New Combinators on the Block

Vincent Danos & Jean-Louis Krivine.
Équipe PPS

Timeline

- CSL'00 "Disjunctive Tautologies as Synchronisation Schemes": interesting behaviours specified by distinguished tautologies (and valid formulas)

- Now: independent decompilation resulting in language extensions (going from mere descriptions to actual computation rules)

- Ongoing: a calculus of communication environments

Perspective

- Usual flow: from programming practice (objects, communications, exceptions) to theory/semantics/logic

- Theory should be more assertive and reverse the flow

- "Obstetrics is good, breeding is better ..."

- Get new/clean/abstract programming forms from logic (such as unification, matching, λs, constraints) !

Summary

- Family of well-typed control/exception related combinators (in a sequential cbn world)

- Introducing a creative piece of syntax: dynamic binders that rescope themselves at run-time

- Potent suggestions of high-level synchronisation combinators

- Aside: introducing Krivine's realizability —a powerful substitute to subject reduction— derived from Tait-Girard-Plotkin's reducibility arguments

Krivine's specification problem

- Needs a logic: second order (classical) predicate calculus (could be ZF as well !)

- needs a language for realizing proofs: variant of Felleisein's λC, a cbn weak head evaluation with a stack-save-and-restore mechanism

Is there any behavior common to all $t : \phi$?

- Needs a tool to read off behaviors from ϕ: Krivine's classical realizability

- Small specification vs big specification: instruction or program?

Generating new programming forms

- Home in on a family of excluded-middle-like tautologies

$$
\begin{array}{l}
(A \to B) \vee A \\
(A \to B) \vee (B \to A) \\
\ldots
\end{array}
$$

- Guess ϕ's specification by trials, prove it by realizability

- Add in new combinators $\mathcal{C}_\phi$ for ϕ decompiling the $t : \phi$ (don't have to prove the decompilation is correct)

- Prove adding in $\vdash \mathcal{C}_\phi : \phi$ is all right by realizability again (a perfectly modular argument)

The language $\lambda\kappa$

- Λ the set of terms & Π the set of stacks

$$t = x, (t\ t), \lambda x.t, \kappa x.t, *_t, *_\pi$$
$$\pi = \epsilon, t \cdot \pi$$

- Usual CBN 'call-with-current-continuation' is $\lambda h.\kappa k.(h\ k)$

- Our variant just makes the analogy between λ and κ obvious: one targets terms, the other stacks. Nothing deep here.

Evaluation

- executables $\in \Lambda \times \Pi$ — a pair of a term and a stack (fun/args)

- evaluation relation, written $\succ$, the smallest preorder such that:

$$
\begin{array}{l}
t\ \ u, \pi \succ t, u \cdot \pi \\
(\lambda x.t), u \cdot \pi \succ t[*u/x], \pi \\
*u, \pi \succ u, \pi \\
(\kappa x.t), \pi \succ t[*\pi/x], \pi \\
*\pi, t \cdot \pi' \succ t, \pi
\end{array}
$$

- set $\delta = \kappa x.x$, then for any π and get a loop:

$$\delta\ \ \delta, \pi \succ \delta, \delta \cdot \pi \succ *\delta \cdot \pi, \delta \cdot \pi \succ \delta, \delta \cdot \pi.$$

Logic or typing system

$$\text{ax}\ \frac{}{\Gamma, x : A \vdash x : A}\ \text{var}$$

$$\to\text{i}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ \text{abs} \qquad \to\text{e}\ \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B}\ \text{app}$$

$$\forall\text{i}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X A} \qquad \forall\text{e}\ \frac{\Gamma \vdash t : \forall X A}{\Gamma \vdash t : A[B/X]}$$

$$\text{peirce}\ \frac{\Gamma, x : A \to B \vdash t : A}{\Gamma \vdash \kappa x.t : A}\ \text{cc}$$

- First five rules: natural deduction for intuitionistic logic

- Sixth rule: Peirce's law makes it one *possible* presentation of second order classical logic

Truth Values

- Formulas are valued by particular sets of terms

- $\perp\!\!\!\perp$ a given set of executables 'good ones' closed by $\succ^{-1}$

- For any set of stacks $\mathcal{Z}$, set $\mathcal{Z} \to \perp\!\!\!\perp$ to be the largest set of terms $\mathcal{X}$ such that $\mathcal{X} \times \mathcal{Z} \subset \perp\!\!\!\perp$: a truth value.

- Largest truth value $\mathsf{\Lambda} = \emptyset \to \perp\!\!\!\perp$, smallest $\bot = \Pi \to \perp\!\!\!\perp$.
(for any $t, \pi \in \perp\!\!\!\perp$, $(*_\pi)t \in \bot$, so $\bot$ is empty iff $\perp\!\!\!\perp$ is).

Models

- Intuition: $|A|^-$ is the set of stacks all $t : A$ get on well with, and ... dually $|A|$ is the set of terms that will form nice executables when paired with $|A|^-$.

- $|.| : \mathsf{Form}(2^\Pi) \to \mathsf{TruthValues} \subset 2^\Lambda$ is defined as: $|F| = |F|^- \to \perp\!\!\!\perp$.

- Given $\perp\!\!\!\perp$ we can inductively extend any $|.|^- : \mathsf{Var} \to 2^\Pi$ to a map $|.|^- : \mathsf{Form}(2^\Pi) \to 2^\Pi$:

$$
\begin{aligned}
|\mathcal{Z}|^- &= \mathcal{Z} \\
|X|^- &= |X|^- \\
|A \to B|^- &= (|A|^- \to \perp\!\!\!\perp) \cdot |B|^- \\
|\forall X A|^- &= \cup_{\mathcal{Z}} |A[\mathcal{Z}/X]|^-
\end{aligned}
$$

(In the last clause, the union ranges over all subsets $\mathcal{Z}$ of Π).

- $|\forall X X|^- = \cup |\mathcal{Z}|^- = \cup \mathcal{Z} = \Pi$, so $|\forall X X| = \perp$.

Adequacy

- If F is closed, $|F|$ only depends on the choice of $\perp\!\!\!\perp$.

- Valuations of classical formulas via a $\neg\neg$-translation.

- If $\perp\!\!\!\perp = \emptyset$, $|.|$ takes only two values: $\emptyset$ and Λ, and for any closed F, $|F| = \Lambda$ iff F is valid. Ie the model collapses to the usual notion of two-valued model.

- adequacy property for any $\perp\!\!\!\perp \subset \Lambda \times \Pi$:

$$\vdash t : F \wedge \pi \in |F|^{-} \Rightarrow t, \pi \in \perp\!\!\!\perp.$$

Consistency Check

- Take a first-order language $\mathcal{L}$ with two constants 0 and 1:

$$Bx = \forall X\, [X0 \to (X1 \to Xx)]$$

- Suppose $\vdash t : B0$, set $\perp\!\!\!\perp = \{e | e \succ a, \pi\}$, $X0^{-} = \{\pi\}$ and $X1 = \Lambda$: $a \in X0$ and $b \in X1$, so $t, a \cdot b \cdot \pi \in \perp\!\!\!\perp$, and hence $tab, \pi \in \perp\!\!\!\perp \succ a, \pi$.

- The system is computationally consistent

- Adequacy gives a means of decoding the behavior specified by a given formula with respect to a given language

A formula

- Coding disjunction to implication (intuitionistic).

$(A \to B) \vee (B \to A)$
$\forall X[((A \to B) \to X) \to ((B \to A) \to X) \to X]$
$(\neg A \vee B) \vee (\neg B \vee A)$.

- Let G be the closure of the second formula.

Example of a new instruction: $\mathcal{C}_G$

- Extend $\lambda\kappa$ with a new combinator $\mathcal{C}_G$:

$$\begin{array}{lll} \mathcal{C}_G \,,\, \sigma_1 \cdot \sigma_2 \cdot \pi & \succ & \sigma_1 \,,\, \alpha \cdot \pi \\ \alpha \,,\, a \cdot \pi' & \succ & \sigma_2 \,,\, \lambda d.\kappa\alpha.a \cdot \pi \end{array}$$

- α is a fresh variable defined by $\mathcal{C}_G$ memoizing σ_2 and π (a better/heavier notation is $*\sigma_2, \pi$)

- When α makes it to head position, it rebinds itself in a !

- Informally: call α an exception; call pushing α to head position 'raising' the exception (the exception is trapped *once* for all)

- Clean local exception handling; no heavier than the usual mechanism, just sound.

Other Solutions

- a ‘lefthanded’ $\mathcal{C}_G$: could have taken a right-handed one of course; even a concurrent=both-handed one, more about this later.

Typing $\mathcal{C}_G$

Subject to a few natural additional conditions on $\perp\!\!\!\perp$:

$$\boxed{\mathcal{C}_G \in |\forall X.[((A \to B) \to X) \to ((B \to A) \to X) \to X]|}$$

(1) $e \succ e' \Rightarrow (e \in \perp\!\!\!\perp \Leftrightarrow e' \in \perp\!\!\!\perp)$ (it's closed by $\succ^{-1}$ and $\succ$ as well);

(2) if $e[t/x] \in \perp\!\!\!\perp$ and $e \not\succ x$, . . . then for any u, $e[u/x] \in \perp\!\!\!\perp$ (that's when x never makes it to head position in e);

(3) $\lambda x.t$, $\pi_0 \notin \perp\!\!\!\perp$ if π_0 is an end-stack (ϵ).

Proof

Take $\sigma_1 \in (A \to B) \to X$, $\sigma_2 \in (B \to A) \to X$ and $\pi \in X^-$ (we drop the $|.|$s everywhere).

We want $\mathcal{C}_G \star \sigma_1 \cdot \sigma_2 \cdot \pi \in \perp\!\!\!\perp$, or by (1) $\sigma_1 \star \alpha \cdot \pi \in \perp\!\!\!\perp$.

There are three cases.

1– $\sigma_1 \star \alpha \cdot \pi \not\succ \alpha \star \ldots$.

If $t \in A \to B$, $\sigma_1 \star t \cdot \pi \in \perp\!\!\!\perp$, and by (2) $\sigma_1 \star \alpha \cdot \pi \in \perp\!\!\!\perp$.

2– $\sigma_1 \star \alpha \cdot \pi \succ \alpha \star \pi_0$ with π_0 an end-stack.

If $t \in B$ then $\lambda x.t \in A \to B$, so $\sigma_1 \star \lambda x.t \cdot \pi \in \perp\!\!\!\perp$, and $\lambda x.t \star \pi_0 \in \perp\!\!\!\perp$ by (1), which contradicts (3).

Proof (2)

3– $\sigma_1 \,, \alpha \cdot \pi \succ \alpha \,, a \cdot \pi'$.

For any $\pi_A \in A^-$, one has: $*\pi_A \in A \to \bot \subset A \to B$, hence $\sigma_1 \,, *\pi_A \cdot \pi \in \perp\!\!\!\perp$, but

$$\sigma_1 \,, *\pi_A \cdot \pi \succ *\pi_A \,, (a \cdot \pi')^{[*\pi_A/\alpha]} \succ a[*\pi_A/\alpha] \,, \pi_A,$$

(because α is fresh in the first step) so everything above is in $\perp\!\!\!\perp$ by $(1)^-$ and therefore $\kappa\alpha.a \,, \pi_A \in \perp\!\!\!\perp$ by (1). This is true for any $\pi_A \in A^-$, hence $\kappa\alpha.a \in A$, and $\lambda d\kappa\alpha.a \in B \to A$. Turning to σ_2 we get $\sigma_2 \,, \lambda d\kappa\alpha.a \cdot \pi \in \perp\!\!\!\perp$, which is what we wanted, since:

$$\mathcal{C}_G \,, \sigma_1 \cdot \sigma_2 \cdot \pi \succ \sigma_1 \,, \alpha \cdot \pi \succ \alpha \,, a \cdot \pi' \succ \sigma_2 \,, \lambda d\kappa\alpha.a \cdot \pi.$$

Peirce's Anamorphosis

- $\mathcal{C}_G$ smarter than its implementations in $\lambda\kappa$ but they're equivalent (horrible to prove). Such as:

$$((C)\sigma_1)\sigma_2 = \kappa k^{X \to B}.(\sigma_1)\lambda x^A.(k)(\sigma_2)\lambda y^B.x$$

That's the way we decompiled it !

- There should be a deadlock-free implementation of the concurrent version.

$$H = (A_1 \to C) \vee (A_2 \to C) \vee (A_1 \wedge A_2)$$

- H is classically equivalent to $\neg A_1 \vee C \vee \neg A_2 \vee C \vee A_1 \wedge A_2$

- Generates another new combinator $\mathcal{C}_H$:

$$\begin{array}{lcl} \mathcal{C}_H \,,\, \sigma_1 \cdot \sigma_2 \cdot \tau \cdot \pi & \succ & \sigma_1 \,,\, \alpha_1 \cdot \pi \\ \alpha_1 \,,\, a_1 \cdot \pi_1 & \succ & \sigma_2 \,,\, \alpha_2 \cdot \pi \\ \alpha_2 \,,\, a_2 \cdot \pi_2 & \succ & \tau \,,\, \kappa\alpha_1.a_1 \cdot \kappa\alpha_2.a_2 \cdot \pi \end{array}$$

- Again there seems to be an obvious synchronisation interpretation: run the two bobs, σ_1 and σ_2, handle them both (τ) if they both raise an exception.

- But it's not a join pattern because of the rescoping, or rebinding, involved; vital to the consistency of the typing.

Conclusions

- What exactly is the family of tautologies for which this analysis is relevant ? it contains our 'disjunctive tautologies'.

- Work out a presentation of local exception handling that would suit programmers.

- Rescoping or rebinding syntax: smarter instruction sets for abstract/virtual machines

- Communication: closed communication contexts, ambients with a handler; in development: a syntax for outs & ins.