An Improved Proof-Theoretic Compilation of Logic Programs

Iliano Cervesato

Carnegie Mellon University
Qatar

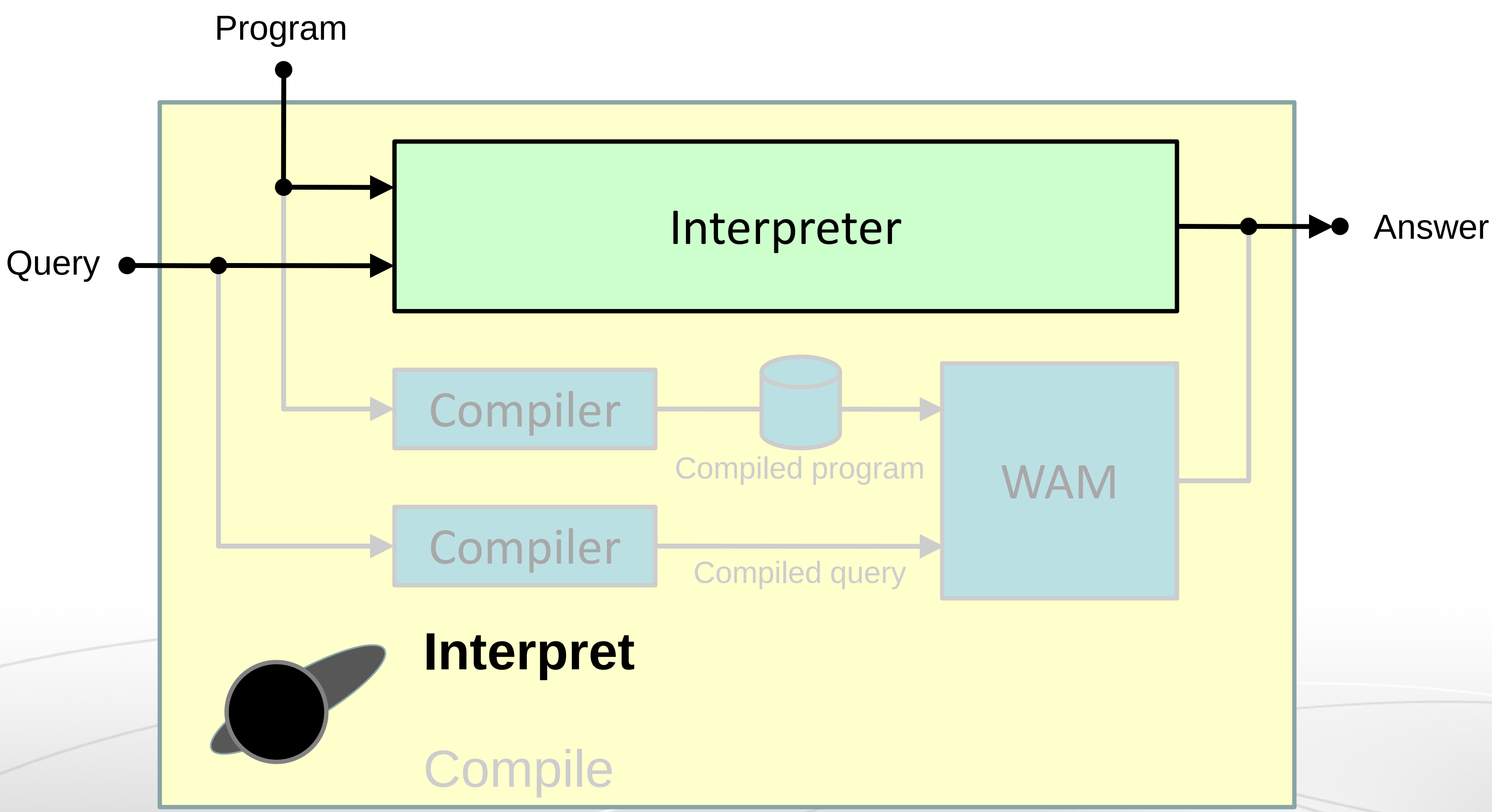
ICLP 2012
Budapest, Hungary – 8 September 2012

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Overview

- Motivations and background
- Abstract logic programming compilation
- Better compilation
- Moded compilation

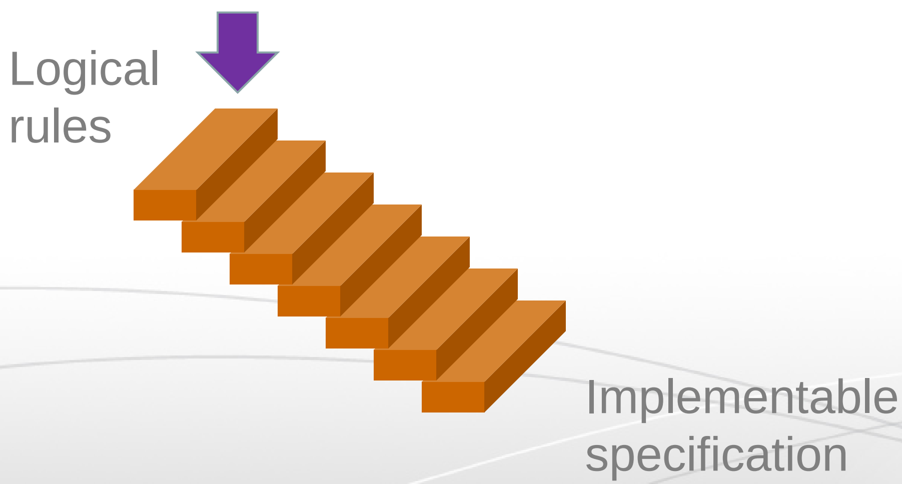
Architecture of a Prolog System



Proof-Theoretic Interpretation

- **Program/queries**: Logical formulas
- **Answers**: Derivability
- **Semantics**: Uniform proof search

ALPL



ALPL [Miller, Nadathur, Pfenning & Scedrov, 91]

- Computation = proof search
  - Connectives in A:
    search directives
  - Clauses in Γ: spec. of how to continue the search when the goal is atomic

$$\Gamma \rightarrow A$$

Program   Goal

- Uniform proofs
  - Goal oriented $\Gamma \longrightarrow A$
  - Focused $\Gamma \longrightarrow B \gg a$

In an ALPL, every provable sequent has a uniform proof

Hereditary Harrop Formulas

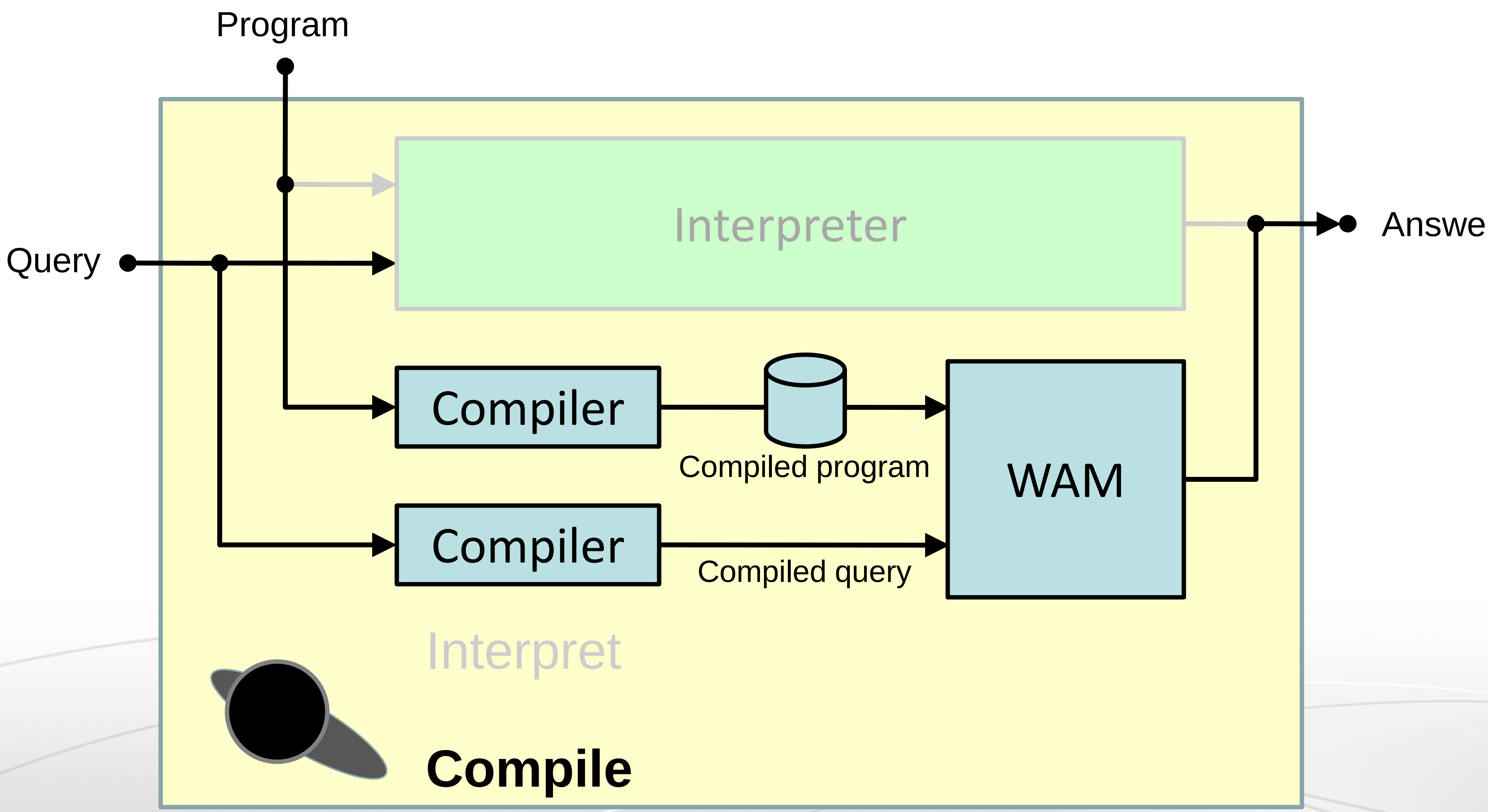
$A ::= a \mid A \supset B \mid \forall x.\, A$ $\qquad$ $a ::= p \mid a\ t$

$\dfrac{\Gamma, A \longrightarrow A \gg a}{\Gamma, A \longrightarrow a}$	$\dfrac{\Gamma, A \longrightarrow B}{\Gamma \longrightarrow A \supset B}$	$\dfrac{c \text{ “new”} \quad \Gamma \longrightarrow [c/x]A}{\Gamma \longrightarrow \forall x.\, A}$
$\dfrac{}{\Gamma \longrightarrow a \gg a}$	$\dfrac{\Gamma \longrightarrow A \quad \Gamma \longrightarrow B \gg a}{\Gamma \longrightarrow A \supset B \gg a}$	$\dfrac{\Gamma \longrightarrow [t/x]A \gg a}{\Gamma \longrightarrow \forall x.\, A \gg a}$

Non-determinism

- Term language is left unspecified
  - Must be predicative

Architecture of a Prolog System



WAM [Warren, 83]

- Interprets a specialized instruction set for Prolog
- Very fast (~40 times)
- Complex
- Specialized to Prolog
  - Then extended to CLP(R), PROTOS-L, λProlog

- <u>No logical status</u>
  - Where do the instructions come from?
  - What does it do?

Correctness of the WAM (as of 1998)

[Russinoff, 92] [Börger & Rosenzweig, 95]

- Starts from highly operational spec. of Prolog's semantics
- Complex
- Do not scale to modern logic programming language

Proof-Theoretic Compilation (JICSLP'98)

- Logic-based
  - Transformation between ALPLs
  - *Target language is an ALPL*
- Logic-independent
  - Applies to any ALPL
- Systematic
  - Easy proofs of correctness
- Abstract and modular
  - Manages gory details
- Used in Twelf and LLF

Proof-Theoretic Foundation of Compilation in Logic Programming Languages

Iliano Cervesato
Department of Computer Science
Stanford University
Stanford, CA 94305-9045
iliano@cs.stanford.edu

Abstract

Commercial implementations of logic programming languages are engineered around a compiler based on Warren's Abstract Machine (WAM) or a variant of it. In spite of various correctness proofs, the logical machinery relating the proof-theoretic specification of a logic programming language and its compiled form is still poorly understood. In this paper, we propose a logic-independent definition of compilation for logic programming languages. We apply this methodology to derive the first cut of a compiler and the corresponding abstract machine for the language of hereditary Harrop formulas and then for its linear refinement.

1 Introduction

Compiled logic programs run over an order of magnitude faster than their interpreted source and constitute therefore a key step to combining the advantages of the declarative nature of logic programming with the efficiency requirements of full-scale applications. For this reason, commercial implementations of logic programming languages come equipped with a compiler to translate a source program into an intermediate language, and an abstract machine to execute this compiled code efficiently. Most systems are based on *Warren's Abstract Machine* (WAM) [1, 22], first developed for *Prolog*. The WAM has now been adapted to other logic programming languages such as *CLP*($\mathcal{R}$) [10] and *PROTOS-L* [2]. Extensions to *λProlog* [14] are under way [12, 16, 17], but no similar effort has been undertaken for other advanced logic programming languages such as *Lolli* [9] or *Elf* [20].

Warren's work appears as a carefully engineered construction, but, for its very pioneering nature, it lacks any logical status. This contrasts strongly with the deep roots that the interpretation semantics of logic programming has in logic and proof-theory [15]. Indeed, the instruction set of the WAM hardly bears any resemblance to the connectives of the logic underlying *Prolog* and seems highly specialized to this language. As a result, the WAM "resembles an intricate puzzle, whose many pieces fit tightly together in a

Appeared in the *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming — JICSLP'98* (J. Jaffar editor), pp ??–??, MIT Press, Manchester, UK, 16–19 June 1998.

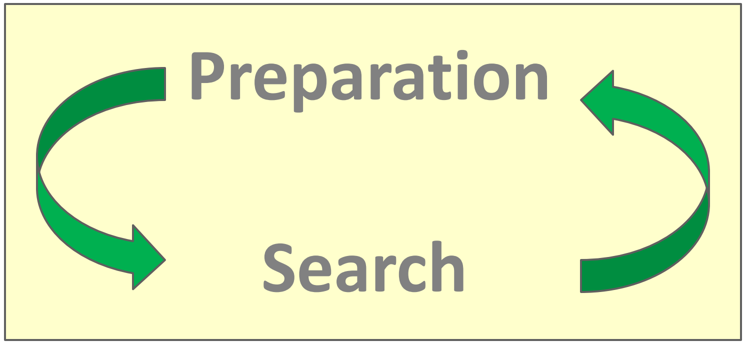
Abstract Compilation

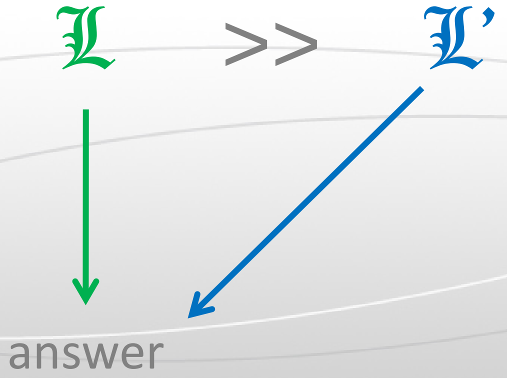
Uniform proofs alternate

- Goal decomposition
- Clause decomposition

Compilation

- First, ***all*** preparation
- Then, ***all*** search





$\mathfrak{L}'$ must be an ALPL
with *right rules only*

Determining the Target ALPL

- Keep every right rule of $\mathbb{L}$
- Add operators that behave on the right like the connectives of $\mathbb{L}$ on the left

$$\frac{\Gamma \rightarrow A \quad \Gamma \rightarrow B \succ a}{\Gamma \rightarrow A \supset B \succ a} \quad >> \quad \frac{\Gamma \rightarrow A \quad \Gamma \rightarrow B}{\Gamma \rightarrow A \wedge B}$$

$$\frac{\Gamma \rightarrow [t/x]A \succ a}{\Gamma \rightarrow \forall x.\, A \succ a} \quad >> \quad \frac{\Gamma \rightarrow [t/x]A}{\Gamma \rightarrow \exists x.\, A}$$

$$\frac{}{\Gamma \rightarrow a \succ a} \quad >> \quad \frac{}{\Gamma \rightarrow a = a}$$

- Logical principle
  - Currying
- Parameterize w.r.t. $\succ a$

Compiled HHF

$G ::= a \mid (\alpha.C) \supset G \mid \forall x.\, G$ $\quad\quad \Psi ::= .\ \mid \alpha.C, \Psi$

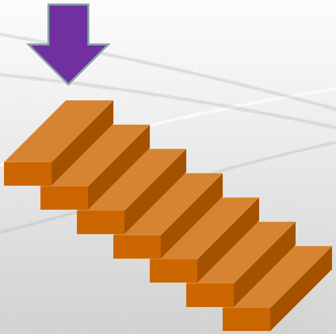
$C ::= \alpha = a \mid C \wedge G \mid \exists x.\, C$

$$\frac{\Psi, \alpha.C \rightarrow [a/\alpha]C}{\Psi, \alpha.C \rightarrow a} \qquad \frac{\Psi, \alpha.C \rightarrow G}{\Psi \rightarrow \alpha.C \supset G} \qquad \frac{c \text{ “new”} \quad \Psi \rightarrow [c/x]G}{\Psi \rightarrow \forall x.\, G}$$

$$\frac{}{\Psi \rightarrow a = a} \qquad \frac{\Psi \rightarrow C \quad \Psi \rightarrow G}{\Psi \rightarrow C \wedge G} \qquad \frac{\Psi \rightarrow [t/x]C}{\Psi \rightarrow \exists x.\, C}$$

Non-determinism

- Non-determinism is preserved



Compilation

Horn clause

$$b \subset a_1 \subset \ldots \subset a_n \quad >> \quad \alpha.\,(\alpha = b \wedge a_1 \wedge \ldots \wedge a_n)$$

- Atoms are not touched

$\dfrac{}{a >> a}$	$\dfrac{A >> \alpha \setminus C \qquad B >> G}{A \supset B >> \alpha.C \supset G}$	$\dfrac{A >> G}{\forall x.\, A >> \forall x.\, G}$
$\dfrac{}{a >> \alpha \setminus \alpha = a}$	$\dfrac{B >> \alpha \setminus C \qquad A >> G}{A \supset B >> \alpha \setminus C \wedge G}$	$\dfrac{A >> G}{\forall x.\, A >> \alpha \setminus \exists x.\, G}$
	$\dfrac{}{.\ >>\ .}$	$\dfrac{\Gamma >> \Psi \qquad A >> \alpha \setminus C}{\Gamma, A >> \Psi, C}$

Concrete Example

$\forall L.$ append nil L L

$\forall K. \forall L. \forall M. \forall X.$
  append (X::K) L (X::M)
  $\subset$ append K L M

$\Downarrow$

$\alpha. \exists L. \alpha$ = (append nil L L)

$\alpha. \exists K. \exists L. \exists M. \exists X.$
  α = (append (X::K) L (X::M))
  $\wedge$ append K L M

```
append1:
        ALLOCATE  L
        UNIFY (append nil L L)
append2:
        ALLOCATE  K
        ALLOCATE  L
        ALLOCATE  M
        ALLOCATE  X
        UNIFY (append (X::K) L (X::M))
        CALL (append K L M)
```

Meta-Theory

$$\begin{array}{ccc} \Gamma & \rightarrow & A \\ \vee\!\vee & & \vee\!\vee \\ \Psi & \rightarrow & G \end{array}$$

Soundness

1. If $\Gamma \rightarrow A$ and
   $\Gamma >> \Psi$ and $A >> G$,
   then $\Psi \rightarrow G$
2. If $\Gamma \rightarrow A \succ a$ and
   $\Gamma >> \Psi$ and $A >> \alpha \setminus C$,
   then $\Psi \rightarrow [a/\alpha]C$

Proof: structural induction

Completeness

1. If $\Psi \rightarrow G$ and
   $\Gamma >> \Psi$ and $A >> G$,
   then $\Gamma \rightarrow A$
2. If $\Psi \rightarrow R$ and
   $R = [a/\alpha]C$ and
   $\Gamma >> \Psi$ and $A >> \alpha \setminus C$,
   then $\Gamma \rightarrow A \succ a$

Proof: structural induction

What is α?

- An ad-hoc second-order mechanism
  - Works very well operationally, but
  - what is its logical status?

- Can we engineer a fully logical compilation scheme?

Idea: Use Term-Level Equality

Generic Horn clause

$$\forall \mathbf{y}.\ (p\ \mathbf{t} \subset a_1 \subset \dots \subset a_n)$$

$$\forall \mathbf{x}.\ (p\ \mathbf{x} \subset \exists \mathbf{y}.\ (\mathbf{x}{=}\mathbf{t} \wedge a_1 \wedge \dots \wedge a_n))$$

- This is currying again, but respecting dependencies
- Compiled head always matches goal for p
- Compiled clauses have the form $\forall \mathbf{x}.\ (p\ \mathbf{x} \subset R)$

$\forall \mathbf{x}.\ (p\ \mathbf{x} \subset R)$

Compiled HHF (2)

$$\frac{\Psi, C \to C \gg a}{\Psi, C \to a}\ * \qquad \frac{\Psi, C \to G}{\Psi \to C \supset G} \qquad \frac{c \text{ “new”} \quad \Psi \to [c/x]G}{\Psi \to \forall x.\ G}$$

$$\frac{\Psi \to R}{\Psi \to R \supset a \gg a}\ * \qquad \frac{\Psi \to [t/x]C \gg a}{\Psi \to \forall x.\ C \gg a}\ *$$

$$\frac{}{\Psi \to t = t} \qquad \frac{\Psi \to R \quad \Psi \to G}{\Psi \to R \wedge G} \qquad \frac{\Psi \to [t/x]R}{\Psi \to \exists x.\ R}$$

- Non-determinism is still preserved
- Minor infrastructure to produce **x=t**
- Remains sound and complete

Macro-Rule

- Builds uniform proofs
  - Necessary sequence of steps to use compiled clause

$$\frac{\Psi\ ,\ \forall \mathbf{x}.(p\ \mathbf{x} \subset R) \rightarrow [\mathbf{t}/\mathbf{x}]R}{\Psi,\ \forall \mathbf{x}.(p\ \mathbf{x} \subset R) \rightarrow p\ \mathbf{t}}$$

  - The backchaining rule
  - View $\forall \mathbf{x}.(p\ \mathbf{x} \subset R)$ as *synthetic connective* $\Lambda_p \mathbf{x}\ .\ R$
    - That's our old α

Concrete Example (2)

$$\forall K.\ \forall L.\ \forall M.\ \forall X.$$
$$\quad \text{append } (X::K)\ L\ (X::M)$$
$$\quad \subset\ \text{append } K\ L\ M$$

$$\forall x_1.\ \forall x_2.\ \forall x_3.$$
$$\quad \text{append } x_1\ x_2\ x_3$$
$$\quad \subset \exists K.\ \exists L.\ \exists M.\ \exists X.$$
$$\qquad (x_1 = (X::K) \quad \wedge$$
$$\qquad x_2 = L \quad \wedge$$
$$\qquad x_3 = (X::M) \quad \wedge$$
$$\qquad \text{append } K\ L\ M)$$

$$\forall L.\ \text{append nil } L\ L$$

$$\forall x_1.\ \forall x_2.\ \forall x_3.$$
$$\quad \text{append } x_1\ x_2\ x_3$$
$$\quad \subset\ \exists L.\ (x_1 = \text{nil} \quad \wedge$$
$$\qquad x_2 = L \quad \wedge$$
$$\qquad x_3 = L)$$

Moded Programs

- Arguments are labeled as input or output
  - Input are ground at call time
  - Output made ground upon return
  - Simple static check
- Moded semantics is based on matching *not unification*
  - Faster for first-order terms (no occurs-check)
  - Decidable for higher-order term
- Sufficient for CLF

Moded Execution

Data flow

Generic Horn clause

$$\forall \mathbf{y}.\ (p\ \mathbf{t}\ \mathbf{t} \subset q_1\ \mathbf{t_1}\ \mathbf{t_1} \subset \ldots \subset q_n\ \mathbf{t_n}\ \mathbf{t_n})$$

Control flow

$$\forall \mathbf{x}\ \mathbf{x}.\ (p\ \mathbf{x}\ \mathbf{x} \subset \exists \mathbf{y}.\ (\mathbf{x{=}t} \wedge \mathbf{x{=}t} \wedge q_1\ \mathbf{t_1}\ \mathbf{t_1} \wedge \ldots \wedge q_n\ \mathbf{t_n}\ \mathbf{t_n}))$$

- **x=t** *matches* input
- **t=x** *assigns* output

Moded Execution

Data flow

Generic Horn clause

$$\forall y. (p\ t\ t \subset q_1\ t_1\ t_1 \subset \dots \subset q_n\ t_n\ t_n)$$

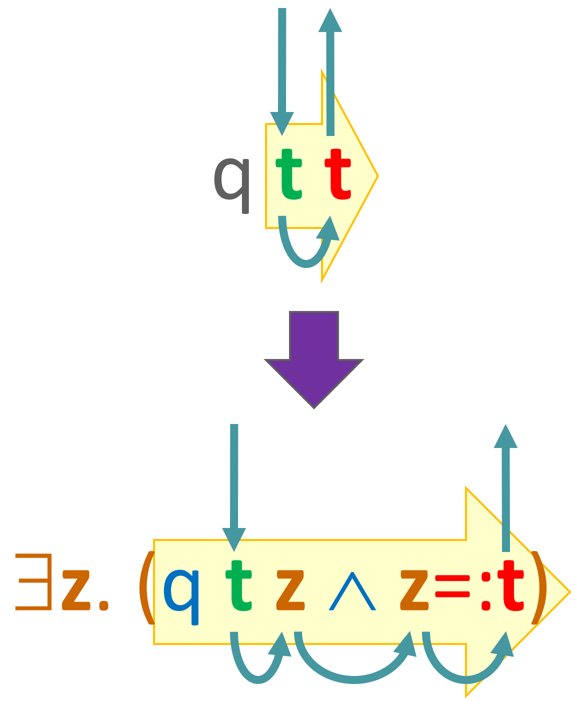
Control flow

$$\forall x\ x. (p\ x\ x \subset \exists y. (x =: t \wedge q_1\ t_1\ t_1 \wedge \dots \wedge q_n\ t_n\ t_n \wedge x := t))$$

- $=:$ *matching* operator – for well-moded programs
- $:=$ *assignment* operator

Moded Atomic Goals

Data flow



Control flow

- Solving an atomic goal is like a function call
  - Non-deterministic partial function

Compiled HHF (3)

$$\frac{\Psi, C \to C \gg a}{\Psi, C \to a}\,{**} \qquad \frac{\Psi, C \to G}{\Psi \to C \supset G} \qquad \frac{c \text{ “new”} \quad \Psi \to [c/x]G}{\Psi \to \forall x.\, G}$$

$$\frac{\Psi \to R}{\Psi \to R \supset a \gg a}\,{*} \qquad \frac{\Psi \to [t/x]C \gg a}{\Psi \to \forall x.\, C \gg a}\,{*}$$

$$\frac{}{\Psi \to t =: t}\,{*} \qquad \frac{}{\Psi \to t := t} \qquad \frac{\Psi \to R \quad \Psi \to G}{\Psi \to R \wedge G}\,{*} \qquad \frac{\Psi \to [t/x]R}{\Psi \to \exists x.\, R}\,{*}$$

- Non-determinism is still preserved
- Additional compilation infrastructure needed
- Remains sound and complete

Consequences of Uniformity

- Two macro-rules

Backchaining

$$\frac{\Psi\ ,\ \forall\ldots \to [t/x,u/y]R}{\Psi,\ \forall x\ x.(p\ x\ x \subset \exists y.\ (R \wedge x := s)) \to p\ t\ [u/y]s}$$

Call

$$\frac{\Psi \to p\ t\ s}{\Psi \to \exists z.\ (p\ t\ z \wedge z =: s))}$$

- Two synthetic connectives
  - $\forall x\ x.(p\ x\ x \subset \exists y.\ (R \wedge x := s))$ as $\Lambda_p x\ .\ \exists y.\ (R\ ;\ \text{return}\ s)$
  - $\exists z.\ (p\ t\ z \wedge z =: s))$ as $\text{call}\ p\ t =: s$

Concrete Example (3)

$\forall K.\ \forall L.\ \forall M.\ \forall X.$

$\quad \text{append } (X::K)\ L\ (X::M)$

$\quad \subset\ \text{append } K\ L\ M$

$\vee$
$\vee$

$\forall x_1.\ \forall x_2.\ \forall x_3.$

$\quad \text{append } x_1\ x_2\ x_3$

$\quad \subset \exists K.\ \exists L.\ \exists M.\ \exists X.$

$\qquad (x_1 =: (X::K) \quad \wedge$

$\qquad x_2 =: L \quad \wedge$

$\qquad \exists z.\ (\text{append } K\ L\ z \wedge z =: M) \wedge$

$\qquad x_3 := (X::M))$

$\forall L.\ \text{append nil } L\ L$

$\vee$
$\vee$

$\forall x_1.\ \forall x_2.\ \forall x_3.$

$\quad \text{append } x_1\ x_2\ x_3$

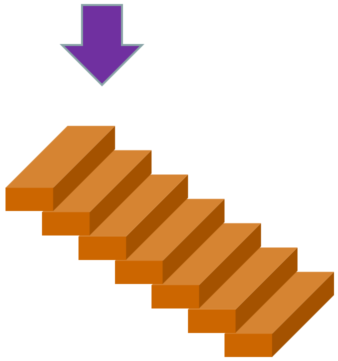
$\quad \subset\ \exists L.\ (x_1 =: \text{nil} \quad \wedge$

$\qquad x_2 =: L \quad \wedge$

$\qquad x_3 := L)$

Future Work

- Prove that matching/assignment are sufficient for moded programs
  - Make goal & term selection explicit
  - Declarative mode checking
  - Semi-functional interpretation



- Implement within CLF prototype
  - Backward + forward chaining semantics
  - Linear/affine operators
  - Complex higher-order term language

Thank you!

Questions?