Compositional C11 Program Transformation

Mark Batty - Mike Dodds - Alexey Gotsman

Imperial Concurrency Workshop, July 2015
Overview

- Context: relaxed memory, axiomatic semantics, fragment of C11 / C++11.
- Immediate aim: program transformation, eg compiler optimisations.
- Approach: summarise context interactions using a set of histories (denotational-ish).
- Under construction!

The C11 model is (arguably) broken: we omit problem features, most importantly no RLX.
Overview

1. Objective: compositional transformation
2. C11 semantics primer
3. Defining execution histories
4. Cutting down contexts
Fragments and Contexts

\[ C \] - whole program

\[ \llbracket C \rrbracket \] - semantics (defined by a set of executions)

\[
\begin{align*}
C(\_\_) & + P = C(P) \\
\text{Context} & \quad \text{Program fragment} & \quad \text{Whole program}
\end{align*}
\]
Motivation: Compiler Optimisations

\[ P_1 \sim P_2 \quad \text{- replace one fragment with another} \]

\[ P_1: \quad \begin{align*}
    r &= \text{read}(x); \\
    r &= \text{read}(x);
\end{align*} \sim P_2: \quad r &= \text{read}(x); \]

Assume operations are \textit{release-acquire} unless otherwise mentioned.

Is this a sound transformation on C11?
Motivation: Compiler Optimisations

\[ P_1 \leadsto P_2 \quad - \text{ replace one fragment with another} \]

**Soundness:**

\[
\forall C. \forall X_2 \in [C(P_2)]. \exists X_1 \in [C(P_1)]. \text{obsv}(X_2) = \text{obsv}(X_1)
\]

- all contexts
- executions of transformed program
- executions of prior program
- equivalent observed behaviour
Approach

\[ [P]_d \] - summarises of all possible interactions

(...a kind of *denotation*)

Adequacy:

\[ [P_2]_d \subseteq [P_1]_d \implies \forall C. \forall X_2 \in [C(P_2)]. \exists X_1 \in [C(P_1)]. \text{obsv}(X_2) = \text{obsv}(X_1) \]

Thus:

\[ [P_2]_d \subseteq [P_1]_d \implies P_1 \sim P_2 \text{ is sound} \]
Approach

We’d also like finiteness:

\[
P \text{ is loop-free code} \implies [P]_d \text{ is finite}
\]

(…possibly with symbolic values?)

This would support e.g. automated checking
1. Objective: compositional transformation

2. **C11 semantics primer**

3. Defining execution histories

4. Cutting down contexts
C11 concurrency semantics

Executions: multiple partial orders on memory actions.

\[ \langle A, \text{rf, hb, mo} \rangle \in [C] \]

- memory actions
- read-writes
- happens before
- per-location coherence
C11 concurrency semantics (II)

C11 semantics is very non-compositional:

1. Generate whole-program execution candidates.
2. Filter on the basis of validity axioms.

Validity forbids eg:

```
write(x,v) \rightarrow[\text{rf}] write(x,v') \rightarrow[\text{hb}] read(x,v)
```

*forbidden!*
Observable behaviour

thread 1

write(y,1)
write(y,2)
write(x,5)

thread 2

P

read(y,1)

rf?
Observable behaviour

thread 1

write(y,1)
hb
down
write(y,2)
hb
down
write(x,5)
rf
down
read(x)

thread 2

read(y,1)
rf?
down

P

read(x)

hb

down

hb

down

hb

down

hb

down
Observable behaviour

thread 1

write(y,1)

hb

write(y,2)

hb

write(x,5)

thread 2

rf?

P

read(x)

hb

read(y,1)
C11 Challenges

1. Semantics is whole-program and axiomatic.
2. No notion of a global state
3. Unclear when orders can be observed
1. Objective: compositional transformation

2. C11 semantics primer

3. Defining execution histories

4. Cutting down contexts
Inspiration

Idea: treat code transformation as library abstraction
Intuition

$C(P)$

execution
Intuition

$C(-)$

$P$

execution
Intuition

'C(-)'

`P`

Projection of `hb` relation to `interface actions`

execution
Interface actions

```
call

read(x, 7)

write(y, 4)

return
```
Interface actions

History includes context reads / writes to locations accessed in code block

- read(y,9)
- read(z,7)
- write(y,8)
- write(y,4)
- write(z,8)
- write(x,5)
- write(z,8)
- read(x,7)
- read(x,9)
- read(y,9)
In history

call

return
In history

Don’t record internal actions in history
Don’t record internal actions in history
In history

write(x,5)

read(x,5)

return
History

```
call

return
```

`read(x, 5)`
History

Some internal order matters!
History

Some internal order matters!
In history as a deny

Prohibits a hb order

Some internal order matters!
History

Some internal order matters!
Building $[P]_d$

1. Generate executions in $[P]$ for a limited collection of contexts.
2. Extract the history from each execution.
Validating the example

\[ \begin{align*}
P_1 : & \quad \text{read}(x); \\
& \quad \text{read}(x); \\
\leftrightsquigarrow & \quad P_2 : \quad \text{read}(x);
\end{align*} \]

Show \[ [P_2]_d \subseteq [P_1]_d : \]
1. Objective: compositional transformation
2. C11 semantics primer
3. Defining execution histories
4. Cutting down contexts
Which contexts matter?

\[ \forall C. \forall X_2 \in [C(P_2)]. \exists X_1 \in [C(P_1)]. \text{obsv}(X_2) = \text{obsv}(X_1) \]

all contexts?
Which contexts matter?

Drop non-interface context actions
Which contexts matter?

Drop duplicate interface reads
Which contexts matter?

Drop interface reads from interface writes
Which contexts matter?

Drop interface writes with siblings in modification-order
Which contexts matter?

\[
\begin{align*}
\text{cut}(X) & \iff \text{“only interface actions”} \\
& \quad \land \\
& \quad \text{“only rf-distinguished reads”} \\
& \quad \land \\
& \quad \text{“only mo-distinguished writes”}
\end{align*}
\]

\[
P \text{ is loop-free code} \implies \{ X \in [P] \mid \text{cut}(X) \} \text{ is finite}
\]

\[
\implies [P]_d \text{ is finite}
\]
Current status

- Proved adequacy for a fragment of C11. *(release-acquire, NA, working on SC)*
- Validated a collection of optimisations.
- Finiteness theorem *(mostly done).*
- *Full abstraction* *(in progress).*
- *Checking tool* *(planning stages).*
Towards a compositional semantics?

Would like to define parallel composition:

\[
[P_1 \parallel P_2]_d \overset{\text{def}}{=} [P_1]_d \oplus [P_2]_d
\]

Would (maybe) like full abstraction:

\[
\forall C. \forall X_2 \in [C(P_2)]. \exists X_1 \in [C(P_1)]. \text{obsv}(X_2) = \text{obsv}(X_1) \quad \Rightarrow 
[P_2]_d \subseteq [P_1]_d
\]