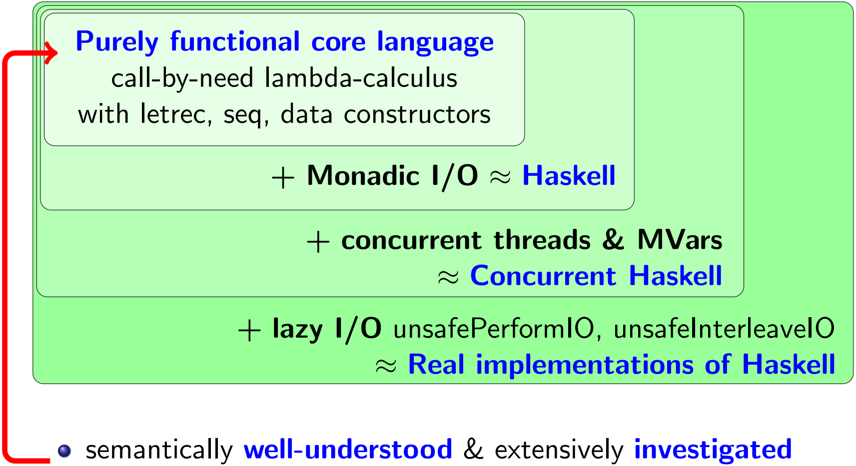
Conservative Concurrency in Haskell

David Sabel and **Manfred Schmidt-Schauß**

Goethe-University, Frankfurt am Main, Germany

LICS'12, Dubrovnik, Croatia

Motivation – a View on Haskell



Purely functional core language
call-by-need lambda-calculus
with letrec, seq, data constructors

+ **Monadic I/O** ≈ **Haskell**

+ **concurrent threads & MVars**
≈ **Concurrent Haskell**

+ **lazy I/O** unsafePerformIO, unsafeInterleaveIO
≈ **Real implementations of Haskell**

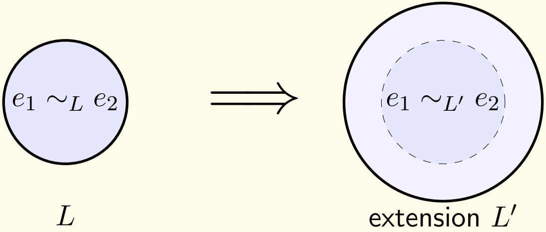
- semantically **well-understood** & extensively **investigated**
- a lot of **correct** of program **transformations** & compiler **optimizations** are known

Issues

- Is the **compiler** still correct after extending the language?
- In short: Are these extensions **safe**?

Safety = Conservativity

An extension is **conservative** if it preserves program equalitites



$\Longrightarrow$ correct **program transformations** of L are also correct in L'

Our Setting

- **Concurrent Haskell** (Peyton Jones, Gordon, Finne 1996) extends Haskell by concurrency
- The **process calculus CHF** (Sabel, Schmidt-Schauß 2011) models **Concurrent Haskell with Futures** operational semantics inspired by (Peyton Jones, 2001)
- Future = variable whose value is computed concurrently by a monadic computation
- allow implicit synchronisation by data dependency
- Concurrent Haskell + `unsafeInterleaveIO` can encode CHF (CHF is a sublanguage of Concurrent Haskell + `unsafeInterleaveIO`)

The Process Calculus CHF

Processes

$$P, P_i \in \mathit{Proc} ::= P_1 \mid P_2 \;\mid\; \nu x.P \;\mid\; \underbrace{x \Leftarrow e}_{\text{future } x} \;\mid\; x = e \;\mid\; \underbrace{x\,\mathbf{m}\,e \;\mid\; x\,\mathbf{m}\,-}_{\text{filled \& empty MVar}}$$

A process has a main thread: $x \overset{\text{main}}{\Longleftarrow} e \mid P$

Expressions

$$\begin{aligned} e, e_i \in \mathit{Expr}_{CHF} ::= {} & x \mid \lambda x.e \mid (e_1\ e_2) \mid \texttt{seq}\ e_1\ e_2 \mid c\ e_1 \ldots e_{\mathrm{ar}(c)} \\ & \mid \texttt{case}_T\ e\ \texttt{of}\ \ldots (c_{T,i}\ x_1 \ldots x_{\mathrm{ar}(c_{T,i})} \to e_i) \ldots \\ & \mid \texttt{letrec}\ x_1 = e_1\ \ldots\ x_n = e_n\ \texttt{in}\ e \\ & \mid \texttt{return}\ e \mid e_1\ \texttt{>>=}\ e_2 \mid \texttt{future}\ e \\ & \mid \texttt{takeMVar}\ e \mid \texttt{newMVar}\ e \mid \texttt{putMVar}\ e_1\ e_2 \end{aligned}$$

Types: standard monomorphic type system

Semantics & Program Equivalence

Operational semantics: Small-step reduction relation $\xrightarrow{CHF}$

Process P is **successful** if P well-formed $\wedge$ $P \equiv \nu \overrightarrow{x_i}.(x \xLeftarrow{\text{main}} \texttt{return}\ e \,|\, P')$

May-Convergence: (a successful process can be reached by reduction)

$$P\downarrow \text{ iff } P \text{ is w.-f. and } \exists P' : P \xrightarrow{CHF,*} P' \wedge P' \text{ successful}$$

Should-Convergence: (every successor is may-convergent)

$$P\Downarrow \text{ iff } P \text{ is w.-f. and } \forall P' : P \xrightarrow{CHF,*} P' \implies P'\downarrow$$

Contextual Equivalence $\sim_{c,CHF}$

On processes:

$$P_1 \sim_{c,CHF} P_2 \text{ iff } \forall \mathbb{D} : (\mathbb{D}[P_1]\downarrow \iff \mathbb{D}[P_2]\downarrow) \wedge (\mathbb{D}[P_1]\Downarrow \iff \mathbb{D}[P_2]\Downarrow)$$

On expressions: $e_1, e_2 :: \tau$

$$e_1 \sim_{c,CHF} e_2 \text{ iff } \forall \mathbb{C} : (\mathbb{C}[e_1]\downarrow \iff \mathbb{C}[e_2]\downarrow) \wedge (\mathbb{C}[e_1]\Downarrow \iff \mathbb{C}[e_2]\Downarrow)$$

Conservativity

PF = Pure, deterministic sublanguage of CHF, no futures, no I/O

$$\begin{aligned} e, e_i \in \mathit{Expr}_{PF} ::= {} & x \mid \lambda x.e \mid (e_1\ e_2) \mid \texttt{seq}\ e_1\ e_2 \mid c\ e_1 \ldots e_{\mathrm{ar}(c)} \\ & \mid \texttt{case}_T\ e\ \texttt{of}\ \ldots (c_{T,i}\ x_1 \ldots x_{\mathrm{ar}(c_{T,i})} \to e_i) \ldots \\ & \mid \texttt{letrec}\ x_1 = e_1\ \ldots\ x_n = e_n\ \texttt{in}\ e \end{aligned}$$

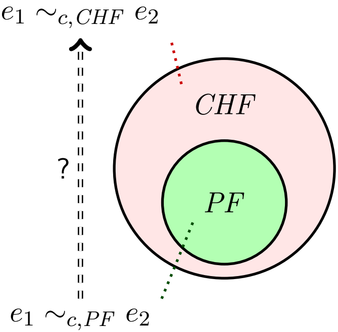
Main Theorem

CHF extends PF conservatively

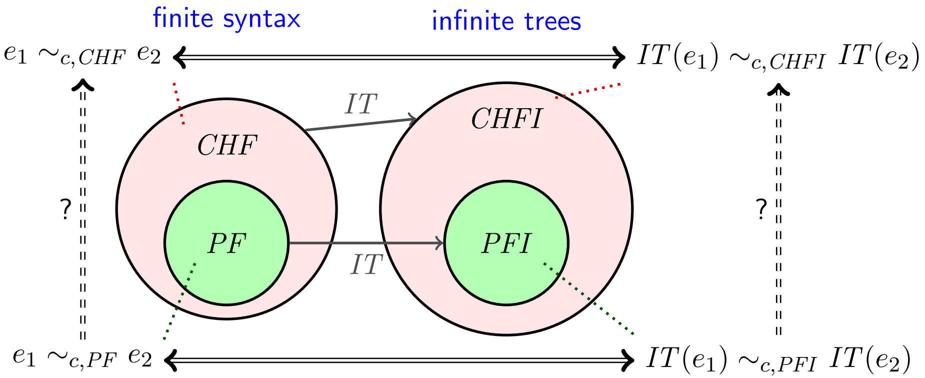
I.e., for all $e_1, e_2 :: \tau \in \mathit{Expr}_{PF}$: $e_1 \sim_{c,PF} e_2 \implies e_1 \sim_{c,CHF} e_2$.

$\Longrightarrow$ correct transformations of the pure core are still valid in CHF

Outline of the Proof



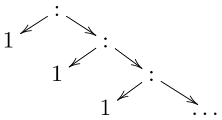
Outline of the Proof



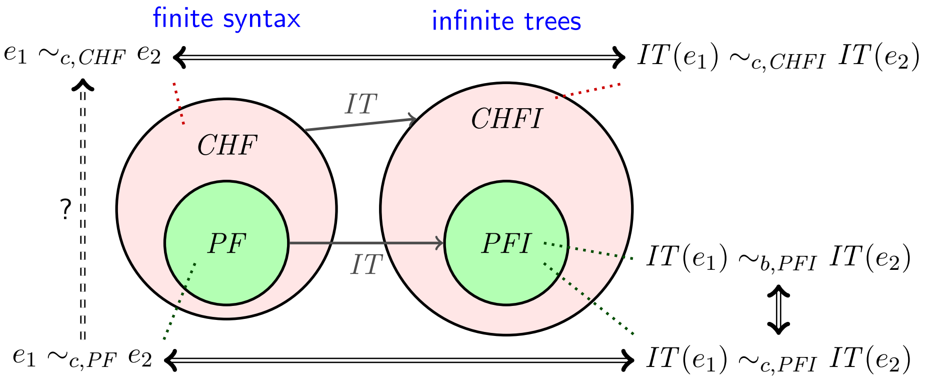
Step 1: transport the problem to calculi with **infinite trees**:

– IT **unfolds all bindings**, $CHFI = IT(CHF)$ and $PFI = IT(PF)$

e.g. $x = 1 : x \quad \xrightarrow{IT}$



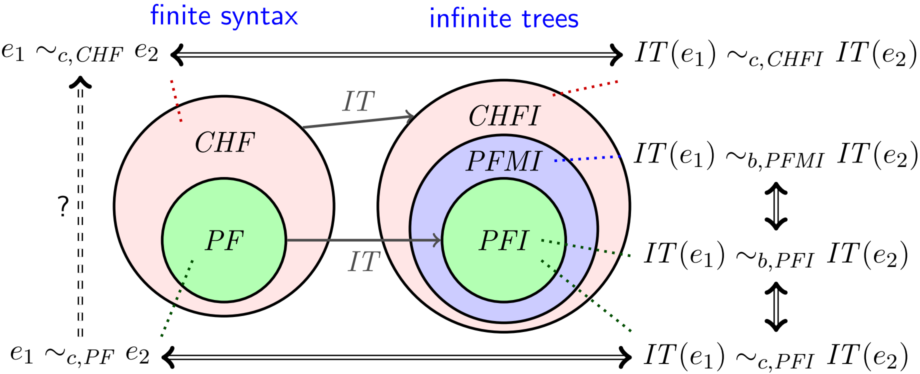
Outline of the Proof



Step 2: define **bisimilarity** $\sim_{b,PFI}$ in PFI

– **Howe's method** shows $\sim_{b,PFI} = \sim_{c,PFI}$

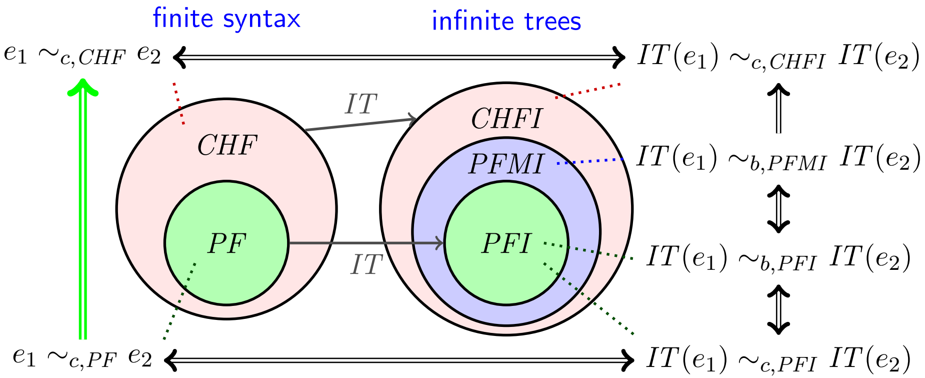
Outline of the Proof



Step 3: add **monadic operators** (interpreted like constants) $= PFMI$

– bisimilarity is unchanged: $\sim_{b,PFI} = \sim_{b,PFMI}$

Outline of the Proof



Step 4: show $e_1 \sim_{b,PFMI} e_2 \implies e_1 \sim_{c,CHFI} e_2$

– syntactical proof by cases

Non-Conservativity Results

$CHFL = CHF +$ **lazy futures**

- lazy future $=$
  concurrent computation starts **only if** the value is **demanded**
- $CHFL$ is **not a conservative extension** of PF
- Counterexample: $\texttt{seq}\ e_2\ (\texttt{seq}\ e_1\ e_1)\ \ \sim_{PF}\ \ (\texttt{seq}\ e_1\ e_2)$

Since lazy futures are encodable with `unsafeInterleaveIO`:

- CHF+`unsafeInterleaveIO` is also
  not a conservative extension of PF

Conclusion & Further Work

Conclusion

- *CHF* (and also Concurrent Haskell) are **conservative extensions** of the pure core language
- result shown w.r.t. **contextual equivalence** based on **may**- and **should**-convergence
- adding `unsafeInterleaveIO` (or even lazy futures) **breaks conservativity**

Future Work

- *CHF* with **polymorphic typing**
- other primitives like **exceptions**