hasChanges} \leftarrow bv'_o = bv_o?$;
$bv_o \leftarrow bv'_o$;
return hasChanges;
```

Summary

```
template <typename TDomainElem, typename TDomainElemRepr
          Direction TDirection, typename TMeetOp>
class Framework : public FunctionPass {
  HashSet Domain;
  // Instruction-Domain Value Mapping
  HashMap InstDomainValMap;
  bool runOnFunction(Function&);
  void initializeDomain(const Function&);
  bool traverseCFG(const Function&);
  DomainVal BC() const;
  DomainValVec MeetOperands(const BasicBlock&) const;
  bool transferFunc(const Instruction&, const DomainVal&,
                    DomainVal&) =0;
};
```

Summary

```
class AvailExpr :
  public Framework<Expression, bool,
                   Forward, Intersect> {
  bool transferFunc(const Instruction&, const BitVector&,
                    BitVector&);
};

class Intersect {
  BitVector operator()(const BitVector&,
                       const BitVector&) const;
  BitVector top(const size_t) const;
};
```

☞ **Homework Assignment**: `AvailExpr` and `Liveness`