Is software that is formally verified fair necessarily also genuinely fair?

A case study on NASA's Formal Requirement Elicitation Tool

Joost J. Joosten [1] Marina López Chamosa [2] Sofía Santiago Fernández [1,2]

[1]Universitat de Barcelona [2]Formal Vindications S.L.

Funded by ICREA Acadèmia, Section Humanities, ICREA; Industrial doctorate program, 2022 DI 051, Generalitat de Catalunya, Departament d'Empresa i Coneixement; PID2023-149556NB-I00, Dynamics of Gödel Incompleteness (DoGI), Spanish Ministry of Science and Innovation; PID2023-151396OB-I00, Enhancing Administrative Decisions through the Use of AI, Experimentation, and Sandboxes: Special Focus on Urban and Housing Context (ADMAIES), Spanish Ministry of Science and Innovation; Grup de Lògica, Generalitat de Catalunya, Suport a grups de recerca consolidats (SGR), 2021 SGR 00348

Error-Free Software and the Role of Structured Natural Language

Formal methods can provably eliminate software bugs and errors, **a prerequisite for fair software**, and *error-free software* is only as reliable as its **formal specification**.

To ensure broader accessibility and structural clarity to the formal specification, **Structured Natural Language (SNL)** is often introduced for two key reasons:

(1) **Improved understandability** of the specification
(2) **Clear high-level structure** that guides formalization

Structured Natural Languages (SNLs) combine **formal precision** with **natural readability** by strictly constraining their **grammar and vocabulary**, which allows a **one-to-one correspondence** with formal logics. A key concern when designing SNLs is ensuring clarity and correctness in translation to formal logic. In particular, it is important to recognize that:

- Applying sound programming and formalization principles can greatly simplify translation algorithms from SNLs to formal languages;
- Even minor design decisions during the creation of an SNL can introduce **counter-intuitive behaviors** that complicate interpretation and verification.

Small design choices can give way to a situation where we have **mathematical proof of fairness properties** of the software while the behaviour of the software **does not really comply** with our **informal understanding of fairness**.

A case study: Formal Requirement Elicitation Tool (FRET)

FRET is a NASA-developed tool that helps engineers write formal requirements in a structured yet readable natural language format. It bridges the gap between human-friendly specifications and the formal logic required for automated analysis like model checking and runtime monitoring.

FRET relies on a domain-specific language called **FRETISH**, designed to express temporal requirements in a structured, precise, and human-readable way. FRETISH requirements follow the format:

Scope Conditions Component* Shall* Timing Responses*

Mandatory Fields		Optional Fields	
Component	system involved	**Timing**	when *response* should occur
Shall	obligation keyword	**Conditions**	applicability context: *void* (unconditional), *trigger* (activates on event), *continual* (holds while true)
Response	expected boolean behavior	**Scope**	temporal intervals of enforcement built from a *mode* (boolean) and an operator

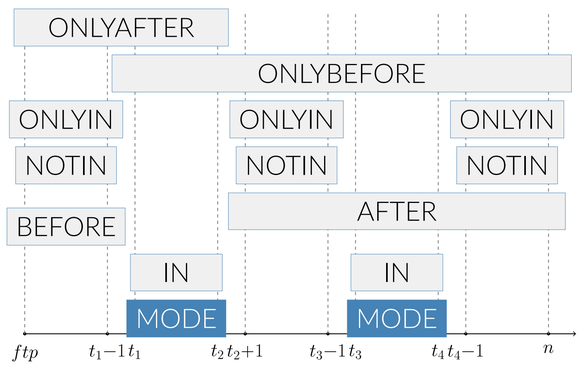

Figure 1. Semantic behavior of Scope kinds

From FRETISH to Logic

FRETISH can be translated to **Metric Temporal Logic** (MTL), which extends classical temporal logic by allowing quantitative timing constraints. The key temporal operators of past-time and future-time MTL are

Past-Time Operators		Future-Time Operators	
$\mathcal{Y}$	Yesterday (true in the previous timepoint)	$\mathcal{X}$	Next (true in the following timepoint)
$\mathcal{O}$	Once (sometime in the past)	$\mathcal{F}$	Finally (eventually)
$\mathcal{H}$	Historically (always in the past)	$\mathcal{G}$	Globally (always in the future)
$\mathcal{S}$	Since (true since past event)	$\mathcal{U}$	Until (true until event)

FRET provides three formal translations: two into Future-time MTL (for finite and infinite models) and one into Past-time MTL. To handle finite models, it introduces a special proposition called **LAST**. Translations are output in SMV or Lustre, easing **integration into model-checking** pipelines.

What we found

While FRET automates the translation of FRETISH requirements into formal temporal logic, the underlying process has long been **opaque** and sometimes **complex**.

- FRET translations tend to be unexpectedly **lengthy** and exhibit substantial **logical depth**.

FRETISH: In Scope upon Condition Component shall before StopCondition satisfy Response
FRET's MTL: (((G ((! (! Scope) & (X Scope))) | (X (((Scope & (X (! Scope))) V (((! Condition) & ((X Condition) & (! (Scope & (X (! Scope)))))) -> ((X (! (((! (! StopCondition) & (Response | (Scope & (X (! Scope)))))) & (! (Scope & (X (! Scope))))) U StopCondition))) & (! (Scope & (X (! Scope)))))) & (Condition -> (! (((! (! StopCondition) & (Response | (Scope & (X (! Scope)))))) & (! (Scope & (X (! Scope))))) U StopCondition)))))) & (Scope -> (((Scope & (X (! Scope))) V (((! Condition) & ((X Condition) & (! (Scope & (X (! Scope)))))) -> ((X (! (((! (! StopCondition) & (Response | (Scope & (X (! Scope)))))) & (! (Scope & (X (! Scope))))) U StopCondition))) & (! (Scope & (X (! Scope)))))) & (Condition -> (! (((! (! StopCondition) & (Response | (Scope & (X (! Scope)))))) & (! (Scope & (X (! Scope))))) U StopCondition)))))

Figure 2. A FRET translation.

- **Counter-intuitive constructs** whose informal interpretation diverges from their actual formal semantics, often leading to misunderstandings:
  - TheParcel shall within 1 day satisfy BeDelivered ⇏ TheParcel shall eventually satisfy BeDelivered
  - **'Only In' Scopes defy logic**: Their semantics aren't logically grounded, but come from engineering intuition.
  - TheDriver shall after 3 hours of driving satisfy Rest forces that the rest cannot occur before 3 hours of driving.

Why this matters: Temporal phrases may not mean what users expect. The logic proves correctness only for an **unfair interpretation**.

Our approach towards fairness

We propose a **fair** and **systematic** translation of FRETISH requirements into MTL, based on **general formula patterns** applied to the translation of each requirement element. The translation formulas account for key factors such as **condition kinds** (trigger vs. continual) and **temporal contexts** (finite vs. infinite traces).

Ex: MTL Formulas Translating FRETISH for Future Time on Infinite Traces

$$\mathcal{G}\left(\llbracket\text{Mode}\rrbracket \wedge \text{Cond} \rightarrow \mathcal{T}_\infty(\text{Resp}, \llbracket\text{Mode}\rrbracket)\right) \quad \text{if Cond is continual}$$

$$\mathcal{G}\left(\mathbf{ChangeTo}\left(\llbracket\text{Mode}\rrbracket \wedge \text{Cond}\right) \rightarrow \mathcal{T}_\infty(\text{Resp}, \llbracket\text{Mode}\rrbracket)\right) \quad \text{if Cond is trigger}$$

$\mathbf{ChangeTo}(\cdot)$	Activates when expression changes from false to true to capture the trigger dynamics
$\llbracket\text{Mode}\rrbracket$	MTL translation of the Scope
$\mathcal{T}_\infty$	MTL translation of the Timing constraint

As a distinguishing feature, we combine both past-time and future-time operators in the MTL translations. This approach yields a **more natural semantic interpretation** and reduces the logical complexity of the resulting formulas.

Methodology and enumeration of semantical templates

Each FRETISH requirement is characterized by a triple of optional fields ⟨Scope, Cond, Timing⟩, known as a **semantic template**, which captures structural variations in the requirement. Therefore, FRETISH requirements can be abstracted into 240* such triples. * parametrized by duration d (see Table 3).

Lexical ID	Semantic Template	FRETISH Requirement
⟨1, 1, 1⟩	⟨In, Continual, Immediately⟩	In Mode Whenever Cond Component shall Immediately satisfy Resp

Figure 3. A semantic template

The semantic templates are **systematically arranged** in lexicographic order, enabling the **individual handling** of each element before integrating them into the general translation formula for the requirement. For each semantic template component, lookup tables provide its corresponding MTL translation.

Ordinal	Scope operator	LTL definition
0	Global Mode	⊤
1	In Mode	Mode
2	Not In Mode	¬ Mode
3	Only In Mode	¬ Mode
4	Before Mode	$\mathcal{H}$ ¬ Mode
5	Only Before Mode	$\mathcal{O}$ Mode
6	After Mode	$\mathcal{O}$ (¬Mode ∧ $\mathcal{Y}$ Mode)
7	Only After Mode	$\mathcal{H}$ ($\mathcal{Y}$ Mode → Mode)

(a) Look-up table for FRETISH Scope

ID	Condition kind
0	void
1	continual
2	trigger

(b) Look-up table for FRETISH Condition

ID	FRETISH Timing	MTL formula
0	eventually	$\llbracket\text{Mode}\rrbracket\ \mathcal{U}\ \left(\text{Resp} \wedge \llbracket\text{Mode}\rrbracket\right)$
1	immediately	Resp
2	next	$\mathcal{X}$ Resp ∨ $\mathcal{X}$ **ChangeTo** ¬$\llbracket$Mode$\rrbracket$
3	always	($\mathcal{X}$ **ChangeTo** ¬$\llbracket$Mode$\rrbracket$) $\mathcal{R}$ Resp
4	never	($\mathcal{X}$ **ChangeTo** ¬$\llbracket$Mode$\rrbracket$) $\mathcal{R}$ ¬Resp
5	within (d)	$\Diamond_{[0,d]}$Resp ∨ $\left(\llbracket\text{Mode}\rrbracket\ \mathcal{U}_{[0,d]} \neg\llbracket\text{Mode}\rrbracket\right)$
6	for (d)	$\Box_{[0,d]}$Resp ∨ $\left(\left(\text{Resp} \wedge \llbracket\text{Mode}\rrbracket\right)\ \mathcal{U}_{[0,d]} \neg\llbracket\text{Mode}\rrbracket\right)$
7	after(d)	$\left(\Box_{[0,d]}\neg\text{Resp} \wedge \Diamond_{[d+1,d+1]}\text{Resp}\right) \vee \left((\neg\text{Resp} \wedge \llbracket\text{Mode}\rrbracket)\ \mathcal{U}_{[0,d+1]} \neg\llbracket\text{Mode}\rrbracket\right)$
8	until (stopCond)	$\Box$Resp ∨ $\left(\text{Resp}\ \mathcal{U}\ \left(\text{stopCond} \vee \mathbf{StrictChangeTo}\left(\neg\llbracket\text{Mode}\rrbracket\right)\right)\right)$
9	before (stopCond)	$\left(\text{Resp}\ \mathcal{R}\ \neg\text{stopCond}\right) \vee \left(\left(\llbracket\text{Mode}\rrbracket \wedge \mathcal{X}\neg\llbracket\text{Mode}\rrbracket\right)\ \mathcal{R}\ \neg\text{stopCond}\right)$

(c) Look-up table for FRETISH Timing for infinite trace when the Scope does not include "only".

Verification through model-checking

We validate the translation of each semantic template through equivalence checking in **nuXmv**, demonstrating the correctness and reliability of our approach.

Results comparison and forward-looking conclusions

- We propose a novel translation from FRETISH to MTL that is **clearer**, more **intuitive**, and **simpler** than previous methods, benefiting from the **combined use of past-time and future-time operators**.

Quantitative Comparison Summary

Metric	Simplified / FRET
Max Operators	28 / 148
Average operators	14 / 32

43.33% of FRET formulas exceed the maximum operator count observed in our simplified translations.

The resulting MTL formulas exhibit significantly reduced complexity, as it is the case for the example shown in Figure 2.

FRETISH: In Scope upon Condition Component shall before StopCondition satisfy Response
Simplified translation: G (ChangeTo (Scope & Condition) -> ((Response V ! StopCondition) | ((Scope & X ! Scope) V ! StopCondition)))

- Our methodology is **robust**, **effective**, and **practical** for real applications. Its **compositional** nature supports straightforward adaptation to modifications on the requirements.
- The simplicity of the results and the **transparency** of the methodology **minimize the risk of errors**. We **enhance fairness** by providing a translation that is closer to the user's natural expectations.
- Our approach is **well-suited for formalization** in proof assistants.

References

[1] Esther Conrad, Laura Titolo, Dimitra Giannakopoulou, Thomas Pressburger, and Aaron Dutle. "A compositional proof framework for FRETish requirements". In: *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2022, pp. 68–81.

[2] Dimitra Giannakopoulou, Thomas Pressburger, Anastasia Mavridou, and Johann Schumann. "Automated formalization of structured natural language requirements". In: *Information and Software Technology* 137 (2021), p. 106590.