Choice Trees

Representing Nondeterministic, Recursive, and Impure Programs in Coq

Nicolas Chappe, Paul He, Ludovic Henrio, Steve Zdancewic and Yannick Zakowski
Representing Nondeterministic, Recursive, and Impure Programs in Coq (POPL’20)

Li-yao Xia, Yannick Zakowski, Paul He, Gregory Malecha, Chung-Kil Hur, Benjamin Pierce, Steve Zdancewic
Introduction: Monadic Definitional Interpreters

Prior Work: Interaction Trees

Choice Trees: Tackling Non-Determinism

CTrees, LTSs and Bisimulations

Conclusion
Introduction: Monadic Definitional Interpreters
Modelling computations in a proof assistant

**Why?**

**Many interesting properties:**

- Does a program respect its specification?
- Are two syntactically different programs equivalent?
- Does a compiler respect the meaning of its input programs?

→ Notions of equivalence and refinement

**How?**

\[
\begin{align*}
    & c_1 | \sigma \rightarrow c_1' | \sigma' \\
\Rightarrow & \quad c_1; c_2 | \sigma \rightarrow c_1'; c_2 | \sigma'
\end{align*}
\]

Small-step

\[
\begin{align*}
    & c_1 | \sigma \downarrow \sigma' \\
\Rightarrow & \quad c_1; c_2 | \sigma \downarrow \sigma'' \\
\Rightarrow & \quad c_2 | \sigma' \downarrow \sigma''
\end{align*}
\]

Big-step

\[
[c_2] \circ [c_1]
\]

Denotational composition of continuous functions over a CPO
Modelling computations in a proof assistant

Why?

Many interesting properties:

• Does a program respect its specification?

• Are two syntactically different programs equivalent?

• Does a compiler respect the meaning of its input programs?

→ Notions of equivalence and refinement

How?

Small-step

\[
\frac{c_1 \mid \sigma \rightarrow c'_1 \mid \sigma'}{c_1; c_2 \mid \sigma \rightarrow c'_1; c_2 \mid \sigma'}
\]

Big-step

\[
\frac{c_1 \mid \sigma \Downarrow \sigma' \quad c_2 \mid \sigma' \Downarrow \sigma''}{c_1; c_2 \mid \sigma \Downarrow \sigma''}
\]

Denotational

\[
[c_2] \circ [c_1]
\]

composition of continuous functions over a CPO

The way we model impacts the ways we can reason
The Semantics Impacts the Reasoning

**Compositionality:** We can reason on parts of the program separately → Simplifies the proof technique

**Modularity:** The semantics is made of several independent parts → Improves maintainability

**Executability:** A complete reference interpreter can be derived from the semantics of a language → Helps with testing
Modelling, but how?

Let’s focus on executability

To model something as complex as C or LLVM IR, a reference interpreter is very valuable!
Modelling, but how?

Let’s focus on executability
To model something as complex as C or LLVM IR, a reference interpreter is very valuable!

ITrees take a simple route (back to the 70’s with Reynolds)

Definitional Interpreters

Describe the language to model via an interpreter written in your host language 🦇
Modelling, but how?

Let’s focus on executability
To model something as complex as C or LLVM IR, a reference interpreter is very valuable!

ITrees take a simple route (back to the 70’s with Reynolds)

Definitional Monadic Interpreters

Describe the language to model via an interpreter written in your host language 🕵️‍♂️
Interpreter for a Modest Language

\[ \text{Imp} \triangleq \bullet | x := e | c_1; c_2 \]

Commands map an initial environment (memory) to a final environment

\[ \text{interp } (c : \text{com}) (s : \text{env}) : \text{env} \]

We thread the state manually

\[ \text{interp } (c_1;c_2) s_1 \triangleq \text{let } s_2 := \text{interp } c_1 s_1 \text{ in } \text{interp } c_2 s_2 \]
Monadic Interpreter for a Modest Language

\[ \text{Imp} \triangleq \cdot \mid x := e \mid c_1; c_2 \]

Commands are stateful computations

\[ \text{interp} \ (c : \text{com}) : \text{state} \ \text{unit} \]

\[
\begin{align*}
\text{state} \ X & \triangleq \text{env} \rightarrow (\text{env} \ast X) \\
\text{maybestate} \ X & \triangleq \text{env} \rightarrow \text{option} \ (\text{env} \ast X)
\end{align*}
\]

The monad tells us how to thread computations

\[ \text{interp} \ (c_1; c_2) \triangleq \text{interp} \ c_1 \ ;; \ \text{interp} \ c_2 \]

Expressions can fail: does not leak into the definition of the sequence
Interaction Trees

or

Representing Recursive, and Impure Programs in Coq
ITree Idea 1: the Free Monad

Stateful computations map initial environments to final environments.

are computations performing reads and writes

My computation is a piece of syntax

able to perform operations specified in $E$

in order to compute a value of type $X$

$$\text{free } E \ X \ = \ X + E \ X + E \ E \ X + \ldots$$
ITree Idea 1: the Free Monad

Stateful computations map initial environments to final environments are computations performing reads and writes

```
iproc interp (c : com) : free Rd_Wr unit
```

```
free E X = X + E X + E E X + ...
```

My computation is a piece of syntax

able to perform operations specified in E

in order to compute a value of type X
Programs as Trees

\[ \text{Imp} \triangleq \cdot | x := e | c_1; c_2 \]

\[ p_2 \triangleq x := 0; x := y \]

\[ p_3 \triangleq x := y \]
Programs as Trees

\[ \text{Imp} \triangleq \bullet \mid x := e \mid c_1; c_2 \]

\[ p_2 \triangleq x := 0; x := y \quad \text{are the same} \]

\[ p_3 \triangleq x := y \]
Programs as Trees

\[ \text{Imp} \triangleq \circ \mid x := e \mid c_1 ; c_2 \]

\[ p_2 \triangleq x := 0 ; x := y \quad \text{are the same} \quad p_3 \triangleq x := y \]
Programs as Trees

\[ \text{Imp} \triangleq \cdot | x := e | c_1 ; c_2 \]

Indeed, they are not the same syntax. We fold over the tree to bring in the semantics.

\[ p_2 \triangleq x := 0; x := y \text{ are the same } p_3 \triangleq x := y \]
Programs as Trees

Indeed, they are not the same **syntax**
We fold over the tree to bring in the **semantics**

\[
\text{Imp} \triangleq \cdot \mid x := e \mid c_1; c_2
\]

\[
p_2 \triangleq x := 0; x := y
\]

\[
p_3 \triangleq x := y
\]
Programs as Trees

$$Imp \triangleq \bullet \mid x := e \mid c_1; c_2$$

Indeed, they are not the same syntax
We fold over the tree to bring in the semantics

$$p_2 \triangleq x := 0; x := y$$ are the same

$$p_3 \triangleq x := y$$

$$\begin{array}{c}
\text{wr } x \ 0 \\
\vdots \\
\text{rd } y \\
\text{wr } x \ 0 \quad \text{wr } x \ 1 \quad \ldots \quad \text{wr } x \ n \\
\text{tt } \quad \text{tt } \quad \text{tt } \\
\end{array}
\quad
\begin{array}{c}
\text{wr } x \ 0 \\
\vdots \\
\text{rd } y \\
\text{wr } x \ 0 \quad \text{wr } x \ 1 \quad \ldots \quad \text{wr } x \ n \\
\text{tt } \quad \text{tt } \quad \text{tt } \\
\end{array}$$
But What About Loops?

\[ \text{Imp} \triangleq \cdot | x := e | c_1; c_2 | \text{while } b \text{ do } c \]

\[ p_1 \triangleq \text{while } \text{true} \text{ do } \cdot \]

What tree should we associate to \( p_1 \)?
ITree Idea 2: Capretta’s Delay Monad

$$Imp \triangleq \mathord{\cdot} | x := e | c_1 ; c_2 | \text{while } b \text{ do } c$$

$$p_1 \triangleq \text{while true do } \mathord{\cdot}$$

Something happened internally
Here, the re-entrance of the loop
ITree Idea 2: Capretta’s Delay Monad

\[ \text{Imp} \triangleq \bullet \mid x := e \mid c_1 ; c_2 \mid \text{while } b \text{ do } c \]

\[ p_1 \triangleq \text{while } \text{true} \text{ do } \bullet \]

We move onto a coinductive datatype, \( p_1 \) is an infinite tree
Programs as Stateful Infinite Trees

\[ \text{Imp} \triangleq \bullet \mid x := e \mid c_1 ; c_2 \mid \text{while } b \text{ do } c \]

\[ p_2 \triangleq x := 0 ; x := y \]

\[ p_3 \triangleq x := y \]
Programs as Stateful Infinite Trees

\[ \text{Imp} \triangleq \bullet \mid x := e \mid c_1 ; c_2 \mid \text{while } b \text{ do } c \]

\[ p_2 \triangleq x := 0; x := y \]

\[ p_3 \triangleq x := y \]
Interaction Trees

A domain of computations shallow embedded in Coq

\[
\begin{align*}
\text{CoInductive } & \text{ itree } (E: \text{ Type } \rightarrow \text{ Type}) (R: \text{ Type}): \text{ Type } := \\
& \mid \text{ Ret } (r: R) \\
& \mid \text{ Later } (t: \text{ itree } E R) \\
& \mid \text{ Vis } \{X: \text{ Type}\} (e: E X) (k: X \rightarrow \text{ itree } E R).
\end{align*}
\]

A value of the datatype \((\text{itree } E R)\) represents:

\begin{itemize}
  \item a potentially diverging computation,
  \item which may return a \text{value} of type \(R\),
  \item while emitting during its execution \text{visible events} from the \text{interface} \(E\).
\end{itemize}
Representing Nondeterministic, Recursive, and Impure Programs in Coq
Nondeterministic branching

\[ \text{Imp} \overset{\triangle}{=} \bullet \mid x := e \mid c_1; c_2 \mid \text{while } b \text{ do } c \mid \text{br } c_1 \text{ or } c_2 \mid \text{stuck} \mid \text{print} \]

\text{br } c_1 \text{ or } c_2 : \text{either branch can be executed}

Sounds quite easy to model as an itree: let’s have a \( (\text{toss} : E \text{ bool}) \) event

\[ [\text{br } c_1 \text{ or } c_2] \overset{\triangle}{=} \begin{cases} \text{toss} \\ \text{true} \\ \text{false} \end{cases} \begin{cases} \text{[c1]} \\ \text{[c2]} \end{cases} \]
Nondeterministic branching

\[ \text{Imp} \triangleq \bullet \mid x := e \mid c_1; c_2 \mid \text{while } b \text{ do } c \mid \text{br } c_1 \text{ or } c_2 \mid \text{stuck} \mid \text{print} \]

\text{br } c_1 \text{ or } c_2 : \text{either branch can be executed}

Sounds quite easy to model as an itree: let’s have a \((toss : E \text{ bool})\) event

\[ [\text{br } c_1 \text{ or } c_2] \triangleq \begin{array}{cc}
\text{true} & \text{false} \\
[c1] & [c2]
\end{array} \quad \& \quad \begin{array}{cc}
\text{true} & \text{false} \\
[c2] & [c1]
\end{array} \]

At this stage, \text{toss} is not commutative nor idempotent, nor associative

Question: what is the structure into which we should interpret \text{toss}?
Nondeterministic branching

Question: what is the structure into which we should interpret toss?

An idea from Vellvm: sets of trees?

$\mathcal{I}([br \; c_1 \; or \; c_2]) \triangleq [c_1] \cup [c_2]$  \hspace{1cm} (In Coq: itree E X -> Prop)

We lose executability, monadic laws, everything becomes harder...
Nondeterministic branching

Question: what is the structure into which we should interpret toss?

An idea from Vellvm: sets of trees?

\[ \mathcal{J}([br \; c_1 \; or \; c_2]) \triangleq [c_1] \cup [c_2] \]  (In Coq: itree E X -> Prop)

We lose executability, monadic laws, everything becomes harder...

This work: ctrees, what we believe to be the right structure
Nondeterministic branching: but what do we mean?

\[ \text{Imp} \triangleq \bullet \mid x := e \mid c_1; c_2 \mid \text{while } b \text{ do } c \mid \text{br } c_1 \text{ or } c_2 \mid \text{stuck} \mid \text{print} \]

\[ p \triangleq \text{br} \text{ (while true do print)} \text{ or stuck} \]

Can the above program \( p \) be stuck?

**Case 1:**

\[
\text{br } c_1 \text{ or } c_2 \rightarrow c_1
\]

\[ p \rightarrow \text{stuck} \text{ is possible} \]

The system may **become** either branch

**Case 2:**

\[
c_1 \rightarrow c'_1
\]

\[
\text{br } c_1 \text{ or } c_2 \rightarrow c'_1
\]

\[ p \rightarrow \text{stuck} \text{ is not possible} \]

The system may **take a transition** offered by either branch
$p \triangleq br \ (\text{while true do print}) \text{ or stuck}$

Let’s take the perspective of an LTS

Case 0 (itree):

\[
\begin{align*}
\text{br } c_1 \text{ or } c_2 & \rightarrow c_1 \\

p^{\text{true}} & \rightarrow \text{stuck possible}
\end{align*}
\]

Case 1:

\[
\begin{align*}
\text{br } c_1 \text{ or } c_2 & \rightarrow c_1 \\

p & \rightarrow \text{stuck possible}
\end{align*}
\]

Case 2:

\[
\begin{align*}
\text{c}_1 & \rightarrow \text{c}'_1 \\

\text{br } c_1 \text{ or } c_2 & \rightarrow c'_1 \\

p & \rightarrow \text{stuck not possible}
\end{align*}
\]

External event, we observe which event happened, what branch we took

true / false
[c1] [c2]
$p \triangleq \text{br (while true do print) or stuck}$

Let’s take the perspective of an LTS

**Case 0 (itree):**

\[
\begin{align*}
\text{br } c_1 \text{ or } c_2 & \rightarrow c_1 \\
\end{align*}
\]

$p \xrightarrow{\text{true}} \text{stuck possible}$

**Case 1:**

\[
\begin{align*}
\text{br } c_1 \text{ or } c_2 & \rightarrow c_1 \\
\end{align*}
\]

$p \rightarrow \text{stuck possible}$

**Case 2:**

\[
\begin{align*}
c_1 & \rightarrow c'_1 \\
\text{br } c_1 \text{ or } c_2 & \rightarrow c'_1 \\
\end{align*}
\]

$p \rightarrow \text{stuck not possible}$

**External event,**
we observe which event happened,
what branch we took

**Stepping branch,**
we observe that a branch has been taken
\[ p \triangleq \text{br (while true do print) or stuck} \]

Let’s take the perspective of an LTS

**Case 0 (itree):**

\[
\begin{align*}
\text{br } c_1 \text{ or } c_2 & \rightarrow c_1 \\
p \text{ true} & \rightarrow \text{ stuck possible}
\end{align*}
\]

**Case 1:**

\[
\begin{align*}
\text{br } c_1 \text{ or } c_2 & \rightarrow c_1 \\
p & \rightarrow \text{ stuck possible}
\end{align*}
\]

**Case 2:**

\[
\begin{align*}
c_1 & \rightarrow c'_1 \\
\text{br } c_1 \text{ or } c_2 & \rightarrow c'_1 \\
p & \rightarrow \text{ stuck not possible}
\end{align*}
\]

**External event,**

we observe which event happened, what branch we took

**Stepping branch,**

we observe that a branch has been taken

**Delayed branch,**

there’s a branch, but we don’t observe it
Choice trees

A \textit{ctree }E\textit{ }R\textit{ models a computation as a potentially infinite tree made of:}

- **Leaves**, pure computations (of type \(R\))
- **External events**, interaction with an environment (as described by \(E\))
- **Stepping branches**, an internal choice which may be observed
- **Delayed branches**, an internal choice that only allows to try reaching an observable action

\[
\text{CoInductive } \text{ctree } (E: \text{Type } \to \text{Type}) \ (R: \text{Type}) : \text{Type} :=  \\
| \text{Ret } (r: R)  \\
| \text{Vis } \{X: \text{Type}\} \ (e: E \ X) \ (k: X \to \text{ctree } E \ R)  \\
| \text{BrS } \{n: \text{nat}\} \ (k: \text{fin } n \to \text{ctree } E \ R)  \\
| \text{BrD } \{n: \text{nat}\} \ (k: \text{fin } n \to \text{ctree } E \ R)
\]
CTrees, LTSs and Bisimulations
Bisimulation over ctrees

Question: when should two ctrees be deemed equivalent?

There has already been a lot of work on equivalence of LTSs,
Let’s build LTSs from ctrees!
Bisimulation over ctrees

**Question:** when should two ctrees be deemed equivalent?

\[ \text{label ::= val } x \mid \text{obs } e \mid x \mid \tau \]

- **Leaves**, pure computations (of type \(R\))

- **External events**, interaction with an environment (as described by \(E\))

- **Stepping branches**, an internal choice which may be observed

- **Delayed branches**, an internal choice that only allows to try reaching an observable action

(inductively)
Bisimulation over LTSs

Question: when should two ctrees be deemed equivalent?

Let \((S, \rightarrow)\) be a LTS, \(\mathcal{R}\) a relation on \(S\) is a simulation if:

\[
P \xrightarrow{\mathcal{R}} Q
\]

\[
l \quad l
\]

\[
P' \quad P'
\]
Bisimulation over LTSs

Question: when should two ctrees be deemed equivalent?

Let \((\mathcal{S}, \rightarrow)\) be a LTS, \(\mathcal{R}\) a relation on \(\mathcal{S}\) is a simulation if:

\[
\begin{align*}
P \xrightarrow{\mathcal{R}} Q \\
\downarrow l & \quad \downarrow l \\
P' \xrightarrow{\mathcal{R}} Q'
\end{align*}
\]

Similarity is then defined as the largest simulation

A whole zoo have been studied: weak, complete, branching, ...
Bisimulation over ctrees

Question: when should two ctrees be deemed equivalent?

Answer: if their underlying LTSs are bisimilar!

\[
\begin{align*}
sb \; \mathcal{R} \; s \; t \triangleq \\
\forall l, t, s, s', s \xrightarrow{l} s' \Rightarrow \exists t', s' \mathcal{R} t' \land t \xrightarrow{l} t' \\
\text{ and }
\forall l, s, t, t', t \xrightarrow{l} t' \Rightarrow \exists s', s' \mathcal{R} t' \land s \xrightarrow{l} s'
\end{align*}
\]

For Coq enthusiasts

We tie the coinductive knot using Pous's coinduction library.
Bisimulation over ctrees

Question: when should two ctrees be deemed equivalent?

Answer: if their underlying LTSs are bisimilar!

We recover the right algebraic laws for non-determinism

- **Idempotent**: $\text{BrD} \sim t = t$
- **Commutative**: $\text{BrD} \sim \text{BrD} = \text{BrD}$
- **Associative**: $\text{BrD} \sim \text{BrD} = \text{BrD}$
- **Insensitive to internal computation**: $\text{BrD} \sim t = t$
Bisimulation over ctrees

Do we have the same with BrS?

Insensitive to internal computation

BrD

| ~ t

t

Insensitive to internal computation (?)

BrS

| ~ t

t
Bisimulation over ctrees

Do we have the same with BrS?

Three main equivalences over ctrees

(Coinductive) structural equality

Strong bisimilarity (∼)

Weak bisimilarity (≈)

And trace equivalence, simulations, and potentially all their variants
CTrees and Interpretation

CTrees are an adequate target monad into which one can interpret toss

\[ \text{h(toss)} \triangleq \text{BrD 2} \]

\[ \text{interp } h : \text{itree } (\text{Toss + E}) \rightarrow \text{ctree E} \]

\[ t \approx u \rightarrow \text{interp } h \; t \sim \text{interp } h \; u \]

They of course themselves still support interpretation
(targets must explain how they internalise branching nodes)

Branching nodes can be « interpreted » as well

\[ \sim \rightarrow \text{low level notion of scheduler} \]
\[ \sim \rightarrow \text{formal refinements (complete simulations) in Coq} \]
\[ \sim \rightarrow \text{practical testing in OCaml} \]
Calculus of Communicating Systems [Milner, 1980]

\[ P ::= 0 \mid l \cdot P \mid P \oplus Q \mid P \parallel Q \mid \nu c \cdot P \mid !P \]

- Communication
- Internal choice
- Parallel composition
- Channel restriction
- Replication

Goal: build a computable model of ccs using ctrees
Calculus of Communicating Systems [Milner, 1980]

\[
P ::= 0 \mid l \cdot P \mid P \oplus Q \mid P \parallel Q \mid vc \cdot P \mid !P
\]

- We establish ccs’s traditional equational theory w.r.t. \(\sim\) on our model
- We prove an adequacy result against ccs’s operational semantics

\[\quad [P] \sim [Q] \text{ iff } P \sim_{op} Q\]

- Our model is computable: we can execute by extraction

\(\leadsto\) With a caveat: restriction kills branches, one needs to avoid these dead branches
Cooperative scheduling

\[
com ::= \bullet \mid x := e \mid c_1; c_2 \mid \text{while } b \text{ do } c \mid \text{fork } c_1 \; c_2 \mid \text{yield}
\]

- Two layered computable model:
  - compositional construction with explicit fork and yield events
  - top-level interleaving combinator

- Combination of non-determinism with stateful computations

- Selected set of algebraic equations (further work needed there)
Ctrees Open Question 1: BrD or BrS?

\[ p_1 \triangleq \text{while true do } \bullet \]

More generally: BrD and strong bisimulation or BrS and weak?
CTrees Open Question 2: Do we have the right LTS?
CTrees Open Question 2: Do we have the right LTS?

\[ t \sim u \]

\[ \text{interp } h t \not\sim \text{interp } h u \]
Choice Trees in a Nutshell

Modelling non-determinism and concurrency as monadic interpreters

- We stick to the tree structure, with two new kinds of branching nodes
- Looking at the tree as an LTS sheds light to reason on their equivalence: the tools from the process algebra literature can be brought in
- Case studies suggest that the approach is viable!
- The representation still feels too large: avenue for improvement?

Implemented as a Coq library: https://github.com/vellvm/ctrees/

Accepted at POPL’23: