Designing a Concurrent Programming Language

Sven Stork

February 5, 2009

Outline

Motivation

Motivation

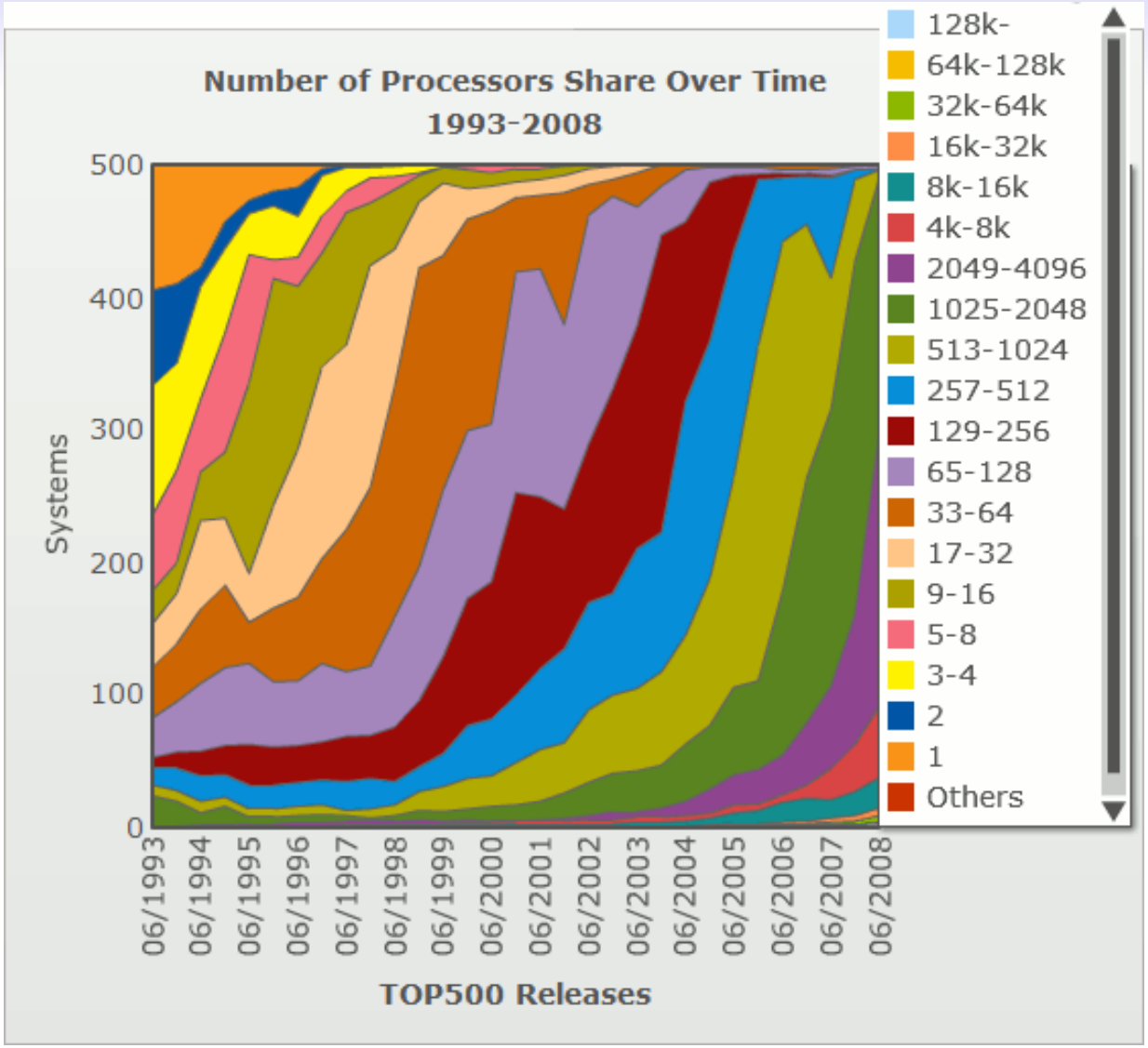
Why are we looking into concurrent programming ?

- concurrent programming is the future (no choice)
- parallel programming was not mainstream in the last decades
  - there is no (good) support for parallel programming in current programming languages
  - only in domain specific areas (e.g HPC) research for parallel programming
  - concepts and approaches of domain specific areas are most of the time no suitable for general purpose programming

glimpse at the future

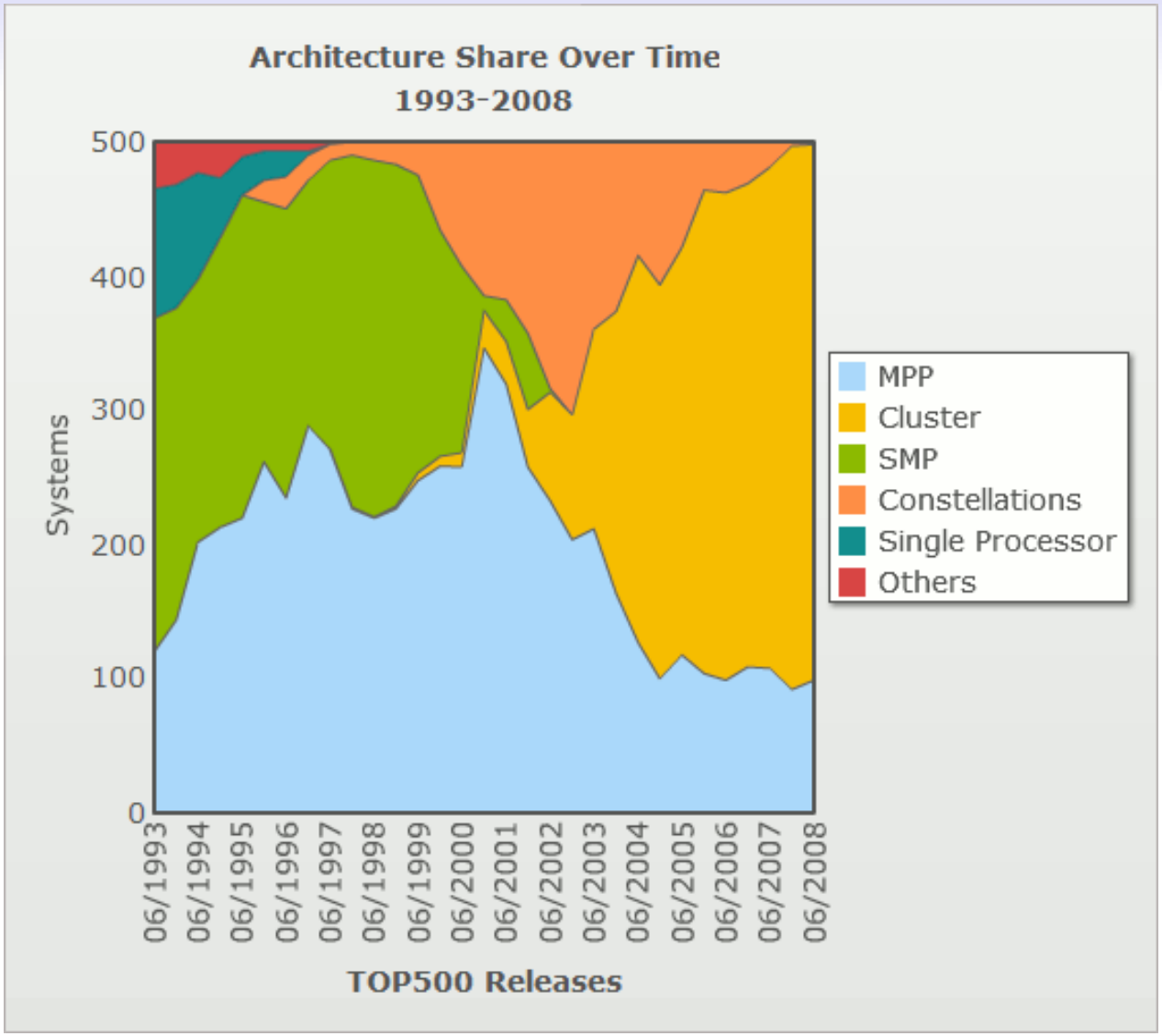
"HPC systems are about 20 year ahead of mainstream systems"

Motivation



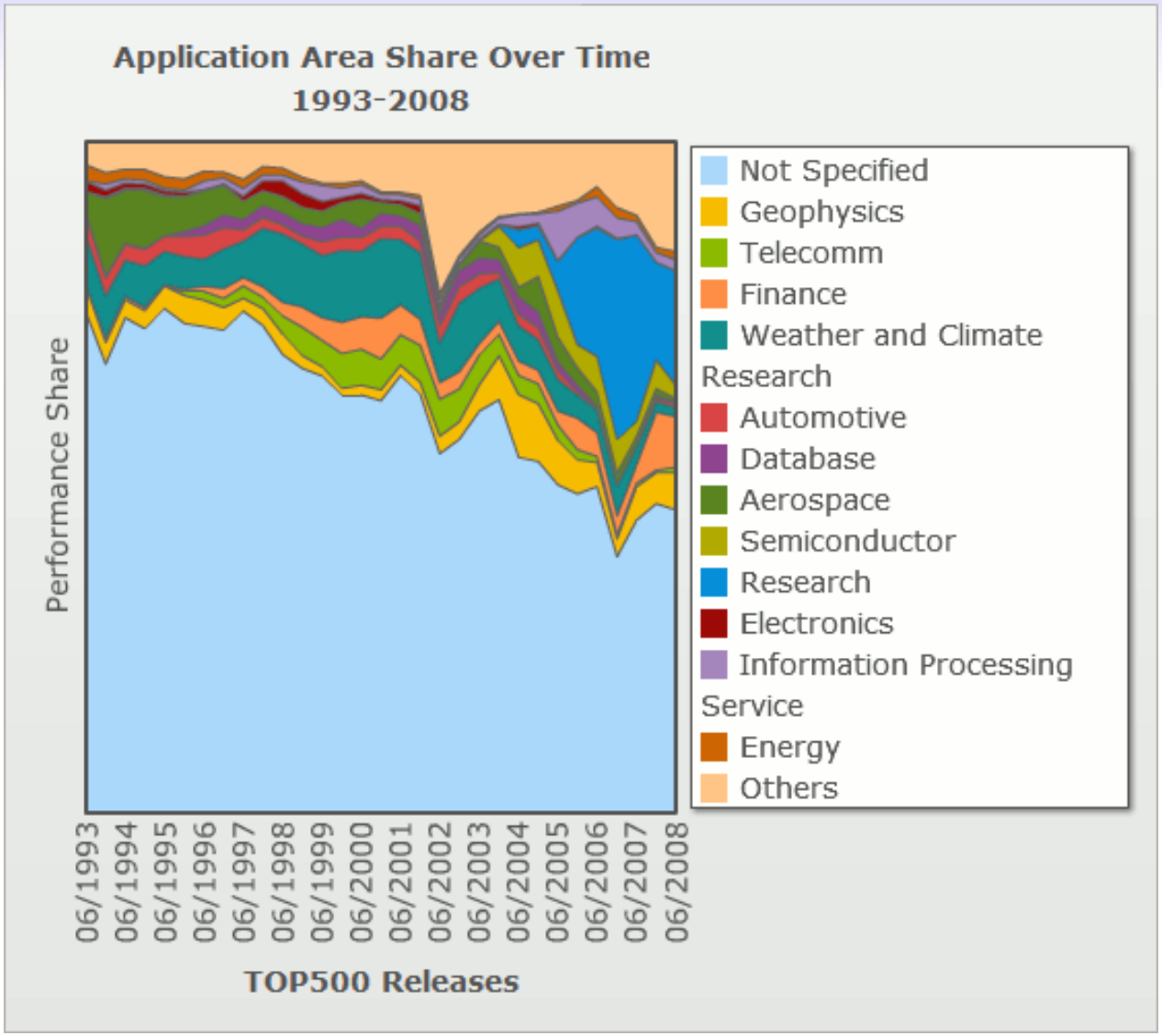
http://www.top500.org

Motivation



http://www.top500.org

Motivation



http://www.top500.org

Motivation

What we are looking for ?

How to write concurrent applications for large-scale concurrent systems ?

- concurrency should be implicit
  - describing what to do versus how to do it
  - no dealing with low level primitives (e.g. threads, locks, . . .)
  - no reasoning about possible execution paths
  - no thinking about concrete data distribution
- the runtime should be in charge for optimal execution and distribution

Language Design

Language Design

Terminology/Concepts

- Syntax
- Semantics
  - Operational Semantics
  - Static Semantics
- Inference Rules
- Grammer

Syntax

Syntax

- **"Notation"**
  only specifies legal language constructs and **not** meaning ($\rightarrow$ operational semantics)
- What's the meaning of : $a = 1$
  - ASSIGNMENT : store 1 in a
  - COMPARISON : is a equals to 1
  - SHIFT : shift a by one bit
  - ADDITION : add 1 to a
  - ...
- Syntax is in general defined by specific grammar (e.g BNF)

Syntax/Grammar

Example

$e ::= \overline{n} \mid tt \mid ff \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e$

e	expression
n	natural number
tt	true value
ff	false value
if ...	conditional

Inference Rules

Definition

Inference Rule

$$\frac{J_1, \ldots, J_n}{J}$$

If $J_1, \ldots, J_n$ are derivable the J is derivable.

Example

$$\frac{\begin{array}{lll} \mathit{Monitor} & \mathit{Mouse} & \mathit{Keyboard} \\ \mathit{Mainboard} & \mathit{CPU} & \mathit{Memory} \\ \mathit{Harddrive} & \mathit{DVD-Drive} & \mathit{Power-Supply} \end{array}}{\mathit{Computer}}$$

Operational Semantics

Arithmetic

$e ::= \overline{n} \mid e + e$

$(1+2+3) \rightarrow ???$

- What does this mean ?
- In which order are the operations executed ?
- What is the result ?

Big-Step Semantics

Example

$e ::= \overline{n} \mid e + e$

$1 + 2 + 3$

Big-Step Semantics

- **Big-Step Semantics** describe the evaluation of an whole expression.
- **Notation** $e \Downarrow n$
- **Example** $1 + 2 + 3 \Downarrow 6$

Small-Step Semantics

Example

$e ::= \overline{n} \mid e + e$

$1 + 2 + 3$

Small-Step Semantics

- **Small-Step Semantics** = "Model of Computation" describe exactly in which order which operation are performed (state $\rightarrow$ state).
- **Notation** $e \mapsto e'$ $\quad$ $(e \mapsto^* e'' \quad KleeneStar, 0 - nsteps)$
- **Example** $1 + 2 + 3 \mapsto 3 + 3$

Small-Step Semantics

Example

$e ::= n \mid e + e$

$1 + 2 + 3$

Small-Step Semantics for Arithmetic

$$\underbrace{\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2} \quad \frac{v_1 \ \mathit{value} \quad e_2 \mapsto e_2'}{v_1 + e_2}}_{\mathit{compatibility\ rules}} \qquad \underbrace{\frac{}{\overline{m} + \overline{n} \mapsto m + n}}_{\text{"}\mathit{reduction}\text{"}\ \mathit{rule}}$$

Small-Step Semantics

Example

$$\frac{}{1 + 2 + 3 \mapsto^*}$$

Small-Step Semantics

Example

$$\dfrac{\dfrac{}{1 + 2 \mapsto} \qquad \dfrac{}{3\ \texttt{value}}}{1 + 2 + 3 \mapsto^*}$$

Small-Step Semantics

Example

$$\dfrac{\dfrac{\dfrac{}{1\ \texttt{value}} \quad \dfrac{}{2\ \texttt{value}}}{1+2 \mapsto} \quad \dfrac{}{3\ \texttt{value}}}{1+2+3 \mapsto^*}$$

Small-Step Semantics

Example

$$\dfrac{\dfrac{\dfrac{}{1\ \texttt{value}} \quad \dfrac{}{2\ \texttt{value}}}{1+2 \mapsto 3\ \texttt{value}} \quad \dfrac{}{3\ \texttt{value}}}{1+2+3 \mapsto^*}$$

Small-Step Semantics

Example

$$\dfrac{\dfrac{\dfrac{}{1\ \texttt{value}} \quad \dfrac{}{2\ \texttt{value}}}{1+2 \mapsto 3\ \texttt{value}} \quad \dfrac{}{3\ \texttt{value}}}{1+2+3 \mapsto^* 6\ \texttt{value}}$$

Static Semantics

Static Semantics

- What is the result of : **1+2+"H"** ?
- *to* "12H"
- *to* 75
- error

Static Semantics

Static Semantics

- What is the result of : **1+2+"H"** ?
- *to* "12H"
- *to* 75
- error
- **We don't know!** We didn't provide a syntax nor operational semantics.

Static Semantics

Arithmetic Language

$e ::= \overline{n} \mid e + e$

$$\underbrace{\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2} \qquad \frac{v_1 \ value \quad e_1 \mapsto e_2'}{v_1 + e_2}}_{compatibility\ rules} \qquad \underbrace{\frac{}{\overline{m} + \overline{n} \mapsto m + n}}_{''reduction''\ rule}$$

- Now he know that **1+2+"H"** is not defined in our language.
- But when to we discover the error ?

Static Semantics

Arithmetic Language

$e ::= \overline{n} \mid e + e$

$$\underbrace{\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2} \quad \frac{v_1 \ value \quad e_1 \mapsto e_2'}{v_1 + e_2}}_{compatibility\ rules} \qquad \underbrace{\frac{}{\overline{m} + \overline{n} \mapsto m + n}}_{''reduction''\ rule}$$

- Now he know that **1+2+"H"** is not defined in our language.
- But when to we discover the error ?
  - After we executed **1+2** and try to execute **3+"H"**
  - So far we have no way of verifying that the specified program is correct (except parser checking).
  - This might after the program executed for a long time

Static Semantics

Types

- To verify that the program doesn't run into undefined expressions use **types**
- **Types** classify terms and allow static checking of programs
- **Notation** $e : \tau$

Definition Stuck

A term e is stuck if e is **not a value** and e does **not take a step**.

Static Semantics

Theorem: Type Safety

If $e : \tau$ and $e \mapsto^* e'$ then e' is **not stuck**.

Intuition

If you start with a well types term/program you will never run into situations where the systems doesn't know what to do (got stuck).

Static Semantics

Lemma: Progress

If $e : \tau$ then either e `value` or e takes a step.

Lemma: Type Preservation

If $e : \tau$ and $e \mapsto e'$ then $e' : \tau$.

Static Semantics

Theorem: Type Safety

If $e : \tau$ and $e \mapsto^* e'$ then e' is **not stuck**.

Proof Type Safety

By induction of $e \mapsto^* e'$.

Static Semantics

Theorem: Type Safety

If $e : \tau$ and $e \mapsto^* e'$ then e' is **not stuck**.

Proof Type Safety

By induction of $e \mapsto^* e'$.

$$\text{CASE} \left[\frac{}{e_1 \mapsto e_1'} \right] \quad \begin{array}{l} e = e_1 \\ e' = e_1' \end{array} \quad :$$

By progress $e(= e')$ is not stuck.

Lemma: Progress

If $e : \tau$ then either e `value` or e takes a step.

Static Semantics

Theorem: Type Safety

If $e : \tau$ and $e \mapsto^* e'$ then e' is **not stuck**.

Proof Type Safety

By induction of $e \mapsto^* e'$.

$$\text{CASE } \left[\frac{}{e_1 \mapsto e_1'}\right] \quad \begin{array}{l} e = e_1 \\ e' = e_1' \end{array} :$$

By progress $e(= e')$ is not stuck.

$$\text{CASE } \left[\frac{e_1 \mapsto e_2 \quad e_2 \mapsto^* e_3}{e_1 \mapsto^* e_3}\right] \quad \begin{array}{l} e = e_1 \\ e' = e_3 \end{array} :$$

By preservation $e_2 : \tau$

By induction, e_3 is not stuck.

Lemma: Type Preservation

If $e : \tau$ and $e \mapsto e'$ then $e' : \tau$.

Static Semantics

Theorem: Type Safety

If $e : \tau$ and $e \mapsto^* e'$ then e' is **not stuck**.

Proof Type Safety

By induction of $e \mapsto^* e'$.

$$\text{CASE}\ \left[\frac{}{e_1 \mapsto e_1'}\right] \quad \begin{array}{l} e = e_1 \\ e' = e_1' \end{array} \quad :$$

By progress $e(= e')$ is not stuck.

$$\text{CASE}\ \left[\frac{e_1 \mapsto e_2 \quad e_2 \mapsto^* e_3}{e_1 \mapsto^* e_3}\right] \quad \begin{array}{l} e = e_1 \\ e' = e_3 \end{array} \quad :$$

By preservation $e_2 : \tau$

By induction, e_3 is not stuck.

Type Safety Proof always the same $\rightarrow$ only prove **Preservation** and **Progress**.

Static Semantics

Example

Grammar .....

$e ::= \overline{n} \mid e + e \mid tt \mid ff \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid e < e$

Static Semantics

Example

Grammar

$e ::= \overline{n} \mid e + e \mid tt \mid ff \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid e < e$

Operational Semantics

$$\frac{}{v\ \texttt{value}} \qquad \frac{}{tt\ \texttt{value}} \qquad \frac{}{ff\ \texttt{value}}$$

Static Semantics

Example

Grammar

$e ::= \overline{n} \mid e + e \mid tt \mid ff \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid e < e$

Operational Semantics

$$\frac{}{v\ \texttt{value}} \qquad \frac{}{tt\ \texttt{value}} \qquad \frac{}{ff\ \texttt{value}}$$

$$\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2} \qquad \frac{v_1\ value \quad e_2 \mapsto e_2'}{v_1 + e_2 \mapsto v_1 + e_2'} \qquad \frac{}{\overline{m} + \overline{n} \mapsto \overline{m + n}}$$

Static Semantics

Example

Grammar

$e ::= \overline{n} \mid e + e \mid tt \mid ff \mid \texttt{if } e \texttt{ then } e \texttt{ else } e \mid e < e$

Operational Semantics

$$\frac{}{v \texttt{ value}} \qquad \frac{}{tt \texttt{ value}} \qquad \frac{}{ff \texttt{ value}}$$

$$\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2} \qquad \frac{v_1 \textit{ value} \quad e_2 \mapsto e_2'}{v_1 + e_2 \mapsto v_1 + e_2'} \qquad \frac{}{\overline{m} + \overline{n} \mapsto \overline{m + n}}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \mapsto \texttt{if } e_1' \texttt{ then } e_2 \texttt{ else } e_3}$$

$$\frac{}{\texttt{if } tt \texttt{ then } e_2 \texttt{ else } e_3 \mapsto e_2} \qquad \frac{}{\texttt{if } ff \texttt{ then } e_2 \texttt{ else } e_3 \mapsto e_3}$$

Static Semantics

Example

Grammar

$e ::= \overline{n} \mid e + e \mid tt \mid ff \mid \texttt{if } e \texttt{ then } e \texttt{ else } e \mid e < e$

Operational Semantics

$$\frac{}{v \texttt{ value}} \qquad \frac{}{tt \texttt{ value}} \qquad \frac{}{ff \texttt{ value}}$$

$$\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2} \qquad \frac{v_1 \textit{ value} \quad e_2 \mapsto e_2'}{v_1 + e_2 \mapsto v_1 + e_2'} \qquad \frac{}{\overline{m} + \overline{n} \mapsto \overline{m+n}}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \mapsto \texttt{if } e_1' \texttt{ then } e_2 \texttt{ else } e_3}$$

$$\frac{}{\texttt{if } tt \texttt{ then } e_2 \texttt{ else } e_3 \mapsto e_2} \qquad \frac{}{\texttt{if } ff \texttt{ then } e_2 \texttt{ else } e_3 \mapsto e_3}$$

$$\frac{e_1 \mapsto e_1'}{e_1 < e_2 \mapsto e_1' < e_2} \qquad \frac{v_1 \textit{ value} \quad e_2 \mapsto e_2'}{v_1 < e_2 \mapsto v_1 < e_2'} \qquad \frac{m < n}{\overline{m} < \overline{n} \mapsto tt} \qquad \frac{\neg(m < n)}{\overline{m} < \overline{n} \mapsto ff}$$

Static Semantics

Example

Grammar

$e ::= \overline{n} \mid e + e \mid tt \mid ff \mid \texttt{if } e \texttt{ then } e \texttt{ else } e \mid e < e$

Operational Semantics

$$\frac{}{v \texttt{ value}} \qquad \frac{}{tt \texttt{ value}} \qquad \frac{}{ff \texttt{ value}}$$

$$\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2} \qquad \frac{v_1 \textit{ value} \quad e_2 \mapsto e_2'}{v_1 + e_2 \mapsto v_1 + e_2'} \qquad \frac{}{\overline{m} + \overline{n} \mapsto \overline{m + n}}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \mapsto \texttt{if } e_1' \texttt{ then } e_2 \texttt{ else } e_3}$$

$$\frac{}{\texttt{if } tt \texttt{ then } e_2 \texttt{ else } e_3 \mapsto e_2} \qquad \frac{}{\texttt{if } ff \texttt{ then } e_2 \texttt{ else } e_3 \mapsto e_3}$$

$$\frac{e_1 \mapsto e_1'}{e_1 < e_2 \mapsto e_1' < e_2} \qquad \frac{v_1 \textit{ value} \quad e_2 \mapsto e_2'}{v_1 < e_2 \mapsto v_1 < e_2'} \qquad \frac{m < n}{\overline{m} < \overline{n} \mapsto tt} \qquad \frac{\neg(m < n)}{\overline{m} < \overline{n} \mapsto ff}$$

Static Semantics

$\tau ::= nat \mid bool$

$$\frac{}{\overline{n} : nat} \qquad \frac{}{tt : bool} \qquad \frac{}{ff : bool}$$

Static Semantics

Example

Grammar

$e ::= \overline{n} \mid e + e \mid tt \mid ff \mid \texttt{if } e \texttt{ then } e \texttt{ else } e \mid e < e$

Operational Semantics

$$\frac{}{v \texttt{ value}} \qquad \frac{}{tt \texttt{ value}} \qquad \frac{}{ff \texttt{ value}}$$

$$\frac{e_1 \mapsto e_1'}{e_1 + e_2 \mapsto e_1' + e_2} \qquad \frac{v_1 \textit{ value} \quad e_2 \mapsto e_2'}{v_1 + e_2 \mapsto v_1 + e_2'} \qquad \frac{}{\overline{m} + \overline{n} \mapsto \overline{m + n}}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \mapsto \texttt{if } e_1' \texttt{ then } e_2 \texttt{ else } e_3}$$

$$\frac{}{\texttt{if } tt \texttt{ then } e_2 \texttt{ else } e_3 \mapsto e_2} \qquad \frac{}{\texttt{if } ff \texttt{ then } e_2 \texttt{ else } e_3 \mapsto e_3}$$

$$\frac{e_1 \mapsto e_1'}{e_1 < e_2 \mapsto e_1' < e_2} \qquad \frac{v_1 \textit{ value} \quad e_2 \mapsto e_2'}{v_1 < e_2 \mapsto v_1 < e_2'} \qquad \frac{m < n}{\overline{m} < \overline{n} \mapsto tt} \qquad \frac{\neg(m < n)}{\overline{m} < \overline{n} \mapsto ff}$$

Static Semantics

$\tau ::= nat \mid bool$

$$\frac{}{\overline{n} : nat} \qquad \frac{}{tt : bool} \qquad \frac{}{ff : bool}$$

$$\frac{e_1 : nat \quad e_2 : nat}{e_1 + e_2 : nat} \qquad \frac{e_1 : nat \quad e_2 : nat}{e_1 < e_2 : bool} \qquad \frac{e_1 : bool \quad e_2 : \tau \quad e_3 : \tau}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

Static Semantics

Example Type Checking

$$\frac{}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\frac{e_1 : \mathit{bool} \quad e_2 : \tau \quad e_3 : \tau}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{}{1 < 2}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\dfrac{e_1 : \mathit{bool} \quad e_2 : \tau \quad e_3 : \tau}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\dfrac{}{1} \quad \dfrac{}{2}}{1 < 2}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\dfrac{e_1 : \mathit{nat} \quad e_2 : \mathit{nat}}{e_1 < e_2 : \mathit{bool}}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\overline{1 : \mathit{nat}} \quad \overline{2 : \mathit{nat}}}{1 < 2} \qquad}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\frac{}{\overline{n} : \mathit{nat}}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\overline{1 : \mathit{nat}} \quad \overline{2 : \mathit{nat}}}{1 < 2 : \mathit{bool}}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\dfrac{e_1 : \mathit{nat} \quad e_2 : \mathit{nat}}{e_1 < e_2 : \mathit{bool}}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\dfrac{}{1 : \mathit{nat}} \quad \dfrac{}{2 : \mathit{nat}}}{1 < 2 : \mathit{bool}} \quad \dfrac{}{0}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\dfrac{e_1 : \mathit{bool} \quad e_2 : \tau \quad e_3 : \tau}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\dfrac{}{1 : \mathit{nat}} \quad \dfrac{}{2 : \mathit{nat}}}{1 < 2 : \mathit{bool}} \quad \dfrac{}{0 : \mathit{nat}}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\dfrac{}{\overline{n} : \mathit{nat}}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\dfrac{}{1 : \mathit{nat}} \quad \dfrac{}{2 : \mathit{nat}}}{1 < 2 : \mathit{bool}} \quad \dfrac{}{0 : \mathit{nat}} \quad \dfrac{}{1 + 1}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\dfrac{e_1 : \mathit{bool} \quad e_2 : \tau \quad e_3 : \tau}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\dfrac{}{1 : nat} \quad \dfrac{}{2 : nat}}{1 < 2 : bool} \quad \dfrac{}{0 : nat} \quad \dfrac{\dfrac{}{1} \quad \dfrac{}{1}}{1 + 1}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\dfrac{e_1 : nat \quad e_2 : nat}{e_1 + e_2 : nat}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\dfrac{}{1 : \mathit{nat}} \quad \dfrac{}{2 : \mathit{nat}}}{1 < 2 : \mathit{bool}} \quad \dfrac{}{0 : \mathit{nat}} \quad \dfrac{\dfrac{}{1 : \mathit{nat}} \quad \dfrac{}{1 : \mathit{nat}}}{1 + 1}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\dfrac{}{\overline{n} : \mathit{nat}}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\dfrac{}{1 : nat} \quad \dfrac{}{2 : nat}}{1 < 2 : bool} \quad \dfrac{}{0 : nat} \quad \dfrac{\dfrac{}{1 : nat} \quad \dfrac{}{1 : nat}}{1 + 1 : nat}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1}$$

Current Rule(s)

$$\dfrac{e_1 : nat \quad e_2 : nat}{e_1 + e_2 : nat}$$

Static Semantics

Example Type Checking

$$\dfrac{\dfrac{\dfrac{}{1 : \mathit{nat}} \quad \dfrac{}{2 : \mathit{nat}}}{1 < 2 : \mathit{bool}} \quad \dfrac{}{0 : \mathit{nat}} \quad \dfrac{\dfrac{}{1 : \mathit{nat}} \quad \dfrac{}{1 : \mathit{nat}}}{1 + 1 : \mathit{nat}}}{\texttt{if } 1 < 2 \texttt{ then } 0 \texttt{ else } 1 + 1 : \mathit{nat}}$$

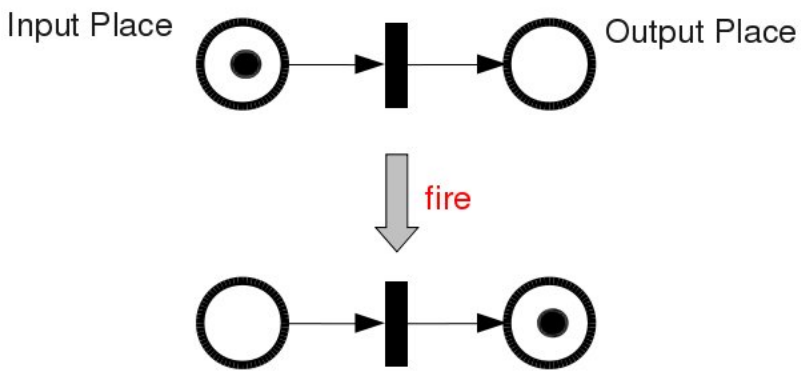
Current Rule(s)

$$\dfrac{e_1 : \mathit{bool} \quad e_2 : \tau \quad e_3 : \tau}{\texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 : \tau}$$

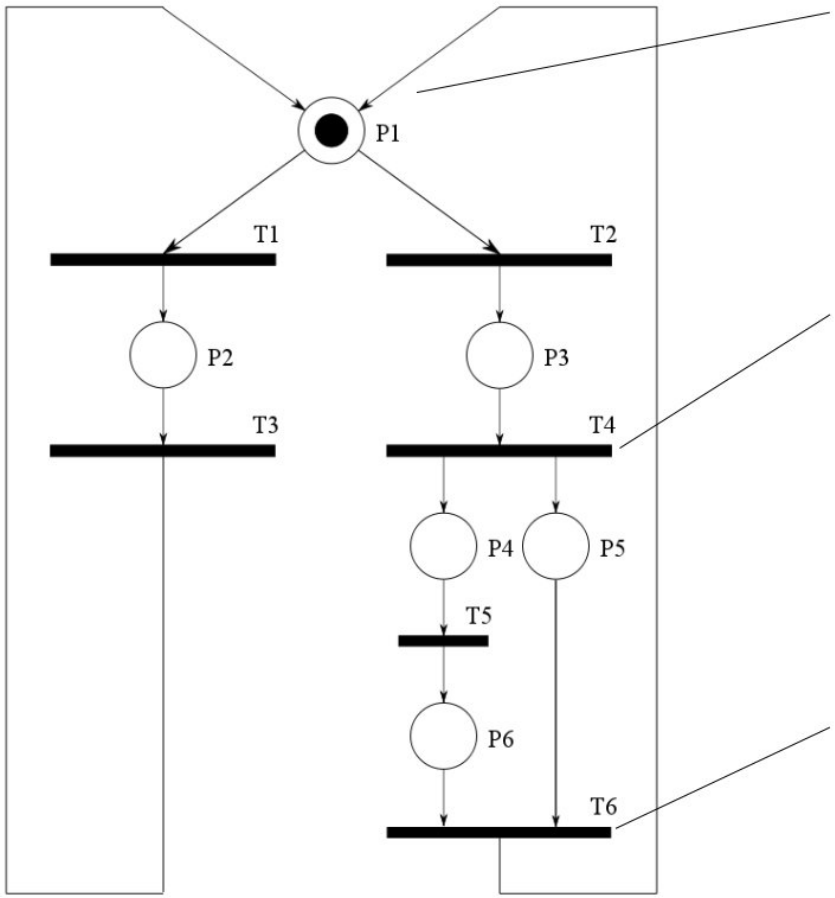
Concurrent Programming Language

Concurrent Programming Language

- A Petri net is a directed bipartite graph
- A Petri net consists of :
  - Tokens
  - Places
  - Transitions



Concurrent Programming Language



Non-Determinism

Trigger multiple actions

Wait for all previous actions to finish

http://de.wikipedia.org/w/index.php?title=Bild:Petrinetz.svg&filetimestamp=20080911233209

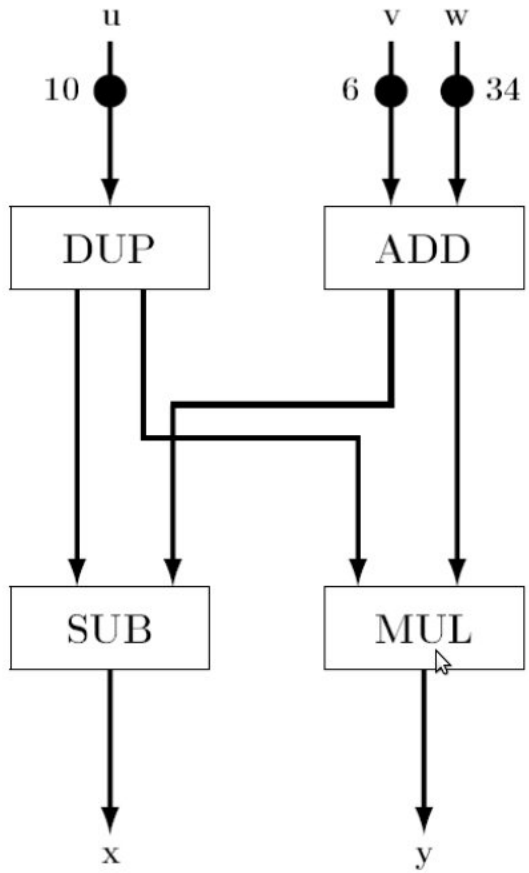
Concurrent Programming Language

Dataflow Architecture

- $\sim$ deterministic Petri Net
- Express data dependencies between statements
  - Program is data dependency graph
- Execution of functions depends on availability of their input data
- At every moment of time all possible execution units are know
  - **MAXIMUM parallelism**

Concurrent Programming Language

```
Input: u, v, w;
   x = u - (v + w);
   y = u * (v + w);
Output: x, y;
```



http://de.wikipedia.org/w/index.php?title=Bild:Datenflussgraph.jpg&filetimestamp=20080425160503

Concurrent Programming Language

Pro

- Expresses all the parallelism in a program
- No shared data
  - Data is consumed and produced
  - No synchronization required

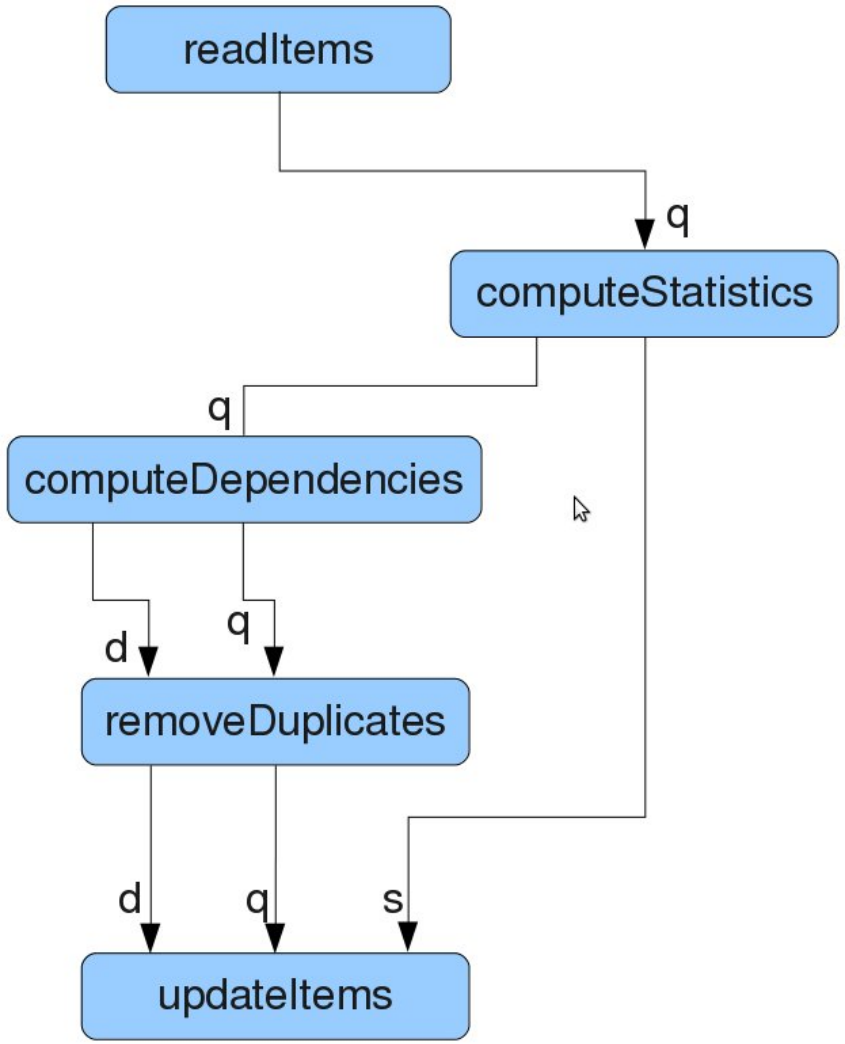
Contra

- Hard to write programs
- Inefficient
  - Always creating and consuming data is expensive

Current Research Directions

```
void readItems(Queue q) { ... }

void updateItems(Queue q,
                 Deps d,
                 Stats s) { ... }

void removeDuplicates(Queue q,
                      Deps d){ ... }

Deps computeDependencies(Queue q) { ... }

Stats computeStatistics(Queue q) { ... }

void main () {
    Queue q = new Queue();

    readItems(q);
    Stats s = computeStatistics(q);
    Deps s  = computeDependencies(q);
    removeDuplicates(q, d);
    updateItems(q, s d);
}
```

Current Research Directions



Current Research Directions

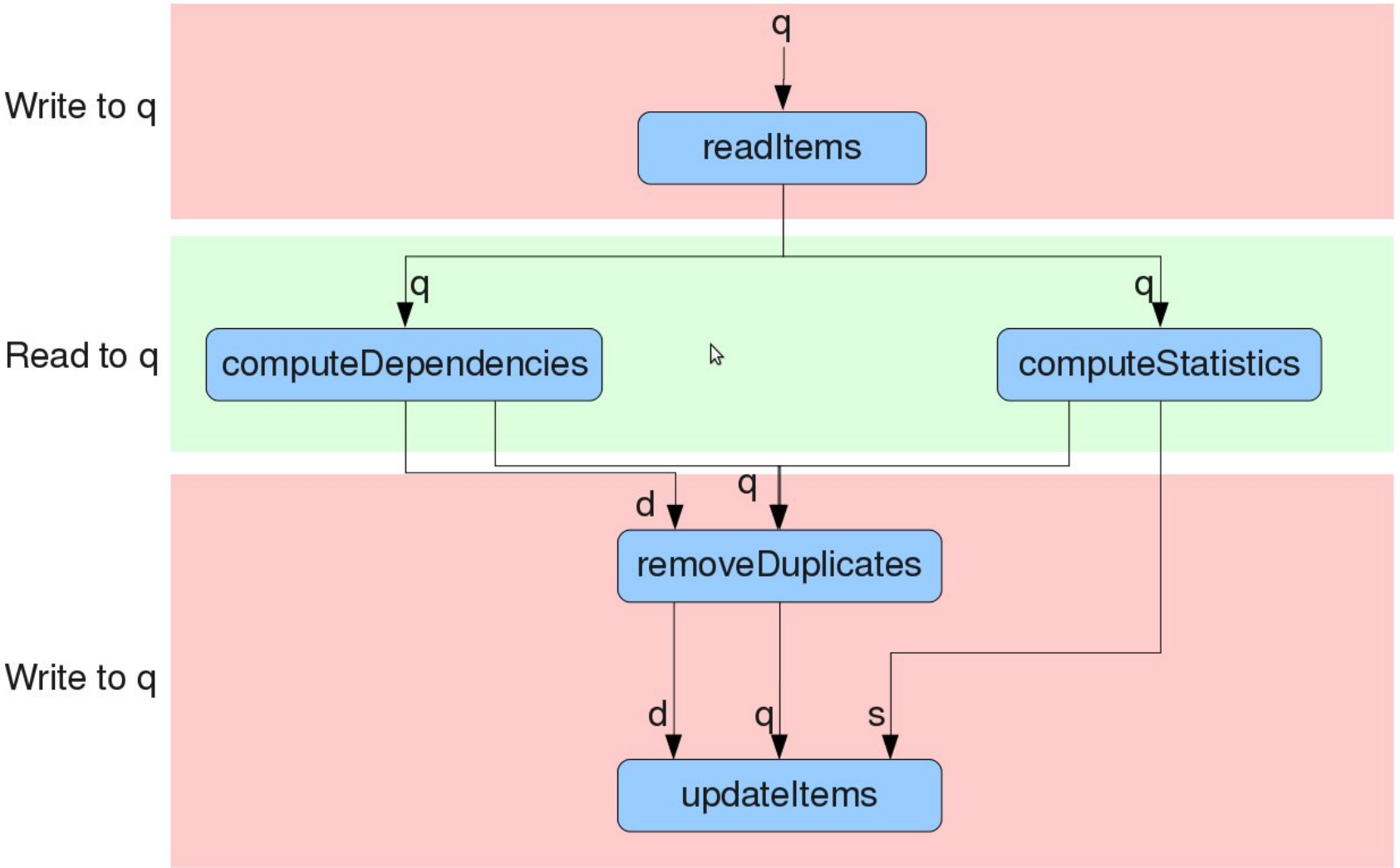
Enhancements

- **Requirements of operations on data**
- Access
  - Read
    - Function only read the specified object
  - Write
    - Function read+write the specified object

Current Research Directions

```
void readItems(@Write Queue q) { ... }

void updateItems(@Write Queue q,
                 @Read Deps d,
                 @Read Stats s) { ... }

void removeDuplicates(@Write Queue q,
                      @Read Deps d){ ... }

Deps computeDependencies(@Read Queue q) { ... }

Stats computeStatistics(@Read Queue q) { ... }

void main () {
    Queue q = new Queue();

    readItems(q);
    Stats s = computeStatistics(q);
    Deps s  = computeDependencies(q);
    removeDuplicates(q, d);
    updateItems(q, s d);
}
```

Current Research Directions

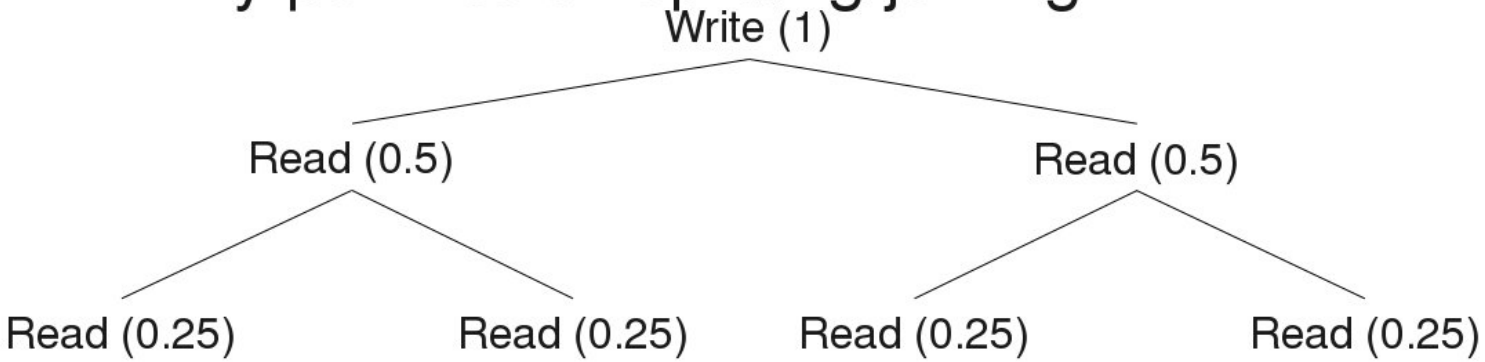


Current Research Directions

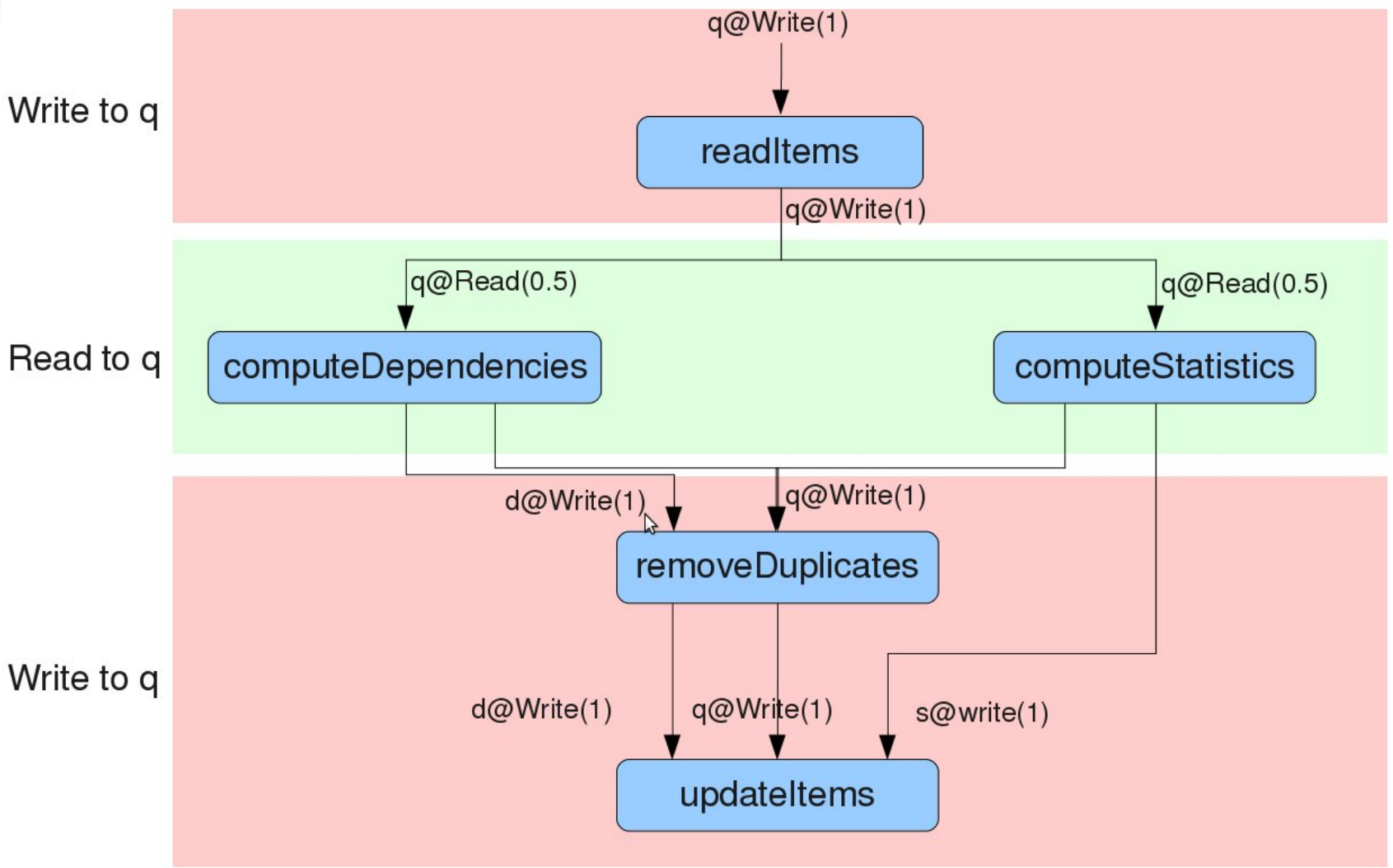
- Ordering
  - Lexical defined

```
...                                  ?    ...
removeDuplicates(q, d);              =    updateItems(q, s d);
updateItems(q, s d);                      removeDuplicates(q, d);
```

  - By permission splitting/joining



Current Research Directions



Core Language

Featherweight Java

Classes	CL	$::= \texttt{class}\ C\ \texttt{extends}\ C'\ \{\ \overline{C}\ \overline{f};\ K\ \overline{M}\ \}$
Constructors	K	$::= C(\overline{C}\ \overline{f})\ \{\ \texttt{super}(\overline{f});\ \texttt{this}.\overline{f} = \overline{f};\ \}$
Methods	M	$::= D\ m(\overline{C}\ \overline{x})\ \{\ \texttt{return}\ e;\ \}$
Expressions	e	$::= x \mid e.f \mid e.m(\overline{e}) \mid \texttt{new}\ C(\overline{e})$

- pure functional object oriented programming language
- true subset of Java

Core Language

Featherweight Java with Permissions

Permissions	p	$::= \mathit{write} \mid \mathit{readonly}$
Classes	CL	$::= \texttt{class}\ C\ \texttt{extends}\ C'\ \{\ \overline{C}\ \overline{f};\ K\ \overline{M}\ \}$
Constructors	K	$::= C(\overline{p}\ \overline{C}\ \overline{f})\ \{\ \texttt{super}(\overline{f});\ \texttt{this}.\overline{f} = \overline{f};\}$
Methods	M	$::= D\ m(\overline{p}\ \overline{C}\ \overline{x})\ p_{this}\ \{\ \texttt{return}\ e;\ \}$
Expressions	e	$::= x \mid e.f \mid e.m(\overline{e}) \mid \texttt{new}\ C(\overline{e}) \mid \underbrace{e_1.f := e_2}_{e_1.f:=e_2 \mapsto e_1}$

- add permissions to parameters of method and method bodies itself
- allow changing state (assignment)

Core Language

Parallel Featherweight Java

Classes	CL	$::= \texttt{class}\ C\ \texttt{extends}\ C'\ \{\ \overline{C}\ \overline{f};\ K\ \overline{M}\ \}$
Constructors	K	$::= C(\overline{C}\ \overline{f})\ \{\ \texttt{super}(\overline{f});\ \texttt{this}.\overline{f} = \overline{f};\ \}$
Methods	M	$::= D\ m(\overline{C}\ \overline{x})\ \{\ S;\ \}$
Sync	S	$::= \texttt{synch}\ \overline{lab}\ \texttt{let}\ lab\ x = a\ \texttt{in}\ S \mid \texttt{return}\ x$
Atom	a	$::= x \mid x.f \mid x.m(\overline{x}) \mid \texttt{new}\ C(\overline{x}) \mid x.f := x$

let-normal form

$$\texttt{synch}\ \underbrace{\overline{labels}}_{dependencies}\ \texttt{let}\ \underbrace{label}_{name}\ \text{x} = \text{exp}\ \texttt{in}\ \ldots$$

Core Language

Example Let-Normal-Form

```
D m(readonly C p) write {
      return new Pair(this.f:=p, new Pair(p, p));
}

D m(C p) {




}
```

Core Language

Example Let-Normal-Form

```
D m(readonly C p) write {
      return new Pair(this.f:=p, new Pair(p, p));
}

D m(C p) {
      synch _ let lab_a a = this.f := p in



}
```

Core Language

Example Let-Normal-Form

D m(readonly C p) write {
    return new Pair(this.f:=p, new Pair(p, p));
}

D m(C p) {
    synch _ let lab_a $a = this.f := p$ in
        synch _ let lab_b $b = new\ Pair(p, p)$ in

}

Core Language

Example Let-Normal-Form

```
D m(readonly C p) write {
        return new Pair(this.f:=p, new Pair(p, p));
}

D m(C p) {
        synch _ let lab_a a = this.f := p in
                synch _ let lab_b b = new Pair(p, p) in
                        synch lab_a, lab_b let lab_c c = new Pair(a, b) in

}
```

Core Language

Example Let-Normal-Form

```
D m(readonly C p) write {
        return new Pair(this.f:=p, new Pair(p, p));
}

D m(C p) {
        synch _ let lab_a a = this.f := p in
                synch _ let lab_b b = new Pair(p, p) in
                        synch lab_a, lab_b let lab_c c = new Pair(a, b) in
                                return c;
}
```

Core Language

class permission context (global)

$$\Omega ::= \emptyset \mid \Omega, C.m.\alpha{:}permission \mid \Omega, C.C.\alpha{:}permission$$

α selects the permission between the permission of the method itself or the specified parameter:

$\alpha = 0$ the permission of the receiver object

$\alpha > 0$ the permission of the N'th parameter

lookup : $\Omega(x) \rightarrow permission$

Core Language

variable permission/label context

$$\Phi ::= \emptyset \mid \{\Phi, (var, permission, \{\overline{labels}\})\}$$

$\Phi(var, permission) \rightarrow \overline{labels}$ lookup

$\Phi(var, unique) \rightarrow label$ latest write operation

$\Phi(var, readonly) \rightarrow \overline{labels}$ all read operations since the last write

$$\Phi' = \left[\frac{(variable, permission, \{\Phi(variable, permission), label_{new}\})}{(variable, permission, \Phi(variable, permission))}\right] \Phi$$

Core Language

S-Context

$$\mathfrak{S} ::= \square \mid \texttt{synch}\ \overline{lab_{deps}}\ \texttt{let}\ lab_x\ x = e_x\ \texttt{in}\ \mathfrak{S} \mid \texttt{return}\ x$$

update : $\mathfrak{S}' = \mathfrak{S}[\texttt{synch}\ lab_{deps}\ \texttt{let}\ lab_x\ x = e_x\ \texttt{in}\ \square]$

$$\Updownarrow$$

$$\mathfrak{S}' = \mathfrak{S}\left[lab_{deps} \Rightarrow lab_x(x, e_x)\right]$$

Core Language

Helper Functions

$typeof(e) = C$ return the type/class a given expression

$fresh\_var() = x$ returns a fresh variable

$var\_to\_label(id) = lab_{id}$ returns a fresh label based on the given variable

$$add\_to\_readset(\Phi, var, label) = \Phi'$$
$$\iff \Phi' = \left[\frac{(var,\{label,\Phi(var,readonly)\})}{(var,readonly,\{\Phi(var,readonly)\})}\right]\Phi$$

$$set\_var(\Phi, var, label) = \Phi'$$
$$\iff \Phi' = \left[\frac{(var,write,\{label\})}{(var,write,\{\Phi(var,write)\})}\right]\left[\frac{(var,readonly,\{label\})}{(var,readonly,\{\Phi(var,readonly)\})}\right]\Phi$$

Core Language

Judgement

$$(\mathfrak{S}_{pre}, \Phi_{in}) \vdash e \rightarrow (\mathfrak{S}_{post}, \Phi_{post}, x)$$

$\mathfrak{S}_{pre}$ S-Context before the evaluation of e

Φ_{pre} Permission-Context before the evaluation of e

e expresssion

$\mathfrak{S}_{post}$ S-Context after the evaluation of e

Φ_{post} S-Context after the evaluation of e

x the variable which contains the result of e

Core Language

R-method

$$\frac{(\emptyset, \square) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y)}{D\ m(C\ p)\{\texttt{return}\ e_1\} \rightarrow D\ m(C\ p)\{\mathfrak{S}_1\ [\ \texttt{return}\ y;\ ]\})}$$

Core Language

R-var

$$\frac{}{(\Phi, \mathfrak{S}_{outer}) \vdash v \longrightarrow (\Phi, \mathfrak{S}_{outer}, v)}$$

Core Language

R-read

$$\frac{}{(\Phi, \mathfrak{S}_{outer}) \vdash e_1.f \rightarrow}$$

Core Language

R-read

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad labs_{deps} = \{\Phi_1(y, unique)\}}{(\Phi, \mathfrak{S}_{outer}) \vdash e_1.f \rightarrow}$$

Core Language

R-read

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad labs_{deps} = \{\Phi_1(y, unique)\} \qquad fresh\_var() = x \qquad var\_to\_label(x) = lab_x}{(\Phi, \mathfrak{S}_{outer}) \vdash e_1.f \rightarrow}$$

Core Language

R-read

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad labs_{deps} = \{\Phi_1(y, unique)\} \qquad fresh\_var() = x \qquad var\_to\_label(x) = lab_x \qquad \Phi_2 = add\_to\_readset(\Phi_1, y, lab_x) \qquad \Phi_3 = set\_var(\Phi_2, x, lab_x)}{(\Phi, \mathfrak{S}_{outer}) \vdash e_1.f \rightarrow}$$

Core Language

R-read

$$\frac{\begin{array}{c}(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad labs_{deps} = \{\Phi_1(y, unique)\} \\ fresh\_var() = x \qquad var\_to\_label(x) = lab_x \\ \Phi_2 = add\_to\_readset(\Phi_1, y, lab_x) \qquad \Phi_3 = set\_var(\Phi_2, x, lab_x)\end{array}}{(\Phi, \mathfrak{S}_{outer}) \vdash e_1.f \rightarrow (\Phi_3, \mathfrak{S}_1 \left[labs_{deps} \Rightarrow lab_x(x, y.f)\right], x)}$$

Core Language

R-assign

$$\frac{}{(\Phi_1, \mathfrak{S}_{outer}) \vdash e_1.f := e_2 \rightarrow}$$

Core Language

R-assign

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z)}{(\Phi_1, \mathfrak{S}_{outer}) \vdash e_1.f := e_2 \rightarrow}$$

Core Language

R-assign

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z) \quad labs_{deps} = \{\Phi_2(y, readonly), \Phi_2(z, unique)\}}{(\Phi_1, \mathfrak{S}_{outer}) \vdash e_1.f := e_2 \rightarrow}$$

Core Language

R-assign

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z) \quad labs_{deps} = \{\Phi_2(y, readonly), \Phi_2(z, unique)\} \quad fresh\_var() = x \qquad var\_to\_label(x) = lab_x}{(\Phi_1, \mathfrak{S}_{outer}) \vdash e_1.f := e_2 \rightarrow}$$

Core Language

R-assign

$$\frac{\begin{array}{c}(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z) \\ labs_{deps} = \{\Phi_2(y, readonly), \Phi_2(z, unique)\} \\ fresh\_var() = x \qquad var\_to\_label(x) = lab_x \\ \Phi_3 = set\_var(\Phi_2, y, lab_x) \qquad \Phi_4 = set\_var(\Phi_3, x, lab_x)\end{array}}{(\Phi_1, \mathfrak{S}_{outer}) \vdash e_1.f := e_2 \rightarrow}$$

Core Language

R-assign

$$\frac{\begin{array}{c}(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z) \\ labs_{deps} = \{\Phi_2(y, readonly), \Phi_2(z, unique)\} \\ fresh\_var() = x \qquad var\_to\_label(x) = lab_x \\ \Phi_3 = set\_var(\Phi_2, y, lab_x) \qquad \Phi_4 = set\_var(\Phi_3, x, lab_x)\end{array}}{(\Phi_1, \mathfrak{S}_{outer}) \vdash e_1.f := e_2 \rightarrow (\Phi_4, \mathfrak{S}_2 \left[labs_{deps} \Rightarrow lab_x(x, y.f := z)\right], x)}$$

Core Language

R-new

$$\frac{}{(\Phi, \mathfrak{S}_{outer}) \vdash \texttt{new}\ C(e_1) \rightarrow}$$

Core Language

R-new

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y)}{(\Phi, \mathfrak{S}_{outer}) \vdash \texttt{new}\ C(e_1) \rightarrow}$$

Core Language

R-new

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \quad \Omega(C.C.1) = p_1 \quad \mathit{fresh\_var}() = x \quad \mathit{var\_to\_label}(x) = \mathit{lab}_x}{(\Phi, \mathfrak{S}_{outer}) \vdash \texttt{new } C(e_1) \rightarrow}$$

Core Language

R-new

$$\frac{\begin{array}{l}(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \\ \Omega(C.C.1) = p_1 \qquad fresh\_var() = x \qquad var\_to\_label(x) = lab_x \\ (p_1 = write)\ ?\ (\Phi_2 = set\_var(\Phi_1, y, lab_x)\ ;\ labs_{deps} = \Phi_1(y, readonly))\end{array}}{(\Phi, \mathfrak{S}_{outer}) \vdash \texttt{new}\ C(e_1) \rightarrow}$$

Core Language

R-new

$$(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y)$$
$$\Omega(C.C.1) = p_1 \qquad \mathit{fresh\_var}() = x \qquad \mathit{var\_to\_label}(x) = \mathit{lab}_x$$
$$(p_1 = \mathit{write})\ ?\ (\Phi_2 = \mathit{set\_var}(\Phi_1, y, \mathit{lab}_x)\ ;\ \mathit{labs}_{deps} = \Phi_1(y, \mathit{readonly}))$$
$$(p_1 = \mathit{readonly})\ ?\ (\Phi_2 = \mathit{add\_to\_readset}(\Phi_1, y, \mathit{lab}_x)\ ;\ \mathit{labs}_{deps} = \Phi_1(y, \mathit{unique}))$$

$$(\Phi, \mathfrak{S}_{outer}) \vdash \texttt{new}\ C(e_1) \rightarrow$$

Core Language

R-new

$$\frac{\begin{array}{l}(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \\ \Omega(C.C.1) = p_1 \qquad fresh\_var() = x \qquad var\_to\_label(x) = lab_x \\ (p_1 = write)\ ?\ (\Phi_2 = set\_var(\Phi_1, y, lab_x)\ ;\ labs_{deps} = \Phi_1(y, readonly)) \\ (p_1 = readonly)\ ?\ (\Phi_2 = add\_to\_readset(\Phi_1, y, lab_x)\ ;\ labs_{deps} = \Phi_1(y, unique)) \\ \Phi_3 = set\_var(\Phi_2, x, lab_x)\end{array}}{(\Phi, \mathfrak{S}_{outer}) \vdash \texttt{new}\ C(e_1) \rightarrow}$$

Core Language

R-new

$$\frac{\begin{array}{l}(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \\ \Omega(C.C.1) = p_1 \qquad fresh\_var() = x \qquad var\_to\_label(x) = lab_x \\ (p_1 = write)\ ?\ (\Phi_2 = set\_var(\Phi_1, y, lab_x)\ ;\ labs_{deps} = \Phi_1(y, readonly)) \\ (p_1 = readonly)\ ?\ (\Phi_2 = add\_to\_readset(\Phi_1, y, lab_x)\ ;\ labs_{deps} = \Phi_1(y, unique)) \\ \Phi_3 = set\_var(\Phi_2, x, lab_x)\end{array}}{(\Phi, \mathfrak{S}_{outer}) \vdash \texttt{new}\ C(e_1) \rightarrow (\Phi_3, \mathfrak{S}_1 \left[lab_{deps} \Rightarrow lab_x(x, \texttt{new}\ C(y))\right], z)}$$

Core Language

R-call

$$\frac{}{(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow}$$

Core Language

R-call

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z)}{(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow}$$

Core Language

R-call

$$\frac{(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z) \qquad typeof(e_1) = C_1 \qquad \Omega(C_1.m.0) = p_0 \qquad \Omega(C_1.m.1) = p_1}{(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow}$$

Core Language

R-call

$$(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z)$$
$$typeof(e_1) = C_1 \qquad \Omega(C_1.m.0) = p_0 \qquad \Omega(C_1.m.1) = p_1$$
$$fresh\_var() = x \qquad var\_to\_label(x) = lab_x$$

$$(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow$$

Core Language

R-call

$$(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z)$$
$$typeof(e_1) = C_1 \qquad \Omega(C_1.m.0) = p_0 \qquad \Omega(C_1.m.1) = p_1$$
$$fresh\_var() = x \qquad var\_to\_label(x) = lab_x$$
$$(p_0 = write)\ ?\ (\Phi_3 = set\_var(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, readonly)))$$

$$(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow$$

Core Language

R-call

$$(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z)$$
$$typeof(e_1) = C_1 \qquad \Omega(C_1.m.0) = p_0 \qquad \Omega(C_1.m.1) = p_1$$
$$fresh\_var() = x \qquad var\_to\_label(x) = lab_x$$
$$(p_0 = write)\ ?\ (\Phi_3 = set\_var(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, readonly)))$$
$$(p_0 = readonly)\ ?\ (\Phi_3 = add\_to\_readset(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, unique)))$$

$$(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow$$

Core Language

R-call

$$(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z)$$
$$typeof(e_1) = C_1 \qquad \Omega(C_1.m.0) = p_0 \qquad \Omega(C_1.m.1) = p_1$$
$$fresh\_var() = x \qquad var\_to\_label(x) = lab_x$$
$$(p_0 = write)\ ?\ (\Phi_3 = set\_var(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, readonly)))$$
$$(p_0 = readonly)\ ?\ (\Phi_3 = add\_to\_readset(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, unique)))$$
$$(p_1 = write)\ ?\ (\Phi_4 = set\_var(\Phi_3, z, lab_x)\ ;\ labs_1 = \Phi_3(z, readonly)))$$

$$(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow$$

Core Language

R-call

$$(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z)$$
$$typeof(e_1) = C_1 \qquad \Omega(C_1.m.0) = p_0 \qquad \Omega(C_1.m.1) = p_1$$
$$fresh\_var() = x \qquad var\_to\_label(x) = lab_x$$
$$(p_0 = write)\ ?\ (\Phi_3 = set\_var(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, readonly)))$$
$$(p_0 = readonly)\ ?\ (\Phi_3 = add\_to\_readset(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, unique)))$$
$$(p_1 = write)\ ?\ (\Phi_4 = set\_var(\Phi_3, z, lab_x)\ ;\ labs_1 = \Phi_3(z, readonly)))$$
$$(p_0 = readonly)\ ?\ (\Phi_4 = add\_to\_readset(\Phi_3, z, lab_x)\ ;\ labs_1 = \Phi_3(z, unique)))$$

$$(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow$$

Core Language

R-call

$$\frac{\begin{array}{l}(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z) \\ typeof(e_1) = C_1 \qquad \Omega(C_1.m.0) = p_0 \qquad \Omega(C_1.m.1) = p_1 \\ fresh\_var() = x \qquad var\_to\_label(x) = lab_x \\ (p_0 = write)\ ?\ (\Phi_3 = set\_var(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, readonly))) \\ (p_0 = readonly)\ ?\ (\Phi_3 = add\_to\_readset(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, unique))) \\ (p_1 = write)\ ?\ (\Phi_4 = set\_var(\Phi_3, z, lab_x)\ ;\ labs_1 = \Phi_3(z, readonly))) \\ (p_0 = readonly)\ ?\ (\Phi_4 = add\_to\_readset(\Phi_3, z, lab_x)\ ;\ labs_1 = \Phi_3(z, unique))) \\ \Phi_5 = set\_var(\Phi_4, x, lab_x) \qquad labs_{deps} = \{labs_0, labs_1\}\end{array}}{(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow}$$

Core Language

R-call

$$\frac{\begin{array}{l}(\Phi, \mathfrak{S}_{outer}) \vdash e_1 \rightarrow (\Phi_1, \mathfrak{S}_1, y) \qquad (\Phi_1, \mathfrak{S}_1) \vdash e_2 \rightarrow (\Phi_2, \mathfrak{S}_2, z) \\ typeof(e_1) = C_1 \qquad \Omega(C_1.m.0) = p_0 \qquad \Omega(C_1.m.1) = p_1 \\ fresh\_var() = x \qquad var\_to\_label(x) = lab_x \\ (p_0 = write)\ ?\ (\Phi_3 = set\_var(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, readonly))) \\ (p_0 = readonly)\ ?\ (\Phi_3 = add\_to\_readset(\Phi_2, y, lab_x)\ ;\ labs_0 = \Phi_2(y, unique))) \\ (p_1 = write)\ ?\ (\Phi_4 = set\_var(\Phi_3, z, lab_x)\ ;\ labs_1 = \Phi_3(z, readonly))) \\ (p_0 = readonly)\ ?\ (\Phi_4 = add\_to\_readset(\Phi_3, z, lab_x)\ ;\ labs_1 = \Phi_3(z, unique))) \\ \Phi_5 = set\_var(\Phi_4, x, lab_x) \qquad labs_{deps} = \{labs_0, labs_1\}\end{array}}{(\Phi, S_{outer}) \vdash e_1.m(e_2) \rightarrow (\Phi_5, \mathfrak{S}_2 \left[labs_{deps} \Rightarrow lab_x(x, y.m(z))\right], x)}$$

Conclusion

Conclusion

Why is this relevant ?

- releave the user from parallel thinking
  - not thinkking about which code paths could execute in parallel with each other
  - thinking about which piece of code accesses which objects in which way
  - annotate code and let the runtime decide
- let the compiler verify that the program is "correct"
  - no race conditions
  - no deadlocks

Conclusion

Current State

- definition of core language
  - use permission to describe data usage
- definition of "intermediate" language
  - explicit represents data dependencies of code paths
- re-writing rules
  - transform core language $\longrightarrow$ intermediate language

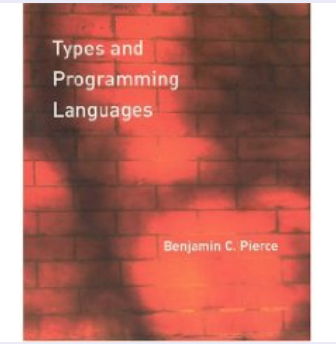
Conclusion

Open Issues

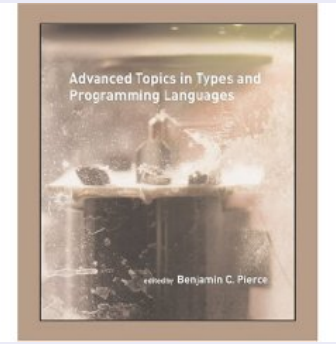
- Handling of shared data accesses
  - for examples like producer-consumer
- proving soundness and correctness of core language / transformation
- need there be another step between the "intermediate" language and the runtime (e.g. byte-code) ?
- develop the runtime itself

Conclusion

More Informations



Types and Programming Languages
Benjamin C. Pierce
The MIT Press



Advanced Topics in Types and Programming Languages
Benjamin C. Pierce
The MIT Press

Questions ?