Promising Event Structure Semantics

Simon Cooksey
`sjc205@kent.ac.uk`

September 28, 2017

Abstract

This document describes the Promising Event Structure Semantics developped with Anton Podkopaev.

1 Definitions

Definition 1.

$$\Sigma' \triangleq (E, \leq, \#) \qquad \text{(Event Structure)}$$

An Event Structure[1] is defined as a set of events extended with a partial order and conflict relation.

Definition 2.

$$\vdash \triangleq \{(x, y) \mid x_{loc} = y_{loc} \wedge x_{val} = y_{val}\} \qquad \text{(Justifies)}$$

$$\Sigma \triangleq (E, \leq, \#, \vdash, \lambda) \qquad \text{(Memory Event Structure)}$$

$$\mathsf{R} = \forall e.e_{\text{READ}} \qquad \text{(Reads)}$$

$$\mathsf{W} = \forall e.e_{\text{WRITE}} \qquad \text{(Writes)}$$

We enhance the Event Structure by adding a labelling function (λ), and a justification relation ($\vdash$).

The λ function gives projections of events in the event set (E), mapping events to their locations and values. For an event x we denote its location as x_{loc} and its value as x_{val}. The λ function also gives us whether an event is a read or a write. This allows us to build sets R and W.

The $\vdash$ relation encodes all possible ways that a particular memory location may get a particular value. These are edges from writes to reads with the same location and same value. They have the flavour of a superset of "reads from" in an axiomatic C/C++ model. We may overload this operator to an infix in the standard way: $x \vdash y \equiv (x, y) \in \vdash$.

Definition 3.

$$[\![\mathcal{P}]\!]_{\text{ES}} \qquad \text{(Program interpretation)}$$

$$
\begin{gathered}
[\![\texttt{r} := \texttt{v}]\!]_{\mathrm{ES}} \ldots \\
[\![\texttt{r} := \texttt{x}]\!]_{\mathrm{ES}} \ldots \\
[\![\texttt{x} := \texttt{v}]\!]_{\mathrm{ES}} \ldots \\
[\![\texttt{x} := \texttt{r}]\!]_{\mathrm{ES}} \ldots \\
[\![\mathcal{P}_1 || \mathcal{P}_2]\!]_{\mathrm{ES}} \ldots
\end{gathered}
$$

Given a program we may build a denotation the program encoded in an event structure. We define $[\![\mathcal{P}]\!]_{\mathrm{ES}}$ inductively over the structure of the program $\mathcal{P}$.

... as before

Definition 4.

$$\mathcal{C} \triangleq \{C \mid \mathrm{down-closed}\ C \wedge \mathrm{conflict-free}\ C\} \tag{Configuration}$$

A Configuration is a member of $\mathcal{C}$. Configurations are conflict free, and downclosed. Meaning that the complete history of a partial program execution is contained within a configuration (down-closed), and it does not represent multiple possible executions overlayed (conflict-free).

2 Promising Event Structures Model

Definition 5.

$$\mathit{maximal}\ C \triangleq \forall e \notin C. \exists g \in C. g\#e \tag{Maximal}$$

A maximal configuration C is a configuration where no element e can be added to C that isn't in conflict with some other element of C.

Definition 6.

$$\mathit{equiv}\ e \triangleq \{f \mid \forall g. (g \vdash e \wedge g \vdash f) \vee (e \vdash g \wedge f \vdash g)\} \tag{Equivalent Event}$$

An equivalent event is an event with the same set of incoming or outgoing justification edges.

Definition 7.

$$\mathrm{certifiable}(e, C) \triangleq \forall Y. (C \subseteq Y) \wedge \mathit{maximal}\ Y \implies (\exists z \in (\mathit{equiv}\ e). z \in Y) \tag{Certifiable}$$

At the core of the Promising Semantics on Event Structures is the certifcation of writes. If a write can be added to some configuration C, it must also be possible to add an equivelantly labelled to all possible maximal extensions of C. More intuitively, for a write to be certified it must occur in all possible executions of the rest of the program.

Definition 8.

$$e \succ C \triangleq e \notin C \land \exists f \in C. f \leq e \land (\nexists g. f \leq g \leq e) \tag{Follows}$$

Definition 9.

$$\mathrm{coh}(\leq, \mathsf{rf}, \mathsf{co}) \triangleq \mathit{acyclic}(\leq \cup \mathsf{rf} \cup \mathsf{co}) \tag{Coherence}$$

This checks all communication edges are acyclic with program order. Cycles in the communication represent coherence violations. co is existentially quantified on the outside of the semantics. [Note: *This may be changed to only check over the edges between events which are already in a given configuration.*]

Definition 10.

$$\textsc{Promise}\ \frac{e \in \mathsf{W} \qquad \mathrm{certifiable}(e, C)}{\langle C, Q, \mathsf{rf}\rangle \xrightarrow{\textsc{Prom}} \langle C, Q \cup \{e\}, \mathsf{rf}\rangle}$$

$$\textsc{Read}\ \frac{\begin{array}{c}\mathrm{coh}(\leq, \mathsf{rf} \cup \{(w, r)\}, \mathsf{co}) \\ r \succ C \qquad r \in \mathsf{R} \qquad \exists w \in Q. w \vdash r\end{array}}{\langle C, Q, \mathsf{rf}\rangle \xrightarrow{\textsc{Prom}} \langle C \cup \{r\}, Q, \mathsf{rf} \cup \{(w, r)\}\rangle}$$

$$\textsc{Fulfill}\ \frac{\begin{array}{c}\mathrm{coh}(\leq, \mathsf{rf}, \mathsf{co}) \\ w \succ C \qquad w \in Q\end{array}}{\langle C, Q, \mathsf{rf}\rangle \xrightarrow{\textsc{Prom}} \langle C \cup \{w\}, Q, \mathsf{rf}\rangle}$$

I think that the $r \in \mathsf{R}$ premise of the READ *rule is redundant with the type of justifies*

The Promising Event Structures model is defined as a transition system which builds valid executions of a program from it's event structure. Transitions have the flavour of promising certifiable writes and putting them in the P set; then either adding a read to the configuration C, or a write from the promise set.??

We write $\xrightarrow{\textsc{Prom}}_{\mathsf{co}}$ to represent transtions with a particular choice of the co relation.

Definition 11.

$$[\![\mathcal{P}]\!]_{\textsc{Prom}} \triangleq \{C \mid \exists \mathsf{co}. \exists P. \langle \emptyset, \emptyset\rangle \xrightarrow{\textsc{Prom}}{}^{*}_{\mathsf{co}} \langle C, P\rangle \land \mathit{maximal}\ C\}$$

(Promising Event Structure Semantics)

The set of accepted behaviours of a program is defined as the set of maximal configurations which are reachable through some number of $\xrightarrow{\textsc{Prom}}$ edges with an existentially quantified co.

References

[1] A. Jeffrey and J. Riely. On thin air reads towards an event structures model of relaxed memory. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 759–767, New York, NY, USA, 2016. ACM.