Rely-Guarantee Reasoning
for Context-Aware Software

Doina Bucur and Marta Kwiatkowska
Computing Laboratory, Oxford University, UK
{doina.bucur, marta.kwiatkowska}@comlab.ox.ac.uk

Introduction Context-aware applications are typically designed with concurrent context handlers. Verification techniques guarantee their behaviour against a specification; to date, contributions include either the verification of models rather than real software, or validation. Of the latter, [3] generates test suites for context-aware Java programs by identifying those *context-aware program points* where a context update influences application behaviour.

No prior specialized techniques exist for the automatic *verification* of *context-aware software*. As an initial step, we add language support to a generic verification tool for C software, so that it verifies assertions in concurrent, context-aware code written for TinyOS sensor nodes [1]. We propose a logical formalism based on Local Rely-Guarantee reasoning (LRG) [2], for the sensor-network C language from [1]. The formalism extends our sensor verification technique so that (i) it is thread-modular, in rely-guarantee fashion, and (ii) it verifies more complex assertions which are specifically context-aware.

The Logic Consider a language syntax extending multithreaded C with function calls for *sensing* and *actuating*:

$$(\textsc{Stmt}) \quad S ::= \mathit{sense}(\&x); \;\; | \;\; \mathit{actuate}(x); \; | \; \ldots$$

with Sen the set of *sensing* methods, $\mathit{sense} \in \mathit{Sen}$, and Act the set of *actuating* methods, $\mathit{actuate} \in \mathit{Act}$. We inherit the style of assertions from LRG, and extend them with assertions over contextual facts:

$$\begin{array}{rrcl} (\textsc{Pre-/postcondition}) & p, q, r & ::= & \mathcal{P}(\mathit{Sen}_n) : \mathrm{Trig}(\mathit{actuate}) \mid \ldots \\ (\textsc{Rely/guarantee}) & R, G & ::= & [p] \mid p \ltimes q \mid \ldots \end{array}$$

In the above, a pre-/postcondition $\mathcal{P}(\mathit{Sen}_n) : \mathrm{Trig}(\mathit{actuate})$ states that the set of sensor readings Sen_n (i.e. a sensing call paired with a constant timestamp n for uniqueness) *triggered* the program's last call to *actuate*; i.e. the call succeeds a context-aware program point. A rely/guarantee is an ordered pair of assertions specifying the effect a code segment S has: $p \ltimes q$ states that p holds before and q holds after S, and $[p]$ is $p \ltimes p$.

Standard rely-guarantee reasoning [2] allows for thread-modular verification with one thread guaranteeing conditions which are relied upon by other threads. We extend this scheme for the typing environments $\Gamma : \mathit{Var} \to \mathcal{P}(\mathit{Sen}_n)$ calculated by a side-effect and escape analysis; if $\mathit{sense}_1 \in \Gamma(v)$, then the value sensed with *sense*$_1$ has "escaped" into v. A thread is then individually verified given the other threads' *global* side-effects Γ_G, as by the rule:

$$\text{(\textsc{Parallel})}\quad \frac{\begin{array}{c}(\Gamma_R \vee \Gamma_{G_2}) * \Gamma_{G_1}; R \vee G_2, G_1 \vdash \{p_1\}\, S_1\, \{q_1\}\\ (\Gamma_R \vee \Gamma_{G_1}) * \Gamma_{G_2}; R \vee G_1, G_2 \vdash \{p_2\}\, S_2\, \{q_2\}\end{array}}{\Gamma_R * (\Gamma_{G_1} \vee \Gamma_{G_2}); R, G_1 \vee G_2 \vdash \{p_1 * p_2\}\, S_1 \parallel S_2\, \{q_1 * q_2\}}$$

As an example, consider the second thread S_2 of an application of the form $S_m; (S_1 \parallel S_2 \parallel S_3)$, with S_2 displaying a video only if the battery power levels are above a minimum threshold `MIN`, and S_1 and S_3 sensing contradictory power values. Given the respective rely/guarantee typing environments and conditions, we show the intermediate pre- and postconditions for the verification of S_2:

$$\Gamma_m \ ::= \begin{cases} ctx1 & \to sense\_power_1 \\ ctx2 & \to display_1 \\ ctx3 & \to sense\_power_2 \end{cases} \qquad \begin{array}{ll} \Gamma_1 & ::= \{power \to sense\_power_1\} \\ \Gamma_2 & ::= \{arg_2 \to display_1\} \\ \Gamma_2' & ::= \Gamma_2 \wedge \{out \to display_1\} \\ \Gamma_3 & ::= \{power \to sense\_power_2\} \end{array}$$

$$\begin{array}{ll} p_1 & ::= \exists X.(ctx_1 = X \wedge X > \texttt{MIN}) \quad p_2 ::= (ctx_2 = Y) \\ p_3 & ::= \exists Z.(ctx_3 = Z \wedge Z \leq \texttt{MIN}) \quad p_4 ::= (power = M) \quad p_c ::= p_1 \wedge p_2 \wedge p_3 \\ q & ::= sense\_power_1 : \mathrm{Trig}(display) \end{array}$$

$$\begin{array}{ll} G_1 & ::= [p_c] \quad G_2 ::= p_c \ltimes (p_c \wedge q) \quad G_3 ::= [p_c] \\ R_1 & ::= G_2 \vee G_3 \quad R_2 ::= G_1 \vee G_3 \quad R_3 ::= G_1 \vee G_2 \end{array}$$

$$\begin{array}{ll} & \boxed{p_c \wedge p_4} \\ \Gamma_m \vee \Gamma_1 \vee \Gamma_3 * \Gamma_2; R_2, G_2 \vdash & \langle arg_2 := ctx2; \rangle \\ & \boxed{p_{21} ::= p_c \wedge (arg_2 = Y)} \\ \Gamma_m \vee \Gamma_1 \vee \Gamma_3 * \Gamma_2; R_2, G_2 \vdash & \langle \texttt{IF !}(power > \texttt{MIN})\ \texttt{EXIT} \rangle \\ & \boxed{p_{22} ::= p_{21} \wedge (M > \texttt{MIN})} \\ \Gamma_m \vee \Gamma_1 \vee \Gamma_3 * \Gamma_2'; R_2, G_2 \vdash & \langle out := DB\ \texttt{sel}\ arg_2; \rangle \\ & \boxed{p_{23} ::= p_{22} \wedge (out = DB\ \texttt{sel}\ Y)} \\ \Gamma_m \vee \Gamma_1 \vee \Gamma_3 * \Gamma_2'; R_2, G_2 \vdash & \langle \texttt{IF}\ (power > \texttt{MIN})\ \ display(out); \rangle \\ & \boxed{p_{23} \wedge q} \end{array}$$

As future work, we plan to implement the logic for the automatic verification of context-aware sensor network applications.

Acknowledgments The authors are supported by the project *UbiVal: Fundamental Approaches to Validation of Ubiquitous Computing Applications and Infrastructures*, EPSRC grant EP/D076625/2.

References

[1] Doina Bucur and Marta Kwiatkowska. Bug-Free Sensors: The Automatic Verification of Context-Aware TinyOS Applications. In *Proceedings of the Third European Conference on Ambient Intelligence (AmI)*. Springer, 2009.

[2] Xinyu Feng. Local rely-guarantee reasoning. In *ACM Symposium on Principles of Programming Languages*, pages 315–327. ACM, 2009.

[3] Zhimin Wang, Sebastian Elbaum, and David S. Rosenblum. Automated Generation of Context-Aware Tests. *International Conference on Software Engineering*, pages 406–415, 2007.