Concurrent Collections are Deterministic

Jens Palsberg

UCLA Computer Science Department

University of California, Los Angeles

palsberg@ucla.edu

This talk

- Determinism = deterministic input-output behavior
- Want: determinism by design; nondeterminism is a bug
- Theme: abstraction ➔ determinism
- Goal 1: run-time check of determinism plus proof
  - Concurrent collections are deterministic: why and how
- Goal 2: compile-time check plus low annotation burden
  - Plasma simulation in X10: is load balancing deterministic?

Analogy

- Determinism in the 2010s = type checking in the 1980s

- The code was designed to satisfy the property
- The developers think the code satisfies the property
- Testing suggests the code satisfies the property
- The expert developers don’t want a new type system
- Average programmers need help!

Concurrent collections are deterministic

- **Idea 1: dynamic single assignment**
  - Or: forbid races
- **Idea 2: blocking read**
  - Or: forbid uninitialized-variable errors

Featherweight CnC syntax

- **Program:** $p ::= f_i \text{ (int a) } \{ d_i \; s_i \} , \quad i \in 1..n$
- **Declaration:** $d ::= n = \text{data.get(e); } d$

  $\quad\quad | \quad \varepsilon$
- **Statement:** $s ::= \text{skip}$

  $\quad\quad | \quad \text{if (e>0) s1 else s2}$

  $\quad\quad | \quad \text{data.put(e1,e2); s}$

  $\quad\quad | \quad \text{prescribe } f_i\text{(e); s}$
- **Expression:** $e ::= \ldots$

Featherweight CnC semantics (1/3)

A state in the semantics: (A, T) or error

T ::= T || T | d s

(A, if (e>0) s1 else s2)) ➔ (A, s1) (if eval(e) >0)

(A, if (e>0) s1 else s2)) ➔ (A, s2) (if eval(e) ≤ 0)

(A, prescribe f_i(e); s) ➔ (A, (d_i, s_i)[a := eval(e)] || s)

Featherweight CnC semantics (2/3)

$$(A, \text{skip} \,||\, T2) \rightarrow (A, T2) \qquad\qquad (A, T1 \,||\, \text{skip}) \rightarrow (A, T1)$$

$$\frac{(A, T1) \rightarrow \text{error}}{(A, T1 \,||\, T2) \rightarrow \text{error}} \qquad\qquad \frac{(A, T2) \rightarrow \text{error}}{(A, T1 \,||\, T2) \rightarrow \text{error}}$$

$$\frac{(A, T1) \rightarrow (A', T1')}{(A, T1 \,||\, T2) \rightarrow (A, T1' \,||\, T2)} \qquad\qquad \frac{(A, T2) \rightarrow (A', T2')}{(A, T1 \,||\, T2) \rightarrow (A, T1 \,||\, T2')}$$

Traditional semantics

(A, item.put(e1,e2); s) ➔ (A[eval(e1) := eval(e2)]; s)

(A, n = data.get(e); d s) ➔ (A, (d s)[n := A(eval(e))])

if A(eval(e)) is defined

(A, n = data.get(e); d s) ➔ error, if A(eval(e)) is undefined

CnC semantics (3/3)

(A, item.put(e1,e2); s) ➔ (A[eval(e1) := eval(e2)]; s)

(dynamic single assignment) if A(eval(e1)) is undefined

(A, item.put(e1,e2); s) ➔ error, if A(eval(e1)) is defined

(A, n = data.get(e); d s) ➔ (A, (d s)[n := A(eval(e))])

(blocking read) if A(eval(e)) is defined

~~(A, n = data.get(e); d s) ➔ error, if A(eval(e)) is undefined~~

UCLA COMPUTER SCIENCE DEPARTMENT

Concurrent collections are deterministic

We use σ to range over states.

A final state is one of:

- **(A, skip)**
- **error**
- **(A, T) that hangs: all reads are blocked**

Theorem: If $\sigma \rightarrow^* \sigma'$ and $\sigma \rightarrow^* \sigma''$ and σ', σ'' are final states,

then $\sigma' = \sigma''$.

UCLA
COMPUTER SCIENCE DEPARTMENT

Proof of determinism

Lemma: (Confluence, Church-Rosser [1936])

If $\sigma \rightarrow^* \sigma'$ and $\sigma \rightarrow^* \sigma''$,

then there exists σ_c such that $\sigma' \rightarrow^* \sigma_c$ and $\sigma'' \rightarrow^* \sigma_c$

$$\begin{tikzcd} & \sigma \arrow[dl, "*"'] \arrow[dr, "*"] & \\ \sigma' \arrow[dr, "*"'] & & \sigma'' \arrow[dl, "*"] \\ & \sigma_c & \end{tikzcd}$$

Switching gears to X10 now

- X10 = imperative language
  + async // ***parallelism***
  + finish // ***synchronization***
- Want: determinism without annotations
- Determinism = may-happen-in-parallel analysis
  + effect analysis

Plasma simulation in Fortran ➔ X10

- Originally programmed in Fortran + MPI + pthreads by Viktor Decyk at UCLA

- Particle-in-cell computation
- Has run on many parallel hardware platforms over 20 years
  - 12 billion particles on 4,096 processors for months
- 3D version: 100,000 lines of Fortran
- 2D version: 10,000 lines of Fortran
  - 4,623 lines of X10

May-happen-in-parallel analysis [PPOPP 2010]

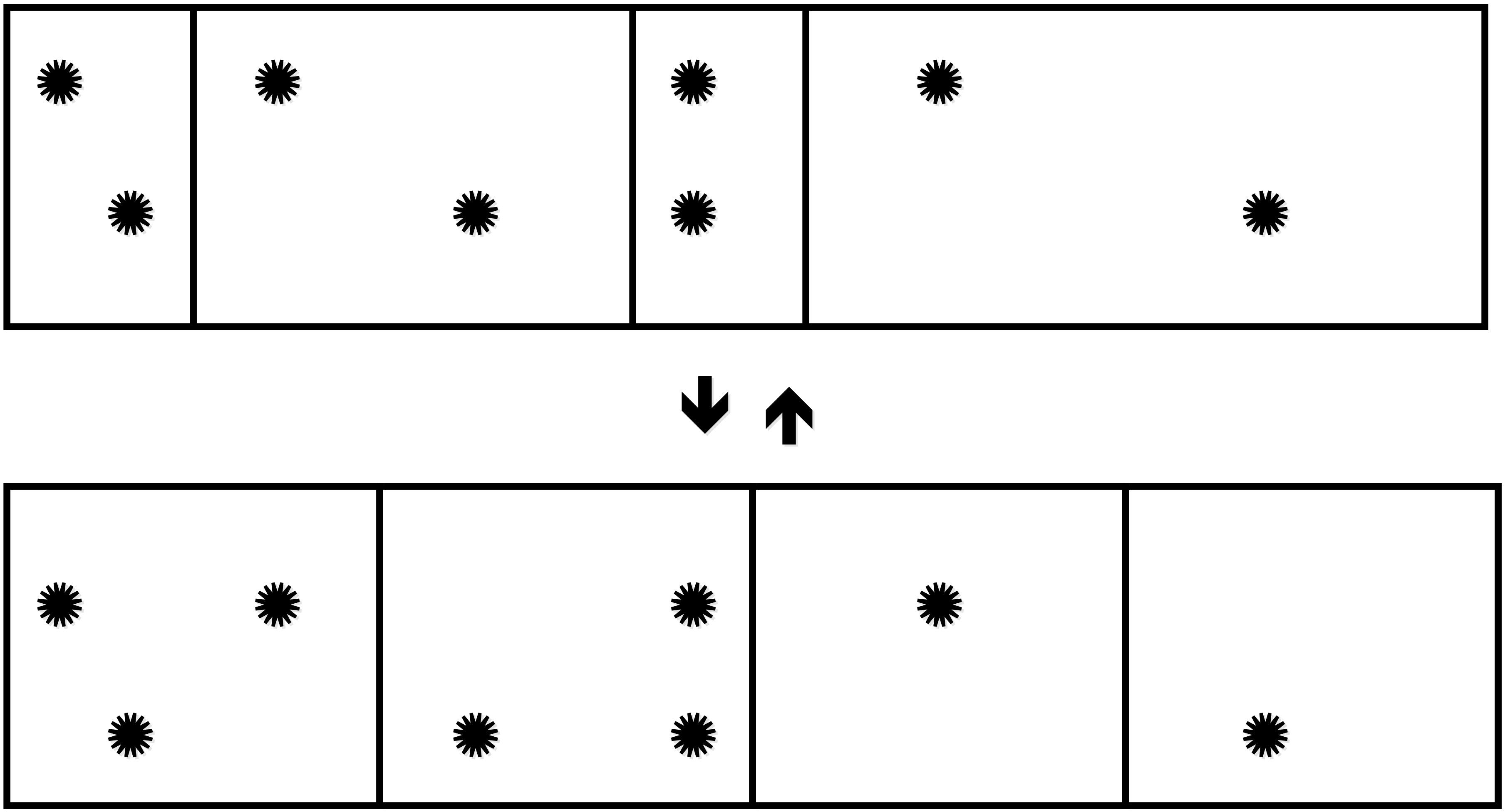
- For a program p,
  let MHP(p) be the true may-happen-in-parallel information
- **Theorem**: if $\vdash p : E$, and $E(main) = (M,O)$
  then $MHP(p) \subseteq M$
- Type inference: given p, find E such that $\vdash p : E$
- Type inference by constraint solving

MHP analysis of the plasma benchmark

- plasma benchmark: 4,623 LOC; 151 async; 84 finish; 505 call
- 11,422 constraints
- About 16 seconds to solve the constraints (257 MB space)
- #pairs of async bodies that may happen in parallel:
  - Self: 134 (in parallel with itself)
  - Same: 120 (two different async bodies in the same method)
  - Diff: 4 (two async bodies in different methods)
  - Total: 258
- We didn't find any false positives!

UCLA
COMPUTER SCIENCE DEPARTMENT

Plasma simulation does load balancing



Plasma simulation benchmark

```
class Particle { … } …
dist(:rank == 1) d = …;
Particle[:rank == 1] a = new Particle[d](…);
while (…) {   // 200,000 iterations
   … a[i] = … a[j] …                          // 2/3 computation
   d = …;
   a = new Particle[d](…a[i]…a[j]…);    // 1/3 communication
}
```

Conclusion

- Average programmers need determinism
- Determinism for CnC !
- Determinism for X10 ?   Annotations?
- Challenge: load balancing for arrays of references