Soundness and Completeness of the NRB Verification Logic

Peter T. Breuer[1(✉)] and Simon J. Pickin[2]

[1] Department of Computer Science, University of Birmingham, Birmingham, UK
Peter.T.Breuer@gmail.com
[2] Facultad de Informática, Universidad Complutense de Madrid, Madrid, Spain
spickin@ucm.es

Abstract. A simple semantic model for the NRB logic of program verification is provided here, and the logic is shown to be sound and complete with respect to it. That provides guarantees in support of the logic's use in the automated verification of large imperative code bases, such as the Linux kernel source. 'Soundness' implies that no breaches of safety conditions are missed, and 'completeness' implies that symbolic reasoning is as powerful as model-checking here.

1 Introduction

NRB ('normal, return, break') program logic was first introduced in 2004 [5] as the theory supporting an automated semantic analysis suite [4] targeting the C code of the Linux kernel. The analyses performed with this kind of program logic and automatic tools are typically much more approximate than that provided by more interactive or heavyweight techniques such as theorem-proving and model-checking [10], respectively, but NRB-based solutions have proved capable of rapidly scanning millions of lines of C code and detecting deadlocks scattered as rarely as one per million lines of code [9]. A rough synopsis of the logic is that it is precise in terms of accurately following the often complex flow of control and sequence of events in an imperative language, but not very accurate at following data values. That is fine in the context of a target like C [1,12], where static analysis cannot reasonably hope to follow all data values accurately because of the profligate use of pointers in a typical program (a pointer may access any part of memory, in principle, hence writing through a pointer might 'magically' change any value) and the NRB logic was designed to work around that problem by focussing instead on information derived from sequences of events.

Modal operators in NRB designate the kind of exit from a code fragment, as **return**, **break**, etc. The logic may be configured in detail to support different abstractions in different analyses; detecting the freeing of a record in memory while it may still be referenced requires an abstraction that counts the possible reference holders, for example, not the value currently in the second field from the right. The technique became known as 'symbolic approximation' [6,7] because of the foundation in symbolic logic and because the analysis is guaranteed to be

S. Counsell and M. Núñez (Eds.): SEFM 2013 Collocated Workshops, LNCS 8368, pp. 389–404, 2014.
DOI: 10.1007/978-3-319-05032-4_28,

inaccurate but on the alarmist side ('approximate from above'). In other words, the analysis does not miss bugs, but does report false positives. In spite of a few years' pedigree behind it now, a foundational semantics for the logic has only just been published [8] (as an Appendix to the main text, in which it is shown that the verification computation can be distributed over a network of volunteer solvers and how such a procedure may be used as the basis of an open certification process). This article aims to provide a yet simpler semantics for the logic and also a completeness result, with the aim of consolidating the technique's bona fides. It fulfils the moral obligation to provide theoretical guarantees for a method that verifies code.

Interestingly, the main formal guarantee ('never miss, always over-report') provided by NRB and symbolic approximation is said not to be desirable in the commercial context by the very practical authors of the Coverity analysis tool [3,11], which also has been used for static analysis of the Linux kernel and many very large C code projects. Allegedly, in the commercial arena, understandability of reports is crucial, not the guarantee that no bugs will be missed. The Coverity authors say that commercial clients tend to dismiss any reports from Coverity staff that they do not understand, turning a deaf ear to all explanations. The reports produced by our tools have been filtered as part of the process [4] before presentation to the client community, so that only the alarms that cannot be dismissed by us as false positives are seen by them. When our process has been organised as a distributed certification task, as reported in [8], then filtering away false positives can be seen as one more 'eyes-on' task for the human part of the anonymous network of volunteer certifiers.

The layout of this paper is as follows. In Sect. 2 a model of programs as sets of 'coloured' transitions between states is set out, and the constructs of a generic imperative language are expressed in those terms. It is shown that the constructs obey certain algebraic laws, which soundly implement the established deduction rules of NRB logic. Section 3 shows that the logic is complete, in that anything that is true in the model introduced in Sect. 2 can be proved using the formal rules of the NRB logic.

Since the model contains at least as many transitions as occur in reality, 'soundness' of the NRB logic means that it may construct *false* alarms for a safety condition possibly being breached at some particular point in a program, but it may not miss any real alarms. 'Completeness' means that the logic flags no more false alarms than are already to be predicted from the model, so if the model says that there ought to be no alarms at all, which implies there really are no alarms, then the logic can prove that. Thus, it is not necessary to construct and examine the complete graph of modelled state transitions ('model checking') in order to be able to give a program a clean bill of health, because the logic does that job, checking 'symbolically'.

2 Semantic Model

This section sets out a semantic model for the full NRBG(E) logic ('NRB' for short) shown in Table 1. The 'NRBG' part stands for 'normal, return, break,

Table 1. NRB deduction rules for triples of assertions and programs. Unless explicitly noted, assumptions $\mathbf{G}_l p_l$ at left are passed down unaltered from top to bottom of each rule. We let $\mathcal{E}_1$ stand for any of $\mathbf{R}$, $\mathbf{B}$, $\mathbf{G}_l$, $\mathbf{E}_k$; $\mathcal{E}_2$ any of $\mathbf{R}$, $\mathbf{G}_l$, $\mathbf{E}_k$; $\mathcal{E}_3$ any of $\mathbf{R}$. $\mathbf{G}_{l'}$ for $l' \neq l$, $\mathbf{E}_k$; $\mathcal{E}_4$ any of $\mathbf{R}$. $\mathbf{G}_l$, $\mathbf{E}_{k'}$ for $k' \neq k$; $[h]$ the body of the subroutine named h.

$$\frac{\triangleright\ \{p\}\ P\ \{\mathbf{N}q \vee \mathcal{E}_1 x\} \quad \triangleright\ \{q\}\ Q\ \{\mathbf{N}r \vee \mathcal{E}_1 x\}}{\triangleright\ \{p\}\ P\,;Q\ \{\mathbf{N}r \vee \mathcal{E}_1 x\}}\,[\text{seq}] \qquad \frac{\triangleright\ \{p\}\ P\ \{\mathbf{B}q \vee \mathbf{N}p \vee \mathcal{E}_2 x\}}{\triangleright\ \{p\}\ \mathbf{do}\ P\ \{\mathbf{N}q \vee \mathcal{E}_2 x\}}\,[\text{do}]$$

$$\frac{}{\triangleright\ \{p\}\ \mathbf{skip}\ \{\mathbf{N}\ p\}}\,[\text{skp}] \qquad \frac{}{\triangleright\ \{p\}\ \mathbf{return}\ \{\mathbf{R}\ p\}}\,[\text{ret}]$$

$$\frac{}{\triangleright\ \{p\}\ \mathbf{break}\ \ \{\mathbf{B}\ p\}}\,[\text{brk}] \quad [p \to p_l]\,\frac{}{\mathbf{G}_l\ p_l \triangleright\ \{p\}\ \mathbf{goto}\ l\ \{\mathbf{G}_l\ p\}}\,[\text{go}]$$

$$\frac{}{\triangleright\ \{p\}\ \mathbf{throw}\ k\ \{\mathbf{E}_k\ p\}}\,[\text{throw}] \quad \frac{}{\triangleright\ \{q[e/x]\}\ x{=}e\ \{\mathbf{N}q\}}\,[\text{let}]$$

$$\frac{\triangleright\ \{q \wedge p\}\ P\ \{r\}}{\triangleright\ \{p\}\ q \to P\ \{r\}}\,[\text{grd}] \quad \frac{\triangleright\ \{p\}\ P\ \{q\} \quad \triangleright\ \{p\}\ Q\ \{q\}}{\triangleright\ \{p\}\ P \mid Q\ \{q\}}\,[\text{dsj}]$$

$$[\mathbf{N}p_l \to q]\,\frac{\mathbf{G}_l\ p_l \triangleright\ \{p\}\ P\ \{q\}}{\mathbf{G}_l\ p_l \triangleright\ \{p\}\ P{:}l\ \{q\}}\,[\text{frm}] \qquad \frac{\mathbf{G}_l\ p_l \triangleright\ \{p\}\ P\ \{\mathbf{G}_l p_l \vee \mathbf{N}q \vee \mathcal{E}_3 x\}}{\triangleright\ \{p\}\ \mathbf{label}\ l.P\ \{\mathbf{N}q \vee \mathcal{E}_3 x\}}\,[\text{lbl}]$$

$$\frac{\triangleright\ \{p\}\ [h]\ \{\mathbf{R}r \vee \mathbf{E}_k x_k\}}{\mathbf{G}_l p_l \triangleright\ \{p\}\ \mathbf{call}\ h\ \{\mathbf{N}r \vee \mathbf{E}_k x_k\}}\,[\text{sub}] \qquad \frac{\triangleright\ \{p\}\ P\ \{\mathbf{N}r \vee \mathbf{E}_k q \vee \mathcal{E}_4 x\} \quad \triangleright\ \{q\}\ Q\ \{\mathbf{N}r \vee \mathbf{E}_k x_k \vee \mathcal{E}_4 x\}}{\triangleright\ \{p\}\ \mathbf{try}\ P\ \mathbf{catch}(k)\ Q\ \{\mathbf{N}r \vee \mathbf{E}_k x_k \vee \mathcal{E}_4 x\}}\,[\text{try}]$$

$$\frac{\triangleright\ \{p_i\}\ P\ \{q\}}{\triangleright\ \{\curlyvee p_i\}\ P\ \{q\}} \qquad \frac{\triangleright\ \{p\}\ P\ \{q_i\}}{\triangleright\ \{p\}\ P\ \{\curlywedge q_i\}} \qquad \frac{\mathbf{G}_l\ p_{li} \triangleright\ \{p\}\ P\ \{q\}}{\curlyvee \mathbf{G}_l\ p_{li} \triangleright\ \{p\}\ P\ \{q\}}$$

$$[p' \to p, q \to q', p'_l \to p_l \mid \mathbf{G}_l q' \to \mathbf{G}_l p'_l]\,\frac{\mathbf{G}_l\ p_l \triangleright\ \{p\}\ P\ \{q\}}{\mathbf{G}_l\ p'_l \triangleright\ \{p'\}\ P\ \{q'\}}$$

goto', and the 'E' part treats exceptions (catch/throw in Java, setjmp/longjmp in C), aiming at a complete treatment of classical imperative languages. This semantics simplifies a *trace model* presented in the Appendix to [8], substituting traces there for state transitions here. The objective in laying out the model is to allow the user of NRB logic to agree that it is talking about what he/she understands a program does, computationally. So the model aims at simplicity and comprehensibility. Agree with it, and one has confidence in what the logic says a program may do.

A standard model of a program is as a relation of type $\mathbb{P}(S \times S)$, expressing possible changes in the program state as a 'set of pairs', consisting of initial and final states of type S. We add a *colour* to this picture. The colour shows if the program has run *normally* through to the end (colour '**N**') or has terminated early via a **return** (colour '**R**'), **break** (colour '**B**'), **goto** (colour '$\mathbf{G}_l$' for some label l) or an exception (colour '$\mathbf{E}_k$' for some exception kind k). This documents the control flow precisely. In our modified picture, a program is a set of 'coloured transitions' of type

$$\mathbb{P}(S \times \star \times S)$$

where the colours $\star$ are a disjoint union

$$\star = \{\mathbf{N}\} \sqcup \{\mathbf{R}\} \sqcup \{\mathbf{B}\} \sqcup \{\mathbf{G}_l \mid l \in L\} \sqcup \{\mathbf{E}_k \mid k \in K\}$$

and L is the set of possible **goto** labels and K the set of possible exception kinds. We write the transition from state s_1 to state s_2 of colour ι as $s_1 \overset{\iota}{\mapsto} s_2$.

Table 2. Models of simple statements.

A **skip** statement is modelled as $$[\![\mathbf{skip}]\!]_g = \{s \overset{\mathbf{N}}{\mapsto} s \mid s \in S\}$$ It makes the transition from a state to the same state again, and ends 'normally'.	A **return** statement has the model $$[\![\mathbf{return}]\!]_g = \{s \overset{\mathbf{R}}{\mapsto} s \mid s \in S\}$$ It exits at once 'via a return flow' after a single, trivial transition.
The model of **skip**; **return** is $$[\![\mathbf{skip}; \mathbf{return}]\!]_g = \{s \overset{\mathbf{R}}{\mapsto} s \mid s \in S\}$$ which is the same as that of **return**. It is made up of the compound of two trivial state transitions, $s \overset{\mathbf{N}}{\mapsto} s$ from **skip** and $s \overset{\mathbf{R}}{\mapsto} s$ from **return**, the latter ending in a 'return flow'.	The **return**; **skip** compound is modelled as: $$[\![\mathbf{return}; \mathbf{skip}]\!]_g = \{s \overset{\mathbf{R}}{\mapsto} s \mid s \in S\}$$ It is made up of of just the $s \overset{\mathbf{R}}{\mapsto} s$ transitions from **return**. There is no transition that can be formed as the composition of a transition from **return** followed by a transition from **skip**, because none of the first end 'normally'.

The programs we usually consider are deterministic, in that only at most one transition from each initial state s appears in the modelling relation, but they are embedded in a more general context where an arbitrary number of transitions may appear. Where the relation is not defined at all on some initial state s, we understand that that initial state leads inevitably to the program getting hung in an infinite loop, instead of terminating. The relations representing deterministic programs have a set of transitions from a given initial state s that is either of size zero ('hangs') or one ('terminates'). Only paths through the program that do not hang are of interest to us, and what the NRB logic will say about a program at some point is true only supposing control reaches that point, which it may never do.

Programs are put together in sequence with the second program accepting as inputs only the states that the first program ends 'normally' with. Otherwise the state with which the first program exited abnormally is the final outcome. That is,

$$\begin{aligned}[\![P; Q]\!] = \{&s_0 \overset{\iota}{\mapsto} s_1 \in [\![P]\!] \mid \iota \neq \mathbf{N}\} \\ &\cup \{s_0 \overset{\iota}{\mapsto} s_2 \mid s_1 \overset{\iota}{\mapsto} s_2 \in [\![Q]\!],\ s_0 \overset{\mathbf{N}}{\mapsto} s_1 \in [\![P]\!]\}\end{aligned}$$

This statement is not complete, however, because abnormal exits with a **goto** from P may still re-enter in Q if the **goto** target is in Q, and proceed. We postpone consideration of this eventuality by predicating the model with the sets of states g_l *hypothesised* as being fed in at the point l in the code. The model with these sets g_l as parameters takes account of the putative extra inputs at the point labeled l:

$$[\![P;Q]\!]_g = \{s_0 \overset{\iota}{\mapsto} s_1 \in [\![P]\!]_g \mid \iota \neq \mathbf{N}\}$$
$$\cup\ \{s_0 \overset{\iota}{\mapsto} s_2 \mid s_1 \overset{\iota}{\mapsto} s_2 \in [\![Q]\!]_g,\ s_0 \overset{\mathbf{N}}{\mapsto} s_1 \in [\![P]\!]_g\}$$

Later, we will tie things up by ensuring that the set of states bound to early exits via a **goto** l in P are exactly the sets g_l hypothesised here as entries at label l in Q. The type of the *interpretation* expressed by the fancy square brackets is

$$[\![-_1]\!]_{-_2} : \mathscr{C} \to (L \nrightarrow \mathbb{P}S) \to \mathbb{P}(S \times \star \times S)$$

where g, the second argument/suffix, has the partial function type $L \nrightarrow \mathbb{P}S$ and the first argument/bracket interior has type $\mathscr{C}$, denoting a simple language of imperative statements whose grammar is set out in Table 3. The models of some of its very basic statements as members of $\mathbb{P}(S \times \star \times S)$ are shown in Table 2. We briefly discuss these and other constructs of the language.

A real imperative programming language such as C can be mapped onto $\mathscr{C}$ – in principle exactly, but in practice rather approximately with respect to data values. A conventional **if**(b) P **else** Q statement in C is written as the choice between two guarded statements $b \to P \mathbin{\triangleright} \neg b \to Q$ in the abstract language $\mathscr{C}$; the conventional **while**(b) P loop in C is expressed as **do**$\{\neg b \to$ **break** $\mathbin{\triangleright} b \to P\}$, using the forever-loop of $\mathscr{C}$. A sequence $P; l : Q$ in C with a label l in the middle should strictly be expressed as $P : l; Q$ in $\mathscr{C}$, but we regard $P; l : Q$ as syntactic sugar for that, so it is still permissible to write $P; l : Q$ in $\mathscr{C}$. As a very special syntactic sweetener, we permit $l : Q$ too, even when there is no preceding statement P, regarding it as an abbreviation for **skip** $: l; Q$.

Curly brackets may be used to group code statements in $\mathscr{C}$, and parentheses may be used to group expressions. The variables are globals and are not formally declared. The terms of $\mathscr{C}$ are piecewise linear integer forms in integer variables, so the boolean expressions are piecewise comparisons between linear forms.

Table 3. Grammar of the abstract imperative language $\mathscr{C}$, where integer variables $x \in X$, term expressions $e \in \mathscr{E}$, boolean expressions $b \in \mathscr{B}$, labels $l \in L$, exceptions $k \in K$, statements $c \in \mathscr{C}$, integer constants $n \in \mathbb{Z}$, infix binary relations $r \in R$, subroutine names $h \in H$. Note that labels (the targets of **goto**s) are declared with '**label**' and a label cannot be the first thing in a code sequence; it must follow some statement. Instead of **if**, $\mathscr{C}$ has guarded statements $b \to P$ and explicit choice $P \mathbin{\triangleright} Q$, for code fragments P, Q. The choice construct is only used in practice in the expansion of **if** and **while** statements, so all its real uses are deterministic (have at most one transition from each initial state), although it itself is not.

$\mathscr{C} ::=$ **skip** | **return** | **break** | **goto** l | $c;c$ | $x{=}e$ | $b{\to}c$ | $c \mathbin{\triangleright} c$ | **do** c | $c : l$ | **label** $l.c$ | **call** h
| **try** c **catch**(k) c | **throw** k

$\mathscr{E} ::= n \mid x \mid n * e \mid e + e \mid b\,?\,e : e$

$\mathscr{B} ::= \top \mid \bot \mid e\ r\ e \mid b \vee b \mid b \wedge b \mid \neg b \mid \exists x.b$

$R ::= < \mid > \mid \leq \mid \geq \mid = \mid \neq$

Table 4. The conventional evaluation of integer and boolean terms of $\mathscr{C}$, for variables $x \in X$, integer constants $\kappa \in \mathbb{Z}$, using $s\,x$ for the (integer) value of the variable named x in a state s. The form $b[n/x]$ means 'expression b with integer n substituted for all unbound occurrences of x'.

$$
\begin{aligned}
[\![-]\!] &: \mathscr{E} \to S \to \mathbb{Z} \\
[\![x]\!]s &= s\,x \\
[\![\kappa]\!]s &= \kappa \\
[\![\kappa * e]\!]s &= \kappa * [\![e]\!]s \\
[\![e_1 + e_2]\!]s &= [\![e_1]\!]s + [\![e_2]\!]s \\
[\![b\,?\,e_1 : e_2]\!]s &= \text{if } [\![b]\!]s \text{ then } [\![e_1]\!]s \text{ else } [\![e_2]\!]s
\end{aligned}
\qquad
\begin{aligned}
[\![-]\!] &: \mathscr{B} \to S \to \mathbf{bool} \\
[\![\top]\!]s &= \top \qquad [\![\bot]\!]s = \bot \\
[\![e_1 < e_2]\!]s &= [\![e_1]\!]s < [\![e_2]\!]s \\
[\![b_1 \vee b_2]\!]s &= [\![b_1]\!]s \vee [\![b_2]\!]s \\
[\![b_1 \wedge b_2]\!]s &= [\![b_1]\!]s \wedge [\![b_2]\!]s \\
[\![\neg b]\!]s &= \neg([\![b]\!]s) \\
[\![\exists x.b]\!]s &= \exists n \in \mathbb{Z}.[\![b[n/x]]\!]s
\end{aligned}
$$

Example 1. A valid integer term is '5x + 4y + 3', and a boolean expression is '5x + 4y + 3 < z − 4 ∧ y ≤ x'.

In consequence another valid integer term, taking the value of the first on the range defined by the second, and 0 otherwise, is '(5x + 4y + 3 < z − 4 ∧ y ≤ x) ? 5x + 4y + 3:0'.

The limited set of terms in $\mathscr{C}$ makes it practically impossible to map standard imperative language assignments as simple as 'x = x ∗ y' or 'x = x | y' (the bitwise or) succinctly. In principle, those could be expressed exactly point by point using conditional expressions (with at most 2^{32} disjuncts), but it is usual to model all those cases by means of an abstraction away from the values taken to attributes that can be represented more elegantly using piecewise linear terms The abstraction may be to how many times the variable has been read since last written, for example, which maps 'x = x ∗ y' to 'x = x + 1; y = y + 1; x = 0'.

Formally, terms have a conventional evaluation as integers and booleans that is shown (for completeness!) in Table 4. The reader may note the notation $s\,x$ for the evaluation of the variable named x in state s, giving its integer value as result. We say that state s *satisfies* boolean term $b \in \mathscr{B}$, written $s \models b$, whenever $[\![b]\!]s$ holds.

The **label** construct of $\mathscr{C}$ declares a label $l \in L$ that may subsequently be used as the target in **goto**s. The component P of the construct is the body of code in which the label is *in scope*. A label may not be mentioned except in the scope of its declaration. The same label may not be declared again in the scope of the first declaration. The semantics of labels and **goto**s will be further explained below.

The only way of exiting the $\mathscr{C}$ **do** loop construct normally is via **break** in the body P of the loop. An abnormal exit other than **break** from the body P terminates the whole loop abnormally. Terminating the body P normally evokes one more turn round the loop. So conventional **while** and **for** loops in C are mapped in $\mathscr{C}$ to a **do** loop with a guarded **break** statement inside, at the head of the body. The precise models for this and every construct of $\mathscr{C}$ as a set of coloured transitions are enumerated in Table 5.

Table 5. Model of programs of language $\mathscr{C}$, given as hypothesis the sets of states g_l for $l \in L$ observable at **goto** l statements. A recursive reference means 'the least set satisfying the condition'. For $h \in H$, the subroutine named h has code $[h]$. The state s altered by the assignment of n to variable x is written $s[x \mapsto n]$.

$$
\begin{aligned}
[\![-]\!]_g : \mathscr{C} &\to \mathbb{P}(S \times \star \times S)\\
[\![\mathbf{skip}]\!]_g &= \{s_0 \overset{\mathbf{N}}{\mapsto} s_0 \mid s_0 \in S\}\\
[\![\mathbf{return}]\!]_g s_0 &= \{s_0 \overset{\mathbf{R}}{\mapsto} s_0 \mid s_0 \in S\}\\
[\![\mathbf{break}]\!]_g &= \{s_0 \overset{\mathbf{B}}{\mapsto} s_0 \mid s_0 \in S\}\\
[\![\mathbf{goto}\ l]\!]_g &= \{s_0 \overset{\mathbf{G}_l}{\mapsto} s_0 \mid s_0 \in S\}\\
[\![\mathbf{throw}\ k]\!]_g &= \{s_0 \overset{\mathbf{E}_k}{\mapsto} s_0 \mid s_0 \in S\}\\
[\![P; Q]\!]_g &= \{s_0 \overset{\iota}{\mapsto} s_1 \in [\![P]\!]_g \mid \iota \neq \mathbf{N}\}\\
&\quad \cup \{s_0 \overset{\iota}{\mapsto} s_2 \mid s_1 \overset{\iota}{\mapsto} s_2 \in [\![Q]\!]_g,\ s_0 \overset{\mathbf{N}}{\mapsto} s_1 \in [\![P]\!]_g\}\\
[\![x = e]\!]_g s_0 &= \{s_0 \overset{\mathbf{N}}{\mapsto} s_0[x \mapsto [\![e]\!]s_0]\} \mid s_0 \in S\}\\
[\![p \to P]\!]_g &= \{s_0 \overset{\iota}{\mapsto} s_1 \in [\![P]\!]_g \mid [\![p]\!]s_0\}\\
[\![P \mid Q]\!]_g &= [\![P]\!]_g \cup [\![Q]\!]_g\\
[\![\mathbf{do}\ P]\!]_g &= \{s_0 \overset{\mathbf{N}}{\mapsto} s_1 \mid s_0 \overset{\mathbf{B}}{\mapsto} s_1 \in [\![P]\!]_g\}\\
&\quad \cup \{s_0 \overset{\iota}{\mapsto} s_1 \in [\![P]\!]_g \mid \iota \neq \mathbf{N}, \mathbf{B}\}\\
&\quad \cup \{s_0 \overset{\iota}{\mapsto} s_2 \mid s_1 \overset{\iota}{\mapsto} s_2 \in [\![\mathbf{do}\ P]\!]_g,\ s_0 \overset{\iota}{\mapsto} s_1 \in [\![P]\!]_g\}\\
[\![P : l]\!]_g &= [\![P]\!]_g\\
&\quad \cup \{s_0 \overset{\mathbf{N}}{\mapsto} s_1 \mid s_0 \in S,\ s_1 \in g_l\}\\
[\![\mathbf{label}\ l\ P]\!]_g &= [\![P]\!]_{g \cup \{l \mapsto g_l^*\}} - g_l^*\\
&\quad \text{where } g_l^* = \{s_1 \mid s_0 \overset{\mathbf{G}_l}{\mapsto} s_1 \in [\![P]\!]_{g \cup \{l \mapsto g_l^*\}}\}\\
[\![\mathbf{call}\ h]\!]_g &= \{s_0 \overset{\mathbf{N}}{\mapsto} s_1 \mid s_0 \overset{\mathbf{R}}{\mapsto} s_1 \in [\![[h]]\!]_{\{\,\}}\}\\
&\quad \cup \{s_0 \overset{\mathbf{E}_k}{\mapsto} s_1 \in [\![[h]]\!]_{\{\,\}} \mid k \in K\}\\
[\![\mathbf{try}\ P\ \mathbf{catch}(k)\ Q\ ]\!]_g &= \{s_0 \overset{\iota}{\mapsto} s_1 \in [\![P]\!]_g \mid \iota \neq \mathbf{E}_k\}\\
&\quad \cup \{s_0 \overset{\iota}{\mapsto} s_2 \mid s_1 \overset{\iota}{\mapsto} s_2 \in [\![Q]\!]_g,\ s_0 \overset{\mathbf{E}_k}{\mapsto} s_1 \in [\![P]\!]_g\}
\end{aligned}
$$

Among the list in Table 5, the semantics of **label** declarations in particular requires explanation because labels are more explicitly controlled in $\mathscr{C}$ than in standard imperative languages. Declaring a label l makes it invisible from the outside of the block (while enabling it to be used inside), working just the same way as a local variable declaration does in a standard imperative programming language. A declaration removes from the model of a labelled statement the dependence on the hypothetical set g_l of the states attained at **goto** l statements. All the instances of **goto** l statements are inside the block with the declaration at its head, so we can take a look to see what totality of states really do accrue at **goto** l statements; they are recognisable in the model because they are the outcomes of the transitions that are marked with $\mathbf{G}_l$. Equating the set of such states with the hypothesis g_l gives the (least) fixpoint g_l^* required in the **label** l model.

The hypothetical sets g_l of states that obtain at **goto** l statements are used at the point where the label l appears within the scope of the declaration. We

say that any of the states in g_l may be an outcome of passing through the label l, because it may have been brought in by a **goto** l statement. That is an overestimate; in reality, if the state just before the label is s_1, then at most those states s_2 in g_l that are reachable at a **goto** l from an initial program state s_0 that also leads to s_1 (either s_1 first or s_2 first) may obtain after the label l, and that may be considerably fewer s_2 than we calculate in g_l^*. Here is a visualisation of such a situation; the curly arrows denote a trace:

$$\begin{array}{ccccc} & \nearrow & \{s_1\} & l: & \{s1, s_2\} \\ \{s_0\} & & \wr & & \\ & & \{s_2\} & \textbf{goto}\ l & \end{array}$$

If the initial precondition on the code admits more than one initial state s_0 then the model may admit more states s_2 after the label l than occur in reality when s_1 precedes l, because the model does not take into account the dependence of s_2 on s_1 through s_0. It is enough for the model that s_2 proceeds from some s_0 and s_1 proceeds from some (possibly different) s_0 satisfying the same initial condition. In mitigation, **goto**s are sparsely distributed in real codes and we have not found the effect pejorative.

Example 2. Consider the code R and suppose the input is restricted to a unique state s:

$$\textbf{label}\ A, B. \overbrace{\underbrace{\textbf{skip};\ \textbf{goto}\ A;\ B\text{: }\textbf{return};\ A}_{Q}\text{: }\textbf{goto}\ B}^{P}$$

with labels A, B in scope in body P, and the marked fragment Q. The single transitions made in the code P and the corresponding statement sequences are:

$$\begin{array}{ll} s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{G}_A}{\mapsto} s & \#\ \textbf{skip};\ \textbf{goto}\ A; \\ s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{G}_B}{\mapsto} s & \#\ \textbf{skip};\ \textbf{goto}\ A; A : \textbf{goto}\ B \\ s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{R}}{\mapsto} s & \#\ \textbf{skip};\ \textbf{goto}\ A; A : \textbf{goto}\ B; B : \textbf{return} \end{array}$$

with observed states $g_A = \{s\}$, $g_B = \{s\}$ at the labels A and B respectively.

The **goto** B statement is not in the fragment Q so there is no way of knowing about the set of states at **goto** B while examining Q. Without that input, the traces of Q are

$$\begin{array}{ll} s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{G}_A}{\mapsto} s & \#\ \textbf{skip};\ \textbf{goto}\ A \\ s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{N}}{\mapsto} s & \#\ \textbf{skip};\ \textbf{goto}\ A; A : \end{array}$$

There are no possible entries at B originating from within Q itself. That is, the model $[\![Q]\!]_g$ of Q as a set of transitions assuming $g_B = \{\,\}$, meaning there are no entries from outside, is $[\![Q]\!]_g = \{s \overset{\mathbf{N}}{\mapsto} s, s \overset{\mathbf{G}_A}{\mapsto} s\}$.

When we hypothesise $g_B = \{s\}$ for Q, then Q has more traces:

$$s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{N}}{\mapsto} s \overset{\mathbf{R}}{\mapsto} s \qquad \#\ \textbf{skip};\ \textbf{goto}\ A; A : \textbf{goto}\ B; B : \textbf{return}$$

Table 6. Extending the language $\mathscr{B}$ of propositions to modal operators $\mathbf{N}$, $\mathbf{R}$, $\mathbf{B}$, $\mathbf{G}_l$, $\mathbf{E}_k$ for $l \in L$, $k \in K$. An evaluation on transitions is given for $b \in \mathscr{B}$, $b^* \in \mathscr{B}^*$.

$$\mathscr{B}^* ::= b \mid \mathbf{N}\, b^* \mid \mathbf{R}\, b^* \mid \mathbf{B}\, b^* \mid \mathbf{G}_l\, b^* \mid \mathbf{E}_k\, b^* \mid b^* \vee b^* \mid b^* \wedge b^* \mid \neg b^*$$

$$\begin{aligned}
[\![b]\!](s_0 \overset{\iota}{\mapsto} s_1) &= [\![b]\!]s_1 \\
[\![\mathbf{N}\, b^*]\!](s_0 \overset{\iota}{\mapsto} s_1) &= (\iota = \mathbf{N}) \wedge [\![b^*]\!](s_0 \overset{\iota}{\mapsto} s_1) \\
[\![\mathbf{R}\, b^*]\!](s_0 \overset{\iota}{\mapsto} s_1) &= (\iota = \mathbf{R}) \wedge [\![b^*]\!](s_0 \overset{\iota}{\mapsto} s_1) \\
[\![\mathbf{B}\, b^*]\!](s_0 \overset{\iota}{\mapsto} s_1) &= (\iota = \mathbf{B}) \wedge [\![b^*]\!](s_0 \overset{\iota}{\mapsto} s_1) \\
[\![\mathbf{G}_l\, b^*]\!](s_0 \overset{\iota}{\mapsto} s_1) &= (\iota = \mathbf{G}_l) \wedge [\![b^*]\!](s_0 \overset{\iota}{\mapsto} s_1) \\
[\![\mathbf{E}_k\, b^*]\!](s_0 \overset{\iota}{\mapsto} s_1) &= (\iota = \mathbf{E}_k) \wedge [\![b^*]\!](s_0 \overset{\iota}{\mapsto} s_1)
\end{aligned}$$

Table 7. Laws of the modal operators $\mathbf{N}$, $\mathbf{R}$, $\mathbf{B}$, $\mathbf{G}_l$, $\mathbf{E}_k$ with $M, M_1, M_2 \in \{\mathbf{N}, \mathbf{R}, \mathbf{B}, \mathbf{G}_l, \mathbf{E}_k \mid l \in L, k \in K\}$ and $M_1 \neq M_2$.

$$\begin{aligned}
M(\bot) &= \bot && \text{(flatness)} \\
M(b_1 \vee b_2) &= M(b_1) \vee M(b_2) && \text{(disjunctivity)} \\
M(b_1 \wedge b_2) &= M(b_1) \wedge M(b_2) && \text{(conjunctivity)} \\
M(Mb) &= Mb && \text{(idempotence)} \\
M_2(M_1 b) = M_1(b) \wedge M_2(b) &= \bot && \text{(orthogonality)}
\end{aligned}$$

corresponding to these entries at B from the rest of the code proceeding to the **return** in Q, and $[\![Q]\!]_g = \{s \overset{\mathbf{N}}{\mapsto} s,\ s \overset{\mathbf{G}_A}{\mapsto} s,\ s \overset{\mathbf{R}}{\mapsto} s\}$. In the context of the whole code P, that is the model for Q as a set of initial to final state transitions.

Example 3. Staying with the code of Example 2, the set $\{s \overset{\mathbf{G}_A}{\mapsto} s,\ s \overset{\mathbf{G}_B}{\mapsto} s,\ s \overset{\mathbf{R}}{\mapsto} s\}$ is the model $[\![P]\!]_g$ of P starting at state s with assumptions g_A, g_B of Example 2, and the sets g_A, g_B are observed at the labels A, B in the code under these assumptions. Thus $\{A \mapsto g_A, B \mapsto g_B\}$ is the fixpoint g^* of the **label** declaration rule in Table 5.

That rule says to next remove transitions ending at **goto** As and Bs from visibility in the model of the declaration block, because they can go nowhere else, leaving only $[\![R]\!]_{\{\,\}} = \{s \overset{\mathbf{R}}{\mapsto} s\}$ as the set-of-transitions model of the whole block of code, which corresponds to the sequence **skip**; **goto** A; A : **goto** B; B : **return**.

We extend the propositional language to $\mathscr{B}^*$ which includes the modal operators $\mathbf{N}$, $\mathbf{R}$, $\mathbf{B}$, $\mathbf{G}_l$, $\mathbf{E}_k$ for $l \in L$, $k \in K$, as shown in Table 6, which defines a model of $\mathscr{B}^*$ on transitions. The predicate $\mathbf{N}p$ informally should be read as picking out from the set of all coloured state transitions 'those normal-coloured transitions that produce a state satisfying p', and similarly for the other operators. The modal operators satisfy the algebraic laws given in Table 7. Additionally, however, for non-modal $p \in \mathscr{B}$,

$$p = \mathbf{N}p \vee \mathbf{R}p \vee \mathbf{B}p \vee \mathbb{W}\, \mathbf{G}_l p \,\mathbb{W} \mathbf{E}_k p \tag{1}$$

because each transition must be some colour, and those are all the colours. The decomposition works in the general case too:

Proposition 1. *Every $p \in \mathscr{B}^*$ can be (uniquely) expressed as*

$$p = \mathbf{N}p_{\mathbf{N}} \vee \mathbf{R}p_{\mathbf{R}} \vee \mathbf{B}p_{\mathbf{B}} \vee \Vbar\, \mathbf{G}_l p_{\mathbf{G}_l} \Vbar \mathbf{E}_k p_{\mathbf{E}_k}$$

for some $p_{\mathbf{N}}$, $p_{\mathbf{R}}$, etc that are free of modal operators.

Proof. *Equation (1) gives the result for $p \in \mathscr{B}$. The rest is by structural induction on p, using Table 7 and boolean algebra. Uniqueness follows because $\mathbf{N}p_{\mathbf{N}} = \mathbf{N}p'_{\mathbf{N}}$, for example, applying $\mathbf{N}$ to two possible decompositions, and applying the orthogonality and idempotence laws; apply the definition of $\mathbf{N}$ in the model in Table 6 to deduce $p_{\mathbf{N}} = p'_{\mathbf{N}}$ for non-modal predicates $p_{\mathbf{N}}$, $p'_{\mathbf{N}}$. Similarly for $\mathbf{B}$, $\mathbf{R}$, $\mathbf{G}_l$, $\mathbf{E}_k$.* □

So modal formulae $p \in \mathscr{B}^*$ may be viewed as tuples $(p_{\mathbf{N}}, p_{\mathbf{R}}, p_{\mathbf{B}}, p_{\mathbf{G}_l}, p_{\mathbf{E}_k})$ of non-modal formulae from $\mathscr{B}$ for labels $l \in L$, exception kinds $k \in K$. That means that $\mathbf{N}p \vee \mathbf{R}q$, for example, is simply a convenient notation for writing down two assertions at once: one that asserts p of the final states of the transitions that end 'normally', and one that asserts q on the final states of the transitions that end in a 'return flow'. The meaning of $\mathbf{N}p \vee \mathbf{R}q$ is the union of the set of the normal transitions with final state that satisfy p plus the set of the transitions that end in a 'return flow' and whose final states satisfy q. We can now give meaning to a notation that looks like (and is intended to signify) a Hoare triple with an explicit context of certain '**goto** assumptions':

Definition 1. *Let $g_l = [\![p_l]\!]$ be the set of states satisfying $p_l \in \mathscr{B}$, labels $l \in L$. Then '$\mathbf{G}_l\, p_l \triangleright \{p\}\ a\ \{q\}$', for non-modal $p, p_l \in \mathscr{B}$, $P \in \mathscr{C}$ and $q \in \mathscr{B}^*$, means:*

$$\begin{aligned}[\![\mathbf{G}_l\, p_l \triangleright \{p\}\ P\ \{q\}]\!] &= [\![\{p\}\ P\ \{q\}]\!]_g \\ &= \forall s_0 \overset{\iota}{\mapsto} s_1 \in [\![P]\!]_g.\ [\![p]\!]s_0 \Rightarrow [\![q]\!](s_0 \overset{\iota}{\mapsto} s_1)\end{aligned}$$

That is read as 'the triple $\{p\}\ P\ \{q\}$ holds under assumptions p_l at **goto** l when every transition of P that starts at a state satisfying p also satisfies q'. The explicit Gentzen-style assumptions p_l are free of modal operators. What is meant by the notation is that those states that may be attainable as the program traces pass through **goto** statements are assumed to be restricted to those that satisfy p_l.

The $\mathbf{G}_l\, p_l$ assumptions may be separated by commas, as $\mathbf{G}_{l_1}\, p_{l_1}, \mathbf{G}_{l_2}\, p_{l_2}, \ldots$, with $l_1 \neq l_2$, etc. Or they may be written as a disjunction $\mathbf{G}_{l_1}\, p_{l_1} \vee \mathbf{G}_{l_2}\, p_{l_2} \vee \ldots$ because the information in this modal formula is only the mapping $l_1 \mapsto p_{l_1}$, $l_2 \mapsto p_{l_2}$, etc. If the same l appears twice among the disjuncts $\mathbf{G}_l\, p_l$, then we understand that the union of the two p_l is intended.

Now we can prove the validity of laws about triples drawn from what Definition 1 says. The first laws are strengthening and weakening results on pre- and postconditions:

Proposition 2. *The following algebraic relations hold:*

$$[\![\{\bot\}\ P\ \{q\}]\!]_g \iff \top \tag{2}$$
$$[\![\{p\}\ P\ \{\top\}]\!]_g \iff \top \tag{3}$$
$$[\![\{p_1 \vee p_2\}\ P\ \{q\}]\!]_g \iff [\![\{p_1\}\ P\ \{q\}]\!]_g \wedge [\![\{p_2\}\ P\ \{q\}]\!]_g \tag{4}$$
$$[\![\{p\}\ P\ \{q_1 \wedge q_2\}]\!]_g \iff [\![\{p\}\ P\ \{q_1\}]\!]_g \wedge [\![\{p\}\ P\ \{q_2\}]\!]_g \tag{5}$$
$$(p_1 \rightarrow p_2) \wedge [\![\{p_2\}\ P\ \{q\}]\!]_g \implies [\![\{p_1\}\ P\ \{q\}]\!]_g \tag{6}$$
$$(q_1 \rightarrow q_2) \wedge [\![\{p\}\ P\ \{q_1\}]\!]_g \implies [\![\{p\}\ P\ \{q_2\}]\!]_g \tag{7}$$
$$[\![\{p\}\ P\ \{q\}]\!]_{g'} \implies [\![\{p\}\ P\ \{q\}]\!]_g \tag{8}$$

for $p, p_1, p_2 \in \mathscr{B}$, $q, q_1, q_2 \in \mathscr{B}^*$, $P \in \mathscr{C}$, *and* $g_l \subseteq g'_l \in \mathbb{P}S$.

Proof. *(2–5) follow on applying Definition 1. (6–7) follow from (4–5) on considering the cases* $p_1 \vee p_2 = p_2$ *and* $q_1 \wedge q_2 = q_1$. *The reason for (8) is that* g'_l *is a bigger set than* g_l, *so* $[\![P]\!]_{g'}$ *is a bigger set of transitions than* $[\![P]\!]_g$ *and thus the universal quantifier in Definition 1 produces a smaller (less true) truth value.* □

Theorem 1 (Soundness). *The following algebraic inequalities hold, for* $\mathcal{E}_1$ *any of* $\mathbf{R}$, $\mathbf{B}$, $\mathbf{G}_l$, $\mathbf{E}_k$; $\mathcal{E}_2$ *any of* $\mathbf{R}$, $\mathbf{G}_l$, $\mathbf{E}_k$; $\mathcal{E}_3$ *any of* $\mathbf{R}$, $\mathbf{B}$, $\mathbf{G}_{l'}$ *for* $l' \neq l$, $\mathbf{E}_k$; $\mathcal{E}_4$ *any of* $\mathbf{R}$, $\mathbf{B}$, $\mathbf{G}_l$, $\mathbf{E}_{k'}$ *for* $k' \neq k$; $[h]$ *the code of the subroutine called* h:

$$\left.\begin{array}{r}[\![\{p\}\,P\,\{\mathbf{N}q \vee \mathcal{E}_1 x\}]\!]_g \\ \wedge\ [\![\{q\}\,Q\,\{\mathbf{N}r \vee \mathcal{E}_1 x\}]\!]_g\end{array}\right\} \implies [\![\{p\}\,P\,;Q\,\{\mathbf{N}r \vee \mathcal{E}_1 x\}]\!]_g \tag{9}$$
$$[\![\{p\}\,P\,\{\mathbf{B}q \vee \mathbf{N}p \vee \mathcal{E}_2 x\}]\!]_g \implies [\![\{p\}\,\mathbf{do}\ P\,\{\mathbf{N}q \vee \mathcal{E}_2 x\}]\!]_g \tag{10}$$
$$\top \implies [\![\{p\}\,\mathbf{skip}\,\{\mathbf{N}\,p\}]\!]_g \tag{11}$$
$$\top \implies [\![\{p\}\,\mathbf{return}\,\{\mathbf{R}\,p\}]\!]_g \tag{12}$$
$$\top \implies [\![\{p\}\,\mathbf{break}\ \{\mathbf{B}\,p\}]\!]_g \tag{13}$$
$$\top \implies [\![\{p\}\,\mathbf{goto}\ l\,\{\mathbf{G}_l\,p\}]\!]_g \tag{14}$$
$$\top \implies [\![\{p\}\,\mathbf{throw}\ k\,\{\mathbf{E}_k\,p\}]\!]_g \tag{15}$$
$$[\![\{b \wedge p\}\,P\,\{q\}]\!]_g \implies [\![\{p\}\,b \rightarrow P\ \{q\}]\!]_g \tag{16}$$
$$[\![\{p\}\,P\ \{q\}]\!]_g \wedge [\![\{p\}\,Q\ \{q\}]\!]_g \implies [\![\{p\}\,P \,\mathbin{!}\, Q\ \{q\}]\!]_g \tag{17}$$
$$\top \implies [\![\{q[e/x]\}\ x{=}e\ \{\mathbf{N}q\}]\!]_g \tag{18}$$
$$[\![\{p\}\ P\ \{q\}]\!]_g \wedge g_l \subseteq \{s_1 \mid s_0 \overset{\mathbf{N}}{\mapsto} s_1 \in [\![q]\!]\} \implies [\![\{p\}\ P : l\ \{q\}]\!]_g \tag{19}$$
$$[\![\{p\}\ P\ \{\mathbf{G}_l p_l \vee \mathbf{N}q \vee \mathcal{E}_3 x\}]\!]_{g \cup \{l \mapsto p_l\}} \implies [\![\{p\}\ \mathbf{label}\ l.P\ \{\mathbf{N}q \vee \mathcal{E}_3 x\}]\!]_g \tag{20}$$
$$[\![\{p\}\ [h]\ \{\mathbf{R}r \vee \mathbf{E}_k x_k\}]\!]_{\{\ \}} \implies [\![\{p\}\ \mathbf{call}\ h\ \{\mathbf{N}r \vee \mathbf{E}_k x_k\}]\!]_g \tag{21}$$
$$\left.\begin{array}{r}[\![\{p\}\ P\ \{\mathbf{N}r \vee \mathbf{E}_k q \vee \mathcal{E}_4 x\}]\!]_g \\ \wedge\ [\![\{q\}\ Q\ \{\mathbf{N}r \vee \mathbf{E}_k x_k \vee \mathcal{E}_4 x\}]\!]_g\end{array}\right\} \implies [\![\{p\}\ \mathbf{try}\ P\ \mathbf{catch}(k)\ Q\ \{\mathbf{N}r \vee \mathbf{E}_k x_k \vee \mathcal{E}_4 x\}]\!]_g \tag{22}$$

Proof By evaluation, given Definition 1 and the semantics from Table 5. □

The reason why the theorem is titled 'Soundness' is that its inequalities can be read as the NRB logic deduction rules set out in Table 1, via Definition 1. The fixpoint requirement of the model at the **label** construct is expressed in the 'arrival from a **goto** at a label' law (19), where it is stated that *if* the hypothesised states g_l at a **goto** l statement are covered by the states q immediately after code block P and preceding label l, *then* q holds after the label l too. However, there is no need for any such predication when the g_l are exactly the fixpoint of the map

$$g_l \mapsto \{s_1 \mid s_0 \overset{\mathbf{G}_l}{\mapsto} s_1 \in [\![P]\!]_g\}$$

because that is what the fixpoint condition says. Thus, while the model in Table 5 satisfies Eqs. (9–22), it satisfies more than they require – some of the hypotheses in the equations could be dropped and the model would still satisfy them. But the NRB logic rules in Table 1 are validated by the model and thus are sound.

3 Completeness

In proving completeness of the NRB logic, we will be guided by the proof of partial completeness for Hoare's logic in K. R. Apt's survey paper [2]. We will need, for every (possibly modal) postcondition $q \in \mathscr{B}^*$ and every construct R of $\mathscr{C}$, a non-modal formula $p \in \mathscr{B}$ that is weakest in $\mathscr{B}$ such that if p holds of a state s, and $s \overset{\iota}{\mapsto} s'$ is in the model of R given in Table 5, then q holds of $s \overset{\iota}{\mapsto} s'$. This p is written $\mathrm{wp}(R, q)$, the 'weakest precondition on R for q'. We construct it via structural induction on $\mathscr{C}$ at the same time as we deduce completeness, so there is an element of chicken versus egg about the proof, and we will not labour that point.

We will also suppose that we can prove any tautology of $\mathscr{B}$ and $\mathscr{B}^*$, so 'completeness of NRB' will be relative to that lower-level completeness.

Notice that there is always a set $p \in \mathbb{P}S$ satisfying the 'weakest precondition' characterisation above. It is $\{s \in S \mid s \overset{\iota}{\mapsto} s' \in [\![R]\!]_g \Rightarrow s \overset{\iota}{\mapsto} s' \in [\![q]\!]\}$, and it is called the weakest *semantic* precondition on R for q. So we sometimes refer to $\mathrm{wp}(R, q)$ as the 'weakest *syntactic* precondition' on R for q, when we wish to emphasise the distinction. The question is whether or not there is a formula in $\mathscr{B}$ that exactly expresses this set. If there is, then the system is said to be *expressive*, and that formula *is* the weakest (syntactic) precondition on R for q, $\mathrm{wp}(R, q)$. Notice also that a weakest (syntactic) precondition $\mathrm{wp}(R, q)$ must encompass the semantic weakest precondition; that is because if there were a state s in the latter and not in the former, then we could form the disjunction $\mathrm{wp}(R, q) \vee (x_1 = sx_1 \wedge \ldots x_n = sx_n)$ where the x_i are the variables of s, and this would also be a precondition on R for q, hence $x_1 = sx_1 \wedge \ldots x_n = sx_n \rightarrow \mathrm{wp}(R, q)$ must be true, as the latter is supposedly the weakest precondition, and so s satisfies $\mathrm{wp}(R, q)$ in contradiction to the assumption that s is not in $\mathrm{wp}(R, q)$. For orientation, then, the reader should note that 'there is a weakest (syntactic) precondition in $\mathscr{B}$' means there is a unique strongest formula in $\mathscr{B}$ covering the weakest semantic precondition.

We will lay out the proof of completeness inline here, in order to avoid excessively overbearing formality, and at the end we will draw the formal conclusion.

A completeness proof is always a proof by cases on each construct of interest. It has the form 'suppose that *foo* is true, then we can prove it like this', where *foo* runs through all the constructs we are interested in. We start with assertions about the sequence construction $P;Q$. We will look at this in particular detail, noting where and how the weakest precondition formula plays a role, and skip that detail for most other cases. Thus we start with *foo* equal to $\mathbf{G}_l\, g_l \triangleright \{p\}\ P;Q\ \{q\}$ for some assumptions $g_l \in \mathscr{B}$, but we do not need to take the assumptions g_l into account in this case.

Case $P;Q$. *Consider a sequence of two statements* $P;Q$ *for which* $\{p\}\ P;Q\ \{q\}$ *holds in the model set out by Definition* 1 *and Table* 5. *That is, suppose that initially the state* s *satisfies predicate* p *and that there is a progression from* s *to some final state* s' *through* $P;Q$. *Then* $s \overset{\iota}{\mapsto} s'$ *is in* $[\![P;Q]\!]_g$ *and* $s \overset{\iota}{\mapsto} s'$ *satisfies* q. *We will consider two subcases, the first where* P *terminates normally from* s, *and the second where* P *terminates abnormally from* s. *A third possibility, that* P *does not terminate at all, is ruled out because a final state* s' *is reached.*

Consider the first subcase. According to Table 5, *that means that* P *started in state* $s_0 = s$ *and finished normally in some state* s_1 *and* Q *ran on from state* s_1 *to finish normally in state* $s_2 = s'$. *Let* r *stand for the weakest precondition wp(Q,q) that guarantees termination of* Q *with* q *holding. By definition,* $\{r\}\ Q\ \{q\}$, *is true and* s_1 *satisfies* r *(if not, then* $r \vee (x_1 = sx_1 \wedge x_2 = sx_2 \wedge \dots)$ *would be a weaker precondition for* q *than* r, *which is impossible). So* $\{p\}\ P\ \{\mathbf{N}r\}$ *is true in this case.*

Now consider the second subcase, when the final state s_1 *reached from* $s = s_0$ *through* P *obtains via an abnormal flow out of* P. *The transition* $s_0 \overset{\iota}{\mapsto} s_1$ *satisfies* q, *but necesarily an abnormal 'error' component of* q *as the flow out of* p *is abnormal, so* $\{p\}\ P\ \{\mathbf{R}q \vee \mathbf{B}q \vee \dots\}$, *as* $\mathbf{R}q \vee \mathbf{B}q \vee \dots$ *is the error component of* q *by Proposition* 1.

Those are the only cases, so $\{p\}\ P\ \{\mathbf{N}r \vee \mathbf{R}q \vee \mathbf{B}q \vee \dots\}$ *is true. By induction, it is the case that there are deductions* $\vdash \{p\}\ P\ \{\mathbf{N}r \vee \mathbf{R}q \vee \mathbf{B}q \vee \dots\}$ *and* $\vdash \{r\}\ Q\ \{q\}$ *in the NRB system. But the following rule*

$$\frac{\{p\}\ P\ \{\mathbf{N}r \vee \mathbf{R}q \vee \mathbf{B}q \vee \dots\} \quad \{r\}\ Q\ \{q\}}{\{p\}\ P;Q\ \{q \vee \mathbf{R}q \vee \mathbf{B}q \vee \dots\}}$$

is a derived rule of NRB logic. It is a specialised form of the general NRB rule of sequence, availing of the 'mixed' error colour $\mathcal{E} = (\mathbf{R} \vee \mathbf{B} \vee \dots)$. *Since* $(q \vee \mathcal{E}q) \to q$, *putting these deductions together, and using weakening, we have a deduction of the truth of the assertions* $\{p\}\ P;Q\ \{q\}$.

That concludes the consideration of the case $P;Q$. The existence of a formula expressing a weakest precondition is what really drives the proof above along, and in lieu of pursuing the proof through all the other construct cases, we note the important weakest precondition formulae below:

– The weakest precondition for sequence is $\mathrm{wp}(a;b,q) = \mathrm{wp}(a, \mathcal{E}q \vee \mathbf{N}\mathrm{wp}(b,q))$ above.

– The weakest precondition for assignment is $\mathrm{wp}(x = e, \mathbf{N}q) = q[e/x]$ for q without modal components. In general $\mathrm{wp}(x = e, q) = \mathbf{N}q[e/x]$.
– The weakest precondition for a **return** statement is $\mathrm{wp}(\mathbf{return}, q) = \mathbf{R}q$.
– The weakest precondition for a **break** statement is $\mathrm{wp}(\mathbf{break}, q) = \mathbf{B}q$. Etc.
– The weakest precondition $\mathrm{wp}(\mathbf{do}\ P, \mathbf{N}q)$ for a **do** loop that ends 'normally' is $\mathbf{wp}(P, \mathbf{B}q) \vee \mathbf{wp}(P, \mathbf{Nwp}(P, \mathbf{B}q)) \vee \mathbf{wp}(P, \mathbf{Nwp}(P, \mathbf{Nwp}(P, \mathbf{B}q))) \vee \ldots$. That is, we might break from P with q, or run through P normally to the precondition for breaking from P with q next, etc. Write $\mathbf{wp}(P, \mathbf{B}q)$ as p and write $\mathbf{wp}(P, \mathbf{N}r) \wedge \neg p$ as $\psi(r)$, Then $\mathrm{wp}(\mathbf{do}\ P, \mathbf{N}q)$ can be written $p \vee \psi(p) \vee \psi(p \vee \psi(p)) \vee \ldots$, which is the strongest solution to $\pi = \psi(\pi)$ no stronger than p. This is the weakest precondition for p after $\mathbf{while}(\neg p)\ P$ in classical Hoare logic. It is an existentially quantified statement, stating that an initial state s gives rise to exactly some n passes through P before the condition p becomes true for the first time. It can classically be expressed as a formula of first-order logic and it is the weakest precondition for $\mathbf{N}q$ after $\mathbf{do}\ P$ here.
  The preconditions for $\mathcal{E}q$ for each 'abnormal' coloured ending $\mathcal{E}$ of the loop $\mathbf{do}\ P$ are similarly expressible in $\mathscr{B}$, and the precondition for q is the disjunction of each of the preconditions for $\mathbf{N}q$, $\mathbf{R}q$, $\mathbf{B}q$, etc.
– The weakest precondition for a guarded statement $\mathrm{wp}(p \to P, q)$ is $p \to \mathrm{wp}(P, q)$, as in Hoare logic; and the weakest precondition for a disjunction $\mathrm{wp}(P \mathbin{\mathrm{I}} Q, q)$ is $\mathrm{wp}(P, q) \wedge \mathrm{wp}(Q, q)$, as in Hoare logic. However, in practice we only use the deterministic combination $p \to P \mathbin{\mathrm{I}} \neg p \to Q$ for which the weakest precondition is $(p \to \mathrm{wp}(P, q)) \wedge (\neg p \to \mathrm{wp}(Q, q))$, i.e. $p \wedge \mathrm{wp}(P, q) \vee \neg p \wedge \mathrm{wp}(Q, q)$.

To deal with labels properly, we have to extend some of these notions and notations to take account of the assumptions $\mathbf{G}_l g_l$ that an assertion $\mathbf{G}_l g_l \triangleright \{p\}\ P\ \{q\}$ is made against. The weakest precondition p on P for q is then $p = \mathrm{wp}_g(P, q)$, with the g_l as extra parameters. The weakest precondition for a label use $\mathrm{wp}_g(P : l, q)$ is then $\mathrm{wp}_g(P, q)$, provided that $g_l \to q$, since the states g_l attained by **goto** l statements throughout the code are available after the label, as well as those obtained through P. The weakest precondition in the general situation where it is not necessarily the case that $g_l \to q$ holds is $\mathrm{wp}_g(P, q \wedge (g_l \to q))$, which is $\mathrm{wp}_g(P, q)$.

Now we can continue the completeness proof through the statements of the form $P : l$ (a labelled statement) and $\mathbf{label}\ l.P$ (a label declaration).

Case Labelled Statement. *If* $[\![\{p\}\ P : l\ \{q\}]\!]_g$ *holds, then (a) every state* $s = s_0$ *satisfying* p *leads through* P *with* $s_0 \overset{\iota}{\mapsto} s_1$ *satisfying* q*, and also (b)* q *contains all the transitions* $s_0 \overset{\mathbf{N}}{\mapsto} s_1$ *where* s_1 *satisfies* g_l*. By (a),* s *satisfies* $\mathrm{wp}_g(P, q)$ *and (b)* $\mathbf{N}g_l \to q$ *holds. Since* s *is arbitrary in* p*, so* $p \to \mathrm{wp}_g(P, q)$ *holds and by induction,* $\vdash \mathbf{G}_l g_l \triangleright \{p\}\ P\ \{q\}$*. Then, by the 'frm' rule of NRB (Table 1), we may deduce* $\vdash \mathbf{G}_l g_l \triangleright \{p\}\ P : l\ \{q\}$.

Case Label Declaration. *The weakest precondition for a declaration* $\mathrm{wp}_g(\mathbf{label}\, l.P, q)$ *is simply* $p = \mathrm{wp}_{g'}(P, q)$*, where the assumptions after the*

declaration are $g' = g \cup \{l \mapsto g_l\}$ *and* g_l *is such that* $\mathbf{G}_l g_l \rhd \{p\}\ P\ \{q\}$. *In other words,* p *and* g_l *are simultaneously chosen to make the assertion hold,* p *maximal and* g_l *the least fixpoint describing the states at* **goto** l *statements in the code* P, *given that the initial state satisfies* p *and assumptions* $\mathbf{G}_l g_l$ *hold. The* $g_l y$ *are the statements that after exactly some* $n \in \mathbb{N}$ *more traversals through* P *via* **goto** l, *the trace from state* s *will avoid another* **goto** l *for the first time and exit* P *normally or via an abnormal exit that is not a* **goto** l.

If it is the case that $[\![\{p\}\ \mathbf{label}\ l.P\ \{q\}]\!]_g$ *holds then every state* $s = s_0$ *satisfying* p *leads through* **label** $l.P$ *with* $s_0 \overset{\iota}{\mapsto} s_1$ *satisfying* q. *That means* $s_0 \overset{\iota}{\mapsto} s_1$ *leads through* P, *but it is not all that do; there are transitions with* $\iota = \mathbf{G}_l$ *that are not considered. The 'missing' transitions are precisely the* $\mathbf{G}_l g_l$ *where* g_l *is the appropriate least fixpoint for* $g_l = \{s_1 \mid s_0 \overset{\mathbf{G}_l}{\mapsto} s_1 \in [\![P]\!]_{g \cup \{l \mapsto g_l\}}$, *which is a predicate expressing the idea that* s_1 *at a* **goto** l *initiates some exactly* n *traversals back through* P *again before exiting* P *for a first time other than via a* **goto** l. *The predicate* q *cannot mention* $\mathbf{G}_l$ *since the label* l *is out of scope for it, but it may permit some, all or no* $\mathbf{G}_l$*-coloured transitions. The predicate* $q \vee \mathbf{G}_l g_l$, *on the other hand, permits all the* $\mathbf{G}_l$*-coloured transitions that exit* P. *transitions. Thus adding* $\mathbf{G}_l g_l$ *to the assumptions means* s_0 *traverses* P *via* $s_0 \overset{\iota}{\mapsto} s_1$ *satisfying* $q \vee \mathbf{G}_l g_l$ *even though more transitions are admitted. Since* $s = s_0$ *is arbitrary in* p, *so* $p \rightarrow \mathrm{wp}_{g \cup \{l \mapsto g_l\}}(P, q \vee \mathbf{G}_l g_l)$ *and by induction* $\vdash \mathbf{G}_l \rhd \{p\}\ P\ \{q \vee \mathbf{G}_l g_l\}$, *and then one may deduce* $\vdash \{p\}$ **label** $l.P\ \{q\}$ *by the 'lbl' rule.* □

That concludes the text that would appear in a proof, but which we have abridged and presented as a discussion here! We have covered the typical case $(P; Q)$ and the unusual cases ($P : l$, **label** $l.P$). The proof-theoretic content of the discussion is:

Theorem 2 (Completeness). *The system of NRB logic in Table 1 is complete, relative to the completeness of first-order logic.*

Theorem 3 (Expressiveness). *The weakest precondition* $wp(P, q)$ *for* $q \in \mathscr{B}^*$, $P \in \mathscr{C}$ *in the interpretation set out in Definition 1 and Table 5 is expressible in* $\mathscr{B}$.

The observation above is that there is a formula in $\mathscr{B}$ that expresses the semantic weakest precondition exactly.

4 Summary

In this article, we have complemented previous work, which guaranteed programs free of semantic defects, with guarantees directed at the symbolic logic used as guarantor. Soundness of the logic is proved with respect to a simple transition-based model of programs, and completeness of the logic with respect to the model is proved.

That shows the logic is equivalent to the model, and reduces the question of its fitness as a guarantor for a program to the fitness for that purpose of the model of programs. The model always overestimates the number of transitions

that may occur, so when the logic is used to certify that there are no program transitions that may violate a given safety condition, it may be believed.

Acknowledgments. Simon Pickin's research has been partially supported by the Spanish MEC project ESTuDIo (TIN2012-36812-C02-01).

References

1. American National Standards Institute. American national standard for information systems - programming language C, ANSI X3.159-1989 (1989)
2. Apt, K.R.: Ten years of Hoare's logic: a survey: part I. ACM Trans. Program. Lang. Syst. **3**(4), 431–483 (1981)
3. Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., Engler, D.: A few billion lines of code later: using static analysis to find bugs in the real world. Commun. ACM **53**(2), 66–75 (2010)
4. Breuer, P.T., Pickin, S.: Checking for deadlock, double-free and other abuses in the Linux kernel source code. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3994, pp. 765–772. Springer, Heidelberg (2006)
5. Breuer, P.T., Valls, M.: Static deadlock detection in the Linux kernel. In: Llamosí, A., Strohmeier, A. (eds.) Ada-Europe 2004. LNCS, vol. 3063, pp. 52–64. Springer, Heidelberg (2004)
6. Breuer, P.T., Pickin, S.: Symbolic approximation: an approach to verification in the large. Innovations Syst. Softw. Eng. **2**(3), 147–163 (2006)
7. Breuer, P.T., Pickin, S.: Verification in the large via symbolic approximation. In: Proceedings of the 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, 2006 (ISoLA 2006), pp. 408–415. IEEE (2006)
8. Breuer, P.T., Pickin, S.: Open source verification in an anonymous volunteer network. Sci. Comput. Program. (2013). doi:10.1016/j.scico.2013.08.010
9. Breuer, P.T., Pickin, S., Petrie, M.L.: Detecting deadlock, double-free and other abuses in a million lines of Linux kernel source. In: Proceedings of the 30th Annual Software Engineering Workshop 2006 (SEW'06), pp. 223–233. IEEE/NASA (2006)
10. Clarke, E., Emerson, E., Sistla, A.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Prog. Lang. Syst. (TOPLAS) **8**(2), 244–253 (1986)
11. Engler, D., Chelf, B., Chou, A., Hallem, S.: Checking system rules using system-specific, programmer-written compiler extensions. In: Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI 2000), pp. 1–16, October 2000
12. International Standards Organisation. ISO/IEC 9899-1999, programming languages - C (1999)