Reasoning about effectful programs: the state of the art
Perspectives

• Abstract structure of effectful programs
• Concrete models for languages with effects
• Logics and reasoning principles
• Proofs of particular programs
• Mechanization
• Automation and tool-building
• Language design and programming patterns
Arrows, premonoidal and Freyd categories

• Power, Robinson, Hughes, Atkey

• Rather than requiring a monad, just ask for identity on objects product-preserving functor from monoidal base category (values) to premonoidal category of computations

• Roughly corresponds to arrow abstraction in Haskell
  – Used for, for example, functional reactive programming

• Nice syntax due to Patterson, Wadler et al
Parameterized monads

- Atkey
- \( T: A^{\text{op}} \times A \times \text{Set} \to \text{Set} \)
- \( \eta_{aX}: X \to T(a,a,X) \)
- \( \mu_{abcXY}: T(a,b,X) \times (X \to T(b,c,Y)) \to T(a,c,Y) \)
- Subject to laws...
- Examples
  - Monads \( T(A,B,X) = MX \)
  - State transformers \( T(S_1,S_2,X) = S_1 \to S_2 \times X \)
    - Monoidal structure on parameters allows separated states too
  - Composable continuations \( T(R_1,R_2,X) = (X \to R_1) \to R_2 \)
- Further applications to, for example, permissions and session types
Algebraic effects

- Plotkin, Power, Hyland, Pretnar, Staton,...
- Focus on operations and equations
- Generate monads from them
- Composite monads from combinations of algebraic theories (sum and tensor (commutation))
- Add generic handlers (destructors) for effects to account for e.g. catching exceptions
- Pretnar and Bauer building a language on these ideas (eff)

```
a : A ⊢ let v := !a in a := v ≈ () : 1
a : A ⊢ let v := !a in let w := !a in (v, w) ≈ let v := !a in (v, v) : V × V
a : A, v, w : V ⊢ a := v; a := w ≈ a := w : 1
a : A, v : V ⊢ a := v; let w := !a in w ≈ a := v; v : V
(a, b) : A ⊗ A, v, w : V ⊢ a := v; b := w ≈ b := w; a := v : 1
(a, b) : A ⊗ A, v : V ⊢ a := v; !b ≈ let w := !b in a := v; w : V
```
Biorthogonality and TT-lifting

- Krivine, Pitts, Katsumata
- Given configurations \((p,e)\) and observation \(O \subseteq P \times E\)
  - If \(P \subseteq \text{Progs}\), \(P^T \subseteq \text{Envs} = \{e \mid \forall p \in P, (p,e) \in O\}\)
  - If \(E \subseteq \text{Envs}\), \(E^T \subseteq \text{Progs} = \{p \mid \forall e \in E, (p,e) \in O\}\)
  - \((.)^{TT}\) is a closure operator on \(2^{\text{Progs}}\) that “contextualizes” properties
- Gives general approach to lifting logical relations to monadic types (via continuation monad transformer)
- Saw this twice already (logical relation for imperative language, QPER closure in relation for effect system)
- Also used in relations for HO store, compiler correctness, realizability
Enriched effect calculus

- Simson, Egger, Mogelberg
- Generalizes Levy’s CBPV and the relationship between the monadic calculus and linear logic to the non-commutative case
- Results on linearly used continuations, parametricity, type isomorphisms etc.
- Promising metalanguage for work on effects

Value types:

\[ A, B, \ldots ::= \alpha | \alpha | 1 | A \times B | A \to B | !A \]
\[ | A \to B | !A \otimes B | 0 | A \oplus B \]

Computation types:

\[ A, B, \ldots ::= \alpha | 1 | A \times B | A \to B | !A \]
\[ | !A \otimes B | 0 | A \oplus B . \]
Separation logic

• Reynolds, O’Hearn, Bornat, Ishtiaq, Yang, Parkinson, and cast of thousands
• Extends Hoare logic to allow local reasoning about programs that manipulate data structures in the heap
  – Separating conjunction
  – Frame rule
• Revitalised theory of program verification, huge influence on theory and practice
• Tools: Smallfoot, jStar, SpaceInvader, SLAyer,…
  – Memory safety proofs for large bits of industrial code
• Extends beautifully to concurrency
  – Resources, permissions
  – Rely-guarantee reasoning
Higher-order extensions of separation logic

- Higher order functions, flat store (Birkedal, Torp-Smith, Yang)
  - Higher-order frame rules, FM-cpos, relational parametricity and TT-lifting

- Higher order functions and higher-order store (Schwinghammer, Birkedal, Reus, Yang)
  - Foundationally challenging because of, e.g. recursion through the store
  - Nested triples, Kripke semantics, worlds defined recursively using ultrametric spaces
Refining the state monad with dependent types

• Nanevski, Morrisett, Birkedal,…
• Hoare Type Theory \{P\}x:A\{Q\} where P,Q are predicates on the state, x binds return value in postcondition
• Embedded in Coq, full dependent types
• Can include separation logic assertions
• Generates VCS, powerful automation
• Cool examples: tricky imperative datastructures, web services, database management system
Relational Hoare Logic

• B, Yang. Extend Hoare logic to reason about multiple runs of programs
• Allows proofs of contextual program transformations enabled by compiler analyses
• Expresses various dependency analyses: information flow, program slicing
• CertiCrypt (Zanella, Barthe,...) probabilistic variant in Coq used to verify digital signature schemes by program transformation
• Relational HTT (Nanevski, Banerjee, Garg) expresses and verifies information flow and access control policies for programs with higher-order functions, dynamically allocated references
Monadic reflection

- Filinski
- Layering of one effect on top of another via monad transformers amounts to translation of language with complex effect into one with simpler effect
- Monadic reflection allows one to move between effects as behaviour and effects as data

\[
\frac{\Gamma \vdash V : T\alpha}{\Gamma \vdash \mu(V) : \alpha} \quad \text{and} \quad \frac{\Gamma \vdash E : \alpha}{\Gamma \vdash [E] : T\alpha}
\]

- Delimited control is a universal effect
- And can be implemented via call/cc and a reference cell

```plaintext
- fun apptwice f = (f 1; f 2; "done");
val apptwice : (int->unit)->string

- val tapptwice = translate ((int->unit)--->string) apptwice;
val tapptwice : (int->unit t)->string t
```
Taming circularity

• Traditional domain theory (Scott,...)
• Step-indexing (Appel, McAllester, Ahmed)
• Ultrametric spaces and the topos of trees $\text{Sets}^{\omega^{\text{op}}}$ (Birkedal et al)
  – Modalities for well-founded recursion
Models of languages with higher-typed store (and other hard features)

- Domain-theoretic (Bohr, Birkedal)
- Step-indexed Kripke logical relations over operational semantics (Ahmed, Dreyer, Rossberg, Neis, Birkedal)
  - Possible worlds become state transition systems
  - Associated modal logics
  - Fine analysis of the effect on reasoning of flat/ho references, control operators
  - Biorthogonalons and much more besides
  - Fully abstract and practically applicable to tricky examples of imperative ADTs
Game semantics

- Abramsky, Ghica, Honda, McCusker, Tzevelekos
- Types as games, terms as strategies
- Has provided fully abstract models for many different kinds of programming languages
- Applied to nu-calc, pi-calc, references Recently extended with nominal structure to provide fully abstract models of ML-like languages
- Game-like ideas apparent in structure of possible worlds in recent logical relations models
Compiler Correctness

• Compcert (Leroy)
• FPCC (Appel,...)
  – Translate high-level types into foundational logic
  – Memory safety
• Compositional compiler correctness (B, Hur, Tabareau, Dreyer)
  – Semantic type soundness
  – Full functional correctness
  – Biorthogonals, separation, logical relations, step indices,...
  – Hur & Dreyer can even verify self-modifying code
Fixpoint semantics_of_types (t:ExpType) (Ra:stateRel) ptr ptr’ struct t :=
  match t with
  | Int P ⇒ lift (P ptr ∧ (ptr = ptr’))
  | Bool P ⇒ lift (P (n2b ptr) ∧ (n2b ptr = n2b ptr’))
  | a * b ⇒ Ex value, Ex value2, Ex value’, Ex value2’,
  (ptr,ptr’→value,value’) ×
  (ptr+1,ptr’+1→value2,value2’) × [b] Ra value value’ × [a] Ra value2 value2’)
  | a → b ⇒ Ex Rprivate,
  (ptr,ptr’ ⇒ Later (Perp (Pre_arrow Rprivate ptr ptr’ Ra ([a]) ([b])))) × Rprivate
end

where "'[[ t ]]'" := (semantics_of_types t).

Definition Post_arrow b (Ra Rc: stateRel) Rc_cloud (n n’ stack_ptr stack_ptr’: nat):=
  Ex ptr_result, Ex ptr_result’,
  (stack_ptr,stack_ptr’ ⇒ ptr_result,ptr_result’) ⊗ (stack_ptr+1,stack_ptr’+1⇒⋯) ⊗
  ((b Ra ptr_result ptr_result’) × Rc_cloud) ⊗ Ra ⊗ Rc ⊗ (spreg→ stack_ptr,stack_ptr’) ⊗
  (envreg→ n,n’) ⊗ unused_space.

Definition Pre_arrow R_private ptr_function ptr_function’ Ra a b:=
  Ex Rc, Ex Rc_cloud, Ex n, Ex n’, Ex ptr_arg, Ex ptr_arg’, Ex stack_ptr, Ex stack_ptr’,
  (stack_ptr,stack_ptr’⇒ ptr_arg,ptr_arg’) ⊗
  (stack_ptr+1,stack_ptr’+1⇒ ptr_function,ptr_function’)
⊗ (R_private × a Ra ptr_arg ptr_arg’ × Rc_cloud) ⊗
  ((n+4,n’+4 ⇒ Later (Perp (Post_arrow b Ra Rc Rc_cloud n n’ stack_ptr stack_ptr’)))) × Rc) ⊗
  Ra ⊗ (spreg→ stack_ptr+1,stack_ptr’+1) ⊗ (envreg→ n,n’) ⊗ unused_space.
Themes

• Monads, algebras, monoidal structure
• Biorthogonalons
• Separation
• Logical relations and parameters
• Recursion and approximation
• Invariants and beyond
• Types versus logics
• Mechanization