Program Verification Using Separation Logic

Dino Distefano
Queen Mary University of London

Lecture 2

Today's plan

- Programming language & semantics
- Small axioms
- Frame Rule
- Tight interpretation of triples

Simple Imperative Language

- Safe commands:
  - S::= skip | x:=E | x:=new()
- Heap accessing commands:
  - A(E) ::= dispose(E) | x:=[E] | [E]:=F

where E is and expression x, y, nil, etc.

- Command:
  - C::= S | A | C1;C2 | if B { C1 } else {C2} | while B do { C }

where B boolean guard E=E, E!=E, etc.

Semantics of Programs

- The concrete semantics of the language is given by a operational semantics:
  - (s,h),C ===>(s',h'),C'
  - (s,h),C ===>(s',h')
  - (s,h),C ===>err (or T)
- err is a special error state indicating a memory violation

Concrete semantics

$$\frac{\mathcal{C}[\![E]\!]s = n}{s, h,\ x := E \implies (s|x \mapsto n), h}$$

$$\frac{\ell \notin dom(h)}{s, h,\ \mathsf{new}(x) \implies (s|x \mapsto \ell), (h|\ell \mapsto n)}$$

$$\frac{\mathcal{C}[\![E]\!]s = \ell \quad h(\ell) = n}{s, h,\ x := [E] \implies (s|x \mapsto n), h}$$

$$\frac{\mathcal{C}[\![E]\!]s = \ell}{s, h * [\ell \mapsto n],\ \mathsf{dispose}(E) \implies s, h}$$

$$\frac{\mathcal{C}[\![E]\!]s = \ell \quad \mathcal{C}[\![F]\!]s = n \quad \ell \in dom(h)}{s, h,\ [E] := F \implies s, (h|\ell \mapsto n)}$$

$$\frac{\mathcal{C}[\![E]\!]s \notin dom(h)}{s, h,\ A(E) \implies \top}$$

Hoare Logic

- A Hoare triple is a formula {P} C {Q} where
  - P, Q are formulae in a base logic (e.g. first order logic, separation logic, etc.)
  - C is a program in our language
  - P is called precondition
  - Q is called postcondition

Semantics of Hoare triples

- Partial correctness: {P} C {Q} is valid iff starting from a state s,h |= P, whenever the execution of C terminates in a state (s',h') then s',h'|= Q
- Total correctness: [P] C [Q] is valid iff starting from a state s,h|= P,
  - Every execution terminates
  - when an execution terminates in a state (s',h') then s',h'|= Q.

Semantics of Hoare triples

- Partial correctness: {P} C {Q} is valid iff starting from a state s,h |= P, whenever the execution of C terminates in a state (s',h') then s',h'|= Q
- Total correctness: [P] C [Q] is valid iff starting from a state s,h|= P,
  - Every execution terminates
  - when an execution terminates in a state (s',h') then s',h'|= Q.

Sequential Composition Rule

$$\frac{\{P\}\ C1\ \{P'\} \qquad \{P'\}\ C2\ \{Q\}}{\{P\}\ C1;C2\ \{Q\}}$$

Example:

Sequential Composition Rule

$$\frac{\{P\}\ C1\ \{P'\} \qquad \{P'\}\ C2\ \{Q\}}{\{P\}\ C1;C2\ \{Q\}}$$

Example:

{ y+z>4 } y:=y+z-1; x:=y+2 { x>5 }

Sequential Composition Rule

$$\frac{\{P\}\ C1\ \{P'\} \qquad \{P'\}\ C2\ \{Q\}}{\{P\}\ C1;C2\ \{Q\}}$$

Example:

$$\frac{\{\ y+z>4\ \}\ y:=y+z-1\ \{y > 3\}}{\{\ y+z>4\ \}\ y:=y+z-1;\ x:=y+2\ \{\ x>5\ \}}$$

Sequential Composition Rule

$$\frac{\{P\}\ C1\ \{P'\} \qquad \{P'\}\ C2\ \{Q\}}{\{P\}\ C1;C2\ \{Q\}}$$

Example:

$$\frac{\{\ y+z>4\ \}\ y:=y+z-1\ \{y > 3\} \qquad \{\ y>3\ \}\ x:=y+2\ \{x > 5\}}{\{\ y+z>4\ \}\ y:=y+z-1;\ x:=y+2\ \{\ x>5\ \}}$$

Conditional rules

$$\frac{\{P \land B\}\ C1\ \{Q\} \qquad \{P \land \lnot B\}\ C2\ \{Q\}}{\{P\}\ \text{if } B \text{ then } C1 \text{ else } C2\ \{Q\}}$$

Example:

Conditional rules

$$\frac{\{P \wedge B\}\ C1\ \{Q\} \qquad \{P \wedge !B\}\ C2\ \{Q\}}{\{P\}\ \text{if } B \text{ then } C1 \text{ else } C2\ \{Q\}}$$

Example:

{ (y>4) } if z>1 then y:=y+z else y:=y-1 { y>3 }

Conditional rules

{P /\ B} C1 {Q}   {P /\ !B} C2 {Q}

{P} if B then C1 else C2 {Q}

Example:

{ (y>4) /\ (z>1) } y:=y+z { y>5 }

{ (y>4) } if z>1 then y:=y+z else y:=y-1 { y>3 }

Conditional rules

$$\frac{\{P \land B\}\ C1\ \{Q\} \qquad \{P \land !B\}\ C2\ \{Q\}}{\{P\}\ \text{if } B \text{ then } C1 \text{ else } C2\ \{Q\}}$$

Example:

$$\frac{\{\ (y>4) \land (z>1)\ \}\ y:=y+z\ \{\ y>5\ \} \qquad \{\ (y>5) \land !(z>1)\}\ y:=y-1\ \{\ y>3\ \}}{\{\ (y>4)\ \}\ \text{if } z>1 \text{ then } y:=y+z \text{ else } y:=y-1\ \{\ y>3\ \}}$$

Structural Rules

$$\frac{P \Longrightarrow P' \qquad \{P'\}\ C\ \{Q'\} \qquad Q' \Longrightarrow Q}{\{P\}\ C\ \{Q\}}$$ consequence

$$\frac{\{P1\}\ C\ \{Q1\} \qquad \{P2\}\ C\ \{Q2\}}{\{P1 \lor P2\}\ C\ \{Q1 \lor Q2\}}$$ disjunction

Note: there are other rules, eg conjuction, quantifiers

Example:

Structural Rules

$$\frac{P \Longrightarrow P' \qquad \{P'\}\ C\ \{Q'\} \qquad Q' \Longrightarrow Q}{\{P\}\ C\ \{Q\}}$$

consequence

$$\frac{\{P1\}\ C\ \{Q1\} \qquad \{P2\}\ C\ \{Q2\}}{\{P1 \lor P2\}\ C\ \{Q1 \lor Q2\}}$$

disjunction

Note: there are other rules, eg conjuction, quantifiers

Example:

$$\{\ (y>4) \land (z>1)\ \}\ y{:=}y+z\ \{\ y>3\ \}$$

Structural Rules

$$\frac{P \Longrightarrow P' \quad \{P'\}\ C\ \{Q'\} \quad Q' \Longrightarrow Q}{\{P\}\ C\ \{Q\}}$$

consequence

$$\frac{\{P1\}\ C\ \{Q1\} \quad \{P2\}\ C\ \{Q2\}}{\{P1 \lor P2\}\ C\ \{Q1 \lor Q2\}}$$

disjunction

Note: there are other rules, eg conjuction, quantifiers

Example:

$$\frac{\{\ y+z>5\ \}\ y:=y+z\ \{y > 5\}}{\{\ (y>4) \land (z>1)\ \}\ y:=y+z\ \{\ y>3\ \}}$$

Structural Rules

$$\frac{P \Longrightarrow P' \qquad \{P'\}\ C\ \{Q'\} \qquad Q' \Longrightarrow Q}{\{P\}\ C\ \{Q\}} \quad \text{consequence}$$

$$\frac{\{P1\}\ C\ \{Q1\} \qquad \{P2\}\ C\ \{Q2\}}{\{P1 \vee P2\}\ C\ \{Q1 \vee Q2\}} \quad \text{disjunction}$$

Note: there are other rules, eg conjuction, quantifiers

Example:

$$\frac{(y>4) \wedge (z>1) \Longrightarrow (y+z>5) \qquad \{\ y+z>5\ \}\ y:=y+z\ \{y > 5\}}{\{\ (y>4) \wedge (z>1)\ \}\ y:=y+z\ \{\ y>3\ \}}$$

Structural Rules

$$\frac{P \Longrightarrow P' \qquad \{P'\}\ C\ \{Q'\} \qquad Q' \Longrightarrow Q}{\{P\}\ C\ \{Q\}}$$

consequence

$$\frac{\{P1\}\ C\ \{Q1\} \qquad \{P2\}\ C\ \{Q2\}}{\{P1 \vee P2\}\ C\ \{Q1 \vee Q2\}}$$

disjunction

Note: there are other rules, eg conjuction, quantifiers

Example:

$$\frac{(y>4) \wedge (z>1) \Longrightarrow (y+z>5) \quad \{\ y+z>5\ \}\ y:=y+z\ \{y > 5\} \quad (y>5) \Longrightarrow (y>3)}{\{\ (y>4) \wedge (z>1)\ \}\ y:=y+z\ \{\ y>3\ \}}$$

Small Axioms

- { x=m /\ emp } x:=E { x=(E[m/x]) /\ emp}
- { E|->- } [E]:=F { E|->F }
- { x=m /\ E|->n } x:=[E] { x=n /\ E[m/x] |->n }
- { E|->- } dispose(E) {emp}
- { x=m /\ emp} x:=new(E1,...,Ek) { x|->E1[m/x],...,Ek[m/x] }

where x,m,n are assumed to be distinct variables

These axioms mention only the local state which is touched, called footprint

Observation

- A Hoare triple only describes the effect an action has on the portion of program store it explicitly mentions.
- It does not say what cells among those not mentioned remain unchanged.

Observation

- A Hoare triple only describes the effect an action has on the portion of program store it explicitly mentions.
- It does not say what cells among those not mentioned remain unchanged.

We want instead to say:

any state alteration not explicitly required by the specification is excluded

Idea: focus on footprint

- Change the interpretation of the Hoare triple {P} C {Q}, so that C must only dereference cells guaranteed to exists by P or allocated by C itself
- Add an inference rule to obtain bigger specifications from small ones.

Idea: focus on footprint

The portion of memory touched by a command

- Change the interpretation of the Hoare triple {P} C {Q}, so that C must only dereference cells guaranteed to exists by P or allocated by C itself
- Add an inference rule to obtain bigger specifications from small ones.

Memory faults

- Some commands can “go wrong” for example:
  - dispose(x) or [x]:=y or x:=[y]
- Examples:

```
x=new();
y:=x;
dispose(x);
[y]:=nil;
```

Memory faults

- Some commands can "go wrong" for example:
  - dispose(x) or [x]:=y or x:=[y]
- Examples:

```
x:=new();
y:=x;
dispose(x);
[y]:=nil;
```

Tight Interpretation of Triples

- The interpretation of the triples in separation logic ensures that a program does not fault!

$$
\{P\}\,C\,\{Q\} \text{ holds} \quad \text{iff} \quad
\begin{array}{l}
\forall s, h.\ \text{if } s, h \models P \text{ then} \\
\neg C, s, h \to^{*} \mathsf{err} \\
\text{and, if } C, s, h \to^{*} s', h' \text{ then } s', h' \models Q
\end{array}
$$

This ensure that a well-specified programs access only the cells guaranteed to exist in the precondition or created by C

Aliasing and Soundness

- In traditional Floyd-Hoare logic, the rule of constancy:

$$\frac{\{P\}\,C\,\{Q\}}{\{P \wedge R\}\,C\,\{Q \wedge R\}}\ \mathsf{Modify(C)} \cap \mathsf{Free(R)} = \emptyset$$

allows modular reasoning for sequential as well as parallel programs.

Aliasing and Soundness

- In traditional Floyd-Hoare logic, the rule of constancy:

$$\frac{\{P\}\,C\,\{Q\}}{\{P \wedge R\}\,C\,\{Q \wedge R\}}\ \mathsf{Modify(C)} \cap \mathsf{Free(R)} = \emptyset$$

allows modular reasoning for sequential as well as parallel programs.

This rule is unsound in presence of pointers

Aliasing and Soundness

- In traditional Floyd-Hoare logic, the rule of constancy:

$$\frac{\{P\}\,C\,\{Q\}}{\{P \wedge R\}\,C\,\{Q \wedge R\}}\ \mathsf{Modify(C)} \cap \mathsf{Free(R)} = \emptyset$$

allows modular reasoning for sequential as well as parallel programs.

This rule is unsound in presence of pointers

$$\frac{\{\ [x]=3\ \}\ [x]:=7\ \{\ [x]=7\ \}}{\{\ [x]=3\ /\backslash\ [y=3]\ \}\ [x]:=7\ \{\ [x]=7\ /\backslash\ [y]=3\}}$$

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}}\ \mathsf{Modifies}(C) \cap \mathsf{FV}(R) = \emptyset$$

R is the frame (it can be added as invariant)

* and err-avoiding triple take care of the heap access of C

The side condition takes care of the stack access

Note:
Modify(x:=E)=Modify(x:=[E])=Modify(x:=new(E1,..,Ek))={x} and Modify([E]:=F)=Modify(dispose(E))={}

Example using the Frame Rule

{x|->-} [x]:=z {x|->z}

{y|->c * x|->-} [x]:=3 {x|->z * y|->c}

Example

Let's assume:

{ x|->1,2 } C { z|-> 3,2 }

and C modifies only the heap.

Example

Let's assume:

{ x|->1,2 } C { z|-> 3,2 }

and C modifies only the heap.

If we give C more heap

{ x|->1,2 * y|->17,42} C { z|-> 3,2* ?????? }

Example

Let's assume:

{ x|->1,2 } C { z|-> 3,2 }

and C modifies only the heap.

If we give C more heap

{ x|->1,2 * y|->17,42} C { z|-> 3,2* y|->17,42 }

Example

Let's assume:

{ x|->1,2 } C { z|-> 3,2 }

and C modifies only the heap.

If we give C more heap

{ x|->1,2 * y|->17,42} C { z|-> 3,2* y|->17,42 }

We are sure that cell y cannot change otherwise we would have a fault and it would contradict the initial assumption where y is dangling

In-place Reasoning

{(x|-> -) * P} [x]:=7 {(x |->7)*P}

{true} [x]:=7 {???}

{(x|-> -) * P} dispose(x) {P}

{true} dispose(x) {???}

{P} x:=new() {(x|-> -) * P}   (x not in Free(P))

Proving a program

```
x = new(3,3);

y = new(4,4);

[x+1] = y;

[y+1] = x;

dispose x;
```

We discuss this more tomorrow

Proving a program

{exists n,m. x=n /\ y=m /\ emp}

```
x = new(3,3);

y = new(4,4);

[x+1] = y;

[y+1] = x;

dispose x;
```

We discuss this more tomorrow

Proving a program

```
{exists n,m. x=n /\ y=m /\ emp}
    x = new(3,3);
  {x|->3,3}
    y = new(4,4);

    [x+1] = y;

    [y+1] = x;

    dispose x;
```

We discuss this more tomorrow

Proving a program

```
{exists n,m. x=n /\ y=m /\ emp}
    x = new(3,3);
   {x|->3,3}
    y = new(4,4);
 {x|->3,3* y|->4,4}
    [x+1] = y;

    [y+1] = x;

    dispose x;
```

We discuss this more tomorrow

Proving a program

```
{exists n,m. x=n /\ y=m /\ emp}
    x = new(3,3);
   {x|->3,3}
    y = new(4,4);
 {x|->3,3* y|->4,4}
    [x+1] = y;
  {x|->3,y* y|->4,4}
    [y+1] = x;


    dispose x;
```

We discuss this more tomorrow

Proving a program

```
{exists n,m. x=n /\ y=m /\ emp}
    x = new(3,3);
  {x|->3,3}
    y = new(4,4);
 {x|->3,3* y|->4,4}
    [x+1] = y;
 {x|->3,y* y|->4,4}
    [y+1] = x;
 {x|->3,y* y|->4,x}
    dispose x;
```

We discuss this more tomorrow

Proving a program

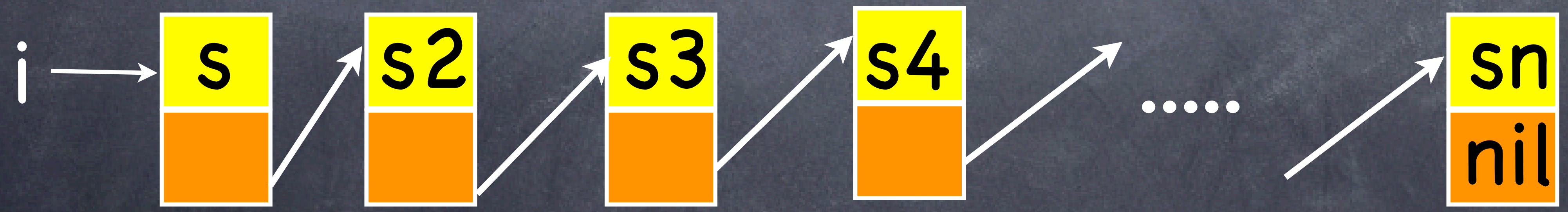
```
{exists n,m. x=n /\ y=m /\ emp}
    x = new(3,3);
  {x|->3,3}
    y = new(4,4);
 {x|->3,3* y|->4,4}
    [x+1] = y;
 {x|->3,y* y|->4,4}
    [y+1] = x;
 {x|->3,y* y|->4,x}
    dispose x;
 {x+1|->y* y|->4,x}
```

We discuss this more tomorrow

Lists

A non circular list can be defined with the following inductive predicate:

```
   list [] i   = emp /\ i=nil
list (s::S) i  = exists j. i|->s,j * list S j
```



Example

```
j:=[i+1];

dispose(i)

dispose(i+1)

i:=j;
```

Example

{list (a::S) i}

j:=[i+1];

dispose(i)

dispose(i+1)

i:=j;

Example

{list (a::S) i}
{exists j. i |->a,j * list S j}

j:=[i+1];

dispose(i)

dispose(i+1)

i:=j;

Example

{list (a::S) i}

{exists j. i |->a,j * list S j}

{ i|->a * exists j. i+1 |->j * list S j}

```
j:=[i+1];


dispose(i)


dispose(i+1)


i:=j;
```

Example

```
{list (a::S) i}
{exists j. i |->a,j * list S j}
{ i|->a * exists j. i+1 |->j * list S j}
j:=[i+1];
{ i|->a * i+1 |->j * list S j}
dispose(i)


dispose(i+1)


i:=j;
```

Example

{list (a::S) i}

{exists j. i |->a,j * list S j}

{ i|->a * exists j. i+1 |->j * list S j}

j:=[i+1];

{ i|->a * i+1 |->j * list S j}

dispose(i)

{ i+1 |->j * list S j}

dispose(i+1)

i:=j;

Example

{list (a::S) i}

{exists j. i |->a,j * list S j}

{ i|->a * exists j. i+1 |->j * list S j}

j:=[i+1];

{ i|->a * i+1 |->j * list S j}

dispose(i)

{ i+1 |->j * list S j}

dispose(i+1)

{ list S j }

i:=j;

Example

```
{list (a::S) i}
{exists j. i |->a,j * list S j}
{ i|->a * exists j. i+1 |->j * list S j}
j:=[i+1];
{ i|->a * i+1 |->j * list S j}
dispose(i)
{ i+1 |->j * list S j}
dispose(i+1)
{ list S j }
i:=j;
{ list S i }
```

Homework

Try to prove this triple (if you cannot do not worry).

```
{list (a::S) x *list(b::S') z }
y:=x;
x:=[x+1];
dispose(y);
dispose(y+1);
y=new(5,5);
[y+1]:=x;
{list (5::S) y *list (b::S) z}
```

look at the next slide

Use these rules:

For proving that program it may be easier to use the following rules (instead of small axioms)

{P} x:=E {exists x'. x=E[x'/x] /\ P[x'/x]}

{P*E|->F} x:=[E] {exists x'.x=F[x'/x] /\ (P*E|->F)[x'/x] }

{P*E|->F} [E]:=G { P*E|->G }

{P} x:=new(E) {exists x'. P[x'/x] * x |->E[x'/x]}

{P*E|->F} dispose(E) { P}

here x' is a fresh variable

References

- H. Yang and P. O'Hearn. A Semantic Basis for Local Reasoning. FOSSACS 2003.
- P. O'Hearn, J. Reynolds, and H. Yang. Local Reasoning about Programs that Alter Data Structures. CSL 2001.