



Saint Vincent College

Department of Computing
and Information Science



Using Semantics for Proofs Memory Management

Fr. Boniface, OSB
Lecture 26

(some slides modified from Tucker/Noonan, copyright McGraw-Hill, 2006)

“There are also many other things that Jesus did, but if these were to be described individually, I do not think the whole world would contain the books that would be written.” (Jn. 21)

Abstract Syntax

Block =
Statement*

- $M(\text{Block } b, \text{State } state)$

= $state$

= $M(b_n, M(\dots, M(b_1, state)\dots))$

if $b = \{\}$

if $b = \{b_1 b_2 \dots b_n\}$

```
int main() {  
    int x,y,z;  
    x = 2;  
    y = 3;  
    z = 0;  
    while (y > 0) {  
        z = z + x;  
        y = y - 1;  
    }  
}
```

Abstract Syntax
Block =
Statement*

- $M(\text{Block } b, \text{State } state)$

= $state$

= $M(b_n, M(\dots, M(b_1, state)\dots))$

if $b = \{\}$

if $b = \{b_1 b_2 \dots b_n\}$

Abstract Syntax

Loop =
Expression test;
Statement body;

- $M(\text{Loop } 1, \text{State } state)$
= $M(1, M(1.\text{body}, state))$ if $M(1.\text{test}, state)$ is true
= $state$ otherwise

Loop



```
int main() {  
    int x,y,z;  
    x = 2;  
    y = 3;  
    z = 0;  
    while (y > 0) {  
        z = z + x;  
        y = y - 1;  
    }  
}
```

Abstract Syntax

Loop =
Expression test;
Statement body;

- $M(\text{Loop } 1, \text{State } state)$
= $M(1, M(1.\text{body}, state))$ if $M(1.\text{test}, state)$ is true
= $state$ otherwise

Expressions



Abstract Syntax

Binary = BinaryOp op; Expression term1, term2;

Unary = UnaryOp op; Expression term;

Variable = String id;

Value = IntValue | BoolValue | FloatValue | CharValue

$M : \text{Expression} \times \text{State} \rightarrow \text{Value}$

Expressions



Abstract Syntax

Binary = **BinaryOp** *op*; **Expression** *term1, term2*;

Unary = **UnaryOp** *op*; **Expression** *term*;

Variable = **String** *id*;

Value = **IntValue** | **BoolValue** | **FloatValue** | **CharValue**

M : *Expression* \times *State* \rightarrow *Value*

M(*Expression e*, *State state*)

Expressions



Abstract Syntax

Binary = BinaryOp *op*; Expression *term1*, *term2*;

Unary = UnaryOp *op*; Expression *term*;

Variable = String *id*;

Value = IntValue | BoolValue | FloatValue | CharValue

$M : \text{Expression} \times \text{State} \rightarrow \text{Value}$

$M(\text{Expression } e, \text{State } state)$

= *e*

if *e* is a *Value*

Expressions



Abstract Syntax

Binary = BinaryOp *op*; Expression *term1*, *term2*;

Unary = UnaryOp *op*; Expression *term*;

Variable = String *id*;

Value = IntValue | BoolValue | FloatValue | CharValue

$M : \text{Expression} \times \text{State} \rightarrow \text{Value}$

$M(\text{Expression } e, \text{State } state)$

= *e*

= *state(e)*

if *e* is a *Value*

if *e* is a *Variable*

Expressions



Abstract Syntax

Binary = BinaryOp *op*; Expression *term1*, *term2*;

Unary = UnaryOp *op*; Expression *term*;

Variable = String *id*;

Value = IntValue | BoolValue | FloatValue | CharValue

$M : \text{Expression} \times \text{State} \rightarrow \text{Value}$

$M(\text{Expression } e, \text{State } state)$

= *e*

= *state(e)*

= *ApplyBinary(e.op,*

M(e.term1, state),

M(e.term2, state))

if *e* is a *Value*

if *e* is a *Variable*

if *e* is a *Binary*

Expressions



Abstract Syntax

Binary = BinaryOp op; Expression term1, term2;

Unary = UnaryOp op; Expression term;

Variable = String id;

Value = IntValue | BoolValue | FloatValue | CharValue

$M : \text{Expression} \times \text{State} \rightarrow \text{Value}$

$M(\text{Expression } e, \text{State } state)$

= e

= state(e)

= ApplyBinary(e.op,
 M(e.term1, state),
 M(e.term2, state))

= ApplyUnary(e.op,
 M(e.term, state))

if e is a *Value*
if e is a *Variable*

if e is a *Binary*

if e is a *Unary*

Binary



ApplyBinary : *Operator* x *Value* x *Value* → *Value*

ApplyBinary(*Operator op*, *Value v1*, *Value v2*)

= *v1* + *v2*

= *v1* - *v2*

= *v1* × *v2*

= *floor*(*|v1* ÷ *v2*) × *sign*(*v1* × *v2*)

= ...

if *op* = int+

if *op* = int-

if *op* = int*

if *op* = int/

```
while (y > 0) {  
    z = z + x;  
    y = y - 1;  
}
```

Proving Termination

```
int main() {  
    int a,b,c;  
    a = 2;  
    b = 3;  
    c = 0;  
    while (b > 0) {  
        c = c + a;  
        b = b - 1;  
    }  
}
```

Proving Termination

```
int main() {  
    int a,b,c;  
    a = 2;  
    b = 3;  
    c = 0;  
    while (b > 0) {  
        c = c + a;  
        b = b - 1;  
    }  
}
```

- Theorem - p always terminates

Proving Termination

```
int main() {  
    int a,b,c;  
    a = 2;  
    b = 3;  
    c = 0;  
    while (b > 0) {  
        c = c + a;  
        b = b - 1;  
    }  
}
```

- Theorem - p always terminates
- Proof - give meaning of $p: M(p)$ and show that it is always defined

Proving Termination



```
int main() {  
    int a,b,c;  
    a = 2;  
    read(b);  
    c = 0;  
    while (b > 0) {  
        c = c + a;  
        b = b - 1;  
    }  
}
```

Proving Termination

```
int main() {  
  int a,b,c;  
  a = 2;  
  read(b);  
  c = 0;  
  while (b > 0) {  
    c = c + a;  
    b = b - 1;  
  }  
}
```

- Theorem - p always terminates

Proving Termination

```
int main() {  
    int a,b,c;  
    a = 2;  
    read(b);  
    c = 0;  
    while (b > 0) {  
        c = c + a;  
        b = b - 1;  
    }  
}
```

- Theorem - p always terminates
- Proof - give meaning of $p: M(p)$ and show that it is always defined

Proof by Induction

- Prove for $i = 1..n$, $\sum i = n(n+1) / 2$
- Show base case (e.g. $n = 1$)
- Using induction hypothesis (n case), show $n+1$.
- Base:
 $1 (1 + 1) / 2 = 1$
- Step:
Assume $\sum i = n(n+1) / 2$
Show $\sum (i+1) = (n+1)((n+1)+1)/2$