Enforcing nondeterminism via linear time temporal logic specifications

niversiteit

Faculteit der Wiskunde
en Informatica

Enforcing Nondeterminism via Linear Time Temporal Logic Specifications

Ruurd Kuiper

March 1987

Enforcing Nondeterminism
via
Linear Time Temporal Logic Specifications

Ruurd Kuiper

March 1987

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science of Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

editor: F.A.J. van Neerven

Enforcing Nondeterminism via Linear Time Temporal Logic Specifications

Ruurd Kuiper

Department of Mathematics and Computing Science,
Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

March 1987

ABSTRACT

It is shown how some amount of nondeterminism can be enforced via linear time temporal logic. This is achieved through extending the notion of specification rather than changing the logic, i.e., no recourse is taken to branching time. The treatment is compared, both in intent and with respect to realization to a similar approach using predicate transformers.

1. Introduction

A specification describes requirements which further developments or implementations must fulfill in order to satisfy it. Usually, many decisions are deliberately left open to be filled in at later stages. Consequently, specifications usually contain nondeterminism which will, perhaps only in part, be resolved later.

For example, if production of either of the actions a,b,c or d will satisfy the user, a component S might, for the moment using without further explanation an intuitively obvious notation, be specified by

$$S \ \ sat \ \ a \vee b \vee c \vee d.$$

The customary interpretation of such a specification is to allow S to be implemented by any process of which the output is in the set $\{a,b,c,d\}$. For instance, by a process $\underline{a}$, which always produces a when activated, but also by $\underline{a \vee c}$, which produces either an a or a c upon different activations.

This kind of nondeterminism, say *allowed* nondeterminism, is not required of the implementation at all and only leaves some freedom to the implementor due to, deliberate, vagueness in the specification.

A completely different kind of nondeterminism, say *required* nondeterminism is the nondeterminism which the implementation should possess.

For example, a random number generator should not always generate the same number when activated. Yet a specification like

$$S \;\; sat \;\; x := x'(x' \in \mathbb{N}),$$

interpreted similarly as above as containing allowed nondeterminism, does not guarantee this. An implementation which always assigns, say, 5 to x would perfectly satisfy this specification.

Usually, specification methods make use of the first kind of nondeterminism to allow general specifications, but cannot handle the second kind. Branching time temporal logic, which describes behaviour as sets of trees is one of the few exceptions. Linear time temporal logic, describing behaviour as sets of sequences does, in its usual form, not have this expressive ability. There are, however, many different considerations which at present leave the debate as to which of the two is the most suitable, wide open.

We will present and discuss a way to enable in the context of linear time temporal logic specification of a modest amount of required nondeterminism. The idea is to limit the extent to which the allowed nondeterminism may be resolved, by additionally specifying a lower bound. This enforces implementations to possess a degree of nondeterminism between the bounds set by the required and the allowed nondeterminism.

For the above examples such lower bounds might be, respectively, $a \vee c$ and $x := x'(x' \in \{1,...,100\})$.

In section 2 we briefly discuss the (only) approach similar to ours we know of, namely [Fr77]. This is carried out in the context of predicate transformers and safety properties, but it will be seen that a more general idea underlies his approach. In section 3 we show how this can be used for linear time temporal logic specifications. The interaction with development is discussed in the next section. In section 5 a brief look is taken at the situation for branching time temporal logic. The last section contains some discussion.

2. A precursor: required nondeterminism and predicate transformers

In [Fr77], Francez addresses specifying required nondeterminism using predicate transformers. We look at the example given above, $S \; sat \; a \vee b \vee c \vee d$ with the extra aim to specify some required nondeterminism.

Let the specification of S be given as $\{\phi\}\, S\, \{\psi\}$.
In the usual weakest precondition approach, only considering allowed nondeterminism, this

means that S has to satisfy

(i) $\phi \Rightarrow wp(S,\psi)$ where, in this example,

$$\phi = \text{true}$$

$$\psi = a \vee b \vee c \vee d.$$

This only gives an upper bound to the allowed nondeterministic behaviour of S and allows implementations like, e.g., $S = \underline{b}$.

The idea in [Fr77] now is, to enforce ψ as a lower bound on required nondeterminism as well, again using weakest preconditions. The extra part of the satisfaction notion is, that S should also satisfy

(ii) $\forall \psi^* \neq \psi[(\psi^* \Rightarrow \psi) \Rightarrow \neg(\phi \Rightarrow wp(S,\psi^*))]$,

where again in this example

$$\phi = \text{true}$$

$$\psi = a \vee b \vee c \vee d.$$

It can be easily seen, that together these requirements limit the implementations to $\underline{a \vee b \vee c \vee d}$ only.

In this example, lower and upper bound coincide. The words lower and upper suggest, although [Fr77] does not claim this, noncoinciding bounds, allowing a range of implementations in between them. This might, for instance, be denoted by

$\{\phi\}\, S\, \{\overline{\psi},\underline{\psi}\}$, where $\overline{\psi}$ is the upper

and $\underline{\psi}$ the lower bound.

Intuitively, expressed in terms of an obvious semantics of i/o pairs, the lower /upper bound approach, in our view, aims at achieving the following kind of constraints.

Let $<i,a>$ denote: on any input, produce a. Take as lower and upper bound requirements respectively

$$\overline{\psi} = a \vee b \vee c \vee d$$

$$\underline{\psi} = a \vee c.$$

Then the desired constraint on S would be

$$\{<i,a>,<i,c>\} \subseteq [\![S]\!] \subseteq \{<i,a>,<i,b>,<i,c>,<i,d>\},$$

i.e., allowing the implementations $\underline{a \vee c}$, $\underline{a \vee b \vee c}$, $\underline{a \vee c \vee d}$ and $\underline{a \vee b \vee c \vee d}$.

Unfortunately, using (ii) with $\underline{\psi}$ as ψ does not give the desired result. Namely (ii) now is of the form

$$\forall \underline{\psi}^* \neq \underline{\psi}[(\underline{\psi}^* \Rightarrow a \vee c) \Rightarrow \neg (\text{true} \Rightarrow wp(S, \underline{\psi}^*))].$$

Consider the implementation $S = \underline{b}$. As S produces only b, S does not satisfy $wp(S, a \vee c)$, which will remain the case if $a \vee c$ is strengthened. So S is, contrary to the intuition, allowed as an implementation of $\{\phi\}\, S\, \{\overline{\psi}, \underline{\psi}\}$. Hence, the approach in [Fr77] is limited to coinciding lower and upper bounds.

In the next section, the lower/upper bound approach will be adapted to linear time temporal logic specifications and extended to enable the use of lower and upper bounds that do not coincide.

3. Enforcing required nondeterminism in linear time temporal logic

In linear time temporal logic (LTL) we take both the specification, ψ, and the semantics, $[\![S]\!]$, of an implementation S to be an LTL formula. Such a formula in turn can be interpreted as characterizing a set of (state) sequences, namely those for which it is true.

The customary satisfaction relation when considering only allowed nondeterminism is then straightforward:

$$S \; sat \; \psi \triangleq [\![S]\!] \Rightarrow \psi.$$

Intuitively this means that the set of sequences that can be generated by S is included in the set allowed by ϕ. It is clear that any less nondeterministic implementation S', meaning that the set of sequences it can generate is smaller, which in turn means that $[\![S']\!] \Rightarrow [\![S]\!]$, satisfies ψ as well. So the implication makes it impossible to specify required fairness. Establishing a lower bound is the solution and, in the LTL framework, can be easily incorporated in a manner reflecting the intuitive set inclusion as mentioned in the previous section.

Define

$$S \; sat \; \langle \underline{\psi}, \overline{\psi} \rangle \triangleq \underline{\psi} \Rightarrow [\![S]\!] \Rightarrow \overline{\psi}.$$

The specification of the example, in the formal notation as used in [BKP84], i.e. assuming sequences to have labels indicating environment (E) steps and component (Π) steps, then becomes:

$$S \; sat \; \langle \underline{\psi}, \overline{\psi} \rangle,$$

where

$$\underline{\psi} = E \; U \; (\Pi \wedge (a \vee c)) \; \mathbf{C} \; fin,$$

(which informally states:
starting with environment steps E,
eventually a component step occurs which produces a or c,

after which the component stops.)

and

$\overline{\psi} = E\,U\,(\Pi \wedge (a \vee b \vee c \vee d))\,C\,fin.$

Remarks

(i) An alternative way to enable specifying required nondeterminism may seem to change the implication to equivalence (this, in fact, is the situation in [Fr77]):

$$P \; sat \; \psi \triangleq [\![P]\!] = \psi.$$

This indeed fulfills the aim, but does not possess the lower and upper bound flexibility. Consequently, extra allowed nondeterminism can now only be obtained by explicitly listing the allowed alternatives, e.g., via exclusive or notation:

$$S \; sat \; \psi_1 \oplus \psi_2 \oplus \cdots \oplus \psi_n \triangleq$$

$$S \; sat \; \psi_1 \oplus S \; sat \; \psi_2 \oplus \cdots \oplus S \; sat \; \psi_n.$$

This is unfortunate, as usually when giving a specification one only has a rough idea about what one wants to allow, but certainly not a full grasp of all possible alternatives. Furthermore, if infinitely many alternatives for implementation exist, as in the case of the random number generator example, it is not possible to list all of these unless infinite $\oplus$ is allowed. In that case, although the first objection remains, both extensions are equivalent.

(ii) In, e.g., [Pn85] a strong notion of expressivity is defined for specification methods: A method is expressive $\triangleq$ for all S there is a characteristic specification, $spec_c$ such that:

(i) For all S', S' sat $spec_c \Leftrightarrow ([\![S]\!] = [\![S']\!])$

(ii) For all $spec$, S sat $spec \Leftrightarrow (spec_c \Rightarrow spec)$

This property usually does not hold; it is obtained for [BKP84] when extended as above.

4. Development

One part of development is concerned with decomposition into subspecifications. The extension of the notion of specification is such, that adapting of this part of existing methods is straightforward.

For instance, a compositional specification method dealing with required nondeterminism can be obtained by using an existing one like described in [BKP84] and just redefining the notion of specification as above and adapting the proof rules as follows.

For the decomposition part, the essential rules are those concerned with syntactical combinators, e.g., sequential and parallel composition, enabling to derive properties of components from properties of their syntactic subcomponents. These rules reflect the semantics of such operators and are of the form

$$\frac{\begin{array}{c} S_1 \; sat \; \Psi_1 \\ S_2 \; sat \; \Psi_2 \end{array}}{C(S_1,S_2) \; sat \; C'(\Psi_1,\Psi_2)}$$

where C is a syntactical combinator on components and C' the corresponding syntactical combinator on specifications.

The translation then is

$$\frac{\begin{array}{c} S_1 \; sat \; \langle \Psi_1, \overline{\Psi}_1 \rangle \\ S_2 \; sat \; \langle \Psi_2, \overline{\Psi}_2 \rangle \end{array}}{C(S_1,S_2) \; sat \; \langle C'(\Psi_1,\Psi_2), C'(\overline{\Psi}_1,\overline{\Psi}_2) \rangle}$$

A concrete example, for sequential composition, using the temporal logic operator **C** (chop) is

$$\frac{\begin{array}{c} S_1 \; sat \; \langle \Psi_1, \overline{\Psi}_1 \rangle \\ S_2 \; sat \; \langle \Psi_2, \overline{\Psi}_2 \rangle \end{array}}{S_1;S_2 \; sat \; \langle \Psi_1 \, \mathbf{C} \, \Psi_2, \overline{\Psi}_1 \, \mathbf{C} \, \overline{\Psi}_2 \rangle}.$$

Another part of development is concerned with extending the requirements on the behaviour. In the context of LTL this intuitively means further narrowing down the sets of sequences allowed by the specification. In the $\Psi \Rightarrow [\![S]\!] \Rightarrow \overline{\Psi}$ framework, this amounts to weakening (!) Ψ and strengthening $\overline{\Psi}$. This gives rise to the following rule:

$$\frac{\begin{array}{c} S \; sat \; \langle \phi, \overline{\phi} \rangle \\ \Psi \Rightarrow \phi \\ \overline{\phi} \Rightarrow \overline{\Psi} \end{array}}{S \; sat \; \langle \Psi, \overline{\Psi} \rangle}$$

Again turning to the previously used example, this means that it can be derived that from

$$S \; sat \; \langle a \vee c \vee d, a \vee c \vee d \rangle$$

it follows that

S *sat* $\langle a \vee c, a \vee b \vee c \vee d \rangle$

This corresponds to the intuition, as the first specification only allows the implementation $S = \underline{a \vee c \vee d}$. This is, as has been seen previously, one of the various implementations allowed by the second specification.

Remark

There is a rather subtle problem in the treatment of required nondeterminism in development. Of variables about which at a certain stage in the development nothing has yet been decided, usually nothing is required, i.e., all sequences are allowed as regards their values.

However, if nothing is required in Ψ about such a variable, this should remain so during further development, because, as seen from the rules, Ψ may only be weakened. Intuitively, as seen from the example, if straightforward strengthening of already mentioned variables is involved, there is no problem, because required nondeterminism for this variable was explicitly stated.

For the decomposition case there is a problem, as one would like, but cannot, formulate that for as yet unused variables no lower bound is yet established. A possible solution for this case is to argue that a decomposition step causes a lower level of abstraction to be used. New variables added to the interface can be viewed as visible only to the subcomponents.
Requirements, especially required nondeterminism, pertaining to these variables can then also be seen as limited to this level only.

The problem then disappears, as Ψ on a higher level of specification cannot impose requirements on these variables. This approach may be formalized by introducing an explicit interface for each level of specification. (See, e.g., [BK83].)

5. Branching time temporal logic

In branching time temporal logic (BTL), the formulae are interpreted not as characterizing sets of sequences, but sets of trees.
It is then obvious, that because sets of such trees are involved, a completely analogous treatment as for the LTL case is, in principle, possible. Whether this is desirable depends on one's view about which objects are more natural as behaviour of programs in certain circumstances.

Consider, for example, required nondeterminism, say $a \vee b$. If one feels, that only a set containing at least a sequence with a and one with b on it is a correct representation of this requirement, then a similar extension as to LTL is needed for BTL. The reason is, that although sequences can be viewed as trees, when required nondeterminism is imposed via sets of these, the same problems with resolving allowed nondeterminism too far as in LTL apply to BTL. If however, one allows this to be expressed via the requirement that each tree has at

least a branch with a and one with b on it, standard BTL is expressive enough already.

As yet, apart from many other arguments about which of these basic varieties is the most suitable (or when), about this particular choice there seems to be no consensus. For more information on BTL, see, e.g., [EL85].

6. Discussion

We presented a way to enforce some amount of required nondeterminism via LTL specifications. It is sometimes argued that specifying required nondeterminism is meaningless, as no test will be able to falsify a claim like, e.g., $\psi = a \vee b$. The idea is, that even after repeated testing with consistently result a, b might still occur at some future test.
One remark here is, that exactly the same argumentation applies to fairness requirements like: eventually b will occur. This concept however now seems quite well accepted.

More direct counter arguments are the following:

(i) When designing a system, it is natural that initially some properties are underdefined. During development these may be strengthened to falsifiable ones, which is certainly the only way in which they can be implemented.

(ii) An implementation will come together with a proof that its specification is met, so testing is not required.

A fortunate consequence of the fact that the extension made to the notion of specification retains the interpretation as a pure LTL formula and does not alter the logic is, that existing decision procedures (see, e.g., [Go83]) can still be used.

An open problem is, whether existing devices that contain nondeterminism, like random number generators, will satisfy abstract specifications of this property. Furthermore, if this is the case, how can this be proven? The link between the formulation of the practical and the theoretical properties seems not obvious.

Acknowledgements

Many thanks to Ron Koymans and Rob Gerth for comments and help at various stages and especially to Willem-Paul de Roever, who provided the link to reference [Fr77].
I am very grateful to Edmé van Thiel for Elastic Time Typing.

References

[EL85] Emerson, E.A., Chin-Laung Lei, *Modalities for Model Checking: Branching Time Logic Strikes Back*, POPL 1985.

[Fr77] Francez, N., *A Case for a Forward Predicate Transformer*, Inf. Proc. Letters IEEE 6:6, 1977.

[Go83] Gough, G.D., M.Sc. Thesis, *Decision Procedures for Temporal Logic*, Univ. of Manchester.

[BK83] Barringer, H., Kuiper, R., *Towards the Hierarchical, Temporal Logic, Specification of Concurrent Systems*, LNCS 207.

[BKP84] Barringer, H., Kuiper, R., Pnueli, A., *Now You May Compose Temporal Logic Specifications*, STOC 1984.

[Pn85] Pnueli, A., *Linear and Branching Structures in the Semantics and Logics of Reactive Systems*, ICALP 1985.

COMPUTING SCIENCE NOTES

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS-circuits
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films
85/04	T. Verhoeff H.M.J.L. Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate
86/01	R. Koymans	Specifying message passing and real-time systems
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specifications of information systems
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several parallel systems
86/05	Jan L.G. Dietz Kees M. van Hee	A framework for the conceptual modeling of discrete dynamic systems
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers

86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987)
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86)
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)
86/14	R. Koymans	Specifying passing systems requires extending temporal logic
87/01	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems
87/04	T.Verhoeff	Delay-insensitive codes - An overview

87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.

Available Reports from the Theoretical Computing Science Group

	Author(s)	Title	Classification EUT	Classification DESCARTES
TIR82.1	R. Kuiper, W.P. de Roever	Fairness Assumptions for CSP in a Temporal Logic Framework		
TIR83.1	R. Koymans, J. Vytopil, W.P. de Roever	Real-Time Programming and Synchronous Message passing (2nd ACM PODC)		
TIR83.2	H. Barringer, R. Kuiper	Towards the Hierarchical, Temporal Logic, Specification of Concurrent Systems		
TIR84.1	R. Gerth, W.P. de Roever	A Proof System for Concurrent Ada Programs (SCP4)		
TIR84.2	R. Gerth	Transition Logic - how to reason about temporal properties in a compositional way (16th ACM FOCS)		
TIR84.3	H.Barringer, R. Kuiper, A. Pnueli	Now you may compose Temporal Logic Specifications (Proc. STOC84)		
TIR84.4	H. Barringer, R. Kuiper	Hierarchical Development of Concurrent Systems in a Temporal Logic Framework		
TIR85.1	W.P. de Roever	The Quest for Compositionality - a survey of assertion-based proof systems for concurrent progams, Part I: Concurrency based on shared variables (IFIP85)		
TIR85.2	O. Grünberg, N. Francez, J. Makowsky, W.P. de Roever	A proof-rule for fair termination of guarded commands (Inf.& Control 1986)		

TIR85.3	F.A. Stomp, W.P. de Roever, R. Gerth	The μ-calculus as an assertion language for fairness arguments (Inf.& Control 1987)		
TIR85.4	R. Koymans, W.P. de Roever	Examples of a Real-Time Temporal Logic Specification (LNCS207)		
TIR85.5	H. Barringer, R. Kuiper, A. Pnueli	A Compositional Approach to a CSP-like Language		
TIR86.1	R. Koymans	Specifying Message Passing and Real-Time Systems (extended abstract)	CSN86/01	
TIR86.2	J. Hooman, W.P. de Roever	The Quest goes on: A Survey of Proof Systems for Partial Correctness of CSP (LNCS227)	EUT-Report 86-WSK-01	
TIR86.3	R. Gerth, L. Shira	On Proving Communication Closedness of Distributed Layers (LNCS236)	CSN86/07	
TIR86.4	R. Koymans, R.K. Shyamasundar, W.P. de Roever, R. Gerth, S. Arun Kumar	Compositional Semantics for Real-Time Distributed Computing (Inf.&Control 1987)	CSN86/08	
TIR86.5	C. Huizing, R. Gerth, W.P. de Roever	Full Abstraction of a Real-Time Denotational Semantics for an OCCAM-like Language	CSN86/09	PE.01
TIR86.6	J. Hooman	A Compositional Proof Theory for Real-Time Distributed Message Passing	CSN86/10	TR.4-1-1(1)
TIR86.7	W.P. de Roever	Questions to Robin Milner - A Responder's Commentary (IFIP86)	CSN86/11	
TIR86.8	R. Gerth, A. Boucher	A Timed Failures Model for Extended Communicating Processes	CSN86/12	TR.4-4(1)
TIR86.9	R. Gerth, W.P. de Roever	Proving Monitors Revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4)	CSN86/13	
TIR86.10	R. Koymans	Specifying Message Passing Systems Requires Extending Temporal Logic	CSN86/14	PE.02

TIR86.11	H. Barringer, R. Kuiper, A. Pnueli	A Really Abstract Temporal Logic Semantics for Concurrency (Proc. POPL86)		
TIR87.1	R. Gerth	On the existence of sound and complete axiomatizations of the monitor concept	CSN87/01	
TIR87.2	R. Kuiper	Enforcing Nondeterminism via Linear Time Temporal Logic Specifications	CSN87/05	
TIR87.3	R. Koymans	Temporele Logica Specificatie van Message Passing en Real-Time Systemen (in Dutch)	CSN87/06	
TIR87.4	R. Koymans	Specifying Message Passing and Real-Time Systems with Real-Time Temporal Logic	CSN87/07	PE.03