Addendum to "Proof Rules for Flush Channels"

Scott Stoller

Abstract— **The logic presented in [1] for processes that communicate using flush channels is inadequate for reasoning about processes that send multiple identical messages along a channel. A modification to the logic and proof system that remedies this deficiency is described herein.**

Keywords— **asynchronous communication, distributed systems, program verification**

The logic presented in [1] for systems of processes that communicate using flush channels is incomplete. The axiomatization given is adequate for reasoning about programs that never send two or more identical messages (*i.e.*, messages with the same contents and the same F-channel message type) along a channel, but inadequate for reasoning about programs that do not satisfy this condition. The source of the weakness is that the mathematical model of flush channels on which the logic is based does not contain enough information to determine exactly the possible message delivery orderings, as defined by the operational semantics of flush channels. (The model of flush channels in [1] is defined implicitly by the choice of auxiliary variables and the updates to the auxiliary variables used to represent communication events.) If several identical messages are sent along a channel, the multiplicity is not taken into account in the ordering $\prec\!\!+_F$. Consequently, the assumption in the satisfaction axiom for receive statements is too weak: it corresponds to an operational semantics that allows a message m to be delivered after delivery of at least one copy of each message that should be delivered before m, even if multiple copies of some of those messages should be delivered before m.

For example, consider the following program:

cobegin
P_1 : **send** $(\mathbf{2F}, 0)$ **on** F;
send $(\mathbf{2F}, 0)$ **on** F;
send $(\mathbf{2F}, 1)$ **on** F
$\|$
P_2 : **receive** $(t1, x1)$ **from** F;
receive $(t2, x2)$ **from** F;
receive $(t3, x3)$ **from** F
coend

Let $m_i =_{def} \langle \mathbf{2F}, i \rangle$. According to the operational semantics of flush channels, both copies of m_0 sent by P_1 must be delivered before m_1, so $x2 = 0 \wedge x3 = 1$ holds when the program terminates. Consider showing satisfaction (using the rules of [1]) for the second **receive**. One possible situation is that all three **send**s have been executed; in this case, $\sigma_F = \{m_0, m_0, m_1\}$, $\rho_F = \{m_0\}$, and $\prec\!\!+_F = \{(m_0, m_1)\}$. The antecedent of the satisfaction formula contains the predicate

$$(\forall m : m \in \sigma_F \wedge m \prec\!\!+_F MTEXT \Rightarrow m \in \rho_F),$$

which is supposed to characterize which messages $MTEXT$ in $\sigma_F \ominus \rho_F$ can be delivered next. This predicate holds for $MTEXT = m_0$ and $MTEXT = m_1$, so the postcondition of this statement must hold if either of these messages is received by this statement. Thus, the strongest fact about $x2$ that can be proved in the postcondition of the second **receive** is $(x2 = 0 \vee x2 = 1)$. Consequently, the strongest fact about $x2$ and $x3$ that can be proved in the postcondition of this program, using the logic presented in [1], is $(x2 = 0 \wedge x3 = 1) \vee (x2 = 1 \wedge x3 = 0)$.

There are various ways to remedy this deficiency, though at the cost of complicating the model and proof system. One approach is to keep track of multiplicities in $\prec$ and $\prec\!\!+$ by making them multisets. The interpretation is: if the multiplicity of (m, m') in $\prec\!\!+$ is i, then i copies of m must be received before m'. This approach fails because $\prec$ does not contain enough information to compute $\prec\!\!+$ (*i.e.*, there is no definition of the "transitive closure" of $\prec$ that yields the correct multiplicities in $\prec\!\!+$). Similar difficulties arise if one tries to modify the proof rules to compute $\prec\!\!+$ directly.

Another approach is to tag messages in a way that ensures that when a tagged message is added to σ_F, it is not already contained in $\sigma_F \ominus \rho_F$. The tagging occurs only in the mathematical model of the network, not in the user's program or in the implementation of flush channels. This approach permits "minimal" tagging schemes but complicates other aspects of the model; for example, it requires that pairs be deleted from $\prec_F$ when a receive occurs.

A simpler approach is to tag each message with an unique identifier. One natural choice for the unique identifier is the multiset of messages previously sent along the channel; another, which I use below, is the size of this multiset. In either case, every element added to σ_F is unique, so σ_F and ρ_F can be regarded as sets rather than multisets.

The corresponding modifications to the proof rules follow. The changes to the **send** axioms are captured neatly by reinterpreting the "macro" m, which is used in these axioms as an abbreviation for $\langle type, data \rangle$, as an abbreviation for $\langle \langle type, data \rangle, |\sigma_F| \rangle$. In the satisfaction and noninterference rules, the substitutions of $MTEXT$ for $\langle mtype, mdata \rangle$

Scott Stoller (stoller@cs.cornell.edu) is with the Department of Computer Science, Cornell University, Ithaca, NY 14853. He is currently supported by an IBM Graduate Fellowship.

should be changed to substitutions of $\pi_1(MTEXT)$ for $\langle mtype, mdata \rangle$, where π_1 projects out the first component of a pair.

In the Appendix of [1], Camp *et al.* argue that their mathematical model accurately describes the behavior of flush channels. The argument fails to uncover the discrepancy noted above because it contains the unstated assumption that all elements added to σ_F are distinct. This assumption is particularly evident in the proof of Lemma 1. However, this assumption does *not* hold for all programs in their model. Since this assumption holds in the model proposed above, their argument shows that this model accurately describes the behavior of flush channels.

References

[1] T. Camp, P. Kearns, and M. Ahuja, "Proof rules for flush channels," *IEEE Trans. on Software Eng.*, vol. 19, no. 4, pp. 366–378, 1993.