Electronic Appendix to Advances in Property-Based Testing for αProlog

James Cheney[1], Alberto Momigliano[2], Matteo Pessina[2]

[1] University of Edinburgh jcheney@inf.ed.ac.uk
[2] Università degli Studi di Milano momigliano@di.unimi.it, matteo.pessina3@studenti.unimi.it

Abstract. We list here some formal definitions and further experiments omitted from the paper for reasons of space

1 Some formal definitions

The effect of a permutation π on a name:

$$\begin{aligned} \mathsf{id}(\mathsf{a}) &= \mathsf{a} \\ ((\mathsf{a}\ \mathsf{b}) \circ \pi)(\mathsf{c}) &= \begin{cases} \mathsf{b} & \pi(\mathsf{c}) = \mathsf{a} \\ \mathsf{a} & \pi(\mathsf{c}) = \mathsf{b} \\ \pi(\mathsf{c}) & \pi(\mathsf{c}) \notin \{\mathsf{a}, \mathsf{b}\} \end{cases} \end{aligned}$$

The swapping operation *ground* terms:

$$\begin{aligned} \pi \cdot \langle\rangle &= \langle\rangle & \pi \cdot f(t) &= f(\pi \cdot t) \\ \pi \cdot \langle t, u\rangle &= \langle \pi \cdot t, \pi \cdot u\rangle & \pi \cdot \mathsf{a} &= \pi(\mathsf{a}) \\ \pi \cdot \langle \mathsf{a}\rangle t &= \langle \pi \cdot \mathsf{a}\rangle \pi \cdot t \end{aligned}$$

Constraint satisfaction:

$$\begin{aligned} \theta &\models \top \\ \theta \models t \approx u &\iff \theta(t) \approx \theta(u) \\ \theta \models t \# u &\iff \theta(t) \# \theta(u) \\ \theta \models C \wedge C' &\iff \theta \models C \text{ and } \theta \models C' \\ \theta \models \exists X{:}\tau.\ C &\iff \text{for some } t : \tau,\ \theta[X := t]^3 \models C \\ \theta \models \reflectbox{N}\mathsf{a}{:}\nu.\ C &\iff \text{for some } \mathsf{b} \# (\theta, C),\ \theta \models C[\mathsf{b}/\mathsf{a}] \end{aligned}$$

A context Γ is a sequence of bindings between variables (or names) and types.

$$\Gamma ::= \cdot \mid \Gamma, X{:}\tau \mid \Gamma \# \mathsf{a}{:}\nu$$

where we write name-bindings as $\Gamma\#\mathsf{a}{:}\nu$, to remind us that a must be fresh for other names and variables in Γ.

Term complementation:

$$
\begin{aligned}
not[\![\tau]\!] &: \tau \to \tau\ set \\
not[\![\tau]\!](t) &= \emptyset \qquad\qquad \text{when } \tau \in \{\mathbf{1}, \nu, \langle\nu\rangle\tau\} \text{ or } t \text{ is a variable} \\
not[\![\tau_1 \times \tau_2]\!](t_1, t_2) &= \{(s_1, \_) \mid s_1 \in not[\![\tau_1]\!](t_1)\} \cup \{(\_, s_2) \mid s_s \in not[\![\tau_2]\!](t_2)\} \\
not[\![\delta]\!](f(t)) &= \{g(\_) \mid g \in \Sigma, g : \sigma \to \delta, f \ \neq g\} \cup \{f(s) \mid s \in not[\![\tau]\!](t)\}
\end{aligned}
$$

The correctness of the algorithm for term complementation can be stated in the following constraint-conscious way, as required by the proof of the main soudness theorem:

Lemma 1 (Term Exclusivity).
Let $\mathcal{K}$ be consistent, $s \in not[\![\tau]\!](t)$, $FV(u) \subseteq \Gamma$ and $FV(s,t) \subseteq \boldsymbol{X}$. It is not the case that both $\Gamma;\mathcal{K} \models \exists \boldsymbol{X}{:}\boldsymbol{\tau}.\ u \approx t$ and $\Gamma;\mathcal{K} \models \exists \boldsymbol{X}{:}\boldsymbol{\tau}.\ u \approx s$.

Inequality and non-freshness:

$$
\begin{aligned}
neq[\![\tau]\!] &: \tau \times \tau \to o \\
neq[\![\mathbf{1}]\!](t,u) &= \bot \\
neq[\![\tau_1 \times \tau_2]\!](t,u) &= neq[\![\tau_1]\!](\pi_1(t), \pi_1(u)) \vee neq[\![\tau_2]\!](\pi_2(t), \pi_2(u)) \\
neq[\![\delta]\!](t,u) &= neq_\delta(t,u) \\
neq[\![\langle\nu\rangle\tau]\!](t,u) &= \text{И}\mathsf{a}{:}\nu.\ neq[\![\tau]\!](t \mathbin{@} \mathsf{a}, u \mathbin{@} \mathsf{a}) \\
neq[\![\nu]\!](t,u) &= t \# u \\
neq_\delta(t,u) &:- \bigvee\{\exists X, Y{:}\tau.\ t \approx f(X) \wedge u \approx f(Y) \wedge neq[\![\tau]\!](X,Y) \\
&\qquad\qquad \mid f : \tau \to \delta \in \Sigma\} \\
&\quad \vee \bigvee\{\exists X{:}\tau, Y{:}\tau'.\ t \approx f(X) \wedge u \approx g(Y) \\
&\qquad\qquad \mid f : \tau \to \delta, g : \tau' \to \delta \in \Sigma, f \neq g\}
\end{aligned}
$$

$$
\begin{aligned}
nfr[\![\nu, \tau]\!] &: \nu \times \tau \to o \\
nfr[\![\nu, \mathbf{1}]\!](a,t) &= \bot \\
nfr[\![\nu, \tau_1 \times \tau_2]\!](a,t) &= nfr[\![\nu, \tau_1]\!](a, \pi_1(t)) \vee nfr[\![\nu, \tau_2]\!](a, \pi_2(t)) \\
nfr[\![\nu, \delta]\!](a,t) &= nfr_{\nu,\delta}(a,t) \\
nfr[\![\nu, \langle\nu'\rangle\tau]\!](a,t) &= \text{И}\mathsf{b}{:}\nu'.\ nfr[\![\tau]\!](a, t \mathbin{@} \mathsf{b}) \\
nfr[\![\nu, \nu]\!](a,b) &= a \approx b \\
nfr[\![\nu, \nu']\!](a,b) &= \bot \quad (\nu \neq \nu') \\
nfr_{\nu,\delta}(a,t) &:- \bigvee\{\exists X{:}\tau.\ t \approx f(X) \wedge nfr[\![\nu, \tau]\!](a, X) \mid f : \tau \to \delta \in \Sigma\}
\end{aligned}
$$

2 Other experiments

Random testing has been present in Isabelle/HOL's since [1] and has been recently enriched with a notion of *smart* test generators to improve its success rate w.r.t. conditional properties. Exhaustive and symbolic testing follow the SmallCheck approach [3]. Notwithstanding all these improvements, QuickCheck requires all code and specs to be *executable* in the underlying functional language, while many of the specifications that we are interested in are best seen as *partial* and *not terminating*.

While not terribly exciting, these benchmarks, proposed and measured in [2] and taken from Isabelle *List.thy* theory are useful to set up a rough comparison with Isabelle's QuickCheck. We show the checks in our logic programming formulation, leaving to the reader the obvious meaning, noting only that we use numerals as datatype.

```
D1: distinct([X|XS]) => distinct(XS).
D2: distinct(XS),remove1(X,XS,YS) => distinct(YS).
D3: distinct(XS),distinct(YS),zip(XS,YS,ZS) => distinct(ZS).
S1: sorted(XS),remove_dupls(XS,YS) => sorted(YS).
S2: sorted(XS),insert(X,XS,YS) => sorted(YS).
S3: sorted(XS),length(XS,N),less_equal(I,J),less(J,N),
                        nth(I,XS,X),nth(J,XS,Y) => less_equal(X,Y).
```

Table 2 shows the TESS run time up to a given size (25), that in our case we interpret as depth-bound. We extrapolated from Table 2 in [2] the S (for *smart generator*) rows. We omit the results for *exhaustive* and *narrowing-based* testing; the point of their inclusion was to show how smart generation outperforms the latter two over checks with hard-to-satisfy premises. Again, these measurements are only suggestive, since QuickCheck's result are taken with another hardware (empty cells denote timeout after 1h as in [2]'s setup). Still, we are largely superior, possibly due to smart generation trying to replicate in a functional setting what logic programming naturally offers. Note however that tests in Isabelle/QuickCheck are efficiently run by code generation at the ML level, while our bounded solver is just a non-optimized logic programming interpreter – to name one, it does not have yet first-argument indexing.

As usual in TESS, negation elimination tends to outperform *NF*, especially when, as here, it does not require extensional quantification. *NEs* only marginally improves on *NE*, because the negated predicates (`distinct,sorted` etc.) are already quite simple.

References

1. S. Berghofer and T. Nipkow. Random testing in Isabelle/HOL. In *SEFM*, pages 230–239. IEEE Computer Society, 2004.
2. L. Bulwahn. Smart testing of functional programs in Isabelle. In N. Bjørner and A. Voronkov, editors, *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2012.
3. C. Runciman, M. Naylor, and F. Lindblad. Smallcheck and lazy SmallCheck: automatic exhaustive testing for small values. In A. Gill, editor, *Haskell Workshop*, pages 37–48. ACM, 2008.

		9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
`D1`	S	0	0	0	0.2	0.7	3.8	22	135	862								
	NF	0	0	0	0	0	0	0	0	0	0.07	0.12	0.2	0.32	0.52	0.83	1.36	2.22
	NE	0	0	0	0	0	0	0	0	0	0.06	0.11	0.18	0.3	0.49	1.8	1.3	2.1
	NEs	0	0	0	0	0	0	0	0	0	0.06	0.11	0.18	0.3	0.4	0.6	1.0	1.7
`D2`	S	0	0	0.1	0.4	2.5	16	98	671									
	NF	0	0	0	0	0	0	0	0	0	0	0.07	0.19	0.32	0.51	0.83	1.36	2.23
	NE	0	0	0	0	0	0	0	0	0	0.6	0.11	0.18	0.3	0.49	0.8	1.32	2.17
	NEs	0	0	0	0	0	0	0	0	0	0.6	0.11	0.18	0.2	0.39	0.6	1.1	1.7
`D3`	S	4.3	157															
	NF	0	0	0	0.08	0.14	0.35	0.76	1	3	6	12	24	45	82	155	286	580
	NE	0	0	0	0.08	0.13	0.32	0.68	1.3	3	6	11	22	42	79	150	280	586
	NEs	0	0	0	0.08	0.13	0.22	0.5	0.9	2.1	4.5	8	17	3	63	121	225	448
`S1`	S	0	0	0	0	0	0	0	0	0.10	0.2	0.3	0.8	1.7	3.6	7.8	17	36
	NF	0	0	0	0	0	0	0	0	0	0	0.6	0.08	0.11	0.15	0.21	0.27	0.35
	NE	0	0	0	0	0	0	0	0	0	0	0.06	0.08	0.11	0.15	0.2	0.27	0.36
	NEs	0	0	0	0	0	0	0	0	0	0	0	0.04	0.06	0.08	0.11	0.16	0.2
`S2`	S	0	0	0	0	0	0.1	0.1	0.2	0.5	1.1	2.5	5.5	12	28	61	135	292
	NF	0	0	0	0	0	0	0	0	0	0	0	0.05	0.07	0.1	0.13	0.18	0.23
	NE	0	0	0	0	0	0	0	0	0	0.06	0.08	0.11	0.15	0.19	0.25	0.33	0.44
	NEs	0	0	0	0	0	0	0	0	0	0.02	0.04	0.04	0.06	0.08	0.11	0.16	0.2
`S3`	S	0	0	0	0	0.1	0.1	0.2	0.4	0.9	2.2	5.1	12	26	59	136	311	708
	NF	0	0	0.05	0.08	0.13	0.2	0.32	0.48	0.73	1	1.5	2.2	3.2	4.5	6.4	8.9	12
	NE	0	0	0	0.05	0.08	0.12	0.18	0.27	0.4	0.57	0.83	1.1	1.6	2.2	3.2	4.3	5.7
	NEs	0	0	0	0	0	0	0.04	0.09	0.1	0.28	0.4	0.5	0.8	1.1	1.5	2.1	2.9

Table 1. TESS for list benchmark.