The Most Awesomest Paper Ever

Automatic and Provably Correct Enforcement of Weak Consistency Guarantees

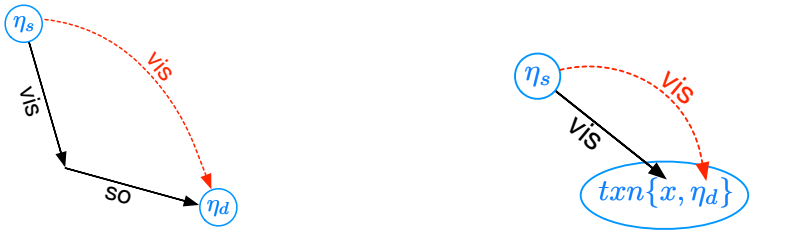
KIA RAHMANI, Purdue University
GOWTHAM KAKI, Purdue University
SURESH JAGANNATHAN, Purdue University

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

ACM Reference format:
Kia Rahmani, Gowtham Kaki, and Suresh Jagannathan. 2017. The Most Awesomest Paper Ever. *PACM Progr. Lang.* 1, 1, Article 1 (January 2017), 4 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

2017. 2475-1421/2017/1-ART1 $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn



(a) Monotonic Read (MR) (b) Monotonic Atomic View (MAV)

Fig. 1. Representation of Contracts

1 SYSTEM MODEL

1.1 Specification Language

. Users in our system are offered with a contract language to specify their application-level consistency requirements. Developers must define a contract for each operation in the application, since the overall correctness of the program requires different levels of consistency for each operation. The constructing blocks of contracts are the relations over the set of effects generated by operations. Relations $vis(a, b)$ and $so(a, b)$ are defined to relate effects a and b, respectively if b was visible to the operation that generated a and if they are the result of two operations submitted by the same session, respecting their submission time. We also introduce *sameobj* relation to maintain per-object instead of system-wide dependencies.

Contracts are basically logical formulae that specify *when* (the pre-condition), *what* effects should be visible to an operation. For example, the Monotonic Read (MR) session guarantee, requires all the effects that are visible to an operation in a session, to be also visible to the later operations in that session. Figure 1a shows how this can be succinctly defined as a contract using the relations over the effects. This contract simply states that all effects η_s that satisfy the relation $(vis^{-1}(so^{-1}))$ to an effect η_d, must also be visible to η_d.

$$\eta_s, \eta_d, x \in \text{EffVar} \qquad Head ::= \eta_h | x \xrightarrow{txso} \eta_h | \eta_h \xrightarrow{txso} x$$

$$R \in \text{Relation} ::= vis | so | R \cup R \qquad Tail ::= \eta_t | x \xrightarrow{txso} \eta_t$$

$$C \in \text{Clause} ::= [R] | R^* \qquad \pi \in \text{Prop} ::= (H \xrightarrow{C} T) | \pi \vee \pi$$

$$\psi \in \text{Contract} ::= \pi \Rightarrow vis(\eta_s, \eta_d)$$

Fig. 2. The Contract Language

All contracts that can be written in our language, are basically consisted of three parts:

(1) Head and tail of the contract - the blue circles in Figure 1a -
(2) The dashed vis edge, that connects head of the contract to its tail, and can be interpreted as the visibility relation *that must be enforced by the system.*
(3) The linear array of relations connecting head and tail, which represents the relation between the tail and all the effects that must be visible to it.

Using *vis* and *so* relations, users can write a broad set of contracts including all session guarantees from (Terry et. al). In order to allow them to specify even broader set of possible scenarios, we add the notion of transactions to the programs and introduce a new relation *txso*, that relates effects which are related by *so*, and are also in the

same transaction. For example, figure 1b represents the Monotonic Atomic View (MAV) guarantee, with a tail consisting of $x \xrightarrow{txso} \eta_t$. Note that when the heads or tails are consisted of x and η_h (or η_t), we implicitly agree to read the dashed vis edge as if it is connected to η_h (or η_t), and the linear array of relations as if it is connected to the variable x.

We structurally limit the use of the $txso$ relation only at the head or the tail of contracts. Figure 2 presents the formal definition of our contract language, that represents this extended notion of the head and tail. Variables Head and Tail are defined to be either a single effect, or two effects related by $txso$ relation. Note that the dashed vis edge is implicitly defined by specifying its two ends, η_h and η_t.

Read My Write (RMW):	$(\eta_h \xrightarrow{so} \eta_h) \Rightarrow vis(\eta_h, \eta_t)$
Monotonic Reads (MR):	$(\eta_h \xrightarrow{vis;so} \eta_h) \Rightarrow vis(\eta_h, \eta_t)$
Monotonic Writes (MW):	$(\eta_h \xrightarrow{so;vis} \eta_h) \Rightarrow vis(\eta_h, \eta_t)$
Writes Follow Reads (WFR):	$(\eta_h \xrightarrow{vis;vis} \eta_h) \vee (\eta_h \xrightarrow{vis;so;vis} \eta_h) \Rightarrow vis(\eta_h, \eta_t)$
Read Committed (RC):	$((\eta_h \xrightarrow{txso} x) \xrightarrow{vis} \eta_h) \vee ((x \xrightarrow{txso} \eta_h) \xrightarrow{vis} \eta_h) \Rightarrow vis(\eta_h, \eta_t)$
Monotonic Atomic View (MAV):	$(\eta_h \xrightarrow{vis} (x \xrightarrow{txso} \eta_h)) \Rightarrow vis(\eta_h, \eta_t)$

Fig. 3. Well-known guarantees specified in our contract language

2 OPERATIONAL SEMANTICS

2.1 Semantics of the Shim Layer

[Auxiliary Reduction]

$$\frac{\begin{array}{cc} E'.vis = E.vis \cup S \times \{\eta\} & \eta = \mathbb{F}_{op}(S) \\ E'.so = E.vis \cup (E.so^{-1}(\eta') \cup \{\eta'\}) \times \{\eta\} & \eta' =< \eta_{SessID}, \eta_{SeqNo} - 1 > \end{array}}{(E, S) \overset{op}{\hookrightarrow} (E', \eta)}$$

[Cache Refresh]

$$\frac{\begin{array}{cc} Cache' = Cache[op \mapsto Cache(op) \cup \{\eta\}] & op = \eta_{op} \\ \mathbb{S}_{\mathrm{DEPS}}(\eta_\psi, \eta) \subseteq Cache(op) & \eta \notin Cache(op) \\ \mathbb{T}_{\mathrm{DEPS}}(\eta_\psi, \eta) \subseteq Cache & \eta \in E.A \end{array}}{(E, Cache) \rightarrow (E, Cache')}$$

[Non-blocking Execution]

$$\frac{\begin{array}{c} \mathbb{TYPE}(op) = non\_blocking \\ (E, Cache(op)) \overset{op}{\hookrightarrow} (E', \eta) \end{array}}{(E, Cache) \overset{op}{\longrightarrow} (E', Cache)}$$

[Blocking Execution]

$$\frac{\begin{array}{cc} & \mathbb{WAIT}_{op}(\eta) \subseteq Cache(op) \\ \mathbb{TYPE}(op) = blocking & (E, Cache(op)) \overset{op}{\hookrightarrow} (E', \eta) \end{array}}{(E, Cache) \overset{op}{\longrightarrow} (E', Cache)}$$