Unifying Theories of Data Flow

Tony Hoare

Beijing October 2009
Unifying...

- **Memory**
  - shared/private, weakly/strongly consistent

- **Communication**
  - synchronised/buffered, reliable/unreliable

- **Allocation**
  - dynamic/nested, disposed/collection

- **Concurrency**
  - threads/processes, coarse/fine-grain
Labelled graphs

- Trace semantics  (Mazurkiewicz)
- Regular expressions  (Kleene)
- Causal nets  (Petri)
- Event structure configurations  (Winskel)
- Message Sequence Charts  (UML)

INSIGHT! They are all labeled graphs
A labeled Graph

- E: a set of nodes (events)
- A: a set of arrows (denoting data flow)
- L: a set of labels (to be determined)
- source, target: A -> E
- label: A -> L (assigning arrow labels)
- label: E -> L (assigning event labels)
Program Execution

is recorded as a trace of

• all events that have occurred
  – drawn as boxes

• all dependency relations between them
  – drawn as arrows
  – the target could not occur without/before source
A Sequential Resource
Allocation and disposal

begin

next

next

next

end
Regular expression

• is a sequence of labels, e.g.,
  – begin ; next ; next ; next ; end
• standing for a set of paths in a graph
  – which have these labels in sequence, e.g.,

\[
\text{begin} \xrightarrow{\text{next}} \text{next} \xrightarrow{\text{next}} \text{next} \xrightarrow{\text{next}} \text{end}
\]

• next* contains next-paths of any length, e.g.,
  – empty $\cup$ next $\cup$ next ; next, ...
indirect dependency

• $d \xrightarrow{r} e$ means

  $r$ leads from $d$ to $e$.

  There is a path of arrows
  with labels in the regular expression $r$;
  its source is $d$
  and its target is $e$. 
A Sequential Resource

paths(sequential resource) \subseteq \text{begin ; next ; next* ; end}

graph(sequential resource) has exactly one path.
Fork

paths(fork) ⊆ fanout

all arrows of its graph have the same source
Join

all arrows of the graph have the same target
Shared Resource

begin

fanout

fanin

end
paths(publication) \subseteq \text{next} \cup \text{fanout} \; ; \; \text{fanin}
Assignment

next

fanout

fanin

:= 3

:= 7

=3

=3

=3

=3
A variable

\[ \text{paths}(\text{variable}) \subseteq \text{begin} ; \text{next} ; (\text{next} \cup \text{fanout} ; \text{fanin})^* ; \text{end} \]
A variable

paths(variable) ⊆ begin ; next ; (next ∪ fanout ; fanin)* ; end
A variable

paths(variable) \subseteq \text{begin} \cdot \text{next} ; (\text{next} \cup \text{fanout} ; \text{fanin})^* ; \text{end}
paths(variable) ⊆ begin ; next ; (next ∪ fanout ; fanin)* ; end
paths(variable) ⊆ begin ; next ; (next ∪ fanout ; fanin)* ; end
A variable

\text{paths}(\text{variable}) \subseteq \text{begin} ; \text{next} ; (\text{next} \cup \text{fanout} ; \text{fanin})^* ; \text{end}
paths(variable) ⊆ begin ; next ; (next ∪ fanout ; fanin)* ; end
Unassigned fetch

begin

:=

next

fanin

fanout

:=

next

fanin

fanout

:=

next

fanout

fanin

end

fanout

fanin
Communication

![Diagram showing communication process with '!' indicating send and '?' indicating receive.](image-url)
Ordering
Exercise

• Define the traces of a channel.

• hint:
  – you can’t do it accurately with a regular expression. Why not?
Regular Expression

\[
\text{paths(channel)} \subseteq \text{begin} ; \text{next} ; \text{next}^{*} ; (\text{empty} \cup \text{send}) ; \text{next}^{*} ; \text{end}
\]
Channel

begin

next

! next

? next

! next

? next

! next

? next

send

send

send

end
d -(! ; send ; next)-> e
≡ d -(next ; send ; ?)-> e
Single-buffered Channel
Single-buffered Channel

\[
\begin{align*}
&d \rightarrow (\text{send} ; \text{sync}) \rightarrow e \equiv d \rightarrow (! ; \text{next} ; !) \rightarrow e \\
&d \rightarrow (\text{sync} ; \text{send}) \rightarrow e \equiv d \rightarrow (? ; \text{next} ; ?) \rightarrow e
\end{align*}
\]
Synchronised
Exercise

• Define the paths of a single-buffered channel and of a synchronised channel.

• Do the same for the unreliable channels which follow.

• hint: you may use intersection to impose additional constraints on earlier definitions.
Lossy channel

\[
\text{loss} = \exists d ! \sim \exists e ? d \rightarrow^-(\text{send}) e
\]
Stuttering channel

stutter = \exists d! \sim \exists a?, b? \ d \rightarrow (send) \rightarrow a \ & \ d \rightarrow (send) \rightarrow b
Fraudulent channel

![Diagram of a fraudulent channel process with nodes labeled '!', '?', and arrows labeled 'next' and 'send'.]
Overtaking

∃a!b!c?d? a →next*→ b & c →next*→ d
& a →send→ d & b →send→ c
Reliable channel

• reliable $\equiv$ $\neg$loss & $\neg$fraud & $\neg$stutter & $\neg$merge & $\neg$overtaking
Threads

begin → fork → next → next → next → end

begin → next → next → end

begin → next → next → next → join → next
An Atomic Assignment

\[ x := x + y \]

\[ x = 3 \]

\[ y = 4 \]

\[ x := 7 \]
An Atomic Assignment

x := x + y

x = 3

y = 4

x := 7
An Atomic Assignment

\[ x := x + y \]

\[ x = 3 \]

\[ y = 4 \]

\[ x := 7 \]
Events and atomic actions

- Each occurrence of an event in the trace of program execution belongs to the trace of exactly one resource (thread, variable, channel,...)

- Atomic actions are groups of synchronised events, including one from the thread which invoked the action, and one (or more) from every resource used by it.
A variable (fully labelled)
A shared variable

v := 4

v := 3

v := 6

thread

interfering thread
Weakly consistent memory

as implemented in multi-core architecture, complicates shared variable behaviour both in definition and in use.

A common architecture is TSO

Total Store Ordering
Total Store Ordering

:=4 \rightarrow :=3 \rightarrow :=6 \rightarrow \text{global memory}

\text{sync}
local memory

:=4
:=3
:=6

val

next
next
next

sync

global memory

local memory
Local memory access

:= 4

:= 3

:= 6

:= 3

:= 4

:= 6

val

next

next

next

global memory

local memory

local memory
The diagram illustrates a process involving multiple threads and memory locations. Here's a breakdown:

1. **Global Memory**: The initial state shows a global memory location set to 4, followed by a sync operation.
2. **Other Thread**: Another thread is shown with a sync operation leading to a global memory location set to 3.
3. **Local Memory**: The diagram indicates multiple local memory locations, each with a value set to 3, 4, and 6, respectively.
4. **Sync Operations**: Arrows labeled 'sync' indicate synchronization points where operations are performed.
5. **Next**: Arrows labeled 'next' show the progression of values through the thread operations.
6. **Val**: Arrows labeled 'val' indicate the transfer of values between different memory locations.
Memory Barrier

:=4

next

:=3

next

:=6

sync

:= 4

:= 3

:= 6

bar

val

global memory

next

local memory
Summary

- Data flow is a primitive concept adequate to describe the dynamic behaviour of many kinds of computing resource.

- Labelled graphs provide a general framework adequate for a unifying theory of data flow.