CS131: Programming Languages

Fall 2015
Week #3

Today

- HW 2 Clarifications
- Java / HW 3

HW2: Naive Parsing of CFGs

- A parser generator
- Submission due: Oct 16, 11:55 pm

parse_prefix: what to return

- parse_prefix should return a matcher that returns Some(derivation, suffix) or None
  - `parse_prefix gram -> matcher`
  - `parse_prefix gram accept frags -> Some(...)|None`
- What derivation to generate?
  - Prefer a full derivation? No.
  - Prefer first derivation accepted, full or partial

Left Recursion / Blind Alleys

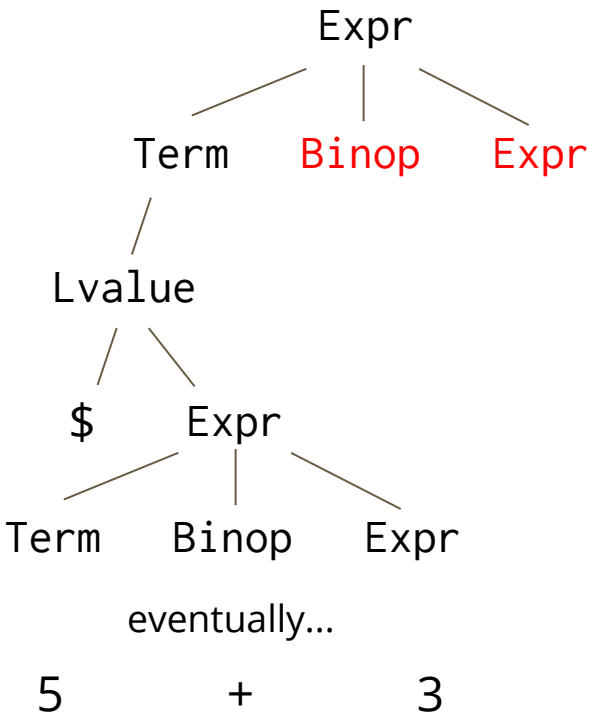
- Don’t need to handle left recursion

  `A -> A | “hello”`

- DO need to handle blind alleys
  - But not like those in hw1
  - Will not be given grammar with blind alley rules

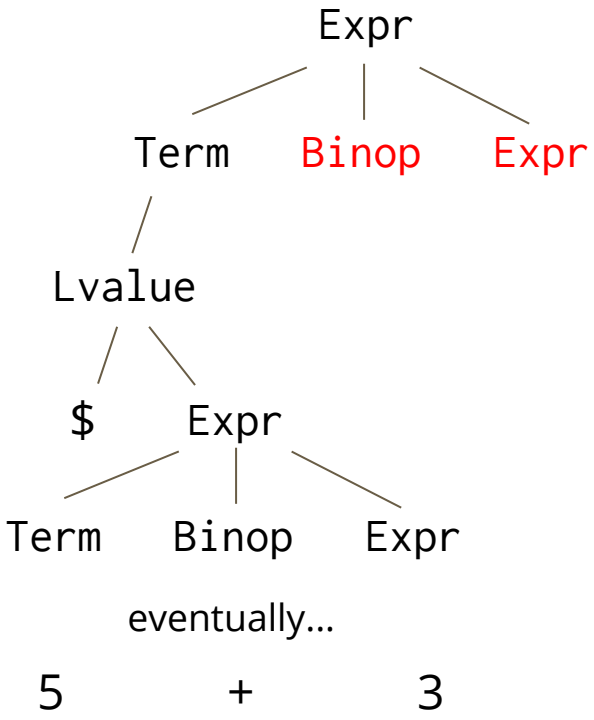
Handling Blind Alleys

- Not like in HW1.
- Blind alley: we matched everything but remain with non terminals
- Ex: [“$”; “5”; “+”; “3”]

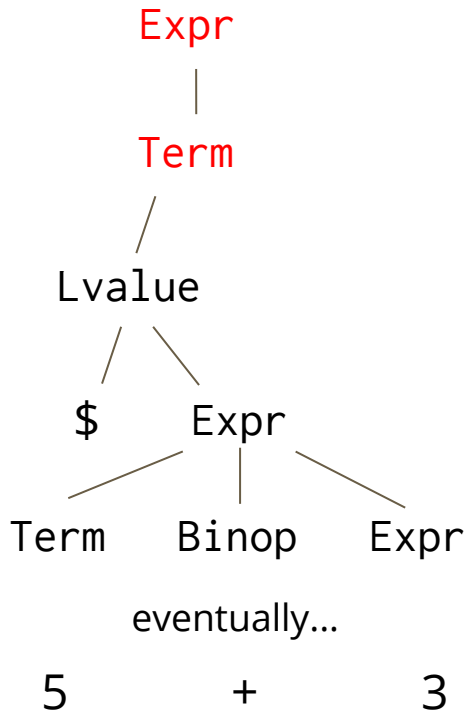


Handling Blind Alleys

- What do we do?

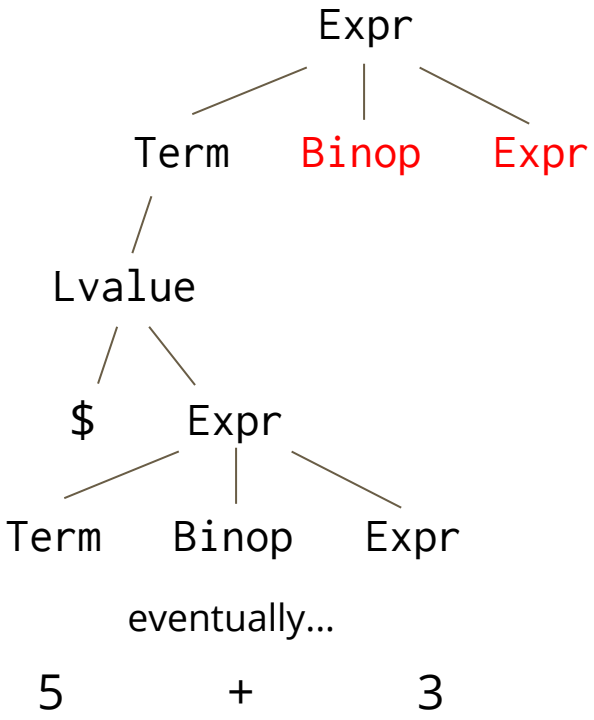


- Naive `or_matcher`

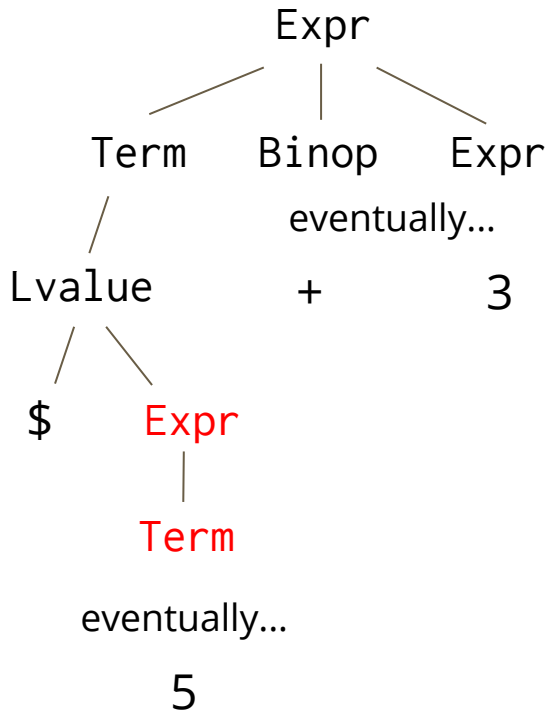


Handling Blind Alleys

- What do we do?



- Test case 4 shows correct behavior:



Handling Blind Alleys

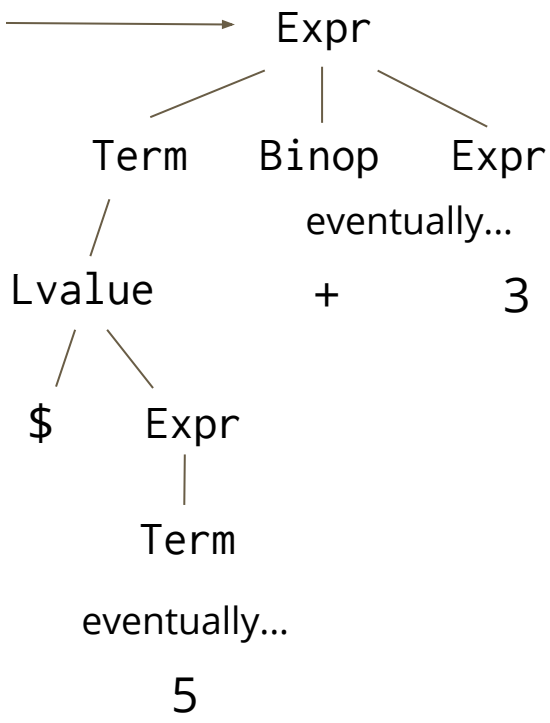
- Must either
  - Backtrack
  - Generate list of all possible derivations
    - Return the first that accepts

When to apply the acceptor

- At each level of match?

When to apply the acceptor

- ~~At each level of match?~~
- No, at top level
- Apply acceptor once, here



When to apply the acceptor

- ~~At each level of match?~~
- No, at top level
- Apply acceptor once
- Intention of `match_empty_suffix`:
  - Match only full derivation
  - Won’t work if we call acceptor at every level

HW3

- Due October 23, 11:55pm
- JMM - Java Memory Model
- Measure performance & reliability of various state models
- Create new state models

Concurrency Issues

- Data race
  - When two threads operate on shared memory
  - Result may depend on order of operations
- Deadlocks
  - When two threads are waiting for each other's operation to finish

Java Memory Model (JMM)

- final
- volatile
- synchronized

Final

- Similar to `const` in C++
- Different in that you can assign to a final variable at most once.

Volatile Keyword

- Thread 1

```
(1)  answer = 22;
(2)  ready = true;
```

- Thread 2

```
(3) if (ready)
(4) println (answer);
```

Shared Memory & Swap

- An **array** of integers in shared memory
  - Integers between 0 and 127 → represent as byte

```
byte[] arr = new byte[5]; // default values: 0

byte[] arr = {1,2,3,4,5};
```

- **Swap**: decrement one element, increment another
  - Sum stays the same

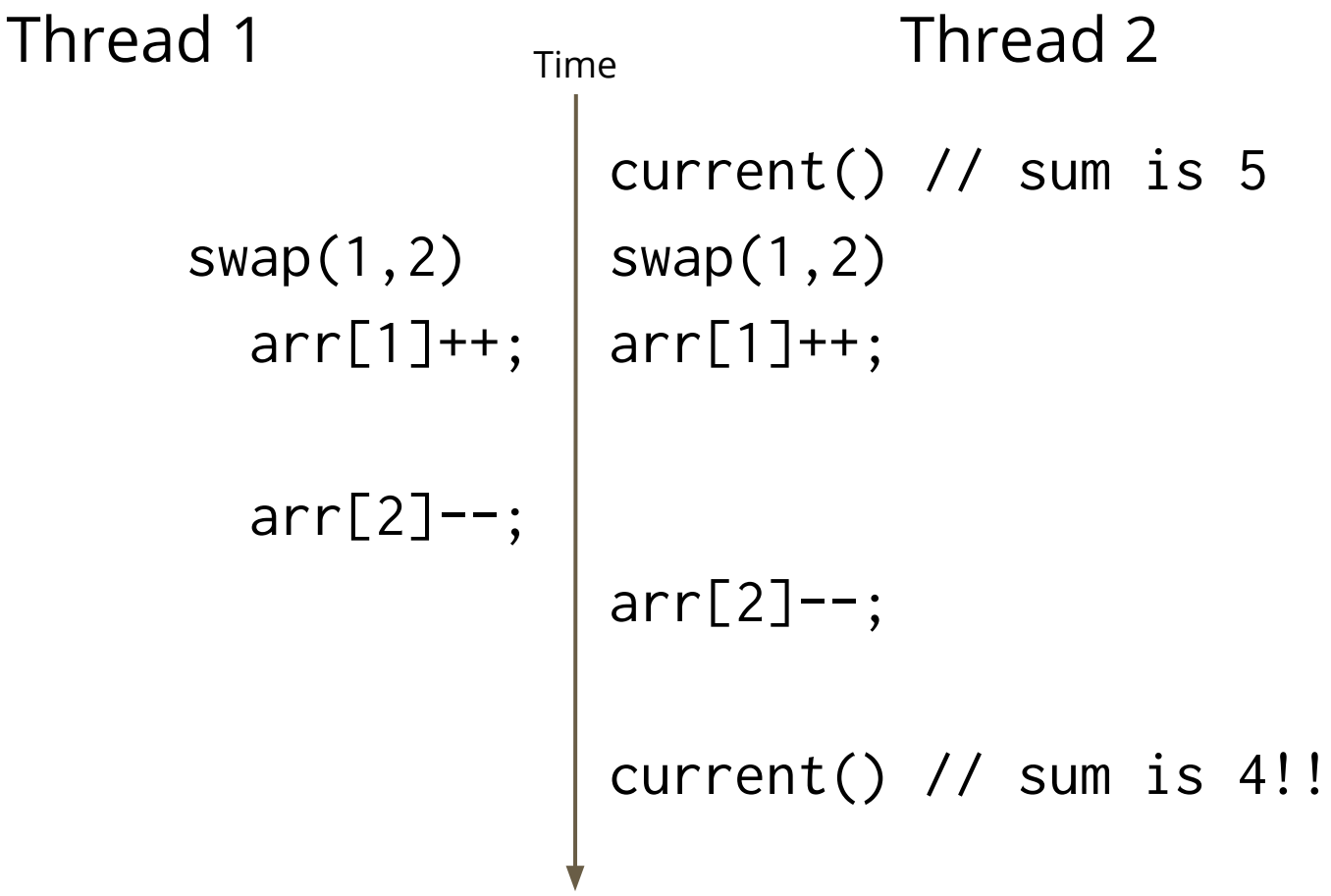
```
arr[0]++; arr[1]--; // arr = {2,1,3,4,5}
```

- Threads will observe a consistent sum
  - Unless there are **data races**

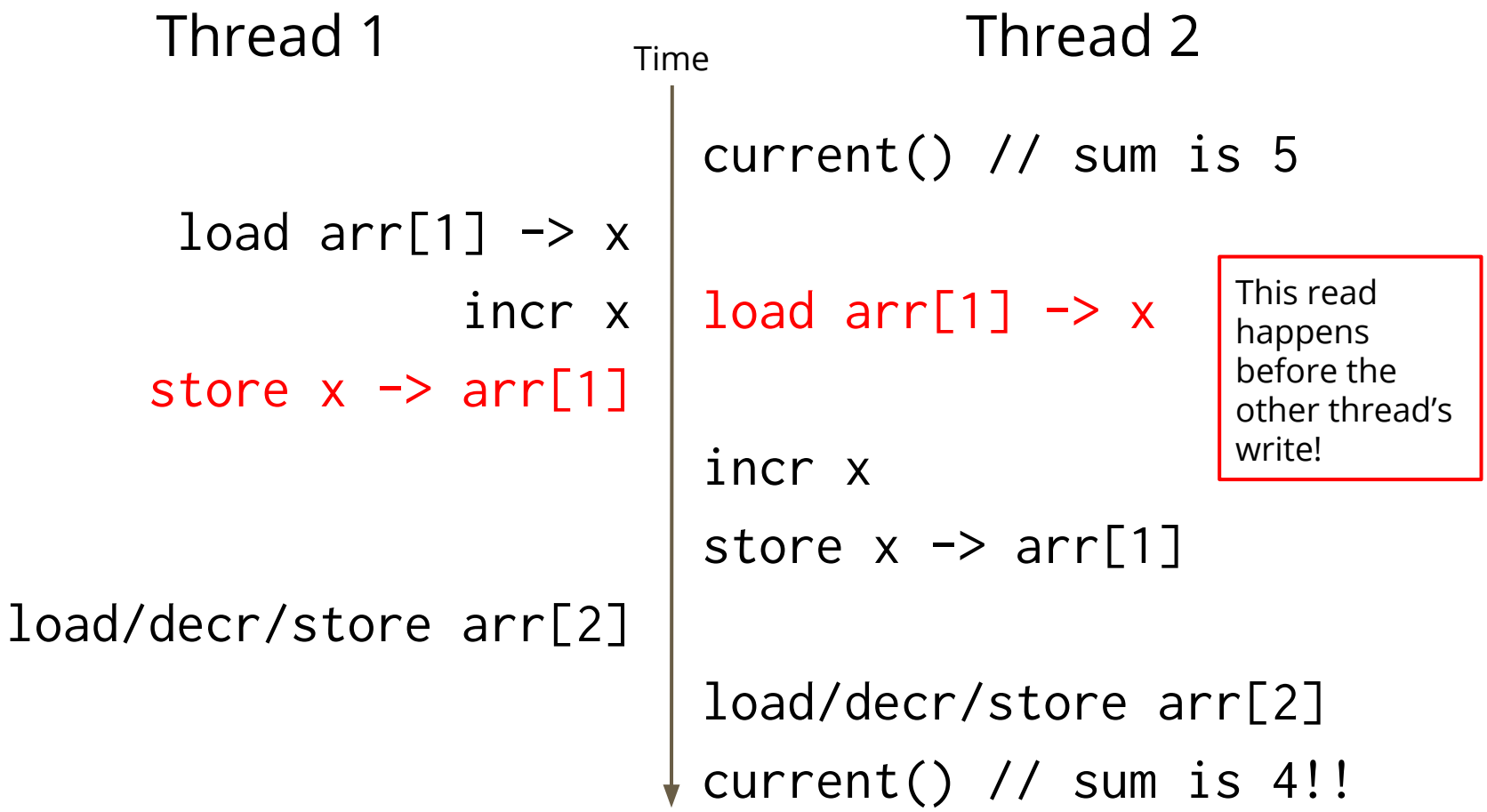
State Model

- Keeps a **reference** to the array in shared memory
- A model is a Java class that implements the `State` interface
  - `public int size()`
    - length of the array
  - `public byte[] current()`
    - See the current state of the array
  - `public boolean swap(int i, int j)`
    - Do the swap operation

Example Data Race



Expanded Example Data Race



SynchronizedState

- Makes `swap` a synchronized method
- `public synchronized boolean swap(int i,int j)`
- The first call to `swap` will block all other calls to `swap` until completion

Models to Create

- `UnsynchronizedState` - like `SynchronizedState`, but without the `synchronized` keyword
- `GetNState - volatile array access`
- `BetterSafeState - better than Synchronized in perf`
- `BetterSorryState - even better, but < 100 reliable`

Building & Testing Performance

- javac
- `java UnsafeMemory SynchronizedState 8 1000000 10 1 2 3 4 5`