

Dr. Liam O'Connor University of Edinburgh LFCS UNSW, Term 3 2020

Big O

We alknow that MERGE SORT has empty time complexity, and that $e^{1}p$ BUBBLESORT has $O(n^{2})$ time complexity, but what does that actually mean?

Big O Notation

Given functions 7,77998 1,

$$\forall x > x_0. \ f(x) \leq m \cdot g(x)$$

What is the codomain of f? We Chat powcoder

When analysing algorithms, we don't usually time how long they take to run on a real machine.

Cost Models

A conting is mathematical tradition of the colt of the

There exist *denotational* cost models, that assign a cost directly to syntax:

https://poweoder.com

However in this course we will focus on operational cost models. Operational Cost Models WeChat powcoder

First, we define a program-evaluating abstract machine. We can determine the time cost by counting the number of steps taken by the abstract machine.

Abstract Machines

Assignment Project Exam Help

Abstract Machines

An abstract machine consists of:

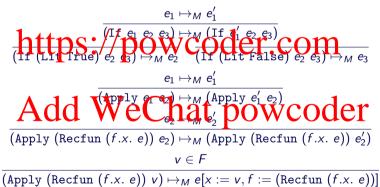
- A set of states / powcoder.com
 A set of initial states / ⊆ Σ,
- **3** A set of final states $F \subseteq \Sigma$, and

We've seen this before in structural operational (or small-step) semantics.

The M Machine

Is just our usual small-step rules:

Assignment Project Exam Help



The M Machine is unsuitable as a basis for a cost model. Why?

Performance

One step in our machine should always only be $\mathcal{O}(1)$ in our language implementation.

Other conting stable illust set in activite testripy of the time esting This makes for two potential problems:

- **1** Substitution occurs in function application, which is potentially $\mathcal{O}(n)$ time.
- Ochtrol Flore to Sexplicate White was to the sound by recursively descending the abstract syntax tree each time.

```
eval (Num n) = n
oneStep (Plus (Num n) (Num m))
oneStep (Plus (Num n) e_2) = Plus (Num n) (oneStep e_2)
oneStep (Plus e_1 e_2)
                                = Plus (oneStep e_1) e_2
. . .
```

The C Machine

We want to define a machine where all the rules are axioms, so there can be no recursive descent into subexpression. How is recursion typically implemented postacks!

https://powcoder.com

Key Idea: States will consist of a current expression to evaluate and a stack of computational contexts that strate it in the overall computation. An example stack would be:

 $(\text{Plus } 3 \square) \triangleright (\text{Times } \square (\text{Num } 2)) \triangleright \circ$

This represents the computational context:

(Times (Plus $3 \square$) (Num 2))

The C Machine

Our states will consist of two modes:

• Assignment of two

2 Return a value v (either a function, integer, or boolean) back into the context in s. written $s \prec v$.

Initial states ar hotel state state of the s $\circ \succ e$. Final states are those that return a value to the empty stack, i.e. $\circ \prec v$. Stack frames are expressions with holes or values in them:

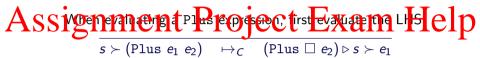
Ad	d	V ₂ /E	eC1	na	t 1	po	AMA (COC	der
	(D1,,	a \Box a) Eram	(יי ום	10.14	□\ E _E	ama	

(Plus $\sqcup e_2$) Frame (Plus $v_1 \sqcup$) Frame

. . .

Evaluating

There are three axioms about Plus now:



https://powered.switcherhee.PHS:
(Plus
$$\square e_2$$
) $\triangleright s \prec v_1 \mapsto_C$ (Plus $v_1 \square$) $\triangleright s \succ e_2$



We also have a single rule about Num that just returns the value:

$$s \succ (\text{Num } n) \mapsto_C s \prec n$$

Example

```
Assignment Project Exam, Help
              \mapsto_{\mathcal{C}} (Plus \square (Num 3)) \triangleright (Plus \square (Num 4)) \triangleright \circ \succ (Num 2)
             https://poweoder.com
              \mapsto_{\mathcal{C}} (Plus 2 \square) \triangleright (Plus \square (Num 4)) \triangleright \circ \succ (Num 3)
                    (Plus 2 \square) \triangleright (Plus \square (Num 4)) \triangleright \circ \prec 3
              Add. We Chat powcoder
             \mapsto_{\mathcal{C}} (Plus 5 \square) \triangleright \circ \succ (Num 4)
             \mapsto_{\mathcal{C}} (Plus 5 \square) \triangleright \circ \prec 4
             \mapsto c \circ \prec 9
```

Other Rules

We have significant the property between the land of t

```
\frac{s \succ (\text{If } e_1 \ e_2 \ e_3) \quad \mapsto_{\mathcal{C}} \quad (\text{If } \square \ e_2 \ e_3) \triangleright s \succ e_1}{\text{https://powcoder.com}}
```

Add We Chat powcoder

 $(\text{If } \square e_2 e_3) \triangleright s \prec \text{False} \mapsto_C s \succ e_3$

Functions

Assignment Project Exam Help $s \succ (\operatorname{Fun}(f.x.\ e)) \mapsto_{\mathcal{C}} s \prec \langle \langle f.x.\ e \rangle \rangle$

Function application is then hardled similarly to Plus COM POWCOGER.COM

$$s \succ (\text{Apply } e_1 \ e_2) \quad \mapsto_C \quad (\text{Apply } \square \ e_2) \triangleright s \succ e_1$$

(Apply Adds We Chat paywe coder > e2

$$\overline{(\operatorname{Apply} \, \langle \langle f.x. \, e \rangle \rangle \, \Box) \, \triangleright \, s \prec v \quad \mapsto_{\mathcal{C}} \quad s \prec e[x := v, f := (\operatorname{Fun} \, (f.x.e))]}$$

We are still using substitution for now.

What have we done?

Assignment Project Exam Help

- All the rules are axioms we can now implement the evaluator with a simple while look or a simple while while look or a simple while look or a simple while look or a sim
- We have a lower-level specification helps with code generation (e.g. in ar assembly language)
- Substitution is still methine occupation we need to find a way to eliminate that.

 Add WeChat powcoder

Correctness

While the M-Machine is reasonably straightforward definition of the language's we wish to prove a theorem that tells us that the C-Machine behaves analogously to the M-Machine.

Refinement

A low-level (concrete) semantics of Oppogram is definement of a high-level (abstract) semantics if every possible execution in the low-level semantics has a corresponding execution in the high-level semantics. In our case:

Functional correctness properties are preserved by refinement, but security properties are not (cf. Dining Cryptographers).

How to Prove Refinement

We can't get away with simply proving that each C machine step has a corresponding step in the M-Machine, because the S-Machine makes multiple steps that are no-ops in the M-Schreiment Project Exam Help

```
\mapsto_{\mathcal{C}} (+ \square (N 4)) \triangleright \circ \succ (+ (N 2) (N 3))
Https://powcoder.com
\mapsto_C (+ \square (N 3)) \triangleright (+ \square (N 4)) \triangleright \circ \prec 2
\mapsto_C (+ 2 \square) \triangleright (+ \square (N 4)) \triangleright \triangleright \succ (N 3)
Add We Chat powcoder
\mapsto_{\mathcal{C}} (+ \square (N 4)) \triangleright \circ \prec 5
                                                                       \mapsto_{M} (+ (N 5) (N 4))
\mapsto_{\mathcal{C}} (+ 5 \square) \triangleright \circ \succ (N 4)
\mapsto_{\mathcal{C}} (+ 5 \square) \triangleright \circ \prec 4
\mapsto c \circ \prec 9
```

How to Prove Refinement

Assignment Project Exam Help Define an Assignment $P_{A}: \Sigma_{C} \to \Sigma_{M}$ that relates C-Machine states to

- M-Machine states, describing how they "correspond".
- Prove that for all initial states $\sigma \in I_C$, that the corresponding state $\mathcal{A}(\sigma) \in I_M$.

 Prove for each trep in the Cocce Little Office.
- - the step is a no-op in the M-Machine and $\mathcal{A}(\sigma_1) = \mathcal{A}(\sigma_2)$, or
 - the step is replicated by the M-Machine $\mathcal{A}(\sigma_1) \mapsto_M \mathcal{A}(\sigma_2)$.
- Prove that Ar of that Av be a first the construction of the $A(\sigma) \in F_M$.

In general this abstraction function is called a *simulation relation* and this type of proof is called a *simulation* proof.

The Abstraction Function

Our abstraction function \mathcal{A} will need to relate states such that each transition that corresponds to a no-op in the M-Mohine will move between \mathcal{A} -equivalent states:

```
\mapsto_C (+ \square (N 4)) \triangleright \triangleright (+ (N 2) (N 3)) -
\mapsto_C (+ \square (N 3)) \triangleright (+ \square (N 4)) \triangleright \circ \triangleright
                                                                              wcoder.com
\mapsto_{\mathcal{C}} (+ 2 \square) \triangleright (+ \square (N 4)) \triangleright \circ \succ (N 3)
                                                               VeChat powcoder
\mapsto_C (+ 5 \square) \triangleright \circ \succ (N 4)
\mapsto_C (+ 5 \square) \triangleright \circ \prec 4
\mapsto c \circ \prec 9 -
                                                                              \xrightarrow{+\rightarrow M} (N 9)
```

Abstraction Function

Given a State With a stack and a current expression (or value), we reconstruct the overall expression to get the corresponding M-Machine state.

$$\begin{array}{ll} \underset{\mathcal{A}((\mathsf{plus}\;\Box\;e_2))}{\text{https://powcoder.com}} \\ \mathcal{A}((\mathsf{plus}\;\Box\;e_2) \triangleright s \succ e_1) &=& \mathcal{A}(s \succ (\mathsf{plus}\;e_1\;e_2)) \end{array}$$

By definition, all the initial/final states of the C-Machine are mapped to initial/final states of the M-Machine. So all that is left is the requirement for each transition.

Showing Refinement for Plus

Assignment Project Exam Help

```
This is a no-oplate parameter S \succ (\text{Plus } e_1 \ e_2) \mapsto_C (\text{Plus } \square \ e_2) \triangleright s \succ e_1
\mathcal{A}(RHS) = \mathcal{A}((\text{Plus } \square \ e_2) \triangleright s \succ e_1)
\text{Add WeChief Powcoder}
```

Showing Refinement for Plus

Assignment Project Exam Help

```
Another no-op interposic in powcoder. Com
A(LHS) = A((Plus \square e_2) \triangleright s \prec v_1)
Add \ We \ A(RHS)
A(RHS) = A(RHS)
```

Showing Refinement for Plus

Assignment Project Exam Help

Technically the reduction step (*) requires induction on the stack.