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, UK
ptb@cs.bham.ac.uk
[2] Facultad de Informática, Universidad Complutense de Madrid
spickin@ucm.es

Abstract. This short paper gives a model for and a proof of completeness of the NRB verification logic for deterministic imperative programs, the logic having been used in the past as the basis for automated semantic checks of large, fast-changing, open source C code archives, such as that of the Linux kernel source. The model is a coloured state transitions model that approximates from above the set of transitions possible for a program. Correspondingly, the logic catches all traces that may trigger a particular defect at a given point in the program, but may also flag false positives.

1 Introduction

NRB 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 the NRB combination has proved capable of rapidly scanning millions of lines of C code and detecting deadlocks scattered at one per million lines of code [9]. A rough synopsis of the characteristics of the logic or an approach using 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 for a target language like C [1, 13], where static analysis cannot reasonably hope to follow all data values accurately because of the profligate use of indirection through 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.

NRB is a logic with modal operators. The modalities do not denote a full range of actions as in Dynamic Logic [12], but rather only the very particular action of the final exit from a code fragment being via a **return**, **break**, or **goto**. The logic is also configurable in detail to support the code abstractions that are of interest 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 on the alarmist side ('approximate from above'); the analysis

does not miss bugs in code, 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), and 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.

Interestingly, the formal guarantee ('never miss, over-report') provided by NRB and the symbolic approximation technique is said not to be desirable in the commercial context by the very practical authors of the Coverity analysis tool [11, 3], 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 that they do not understand, turning a deaf ear to explanations. However, the reports produced by our tools have always been filtered before presentation, so only the alarms that cannot be dismissed as false positives are seen.

The layout of this paper is as follows. In Section 2 a model of programs as sets of 'coloured' transitions between states is introduced, 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 for deterministic programs, in that anything that is true in the model introduced in Section 2 can be proved using the formal rules of the NRB logic.

Since the model contains at least as many state transitions as occur in reality, 'soundness' of the NRB logic means that it may construct false alarms for when a particular condition may be breached at some particular point in a program, but that 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 means that there really are no alarms), then the logic can prove that. Thus, reasoning symbolically is not in principle an approximation here; it is not necessary to laboriously construct and examine the complete graph of modelled state transitions in order to be able to give a program a 'clean bill of health' with reference to some potential defect, because the logic can always do the job as well.

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, 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.

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

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.

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

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

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

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

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

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

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

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

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

deterministic program may be modelled as 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.

The programs we consider are in fact deterministic, but we will use the general setting. Where the relation is not defined on some initial state s, we understand that the initial state s leads to the program getting hung up in an infinite loop, instead of terminating. Relations representing deterministic programs thus have a set of images for any given initial state that is either of size zero ('hangs') or one ('terminates'). Only paths through the program that do not 'hang' in an infinite loop are of interest to us, and what the NRB logic will say about a program at some point will be 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,

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

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 $\mathbf{skip}; \mathbf{return}$ 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 $\mathbf{return}; \mathbf{skip}$ 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'.

Table 2: Models of simple statements.

This statement is not complete, however, because abnormal exits with a **goto** from P may still re-enter in Q if the **goto** label 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 label l in the code. The model of P and Q with these sets as assumptions produce outputs that take account of these putative extra inputs at label l:

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

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 (and vice versa). 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 and we will discuss them and the interpretations of other language constructs below.

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, as will be indicated below. A conventional $\mathbf{if}(b)\ P\ \mathbf{else}\ Q$ statement in C is written as the nondeterministic choice between two guarded statements $b \to P \,\mathbin{|}\, \neg b \to Q$ in the abstract language $\mathscr{C}$; the conventional $\mathbf{while}(b)\ P$ loop in C is expressed as $\mathbf{do}\{\neg b \to \mathbf{break} \,\mathbin{|}\, b \to P\}$, using the forever-loop of $\mathscr{C}$, etc. A sequence $P; l : Q$ in C with a label l in the

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, and explicit nondeterminism, which, however, is only to be used here in the deterministic construct $b{\rightarrow}P \mathbin{▯} \neg b{\rightarrow}Q$ for code fragments P, Q.

$\mathscr{C}$ ::= $\mathbf{skip} \mid \mathbf{return} \mid \mathbf{break} \mid \mathbf{goto}\ l \mid c;c \mid x{:=}e \mid b{\rightarrow}c \mid c \mathbin{▯} c \mid \mathbf{do}\ c \mid c : l \mid \mathbf{label}\ l.c \mid \mathbf{call}\ h$
$\mid\ \mathbf{try}\ c\ \mathbf{catch}(k)\ c \mid \mathbf{throw}\ k$

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

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

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

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 $\mathbf{skip} : l; Q$.

Curly brackets may be used to group code statements for clarity 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.

Example 1. A valid integer term is '$\mathrm{5x} + \mathrm{4y} + 3$', and a boolean expression is '$\mathrm{5x} + \mathrm{4y} + 3 < \mathrm{z} - 4 \wedge \mathrm{y} \leq \mathrm{x}$'.

In consequence another valid integer term, taking the value of the first on the range defined by the second, and 0 otherwise, is '$(\mathrm{5x}{+}\mathrm{4y}{+}3 < \mathrm{z}{-}4 \wedge \mathrm{y} \leq \mathrm{x})\,?\,\mathrm{5x}{+}\mathrm{4y}{+}3 : 0$'.

The limited set of terms in $\mathscr{C}$ makes it practically impossible to map standard imperative language assignments as simple as '$\mathrm{x} = \mathrm{x} * \mathrm{y}$' or '$\mathrm{x} = \mathrm{x} \mid \mathrm{y}$' (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 '$\mathrm{x} = \mathrm{x} * \mathrm{y}$' to '$\mathrm{x} = \mathrm{x} + 1; \mathrm{y} = \mathrm{y} + 1; \mathrm{x} = 0$'.

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.

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'.

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

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 need to be mapped 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.

Among the list of models in Table 5, that 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:

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

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

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 \shortmid 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}
$$

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:

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

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:

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

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 $b \in \mathscr{B}$, $b^* \in \mathscr{B}^*$.

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

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

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

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

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

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:

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

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 $\mathbf{skip}; \mathbf{goto}\ A; A:\ \mathbf{goto}\ B; B:\ \mathbf{return}$.

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 $p \in \mathscr{B}$,

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

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 $M_1 \neq M_2$.

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

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*

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

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 $\mathbf{goto}\ l$ 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:*

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

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:*

$$\left.\begin{array}{r}[\![\{p\}\,P\,\{\mathbf{N}q \vee \mathcal{E}_1 x\}]\!]_g \\ \wedge\, [\![\{q\}\,Q\,\{\mathbf{N}r \vee \mathcal{E}_1 x\}]\!]_g\end{array}\right\} \Rightarrow [\![\{p\}\,P\,;Q\,\{\mathbf{N}r \vee \mathcal{E}_1 x\}]\!]_g \tag{9}$$

$$[\![\{p\}\,P\,\{\mathbf{B}q \vee \mathbf{N}p \vee \mathcal{E}_2 x\}]\!]_g \Rightarrow [\![\{p\}\,\mathbf{do}\ P\,\{\mathbf{N}q \vee \mathcal{E}_2 x\}]\!]_g \tag{10}$$

$$\top \Rightarrow [\![\{p\}\,\mathbf{skip}\,\{\mathbf{N}\,p\}]\!]_g \tag{11}$$

$$\top \Rightarrow [\![\{p\}\,\mathbf{return}\,\{\mathbf{R}\,p\}]\!]_g \tag{12}$$

$$\top \Rightarrow [\![\{p\}\,\mathbf{break}\ \{\mathbf{B}\,p\}]\!]_g \tag{13}$$

$$\top \Rightarrow [\![\{p\}\,\mathbf{goto}\ l\,\{\mathbf{G}_l\,p\}]\!]_g \tag{14}$$

$$\top \Rightarrow [\![\{p\}\,\mathbf{throw}\ k\,\{\mathbf{E}_k\,p\}]\!]_g \tag{15}$$

$$[\![\{b \wedge p\}\,P\,\{q\}]\!]_g \Rightarrow [\![\{p\}\,b \rightarrow P\ \{q\}]\!]_g \tag{16}$$

$$[\![\{p\}\,P\ \{q\}]\!]_g \wedge [\![\{p\}\,Q\ \{q\}]\!]_g \Rightarrow [\![\{p\}\,P \shortmid Q\ \{q\}]\!]_g \tag{17}$$

$$\top \Rightarrow [\![\{q[e/x]\}\ x{=}e\ \{\mathbf{N}q\}]\!]_g \tag{18}$$

$$[\![\{p\}\ P\ \{q\}]\!]_g \wedge g_l \subseteq \{s_1 \mid s_0 \overset{\mathbf{N}}{\mapsto} s_1 \in [\![q]\!]\} \Rightarrow [\![\{p\}\ P : l\ \{q\}]\!]_g \tag{19}$$

$$[\![\{p\}\ P\ \{\mathbf{G}_l p_l \vee \mathbf{N}q \vee \mathcal{E}_3 x\}]\!]_{g \cup \{l \mapsto p_l\}} \Rightarrow [\![\{p\}\ \mathbf{label}\ l.P\ \{\mathbf{N}q \vee \mathcal{E}_3 x\}]\!]_g \tag{20}$$

$$[\![\{p\}\ [h]\ \{\mathbf{R}r \vee \mathbf{E}_k x_k\}]\!]_{\{\ \}} \Rightarrow [\![\{p\}\ \mathbf{call}\ h\ \{\mathbf{N}r \vee \mathbf{E}_k x_k\}]\!]_g \tag{21}$$

$$\left.\begin{array}{r}[\![\{p\}\,P\,\{\mathbf{N}r \vee \mathbf{E}_k q \vee \mathcal{E}_4 x\}]\!]_g \\ \wedge\, [\![\{q\}\,Q\,\{\mathbf{N}r \vee \mathbf{E}_k x_k \vee \mathcal{E}_4 x\}]\!]_g\end{array}\right\} \Rightarrow [\![\{p\}\ \mathbf{try}\ P\ \mathbf{catch}(k)\ Q\ \{\mathbf{N}r \vee \mathbf{E}_k x_k \vee \mathcal{E}_4 x\}]\!]_g \tag{22}$$

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 equations (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 for deterministic programs

In proving completeness of the NRB logic, at least for deterministic programs, 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 $\text{wp}(R, q)$, 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 $\text{wp}(R, q)$ 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, $\text{wp}(R, q)$. Notice also that a weakest (syntactic) precondition $\text{wp}(R, q)$ 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 $\text{wp}(R, q) \vee (x_1 = sx_1 \wedge \ldots x_n = sx_n)$ where the x_i are the variables of s, and this would also be a precondition on R for q, hence $x_1 = sx_1 \wedge \ldots x_n = sx_n \rightarrow \text{wp}(R, q)$ must be true, as the latter is supposedly the weakest precondition, and so s satisfies $\text{wp}(R, q)$ in contradiction to the assumption that s is not in $\text{wp}(R, q)$. 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, which means that we think of s as confined to wp$(P, \mathbf{N}\top)$. 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$(Q, \mathbf{N}q)$ that guarantees a normal termination of Q with q holding. By definition of weakest precondition, $\{r\}\ Q\ \{\mathbf{N}q\}$, is true and s_1 satisfies r (if not, then $r \vee (x_1 = sx_1 \wedge x_2 = sx_2 \wedge \ldots)$ would be a weaker precondition for $\mathbf{N}q$ than r, which is impossible). The latter is true whatever s_0 satisfying p and wp$(P, \mathbf{N}\top)$ we started with, so by definition of weakest precondition, $p \wedge$

wp$(P, \mathbf{N}\top)\rightarrow$*wp*$(P, \mathbf{N}r)$ *must be true, which is to say that* $\{p \wedge$ *wp*$(P, \mathbf{N}\top)\}\ P\ \{\mathbf{N}r\}$ *is true.*

By induction, it is the case that there are deductions $\vdash \{p \wedge$ *wp*$(P, \mathbf{N}\top)\}\ P\ \{\mathbf{N}r\}$ *and* $\vdash \{r\}\ Q\ \{\mathbf{N}q\}$ *in the NRB system. But the following rule*

$$\frac{\{p \wedge \textit{wp}(P, \mathbf{N}\top)\}\ P\ \{\mathbf{N}r\} \quad \{r\}\ Q\ \{\mathbf{N}q\}}{\{p \wedge \textit{wp}(P, \mathbf{N}\top)\}\ P; Q\ \{\mathbf{N}q\}}$$

is a derived rule of NRB logic. It is a specialised form of the general NRB rule of sequence. Putting these deductions together, we have a deduction of the truth of the assertions $\{p\wedge$*wp*$(P, \mathbf{N}\top)\}\ P; Q\ \{\mathbf{N}q\}$*. By weakening on the conclusion, since* $\mathbf{N}q\rightarrow q$ *is (always) true, we have a deduction of* $\{p \wedge$ *wp*$(P, \mathbf{N}\top)\}\ P; Q\ \{q\}$*.*

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*. This means that we think of* s *as confined to wp*$(P, \neg\mathbf{N}\top)$*. Now the transition* $s_0 \overset{\iota}{\mapsto} s_1$ *in* $[\![P]\!]_g$ *satisfies* q*, and* s *is arbitrary in* $p \wedge$ *wp*$(P, \neg\mathbf{N}\top)$*, so* $\{p \wedge$ *wp*$(P, \neg\mathbf{N}\top)\}\ P\ \{q\}$*. However, 'not ending normally' (and getting to a termination, which is the case here) means 'ending abnormally', i.e.,* $\mathbf{R}\top \vee \mathbf{B}\top \vee \ldots$ *through all of the available colours, as per Proposition 1, and we may write the assertion out as* $\{p \wedge$ *wp*$(P, \mathbf{R}\top \vee \mathbf{B}\top \ldots)\}\ P\ \{q\}$*. Considering the cases separately, one has* $\{p \wedge$ *wp*$(P, \mathbf{R}\top)\}\ P\ \{\mathbf{R}q\}$ *(since* $\mathbf{R}q$ *is the component of* q *that expects an* $\mathbf{R}$*-coloured transition), and* $\{p \wedge$ *wp*$(P, \mathbf{B}\top)\}\ P\ \{\mathbf{B}q\}$*, and so on, all holding. By induction, there are deductions* $\vdash \{p \wedge$ *wp*$(P, \mathbf{R}\top)\}\ P\ \{\mathbf{R}q\}$*,* $\vdash \{p \wedge$ *wp*$(P, \mathbf{B}\top)\}\ P\ \{\mathbf{B}q\}$*, etc. But the following rule*

$$\frac{\{p \wedge \textit{wp}(P, \mathcal{E}\top)\}\ P\ \{\mathcal{E}q\}}{\{p \wedge \textit{wp}(P, \mathcal{E}\top)\}\ P; Q\ \{\mathcal{E}q\}}$$

is a derived rule of NRB logic for each 'abnormal' colouring $\mathcal{E}$*, and hence we have a deduction* $\vdash \{p \wedge$ *wp*$(P, \mathcal{E}\top)\}\ P; Q\ \{\mathcal{E}q\}$ *for each of the 'abnormal' colours* $\mathcal{E}$*. By weakening on the conclusion, since* $\mathcal{E}q\rightarrow q$*, for each of the colours* $\mathcal{E}$*, we have a deduction* $\vdash \{p \wedge$ *wp*$(P, \mathcal{E}\top)\}\ P; Q\ \{q\}$ *for each of the colours* $\mathcal{E}$*.*

By the rule on disjunctive hypotheses (fourth from last in Table 1) we now have a deduction $\vdash \{p \wedge ($*wp*$(P, \mathbf{N}\top) \vee$ *wp*$(P, \mathbf{R}\top) \vee \ldots)\}\ P; Q\ \{q\}$*. But the weakest precondition is monotonic, so* *wp*$(P, \mathbf{N}\top) \vee$ *wp*$(P, \mathbf{R}\top) \vee \ldots$ *is covered by* *wp*$(P, \mathbf{N}\top \vee \mathbf{R}\top \vee \ldots)$*, which is* *wp*$(P, \top)$ *by Proposition 1. But for a deterministic program* P*, the outcome from a single starting state* s *can only be uniquely a normal termination, or uniquely a return termination, etc, and* *wp*$(P, \mathbf{N}\top) \vee$ *wp*$(P, \mathbf{R}\top) \vee \cdots =$ *wp*$(P, \mathbf{N}\top \vee \mathbf{R}\top \vee \ldots) =$ *wp*$(P, \top)$ *exactly. The latter is just* $\top$*, so we have a proof* $\vdash \{p\}P; Q\ \{q\}$*. As to what the weakest precondition* *wp*$(P; Q, q)$ *is, it is* *wp*$(P, \mathbf{N}$*wp*$(Q, q))\vee$*wp*$(P, \mathbf{R}q)\vee$ *wp*$(P, \mathbf{B}q) \vee \ldots$*, the disjunction being over all the possible colours.*

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 assignment is wp$(x = e, \mathbf{N}q) = q[e/x]$ for q without modal components. In general wp$(x = e, q) = \mathbf{N}q[e/x]$.

– The weakest precondition for a **return** statement is $\text{wp}(\mathbf{return}, q) = \mathbf{R}q$.
– The weakest precondition for a **break** statement is $\text{wp}(\mathbf{break}, q) = \mathbf{B}q$. Etc.
– The weakest precondition $\text{wp}(\mathbf{do}\ P, \mathbf{N}q)$ for a **do** loop that ends 'normally' is $\mathbf{wp}(P, \mathbf{B}q) \vee \mathbf{wp}(P, \mathbf{N}\mathbf{wp}(P, \mathbf{B}q)) \vee \mathbf{wp}(P, \mathbf{N}\mathbf{wp}(P, \mathbf{N}\mathbf{wp}(P, \mathbf{B}q))) \vee \ldots$. 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 $\text{wp}(\mathbf{do}\ P, \mathbf{N}q)$ 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 $\text{wp}(p{\rightarrow}P, q)$ is $p{\rightarrow}\text{wp}(P, q)$, as in Hoare logic; and the weakest precondition for a disjunction $\text{wp}(P \mathrel{\triangleleft} Q, q)$ is $\text{wp}(P, q) \wedge \text{wp}(Q, q)$, as in Hoare logic. However, we only use the deterministic combination $p{\rightarrow}P \mathrel{\triangleleft} \neg p{\rightarrow}Q$ for which the weakest precondition is $(p{\rightarrow}\text{wp}(P, q)) \wedge (\neg p{\rightarrow}\text{wp}(Q, q))$, i.e. $p \wedge \text{wp}(P, q) \vee \neg p \wedge \text{wp}(Q, q)$.

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 $\mathbf{G}_l g_l \rhd \{p\}\ P\ \{q\}$ is made against. The weakest precondition p on P for q is then $p = \text{wp}_g(P, q)$, with the g_l as extra parameters. The weakest precondition for a label use $\text{wp}_g(P : l, q)$ is then $\text{wp}_g(P, q)$, provided that $g_l{\rightarrow}q$, since the states g_l attained by $\mathbf{goto}\ l$ 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 $g_l{\rightarrow}q$ holds is $\text{wp}_g(P, q \wedge (g_l{\rightarrow}q))$, which is $\text{wp}_g(P, q)$.

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 every state $s = s_0$ satisfying p leads through P with $s_0 \overset{\iota}{\mapsto} s_1$ satisfying q, and also q must contain all the transitions $s_0 \overset{\mathbf{N}}{\mapsto} s_1$ where s_1 satisfies g_l. Thus s satisfies $\text{wp}_g(P, q)$ and $\mathbf{N}g_l{\rightarrow}q$ holds. Since s is arbitrary in p, so $p{\rightarrow}\text{wp}_g(P, q)$ holds and by induction, $\vdash \mathbf{G}_l g_l \rhd \{p\}\ P\ \{q\}$. Then, by the 'frm' rule of NRB (Table 1), we may deduce $\vdash \mathbf{G}_l g_l \rhd \{p\}\ P : l\ \{q\}$.*

Case label declaration. *The weakest precondition for a declaration $\text{wp}_g(\mathbf{label}\, l.P, q)$ is simply $p = \text{wp}_{g'}(P, q)$, 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 $\mathbf{goto}\ l$ statements in the code P, given that the initial state satisfies p and assumptions $\mathbf{G}_l g_l$ hold. The g_ly are the statements that after exactly some $n \in \mathbb{N}$ more traversals through P via $\mathbf{goto}\ l$, the trace from state s will avoid another $\mathbf{goto}\ l$ for the first time and exit P normally or via an abnormal exit that is not a $\mathbf{goto}\ l$.*

If it is the case that $[\![\{p\}\ \mathbf{label}\ l.P\ \{q\}]\!]_g$ *holds then every state* $s = s_0$ *satisfying* p *leads through* $\mathbf{label}\ l.P$ *with* $s_0 \overset{\iota}{\mapsto} s_1$ *satisfying* q*. That means that* $s_0 \overset{\iota}{\mapsto} s_1$ *leads through* P*, but it is not all that do; there are extra 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* $\mathbf{goto}\ l$ *initiates some exactly* n *traversals back through* P *again before exiting* P *for a first time other than via a* $\mathbf{goto}\ l$*. 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 that* 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* $p \rightarrow wp_{g \cup \{l \mapsto g_l\}}(P, q \vee \mathbf{G}_l g_l)$ *and by induction* $\vdash \mathbf{G}_l \triangleright \{p\}\ P\ \{q \vee \mathbf{G}_l g_l\}$*, and then one may deduce* $\vdash \{p\}\ \mathbf{label}\ l.P\ \{q\}$ *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 ($P : l$, $\mathbf{label}\ l.P$). The proof-theoretic content of the discussion is:

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

We do not know if the result holds for non-deterministic programs too, but it seems probable. A different proof technique would be needed (likely showing that attempting to construct a proof backwards either succeeds or yields a counter-model).

Along with that we note

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

We have proven the NRB logic sound with respect to a simple transition-based model of programs, and showed that it is complete for deterministic programs.

References

1. American National Standards Institute. American national standard for information systems – programming langu age C, ANSI X3.159-1989, 1989.
2. Krzysztof R. Apt. Ten years of Hoare's logic: A survey: Part I. *ACM Trans. Program. Lang. Syst.*, 3(4):431–483, October 1981.
3. Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. A few billion lines of code later: using static analysis to find bugs in the real world. *Commun. ACM*, 53(2):66–75, February 2010.

4. Peter Breuer and Simon Pickin. Checking for deadlock, double-free and other abuses in the linux kernel source code. In *Proc. Computational Science – ICCS 2006*, number 3994 in LNCS, pages 765–772. Springer, May 2006.
5. Peter T Breuer and Marisol Garcia Valls. Static deadlock detection in the linux kernel. In *Proc. Reliable Software Technologies/Ada-Europe 2004*, number 3063 in LNCS, pages 52–64. Springer Berlin/Heidelberg, June 2004.
6. Peter T Breuer and Simon Pickin. Symbolic approximation: an approach to verification in the large. *Innovations in Systems and Software Engineering*, 2(3):147–163, 2006.
7. Peter T Breuer and Simon Pickin. Verification in the large via symbolic approximation. In *Proc. 2nd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, 2006 (ISoLA 2006)*, pages 408–415. IEEE, 2006.
8. Peter T Breuer and Simon Pickin. Open source verification in an anonymous volunteer network. *Science of Computer Programming*, 2013. To appear.
9. Peter T Breuer, Simon Pickin, and Maria Larrondo Petrie. Detecting deadlock, double-free and other abuses in a million lines of linux kernel source. In *Proc. 30th Annual Software Engineering Workshop 2006 (SEW'06)*, pages 223–233. IEEE/NASA, 2006.
10. E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using tempora l logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–253, 1986.
11. D. Engler, B. Chelf, A. Chou, and S. Hallem. Checking system rules using system-specific, programmer-written compiler extensions. In *Proc. 4th Symposium on Operating System Design and Implementati on (OSDI 2000)*, pages 1–16, October 2000.
12. David Harel, Jerzy Tiuryn, and Dexter Kozen. *Dynamic Logic*. MIT Press, Cambridge, MA, USA, 2000.
13. International Standards Organisation. ISO/IEC 9899-1999, programming languages - C, 1999.