Advanced T⚙pics *in* C⚙mpilers

Task

Simone Campanoni
simone.campanoni@northwestern.edu

Task in NOELLE

- Sources:
  src/core/task

- Main headers:
  install/noelle/core/Task.hpp

- Examples of passes using the abstraction:
  examples/passes/task

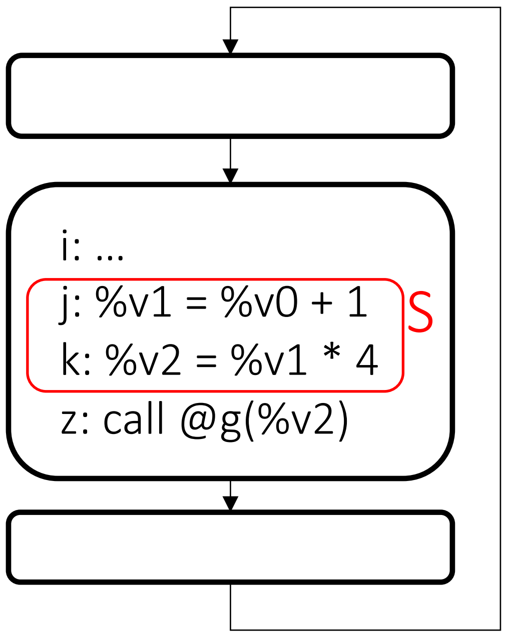
Outline

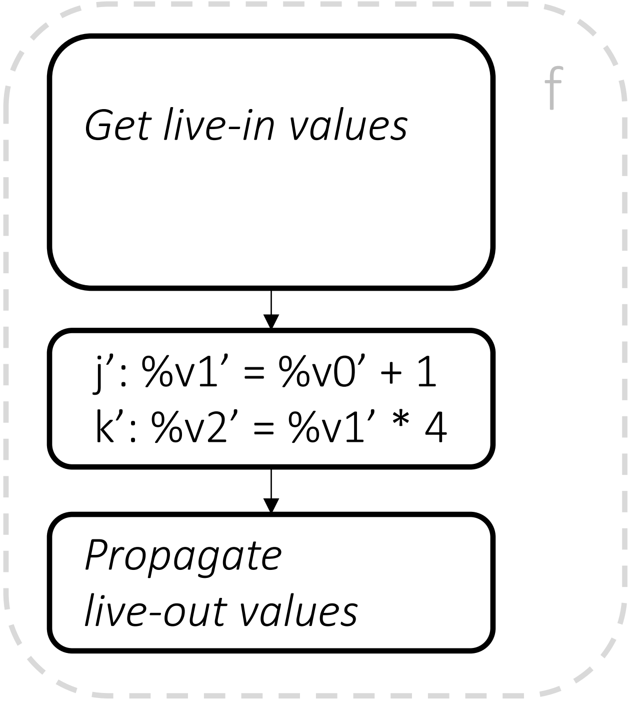
- What is a Task in NOELLE
- Creation of a Task
- Invoking a Task

Task in NOELLE

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S





Code mapping

Original	Clone
%v0	%v0'
%v1	%v1'
%v2	%v2'

e

Value	Live-In ?
%v0	True
%v2	False

Task in NOELLE

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S

- f is called “task body”
- e is called “task environment”
- t has a static unique ID (uint64_t) and a dynamic instance ID
  - The static ID is set by the Task abstraction automatically
  - The instance ID is a Value * and whoever defines t is responsible for creating it and store it in the field Task::instanceIndexV

Task in NOELLE

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S

Task definition

Task invocation

Task in NOELLE: task signature

A task t is a wrapper of

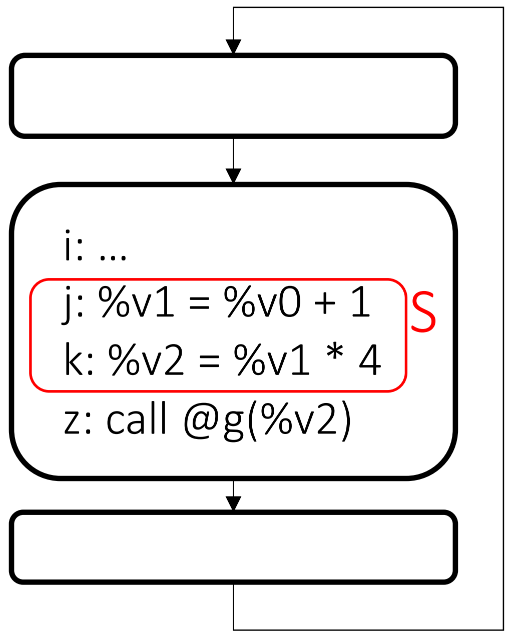
1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e includes pointers to all live-in and live-out variables of S

- Whoever creates t is responsible to define the signature of f
  - f needs to obtain as inputs everything that it needs to execute
  - An instance of e (of some shape/form) needs to be an input of f
  - The return type of the signature of f can only be void
  - The signature is an input to the Task constructor

Task in NOELLE: body definition

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S



void f (int8 *%e)

Code mapping

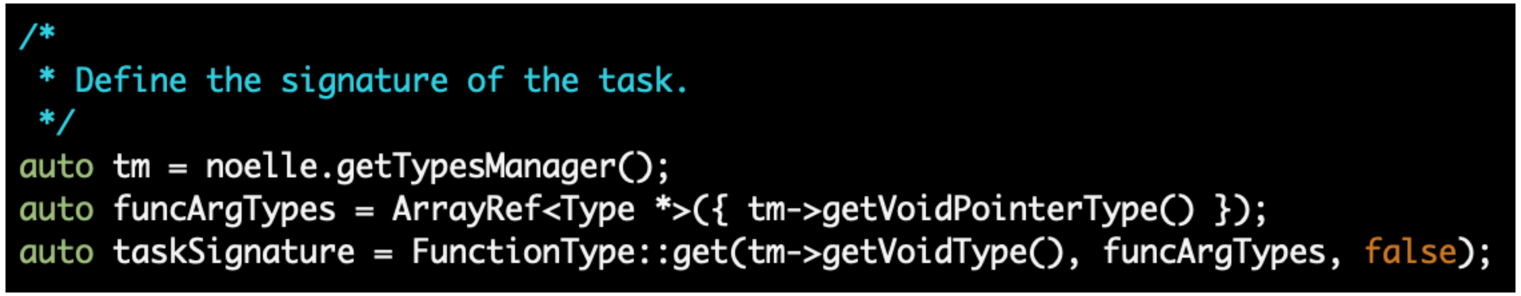
e

Value	Live-In ?
%v0	True
%v2	False

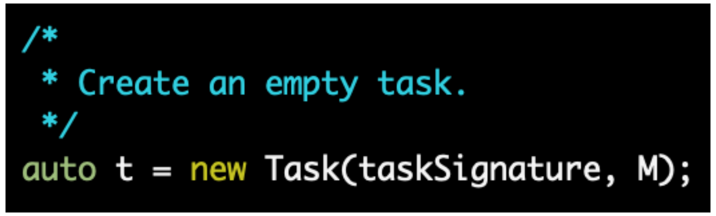
Task in NOELLE: task signature

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e includes pointers to all live-in and live-out variables of S



```
/*
 * Define the signature of the task.
 */
auto tm = noelle.getTypesManager();
auto funcArgTypes = ArrayRef<Type *>({ tm->getVoidPointerType() });
auto taskSignature = FunctionType::get(tm->getVoidType(), funcArgTypes, false);
```



```
/*
 * Create an empty task.
 */
auto t = new Task(taskSignature, M);
```

Task in NOELLE: task definition

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S

- Whoever creates t is responsible to define the body of f
  - The body is first defined by the constructor of Task by creating two basic blocks
    - Entry basic block: first code executed when f is invoked
    - Exit basic block: last code executed before leaving f
    - Both basic blocks are empty

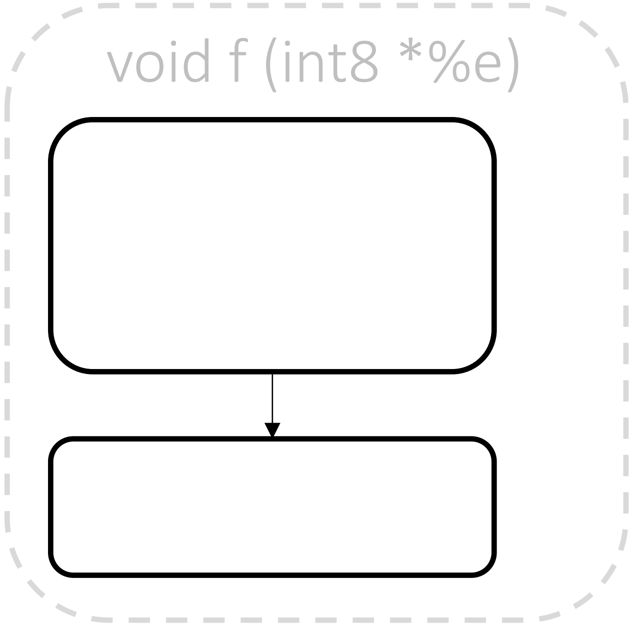
Task in NOELLE: task signature

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e includes pointers to all live-in and live-out variables of S

```
/*
 * Define the signature of the task.
 */
auto tm = noelle.getTypesManager();
auto funcArgTypes = ArrayRef<Type *>({ tm->getVoidPointerType() });
auto taskSignature = FunctionType::get(tm->getVoidType(), funcArgTypes, false);
```

```
/*
 * Create an empty task.
 */
auto t = new Task(taskSignature, M);
```



Task in NOELLE: body definition

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S

- Whoever creates t is responsible to define the body of f
  - The body is first defined by the constructor of Task by creating two basic blocks
  - New basic blocks are then created by cloning S

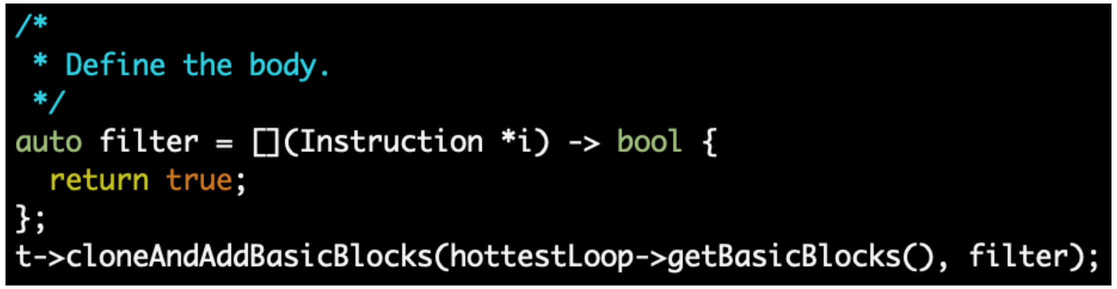
```
void Task::cloneAndAddBasicBlocks(
    const std::unordered_set<BasicBlock *> &bbs,
    std::function<bool(Instruction *origInst)> filter);
```

Task in NOELLE: body definition

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S

- Whoever creates t is responsible to define the body of f
  - The body is first defined by the creation of two basic blocks
  - New basic blocks are then created by cloning S

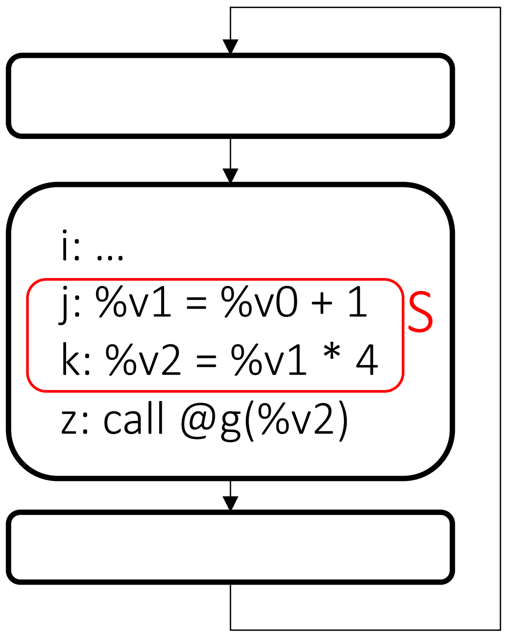


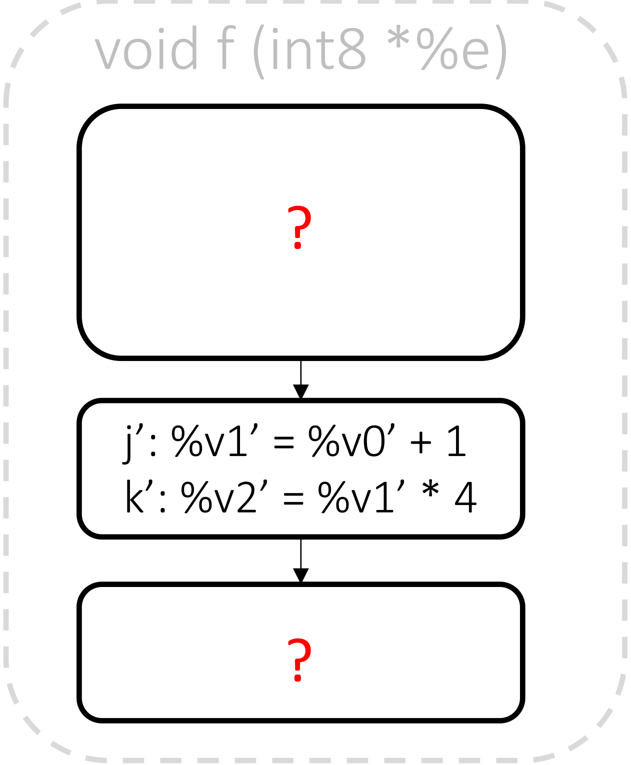
```
/*
 * Define the body.
 */
auto filter = [](Instruction *i) -> bool {
  return true;
};
t->cloneAndAddBasicBlocks(hottestLoop->getBasicBlocks(), filter);
```

Task in NOELLE: body definition

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S





Code mapping

Original	Clone
%v0	%v0'
%v1	%v1'
%v2	%v2'

e

Value	Live-In ?
%v0	True
%v2	False

Task in NOELLE: environment definition

A task t is a wrapper of

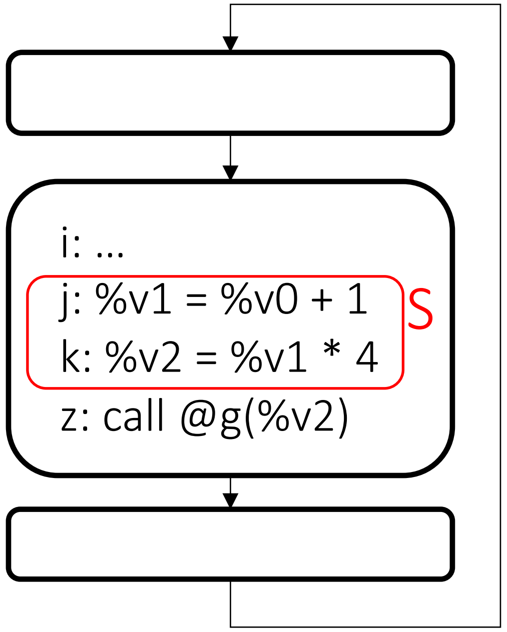
1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S

- Whoever creates t is responsible to identify and instantiate e correctly
  - t sees e as a Value * to be the pointer from which
    you can reach all live-in and live-out variables of the code wrapped into f
  - The data layout of the object pointed by e is decided by whoever designs a task
    (rather than Task itself)
  - In other words, Task ignores the details about how e looks in memory

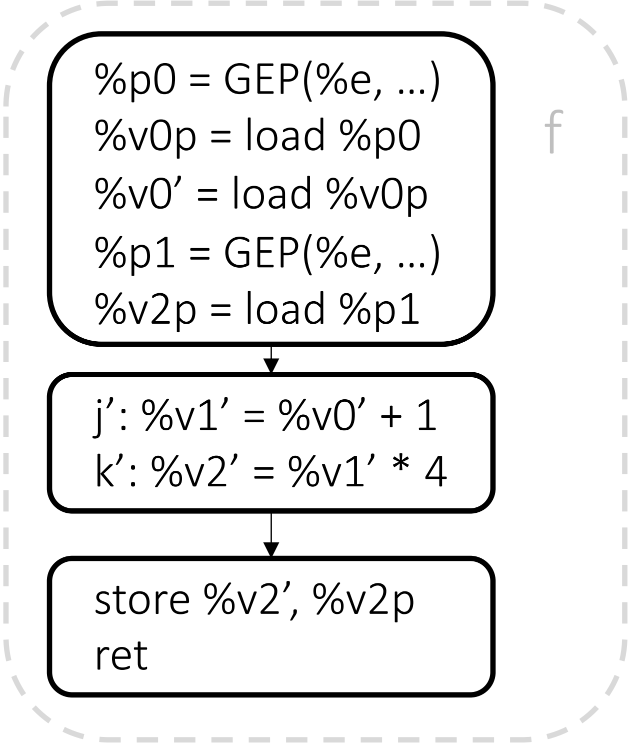
Task in NOELLE: environment

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S



```
i: ...
j: %v1 = %v0 + 1
k: %v2 = %v1 * 4
z: call @g(%v2)
```



```
%p0 = GEP(%e, ...)
%v0p = load %p0
%v0' = load %v0p
%p1 = GEP(%e, ...)
%v2p = load %p1
```

```
j': %v1' = %v0' + 1
k': %v2' = %v1' * 4
```

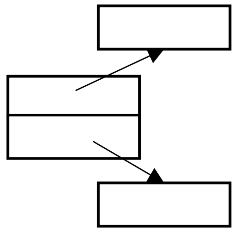
```
store %v2', %v2p
ret
```

Code mapping

Original	Clone
%v0	%v0'
%v1	%v1'
%v2	%v2'

e

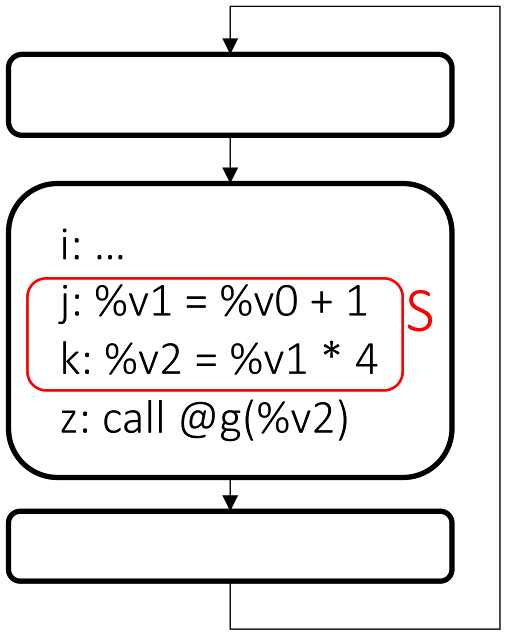
Value	Live-In ?
%v0	True
%v2	False



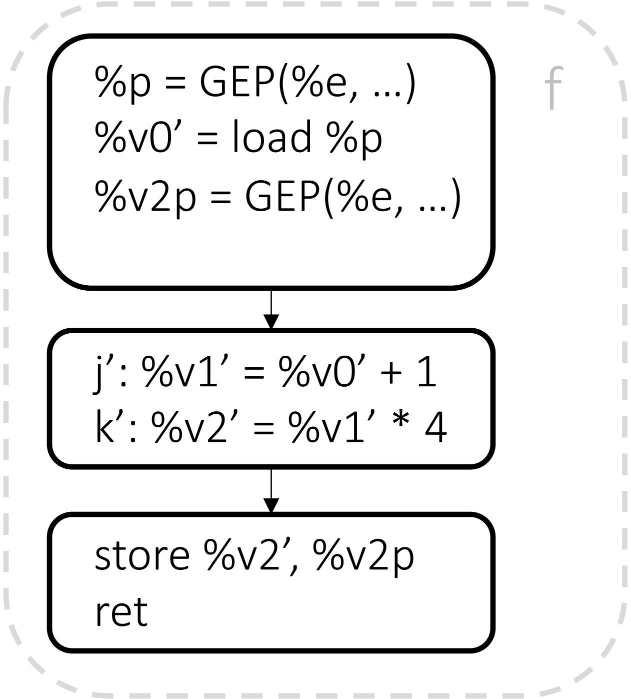
Task in NOELLE: environment

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S



```
i: …
j: %v1 = %v0 + 1
k: %v2 = %v1 * 4
z: call @g(%v2)
```



```
%p = GEP(%e, …)
%v0' = load %p
%v2p = GEP(%e, …)
```

```
j': %v1' = %v0' + 1
k': %v2' = %v1' * 4
```

```
store %v2', %v2p
ret
```

Code mapping

Original	Clone
%v0	%v0'
%v1	%v1'
%v2	%v2'

e

Value	Live-In ?
%v0	True
%v2	False

Outline

- What is a Task in NOELLE
- Creation of a Task
- Invoking a Task

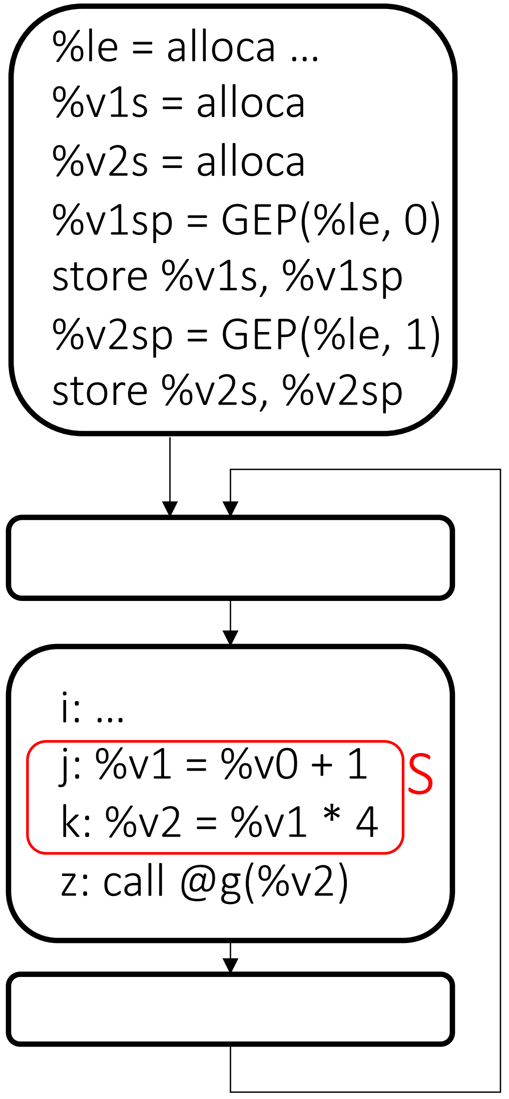
Task in NOELLE: task invocation

A task t is a wrapper of

1. A set of instructions S organized in basic blocks cloned from the original code
2. Instructions S are wrapped into a new function f and
3. An environment e that includes live-in and live-out variables of S

- t is invoked by calling f
  - The code that invokes f needs to setup a memory instance of e consistently with the data layout chosen by whoever defined the Task

Task in NOELLE: example0



```
%le = alloca …
%v1s = alloca
%v2s = alloca
%v1sp = GEP(%le, 0)
store %v1s, %v1sp
%v2sp = GEP(%le, 1)
store %v2s, %v2sp
```

```
i: …
j: %v1 = %v0 + 1
k: %v2 = %v1 * 4
z: call @g(%v2)
```

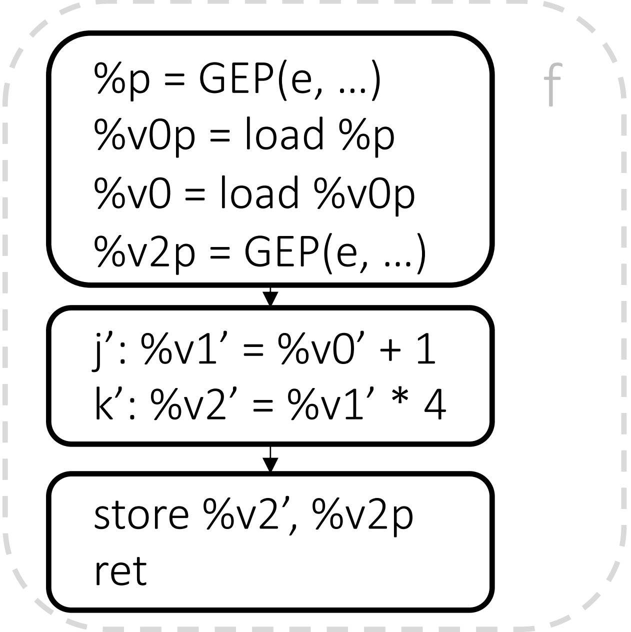
```
store %v1, %v1s
call @f (%le)
%v2 = load %v2s
```

Original	Clone
%v0	%v0’
%v1	%v1’
%v2	%v2’

Code mapping

Value	Live-In ?
%v0	True
%v2	False

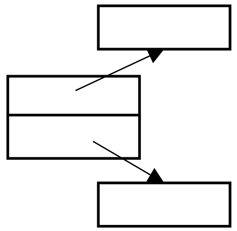
e



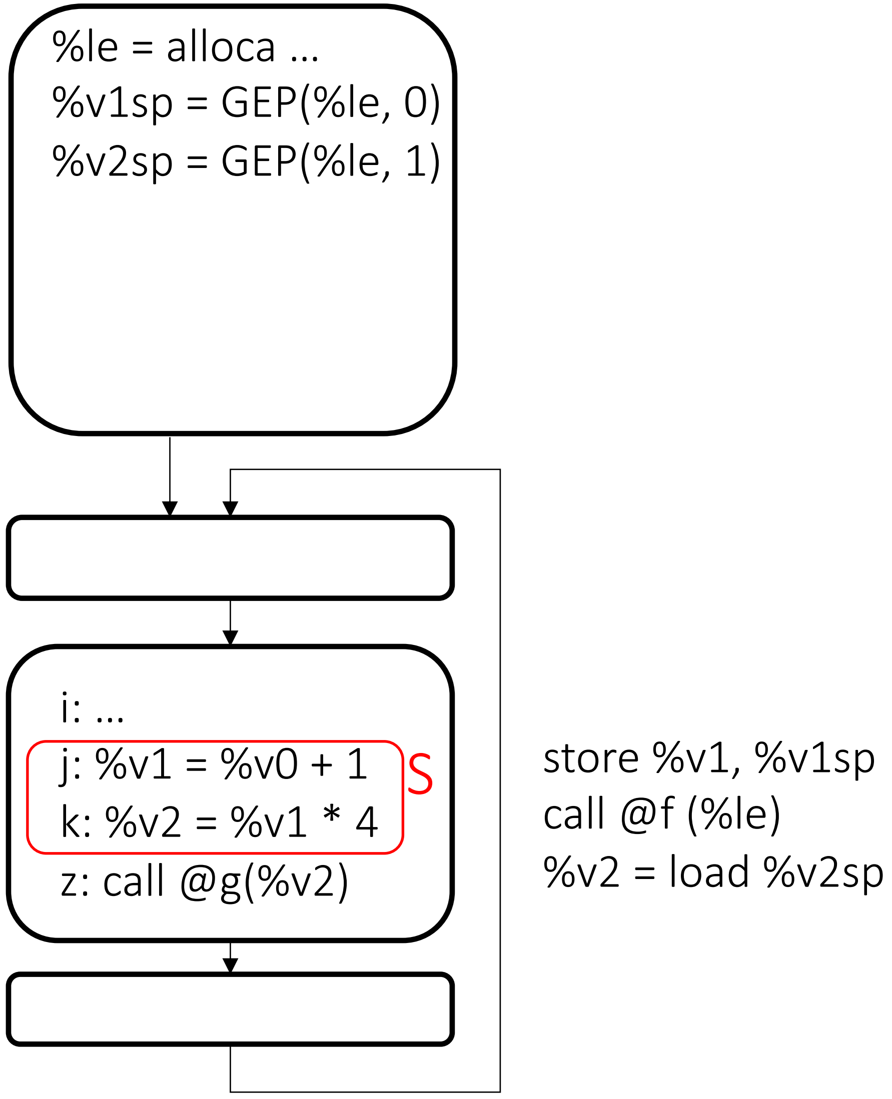
```
%p = GEP(e, …)
%v0p = load %p
%v0 = load %v0p
%v2p = GEP(e, …)
```

```
j’: %v1’ = %v0’ + 1
k’: %v2’ = %v1’ * 4
```

```
store %v2’, %v2p
ret
```



Task in NOELLE: example1

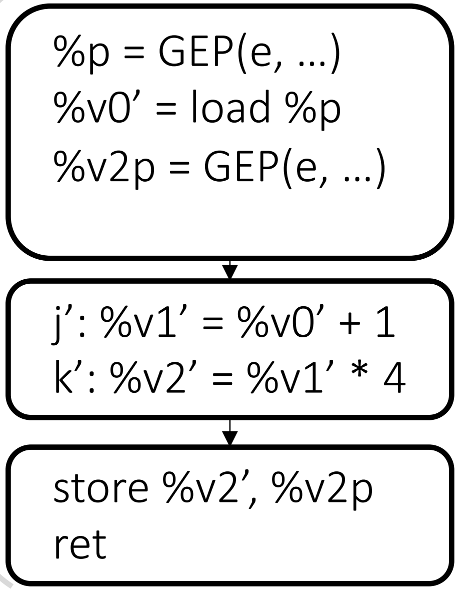


Original	Clone
%v0	%v0’
%v1	%v1’
%v2	%v2’

Code mapping

Value	Live-In ?
%v0	True
%v2	False

e



f

Always have faith in your ability

Success will come your way eventually

Best of luck!