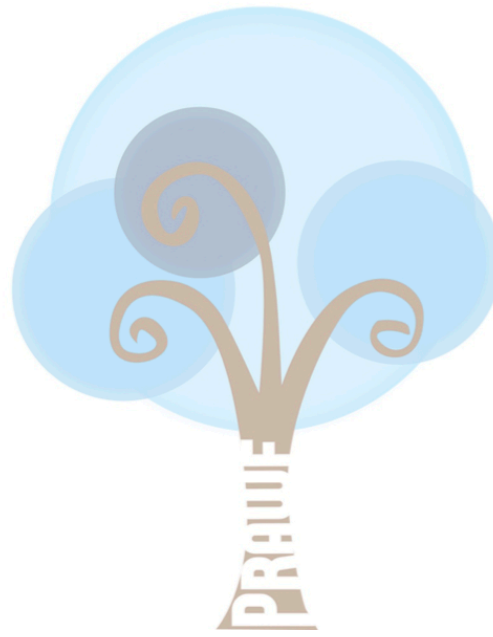
Prifysgol Abertawe
Swansea University

Operational Semantics for PRAWF

Olga Petrovska

Proof and Computation 2022
Celebrating Prof. Schwichtenberg's 80th Birthday
2 June 2022, Schlehdorf, Germany

What is Prawf?

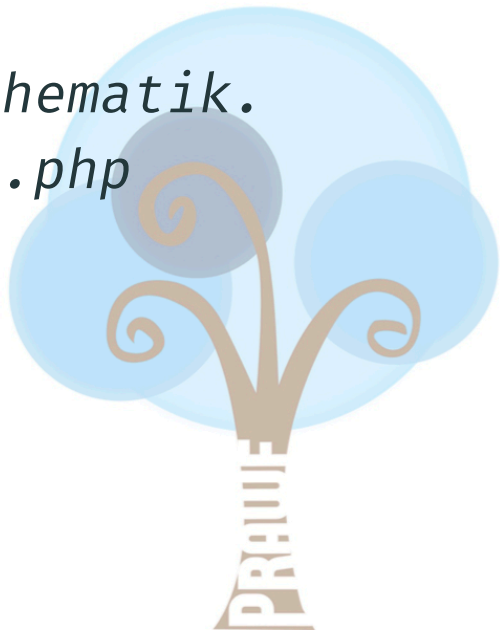


Prawf

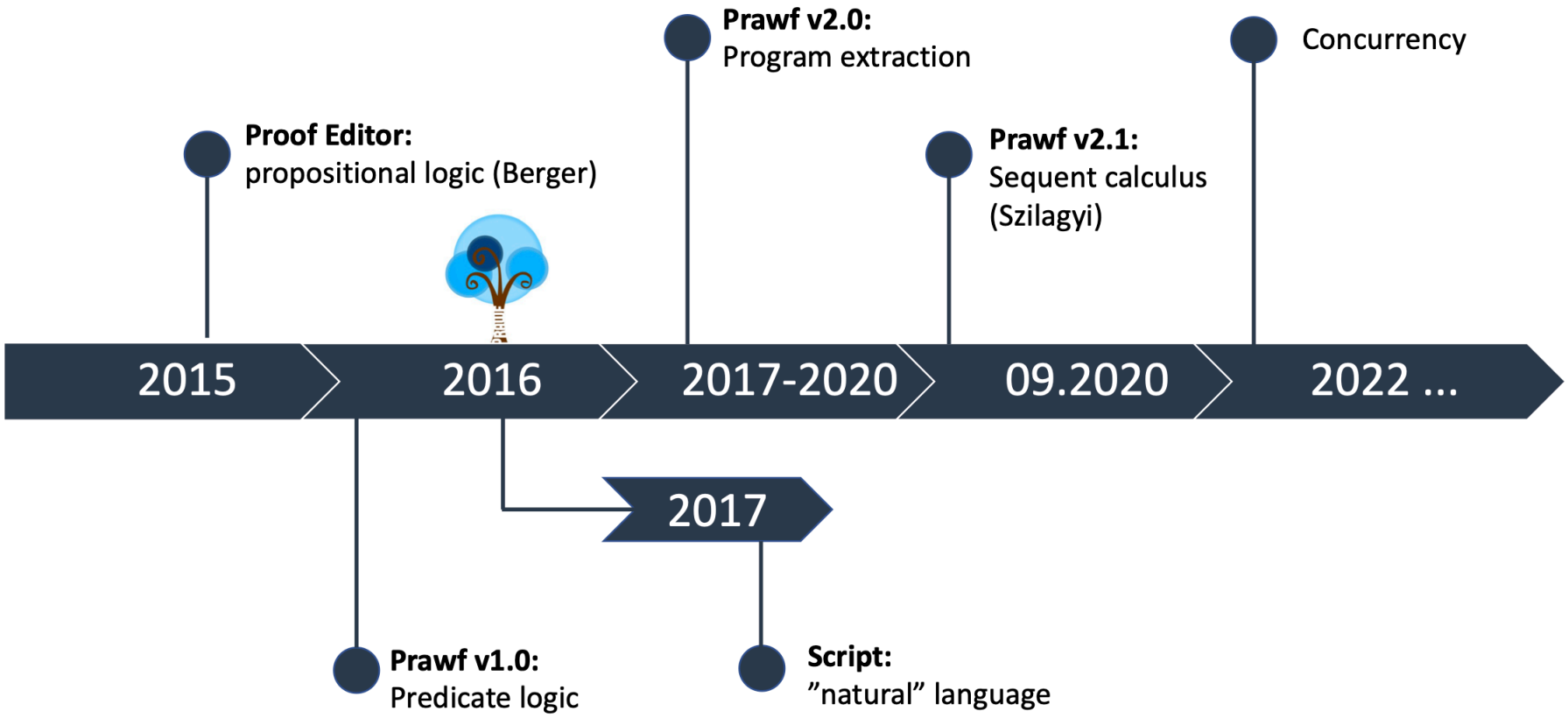
Prawf (Welsh for “proof”) is an interactive Haskell-based proof assistant and program extraction tool. The underlying logical framework is Intuitionistic Fixed-Point Logic.

Existing systems:

- Minlog (H. Schwichtenberg): *http://www.mathematik.uni-muenchen.de/~logik/minlog/index.php*
- Nuprl, Isabelle, Coq etc.



PRAWF: overview



Intuitionistic Fixed Point Logic

Intuitionistic Fixed Point Logic (IFP) as a schema

First-order logic with lambda abstractions and fixed point operators

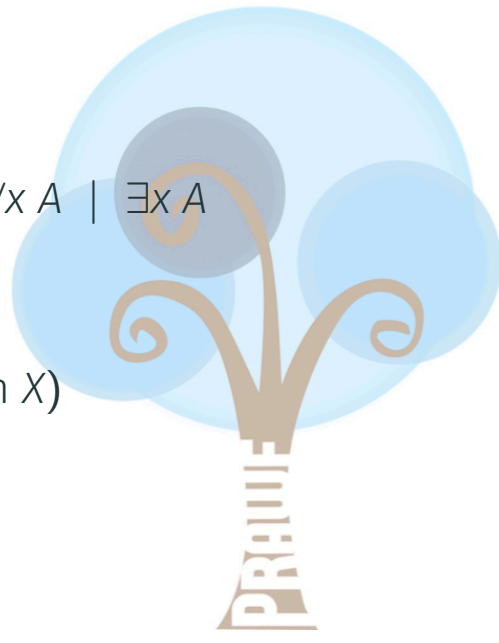
IFP is a schema

(1) *Sorts* $\iota, \iota_1, \ldots$ as names for spaces of abstract mathematical objects.

(2) *Terms* $(\vec{t})$ that include *variables*, *constants* and *function symbols* .

(3) *Predicate constants* of fixed arities $(\vec{\iota})$.

$$
\begin{array}{lll}
\textit{Formulas } \ni A, B & ::= & P(\vec{t}) \\
 & \mid & A \wedge B \mid A \vee B \mid A \to B \mid \forall x\, A \mid \exists x\, A \\
\textit{Predicates } \ni P, Q & ::= & X \mid P_0 \mid \lambda \vec{x}\, A \mid \mu\, \Phi \mid \nu\, \Phi \\
\textit{Operators } \ni \Phi, \Psi & ::= & \lambda X\, P \text{ } (P \text{ is strictly positive in } X)
\end{array}
$$



Intuitionistic Fixed Point Logic (IFP)

- Intuitionistic Predicate Logic

  Natural deduction with equality

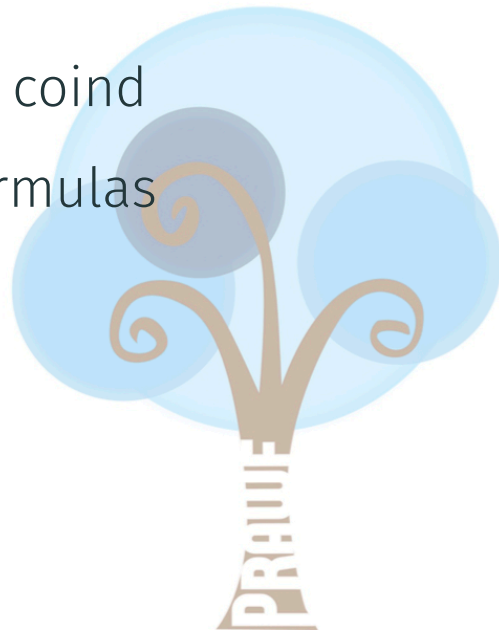
- Induction and Coinduction

$$\frac{}{\Phi(\mu\,\Phi) \subseteq \mu\,\Phi}\ \mathrm{cl} \qquad \frac{\Phi(P) \subseteq P}{\mu\,\Phi \subseteq P}\ \mathrm{ind}$$

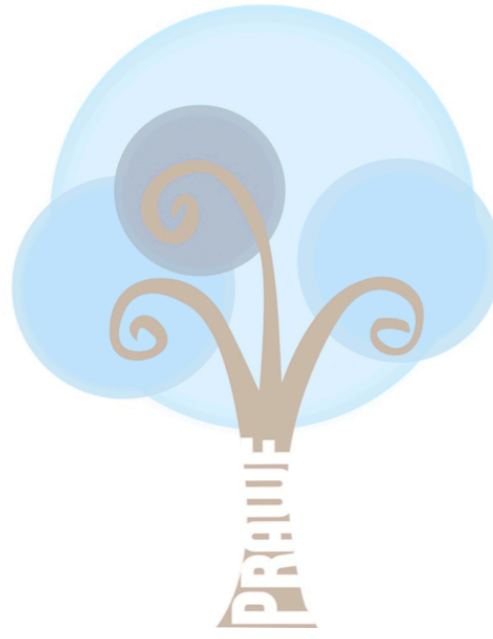
$$\frac{}{\nu\,\Phi \subseteq \Phi(\nu\,\Phi)}\ \mathrm{cocl} \qquad \frac{P \subseteq \Phi(P)}{P \subseteq \nu\,\Phi}\ \mathrm{coind}$$

- Axioms consisting of closed disjunction-free formulas

  e.g., $\forall x\,, y(x + y = y + x)$



Realizability



Realizability and RIFP

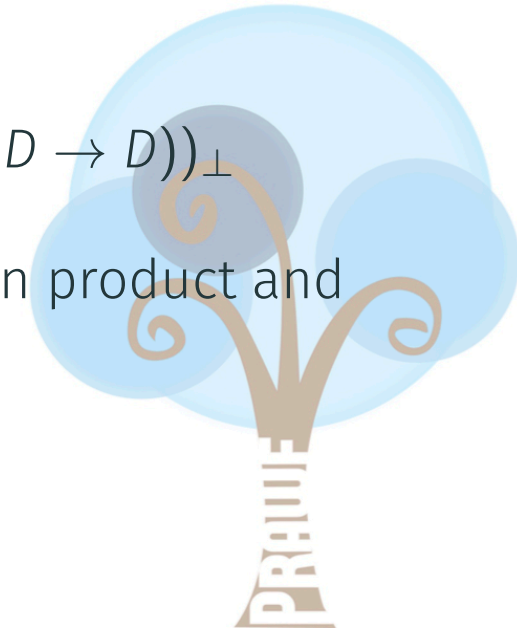
A *realizer* is an object that "realizes" a formula from a formal theory, i.e. serves as a confirmation of its truth.

IFP for Realizers (RIFP)

The Scott domain of realizers is defined by the recursive domain equation

$$D = (\mathbf{Nil} + \mathbf{Lt}(D) + \mathbf{Rt}(D) + \mathbf{Pair}(D \times D) + \mathbf{F}(D \to D))_\perp$$

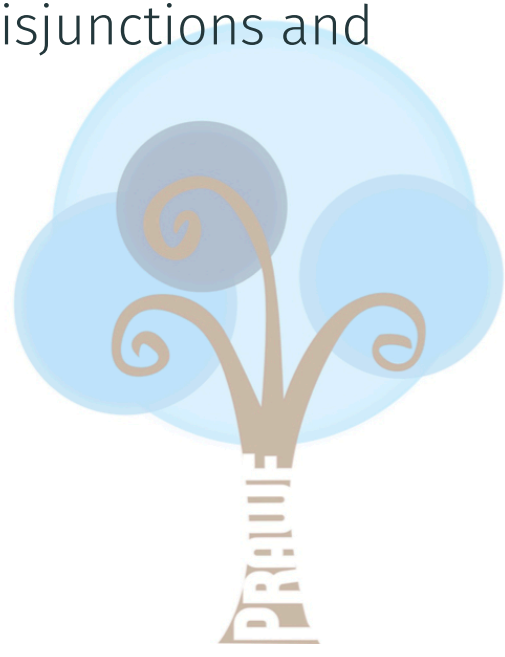
where $+$ denotes the disjoint union, $\times$ the Cartesian product and $D \to D$ is the continuous function space.



Non-computational and Harrop expressions

An expression is ***Harrop*** if it contains neither disjunction nor free predicate variable at a strictly positive position.

A ***non-computational*** expression contains no disjunctions and no free predicate variable.



Realizability interpretation

For every formula A we define, by structural recursion, a unary predicate

$$\mathsf{R}(A)$$

on D in the expected way.

$\mathsf{R}(A)(a)$ means a realizes A. We write $a \,\mathbf{r}\, A$.

Simultaneously, we define for every Harrop formula A a formula

$$\mathsf{H}(A)$$

$\mathsf{H}(A)$ can be thought of a $\mathsf{R}(A)$ with a suppressed realizer.

Realizability interpretation (details)

$$\mathbf{R}(A) = \lambda a\ (\mathbf{H}(A) \wedge a = \mathbf{Nil}) \qquad (A \text{ Harrop})$$

$$\mathbf{R}(P(t)) = \lambda a\ \mathbf{R}(P)(t,a) \qquad (P(t) \text{ non-Harrop; } P \text{ is not an abstraction})$$

$$\mathbf{H}(P(t)) = \mathbf{H}(P)(t) \qquad (P(t) \text{ Harrop})$$

$$\mathbf{R}(A \wedge B) = \lambda c\ (\mathbf{R}(A)(\pi_{\mathbf{Lt}} c) \wedge \mathbf{R}(B)(\pi_{\mathbf{Rt}} c)) \qquad (A, B \text{ non-Harrop})$$

$$\mathbf{R}(A \wedge B) = \lambda a\ (\mathbf{R}(A)(a) \wedge \mathbf{H}(B)) \qquad (A \text{ non-Harrop, } B \text{ Harrop})$$

$$\mathbf{R}(A \wedge B) = \lambda b\ (\mathbf{H}(A) \wedge \mathbf{R}(B)(b)) \qquad (A \text{ Harrop, } B \text{ non-Harrop})$$

$$\mathbf{H}(A \wedge B) = \mathbf{H}(A) \wedge \mathbf{H}(B) \qquad (A, B \text{ Harrop})$$

$$\mathbf{R}(A \vee B) = \lambda c\ (\exists a(c = \mathbf{Lt}(a) \wedge \mathbf{R}(A)(a)) \vee \exists b(c = \mathbf{Rt}(b) \wedge \mathbf{R}(B)(b)))$$

$$\mathbf{R}(A \to B) = \lambda f\ (\forall a(\mathbf{R}(A)(a) \to \mathbf{R}(B)(f\,a))) \qquad (A, B \text{ non-Harrop})$$

$$\mathbf{R}(A \to B) = \lambda b\ (\mathbf{H}(A) \to \mathbf{R}(B)(b)) \qquad (A \text{ Harrop, } B \text{ non-Harrop})$$

$$\mathbf{H}(A \to B) = \exists a\ \mathbf{R}(A)(a) \to \mathbf{H}(B) \qquad (B \text{ non-Harrop})$$

$$\mathbf{R}(\diamond x\, A) = \lambda a\ \mathbf{R}(\diamond x\ (A))(a) = \lambda a\ \diamond x\ (\mathbf{R}(A)(a)) \qquad (A \text{ non-Harrop; } \diamond \in \{\forall, \exists\})$$

$$\mathbf{H}(\diamond x\, A) = \diamond x\ \mathbf{H}(A) \qquad (A \text{ Harrop; } \diamond \in \{\forall, \exists\})$$

Realizability interpretation (details)

$$\mathbf{R}(P) = \lambda(\vec{x}, a)(\mathbf{H}(P) \wedge a = \mathbf{Nil}) \qquad (P \text{ Harrop})$$

$$\mathbf{R}(X) = \tilde{X} \qquad (X \text{ predicate variable})$$

$$\mathbf{H}(P_0) = P_0 \qquad (P_0 \text{ predicate constant})$$

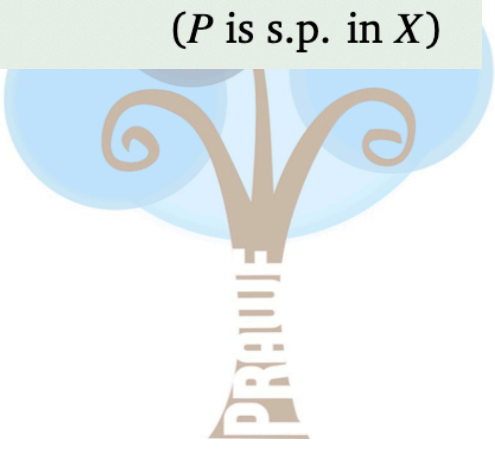
$$\mathbf{R}(\lambda\vec{x}\, A) = \lambda(\vec{x}, a)\ \mathbf{R}(A)(a) \qquad (\text{Abstraction})$$

$$\mathbf{R}(\diamond(\Phi)) = \diamond(\mathbf{R}(\Phi)) \qquad (\Phi \text{ non-Harrop}; \diamond \in \{\mu, \nu\})$$

$$\mathbf{H}(\diamond(\Phi)) = \diamond(\mathbf{H}(\Phi)) \qquad (\Phi \text{ Harrop}; \diamond \in \{\mu, \nu\})$$

$$\mathbf{R}(\lambda X\ P) = \lambda\tilde{X}\ \mathbf{R}(P) \qquad (P \text{ any predicate})$$

$$\mathbf{H}(\lambda X\ P) = \lambda X\ \mathbf{H}_X(P) \qquad (P \text{ is s.p. in } X)$$

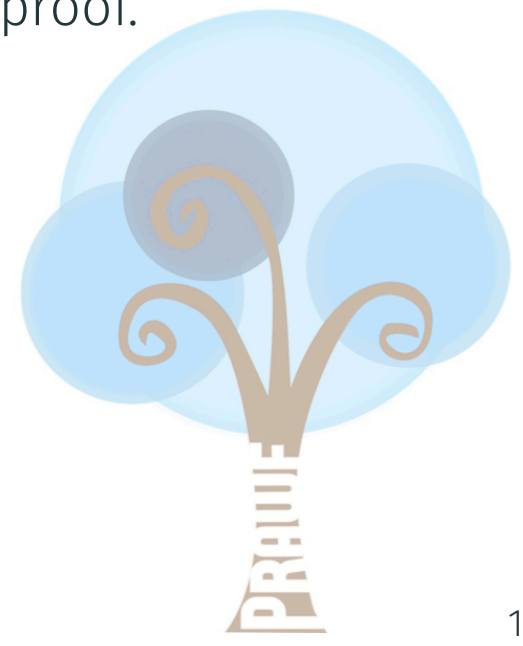


Soundness

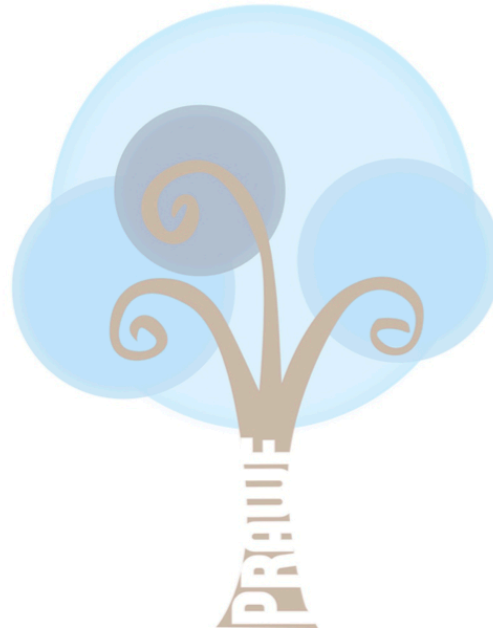
Theorem (Soundness)

If $\Gamma \vdash_{IFP} A$, where Γ consists of non-computational formulas, then $\Gamma \vdash_{RIFP} p\,\mathbf{r}\,A$ for some program p extracted from the proof.

Proof: By induction on the length of derivations.



Operational Semantics



Operational semantics: closures and values

A ***closure*** is, inductively, a pair of a program with the corresponding environment, for instance (M, η), where *environment* is a finite mapping from object variables to closures.

A ***value*** is a closure (M, η), where the program M begins with a constructor.

Big-step reduction rules

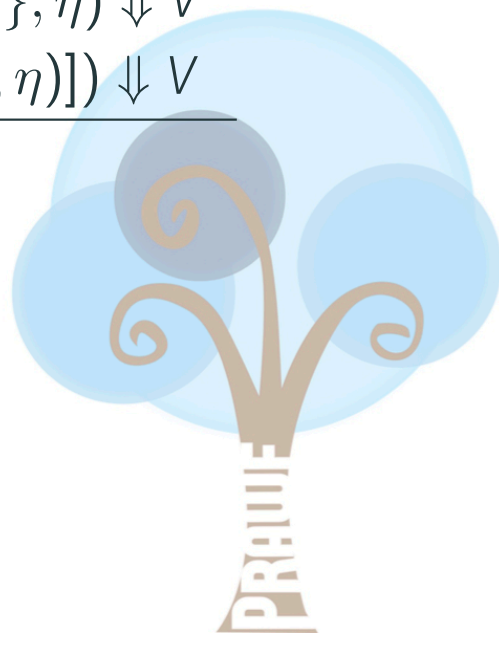
$$\text{(I, big)} : V \Downarrow V$$

$$\text{(II, big)} : \frac{\eta(a) \Downarrow V}{(a, \eta) \Downarrow V}.$$

$$\text{(III, big)} : \frac{\begin{array}{c}(M, \eta) \Downarrow (C(M_1, \ldots, M_k), \eta') \\ (N, \eta[u_1 \mapsto (M_1, \eta'), \ldots, u_k \mapsto (M_k, \eta')]) \Downarrow V\end{array}}{(\textsf{case } M \textsf{ of } \{\ldots C(u_1, \ldots, u_k) \to N; \ldots\}, \eta) \Downarrow V}$$

$$\text{(IV, big)} : \frac{(M, \eta) \Downarrow (\lambda a\, M', \eta') \qquad (M', \eta'[a \mapsto (N, \eta)]) \Downarrow V}{(MN, \eta) \Downarrow V}$$

$$\text{(V, big)} : \frac{(M(\textsf{rec } M), \eta) \Downarrow V}{(\textsf{rec } M, \eta) \Downarrow V}$$



Small-step reduction rules (old version)

(i) $V \rightsquigarrow V$

(ii) $(a, \eta) \rightsquigarrow \eta(a)$

(iii) $(\textbf{case}\ C(M_1, \ldots, M_k)\ \textbf{of}\ \{\ldots; C(u_1, \ldots, u_k) \to N; \ldots\}, \eta) \rightsquigarrow (\mathrm{N}, \eta[u_1 \mapsto (M_1, \eta), \ldots, u_k \mapsto (M_k, \eta)])$

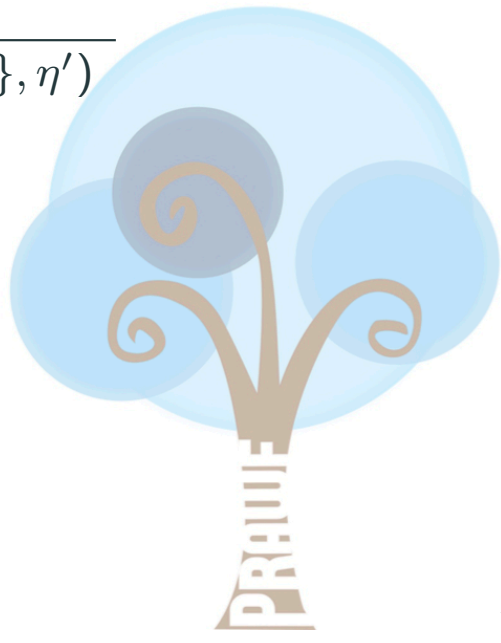
(iv) $((\lambda a\, M)\ N, \eta) \rightsquigarrow (M, \eta[a \mapsto (N, \eta)])$

(v) $(\textbf{rec}(\lambda a\, M), \eta) \rightsquigarrow (M, \eta[a \mapsto (\textbf{rec}(\lambda a\, M), \eta)])$

(vi) $$\frac{(M, \eta) \rightsquigarrow (M', \eta')}{(\textbf{case}\ M\ \textbf{of}\ \{Cl_1, \ldots, Cl_n\}, \eta) \rightsquigarrow (\textbf{case}\ M'\ \textbf{of}\ \{Cl_1, \ldots, Cl_n\}, \eta')}$$

(ix) $$\frac{(M, \eta) \rightsquigarrow (M', \eta')}{(MN, \eta) \rightsquigarrow (M', N, \eta')}$$

(x) $$\frac{(M, \eta) \rightsquigarrow (M', \eta')}{(\textbf{rec}(M)\, N, \eta) \rightsquigarrow (\textbf{rec}(M')\, N, \eta)}$$



Old rules with closures

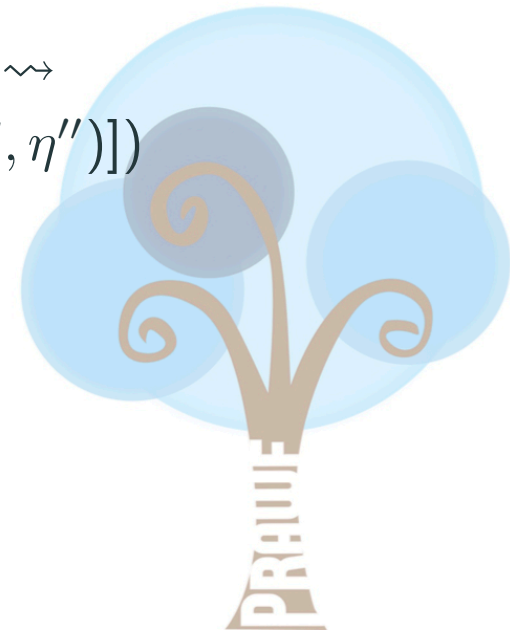
Rules with closures:

(iv) $((\lambda a\, M)\ N, \eta) \rightsquigarrow (M, \eta[a \mapsto (N, \eta)])$

(ix) $$\frac{(M, \eta) \rightsquigarrow (M', \eta')}{(MN, \eta) \rightsquigarrow (M'N, \eta')}$$

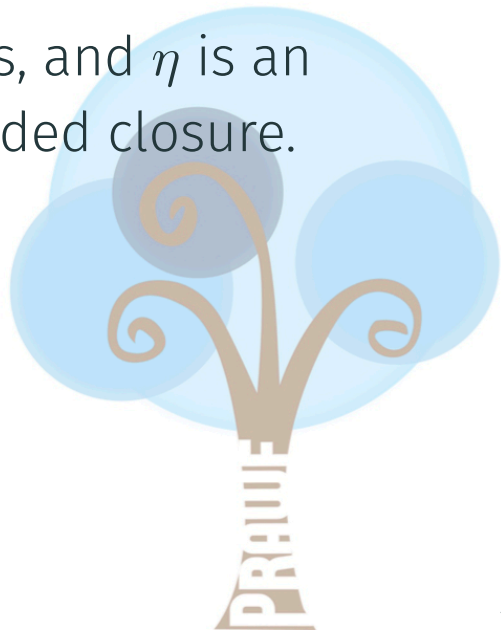
Applying rules:

$(MN, \eta) \rightsquigarrow$
$(M'N, \eta') \rightsquigarrow \ldots$
$((\lambda a\, M'')N, \eta'') \rightsquigarrow$
$(M'', \eta''[a \mapsto (N, \eta'')])$



Extended closures

- Every closure is an extended closure;
- if U is an extended closure and U_0 is a closure, then $U\,U_0$ is an extended closure;
- if U is an extended closure, $\vec{Cl}$ is a list of clauses, and η is an environment, then **case** U **of** $(\vec{Cl}, \eta)$ is an extended closure.



Small-step reduction rules

(II, small) : $(a, \eta) \rightsquigarrow \eta(a)$

(III a, small) : $(\textbf{case } M \textbf{ of } \{Cl_1, ..., Cl_n\}, \eta) \rightsquigarrow \textbf{case } (M, \eta) \textbf{ of } (\{Cl_1, ..., Cl_n\}, \eta)$

(III b, small) : $\textbf{case } (C(M_1, ..., M_k), \eta) \textbf{ of } (\{...; C(u_1, ..., u_k) \to N; ...\}, \eta_0) \rightsquigarrow (\mathrm{N}, \eta_0[u_1 \mapsto (M_1, \eta), ..., u_k \mapsto (M_k, \eta)])$

(III c, small) : $$\frac{U \rightsquigarrow U'}{\textbf{case } U \textbf{ of } (\{Cl_1, ..., Cl_n\}, \eta) \rightsquigarrow \textbf{case } U' \textbf{ of } (\{Cl_1, ..., Cl_n\}, \eta)}$$

(IV a, small) : $(MN, \eta) \rightsquigarrow (M, \eta)(N, \eta)$

(IV b, small) : $(\lambda a\, M, \eta)U_0 \rightsquigarrow (M, \eta[a \mapsto U_0])$

(IV c, small) : $$\frac{U \rightsquigarrow U'}{UU_0 \rightsquigarrow U'U_0}$$

(V, small) : $(\textbf{rec } M, \eta) \rightsquigarrow (M(\textbf{rec } M), \eta)$

New rules with extended closures

Rules with extended closures:

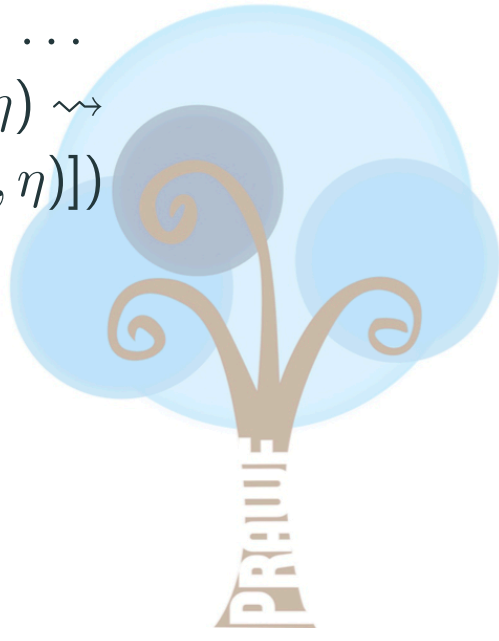
(a) $(MN, \eta) \rightsquigarrow (M, \eta)(N, \eta)$

(b) $(\lambda a\ M, \eta)U_0 \rightsquigarrow (M, \eta[a \mapsto U_0])$

(c) $$\frac{U \rightsquigarrow U'}{UU_0 \rightsquigarrow U'U_0}$$

Applying rules:

$(MN, \eta) \rightsquigarrow$
$(M, \eta)(N, \eta) \rightsquigarrow$
$(M', \eta')(N, \eta) \rightsquigarrow \ldots$
$(\lambda a\ M'', \eta'')(N, \eta) \rightsquigarrow$
$(M'', \eta''[a \mapsto (N, \eta)])$



Small-step reduction rules with substitution

(i) $\mathbf{case}\, C(\vec{M})\, \mathbf{of}\, \{\ldots; C(\vec{y}) \to N; \ldots\} \rightsquigarrow N[\vec{M}/\vec{y}]$

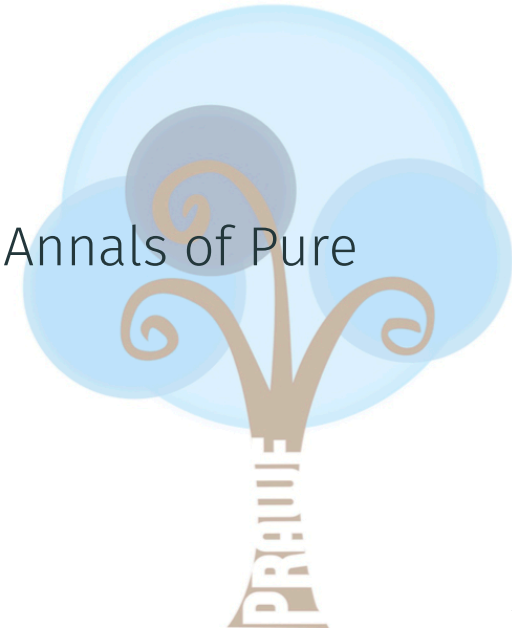
(ii) $(\lambda x.\, M)\ N \rightsquigarrow M[N/x]$

(iii) $\mathbf{rec}\, M \rightsquigarrow M\, (\mathbf{rec}\, M)$

(iv) $$\frac{M \rightsquigarrow M'}{\mathbf{case}\, M\, \mathbf{of}\, \{\vec{C}l\} \rightsquigarrow \mathbf{case}\, M'\, \mathbf{of}\, \{\vec{C}l\}}$$

(v) $$\frac{M \rightsquigarrow M'}{M\, N \rightsquigarrow M'\, N}$$

(U. Berger, H. Tsuiki: Intuitionistic Fixed Point Logic. Annals of Pure and Applied Logic 172.3 (2021): 102903.)

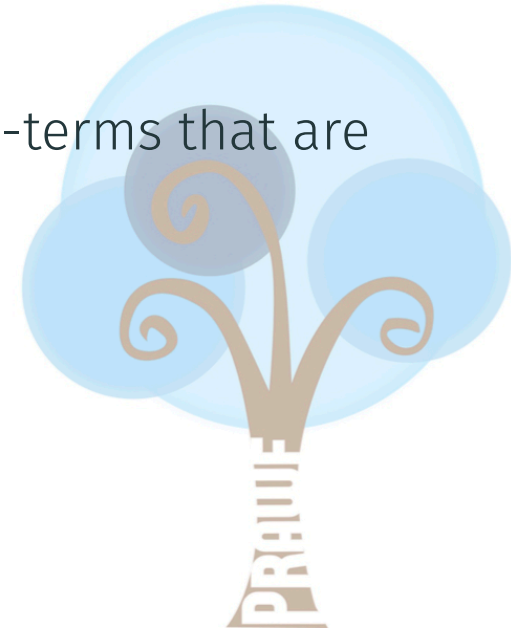


Adequacy

Theorem (Computational Adequacy)
If $[\![M]\!] \in E$*, then* $M^{(\infty)} = [\![M]\!]$*, where*

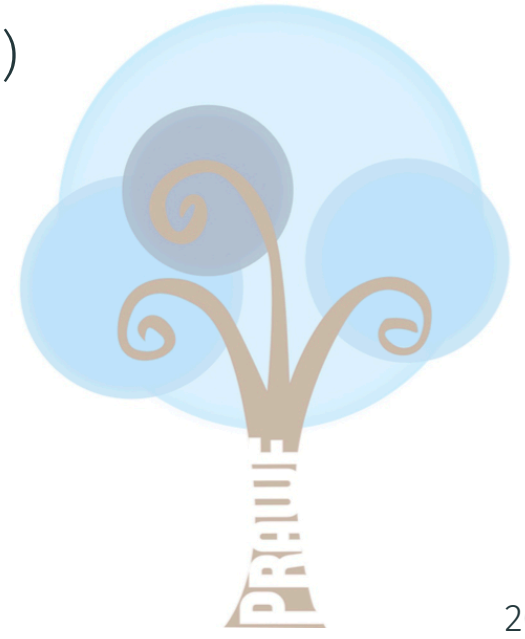
$$E = (\mathbf{Nil} + \mathbf{Lt}(E) + \mathbf{Rt}(E) + \mathbf{Pair}(E \times E))_{\perp}$$

$M^{(\infty)}$ is the infinite normal form of M, where all sub-terms that are not constructor terms are replaced by $\perp$.



Future work

- Extraction of proof of correctness with the program
- Developing a substantial theorems database in Prawf
- Extension for CFP (Concurrent Fixed Point Logic)



Happy Birthday!