Interoperability via Institutions and Unifying Theories of Programming

Rosemary Monahan, Jim Woodcock, Marie Farrell, Andrew Butterfield

IFIP WG 1.9/2.15 — 14th April 2026 — Torino

IFIP WG 1.9/2.15 July 2025

Reality: Our systems are heterogeneous—discrete, probabilistic, hybrid, and human-in-the-loop.

Problem: Tools and underlying logics don't talk to each other; calculi don't scale across them.

Thesis: Parameterising UTP theories with Institutions provides a hybrid semantic framework.

Payoff: Refinement-style calculation and logic-of-logics interoperability.

Motivation

- Introduction
  - UTP's strength in modelling and reasoning about program semantics.
  - Institutions' utility for abstracting over logics and supporting heterogeneity.
- Problem
  - UTP lacks a general theory of logic interoperability.
  - Institutions lack detailed semantics and fine-grained program reasoning.
- Proposal
  - A hybrid semantic framework: embed Institutions in UTP Theories, preserving internal structure while enabling meta-level reasoning.

Unifying Theories of Programming

- UTP: Mathematical framework introduced by Hoare and He.
- Provides single semantic foundation for diverse programming language paradigms.
- Common basis for understanding semantics of various programming notations.
- Represents all programming constructs uniformly as predicates over observations.
- Models programs as relations between states described by logical predicates.
- Axiomatic semantics, predicative programming, predicate transformers.
- All unified as special cases of a general pointwise relational calculus.
- Framework enriched with healthiness conditions.
- Constraints that determine the well-formedness of program predicates.

UTP Foundations

- UTP models program behaviours using alphabetised relations.
- Predicates over variables represent before and after states of program execution.
- Relational view enables programs to be interpreted pointwise.
- No separate denotational mapping from syntax to semantics.
- For example: program $x := x + y$ is the predicate

$$(x' = x + y)(y' = y)$$

1. Variable alphabet Program variables and auxiliary observations.
2. Signature Defining program constructs.
3. Semantic domain Interprets constructs as alphabetised predicates.
4. Healthiness conditions Identify well-formed members of the theory.

Relational and Design Theories

Relational Theory

- Simplest UTP theory: models partial correctness using relational semantics.
- Refinement $P \sqsubseteq Q$ is defined by universal implication: [Q $\Rightarrow$ P]
- Q refines P if it behaves no more nondeterministically and no less correctly.

Design theory models total correctness

- Design $(P \vdash Q)$: precondition P postcondition Q, defined by: $ok \wedge P \;\Rightarrow\; ok' \wedge Q$.
- If program starts with precond P, it must terminate satisfying postcond Q.
- Observations ok and ok' allow reasoning about failure and termination.

Designs obey some healthiness conditions

1. H1: programs cannot observe anything before starting.
2. H2: programs cannot demand nontermination.
3. H3 and H4: ensure sensible preconditions and no miracles.

The problem

- Pain point UTP developments often look logic-neutral.
- But key laws silently assume classical reasoning and total evaluation.
- There are many different logics we might want to work with.
- One key area is logics that deal with undefined expressions.
- Concrete mental image Undefined expression means a term like x/y at $y = 0$.
- Evaluation order matters Commuting $\wedge$ and reordering guards can change definedness under short-circuit logics.
- Precise objective Define UTP theories once, wrt a class of logics.
- But allow the underlying logic policy to vary (e.g., strict, Kleene, McCarthy, etc.).
- Next Institutions give us exactly the right abstraction boundary.

Overview

- We make the logic layer in UTP explicit and swappable.
- The aim is to avoid baking in classical reasoning in UTP theories.
- For example, logics with different treatments of partiality and undefinedness.
- Our technique Create logic-parametrised UTP theories.
- Roadmap We will
  1. Explain the institution parameter.
  2. Show how it threads through alphabetisation and satisfaction.
  3. Give a case study: McCarthy short-circuit logic.
  4. Extract a practical proof discipline for UTP theories.
  5. Mechanise the theory in Isabelle/HOL.
- Why should you care? We'll use one small worked example: division by zero.
- Linux kernel bug CVE-2025-68350 Divide-by-zero in exFAT filesystem code crashes the system; fixed via a patch once identified.
- Divide-by-zero semantics shows refinement genuinely changes as the logic varies.

Institutions: Only what we need

- Lightweight description Institution = logic package: $(\mathbf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models)$.
- Why should we care?

  Satisfaction condition ensures truth is preserved when we change signatures.
- UTP constantly changes alphabets: priming, adding mid-state variables, hiding.
- Truth invariance is the backbone of compositional semantics.
- The satisfaction condition is the key.
- Truth is preserved under signature morphisms.
- Satisfaction condition (for $\sigma : \Sigma \to \Sigma'$ and $\varphi \in \mathbf{Sen}(\Sigma)$):

$$M' \models_{\Sigma'} \mathbf{Sen}(\sigma)(\varphi) \iff \mathbf{Mod}(\sigma)(M') \models_{\Sigma} \varphi$$

- This is what makes alphabet-changing semantics compositional.
- Example If a predicate doesn't mention a fresh observation, extending the alphabet doesn't change the predicate's truth.

Previous work...

- Event-B formal specification language

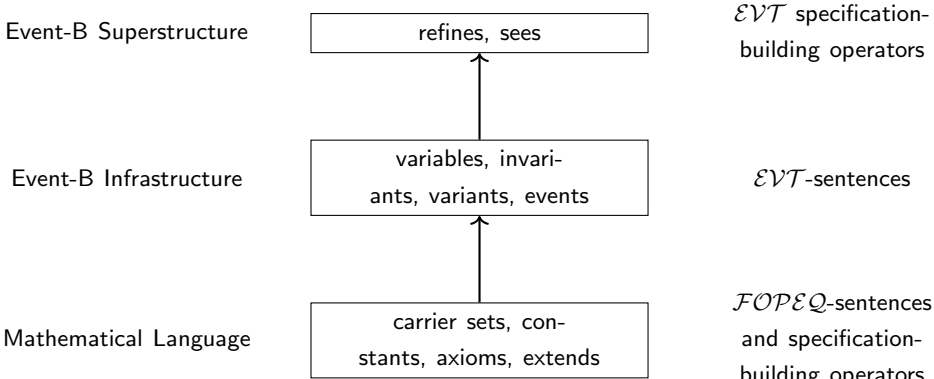
```
CONTEXT ctx
   extends ctx0
   SETS S
   CONSTANTS c
   AXIOMS
      A(s,c)
```

```
MACHINE m  refines m0
SEES ctx
  VARIABLES x
  INVARIANTS I(x)
  VARIANT n(x)
  EVENTS
    INITIALISATION, e1,...,en
```

```
Event ei ≙ status
   any p
   when G(x,p)
   with W(x,p)
   then BA(x,p,x')
end
```

Previous work...

- $\mathcal{EVT}$ - the institution for Event-B



An Event-B Institution: $\mathcal{EVT}$

Signatures: $\langle S, \Omega, \Pi, E, V \rangle$ where S is a set of sort names, Ω is a set of operation names, Π is a set of predicate names, E is a set of event-name, status pairs and V is a set of sort-indexed variable names.

Sentences: $\langle e, \phi(\overline{x}, \overline{x\prime}) \rangle$ or $\langle inv, \phi(\overline{x}, \overline{x\prime}) \rangle$ where e is an event name drawn from E, inv is an identifier for invariant sentences and $\phi(\overline{x}, \overline{x\prime})$ is an open $\mathcal{FOPEQ}$-formula over the variables in V and their primed versions.

Models: $\langle A, L, R \rangle$ where A is a $\mathcal{FOPEQ}$-model L is an initialising set of variable to value mappings corresponding to the `Init` event and R is a relation over before and after variable to value mappings.

Satisfaction: $\mathcal{FOPEQ}$ satisfaction by embedding $\mathcal{EVT}$-models to $\mathcal{FOPEQ}$ and closing the previously open $\mathcal{FOPEQ}$-formulae in the $\mathcal{EVT}$-sentences.

Our Objective

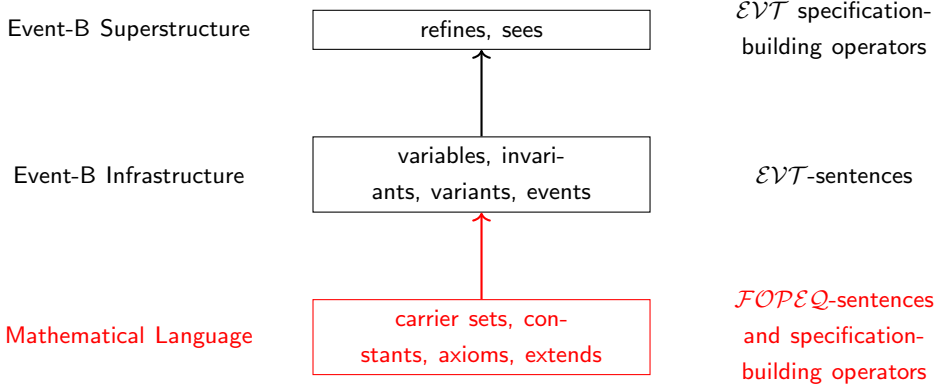
Given an Event-B machine we want to be able to express and reason about LTL properties:

$$(\neg\, GF[\texttt{selectBiscuit}]) \Rightarrow G([\texttt{selectChoc}] \Rightarrow F[\texttt{dispenseChoc}])$$

$$G(\texttt{item=2} \Rightarrow (X(\texttt{item=1})))$$

where `selectBiscuit`, `selectChoc` and `dispenseChoc` are Event-B events, and `item` is a variable in the Event-B machine.

Basic Idea: Combine $\mathcal{EVT}$ and $\mathcal{LTL}$ by replacing $\mathcal{FOPEQ}$



How can UTP help?

- Confusion Are we parametrising UTP, or parametrising institutions?

(A) Parametrise UTP by institutions

- Keep UTP theory constructors fixed (designs, reactive designs, ...)
- Vary the logic instance $\mathcal{I}$.
- Re-run the same development per $\mathcal{I}$.

(B) Parametrise institutions for UTP

- Often keep **Sig**, **Sen**, **Mod** fixed.
- Vary $\models$ satisfaction relation.
- Definedness evaluation policy.
- This is where the parameter lives.

- We use (A) as the story: $\mathcal{I} \mapsto \mathsf{UTP}(\mathcal{I})$ (theory family indexed by logic instance).
- But acknowledge (B): Operationally, the parameter lives in the satisfaction relation $\models$ (evaluation and definedness policy).
- Whenever we suppress $\mathcal{I}$, it is still there inside every connective via $\models$.

Parameterising UTP: Where does the institution go?

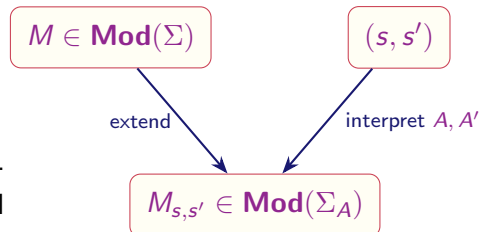
- Notational convention: We fix Σ and $\mathcal{I}$, then drop the indices to avoid clutter.
- Explicitly: $\wedge$, $\Rightarrow$, and $\exists$ are interpreted according to $\mathcal{I}$.
- Change $\mathcal{I}$ and these symbols change meaning.
- Consequence: Validity $\models_A$ and refinement $\sqsubseteq$ are really $\models_A^{\mathcal{I}}$ and $\sqsubseteq_{\mathcal{I}}$.
- Re-introduce indices only when comparing instances (e.g., $\sqsubseteq_{\mathsf{Cl}}$ vs $\sqsubseteq_{\mathsf{Mc}}$).
- Or when stating instance obligations (the logic interface).
- Pay-off UTP theory constructors (designs, reactive designs, ...) are written once.
- Swapping $\mathcal{I}$ replays theory development over different logics.
- Next Let's see how UTP's alphabetisation becomes a signature extension.

Alphabetisation as a signature extension

- Key trick: We turn free variables of UTP observations into part of the signature.
- UTP predicate: ordinary institution sentence with institution satisfaction.
- Σ: program/expression vocabulary.
- A: pre-state observations (reified as constant symbols in Σ_A).
- A': post-state observations (reified as constant symbols in Σ_A).
- A and A' are the unprimed/primed observations (pre/post): $\Sigma_A = \Sigma \uplus A \uplus A'$.
- Constants vs variables subtlety: syntactically variables, but semantically reified as constant symbols. Satisfaction needs no extra valuation parameter.
- Next Extended signature, a state-pair (s, s'), becomes a model expansion $M_{s,s'}$.

Canonical model expansion from a state-pair

- M interprets base vocabulary.
- (s, s') supplies values for observations.
- Together they yield $M_{s,s'}$ over Σ_A.
- Intuition: A state-pair is just a convenient way to produce a model for the extended signature.
- Foreshadow undefinedness: If M gives a partial interpretation of division, the expanded model inherits that behaviour.



- Satisfaction definition is a one-liner. Threads institution parameter through $\models$.
- Next Define state-pair satisfaction as ordinary satisfaction in the expanded model.

Key definition: State-pair satisfaction

- Consider UTP satisfaction of a predicate in a pair of states: $(M, s, s') \models_A P$.
- This becomes ordinary institutional satisfaction of a sentence in a single model.
- $M_{s,s'}$ is a model expansion of M from Σ to Σ_A.
- This is the semantic bridge: $(M, s, s') \models_A P$ is defined by $M_{s,s'} \models_{\Sigma_A} P$.
- Payoff UTP predicates become plain sentences.
- This preserves Hoare/He/Hehner mantra programs are predicates.
- The only place logic enters is via $\models$ in the chosen institution instance $\mathcal{I}$.
- Changing $\mathcal{I}$ changes $\models$, hence changes which laws and refinements hold.
- Next Satisfaction, validity, refinement drop out as usual in UTP, indexed by $\mathcal{I}$.

Instance-parametric validity and refinement

- UTP with indices made honest.
- Key claim Changing $\mathcal{I}$ changes $\Rightarrow$ (and often definedness), so it changes $\sqsubseteq$.
- Definitions (indexed by $\mathcal{I}$):

$$\models_A \varphi \;\widehat{=}\; \forall M \in \mathbf{Mod}(\Sigma) \bullet \forall s, s' \bullet (M, s, s') \models_A \varphi$$

$$P \sqsubseteq Q \;\widehat{=}\; \models_A (Q \Rightarrow P)$$

- Comparing instances Write $\sqsubseteq_{\mathcal{I}}$ or $\models_A^{\mathcal{I}}$. Most of the time we suppress $\mathcal{I}$.
- This is standard UTP, but now explicitly instance-parametric.
- Important Changing $\mathcal{I}$ changes the refinement order, not just which proofs work.
- Next Let's see a familiar UTP theory (designs) developed against this interface.

Parametric case study: Designs

- Designs are a testbed for these ideas.
- Canonical UTP contract theory: stress the framework without being exotic.
- Design theory Develop designs in small logic interface. Instantiate later.
- Design constructor (standard UTP shape) $P \vdash Q \;\widehat{=}\; ok \wedge P \Rightarrow ok' \wedge Q$.
  Here ok means this design has started; ok' means this design terminates.
- Sequential composition $P \,;\, Q \;\widehat{=}\; \exists\, s_0 \bullet P[s/s_0] \wedge Q[s_0/s']$. Uses only
  1. Substitution/renaming.
  2. Conjunction $\wedge$.
  3. Existential quantification $\exists$.
- Message Modular construction: define the theory once.
  Instantiate it by supplying institution and whatever algebraic laws it satisfies.
- Next What must logic instance provide for usual calculation rules to work?

What an instance must provide

- Instance contract Minimal set of laws required to replay UTP calculations.
- Three main ingredients:
  1. Substitution/renaming.
  2. $\exists$-calculus.
  3. Propositional principles appropriate to instance.
- Methodology Separate UTP theory from proof obligations about modularity.
- Natural fit for Isabelle locales and type classes: replay proofs per instance.
- Next Instantiate with order-sensitive operators: McCarthy's left-sequential logic.

  Objective See if there's a genuine semantic difference.

Instantiation: McCarthy (left-sequential) logic

- McCarthy-ism: Sequential, left to right logic.
- Formalises programmer-style short-circuiting; evaluation order becomes semantic.
- Truth values $\{T, F, U\}$ where U means undefined.
- Left-sequential conjunction:

$$a \wedge_{\mathsf{Mc}} b = \begin{cases} F & \textbf{if } a = F \\ b & \textbf{if } a = T \\ U & \textbf{if } a = U \end{cases} \qquad a \Rightarrow_{\mathsf{Mc}} b \;=\; (\neg\, a) \vee_{\mathsf{Mc}} b$$

- Evaluation order is observable

  Swapping conjuncts changes definedness, classically equivalent predicates diverge.
- Next Smallest example that changes refinement: division-by-zero.

Worked example: Division-by-zero (why refinement changes)

- Example Let division be partial: x/y undefined when $y = 0$.
- Atomic predicate and guard: $A = (x/y > 1)$ $G = (y \neq 0)$.
- Two preconditions: $P_{\text{safe}} = G \wedge_{\text{Mc}} A$ $P_{\text{unsafe}} = A \wedge_{\text{Mc}} G$.

Expression	Value at $y = 0$
G	F
A	U
$G \wedge_{\text{Mc}} A$	F
$A \wedge_{\text{Mc}} G$	U

- At $y = 0$: $P_{\text{safe}} = F$ (guard short-circuits).
- At $y = 0$: $P_{\text{unsafe}} = U$ (division evaluated first).
- Key message Classical rewrites that commute conjuncts aren't sound.
- Punchline This is not cosmetic: it changes which predicates refine which.
- Next Embedding into designs shows this isn't a toy: it affects UTP refinement.

Design consequence

- Preconditions Put the same postcondition under both: $Q = (z' = x/y)$.
- The same postcondition yields different designs depending on precondition order.
- Designs $D_{\text{safe}} = P_{\text{safe}} \vdash Q \qquad D_{\text{unsafe}} = P_{\text{unsafe}} \vdash Q$.
- At $y = 0$
  1. The safe design can be satisfied (precondition fails cleanly).
  2. The unsafe one becomes undefined purely due to expression order.
  3. False precondition Still a proper contract.
  4. Undefined precondition Design calculus becomes unreliable.
  5. That is a genuine, instance-induced refinement difference.
- Intuition When the guard fails cleanly, the design behaves as expected.
- When the precondition becomes undefined, the whole contract is poisoned.
- Take-away Parameter affects specification structure, not just proof convenience.
- Discipline for writing and manipulating predicates when evaluation order matters.
- Next Simplest discipline is to put the guard first.

Isabelle/HOL Mechanisation of Institutions for Logic-Parametrised UTP

- Mechanisation treats underlying logic of a UTP theory as a parameter.
- We don't fix a single notion of truth or definedness.
- We have a single semantic infrastructure instantiated with different logics.
- Instantiations can be compared inside one framework.
- Development connects three ingredients:
  1. Institution-theoretic structure.
  2. UTP semantic operators.
  3. Isabelle locales and interpretations.
- Aim: Reusable semantic infrastructure.
- Define generic theory once, then instantiate it many times.

Isabelle Architecture: Locales, Interpretations, and Reuse

- Core theory encoded using Isabelle's locale structure.
- Locale Named, parametrised proof context.
  - Fixes assumptions, definitions, notation for theories.
  - Abstract development and reuse before concrete instantiation.
  - Captures abstract assumptions on chosen logic and semantic operators.
- Interpretation Concrete instantiation of a locale.
  - Delivers particular UTP theories under specific logical settings.
  - Proof that concrete structure satisfies locale's assumptions.
  - Imports locale's definitions, theorems, and notation for that instance.
- Mechanisation includes infrastructure for:
  - Abstract consequence/truth structure. State and alphabet construction.
  - Generic UTP operators such as assignment and sequencing.
  - Renaming/morphism machinery for transport between signatures and theories.
- Major benefit Generic lemmas proved once in the abstract setting.
- Reuse These are then inherited by each concrete instantiation.

Mechanised Examples: Small Languages and Semantic Comparisons

- Mechanised small imperative-language examples.
- Objective Validate generic UTP operators on manageable test cases.
- TinyLang-style examples Simple relation programming language.
- One abstract UTP core instantiated repeatedly across different logical settings.
- Examples compare instantiations with different induced theory behaviours.
  - Demonstrate uniformity of the framework.
  - Expose semantic distinctions between logics being instantiated.
- Examples Strict and McCarthy logics for undefined expressions.
- Examples Relational, probabilistic, and Bayesian semantics.
- Isabelle is used in two ways:
  - To prove common laws across all (constrained) instantiations.
  - To show where particular laws fail in specific instantiations.

Why the Mechanisation Matters

- Development gives a machine-checked route:
  - From abstract institutional semantics to concrete instantiated UTP theories.
- Supports disciplined comparison of different notions of logic and definedness:
  - Within one Isabelle framework.
- Proof infrastructure scales beyond smallest examples.
- Applies to richer state-based and application-oriented case studies.
- Shows institution theory + logic-parametrised UTP + Isabelle locales provides a practical basis for modular semantic engineering.
- Build generic theory once, reuse it systematically across family of UTP theories.

Take-home messages

What we've already done

1. Institutions give a clean, compositional logic parameter for UTP.
2. The parameter is carried by $\models$, hence by $\models_A$, validity, and refinement.
3. Developed UTP theories once against a logic interface, then replayed per instance.
4. McCarthy short-circuiting is a compact case study showing real semantic impact.
5. Mechanised the theory in Isabelle/HOL: can compare families of UTP theories without rebuilding the semantic infrastructure from scratch each time.

- Next steps Larger instance library, more paradigms (reactive, probabilistic, hybrid), more logics (constructive or intuitionistic, substructural, modal or temporal, epistemic and information-flow, deontic contract, probabilistic and quantitative, hybrid and differential, ...), catalogue of laws preserved across all instances.
- Research Motivation:
  Which UTP laws do you want to preserve. What logic interface is required?