

# Inductive Programming

## A Unifying Framework for Analysis and Evaluation of Inductive Programming Systems

Hofmann, Kitzelmann, Schmid

Cognitive Systems Group  
University of Bamberg



AGI 2009



# Inductive Program Synthesis (IP)

**Inductive Program Synthesis (IP)** researches the automatic construction of (recursive) programs from *incomplete* specifications, i.e. input/output examples (I/O examples)

## Example (reverse)

I/O-examples:

```
reverse [] = []  
reverse [a] = [a]  
reverse [a,b] = [b,a]  
reverse [a,b,c] = [c,b,a]
```

Induced functional program:

```
reverse [] = []  
reverse (x:xs) = (reverse xs) ++ [x]
```

# Key Concepts

**Preference Bias** **criteria** to choose among (**semantically** different!) candidate solutions, i.e syntactic size, number of case distinctions, runtime (search strategy).

**Restriction Bias** **Restricts** the inducible class of problems, through **syntactic** constraints, i.e. linear recursion as sole kind of recursion (hypotheses language)

**Background Knowledge** already **implemented functions**, which can be used for synthesis, i.e. `append` and `partition` for `quicksort`

**Sub Functions** Functions **neither** defined as **target functions** nor in the **background knowledge**, but automatically introduced as auxiliary functions by the IP algorithm

# Key Concepts

**Preference Bias criteria** to choose among (**semantically** different!) candidate solutions, i.e syntactic size, number of case distinctions, runtime (search strategy).

**Restriction Bias Restricts** the inducible class of problems, through **syntactic** constraints, i.e. linear recursion as sole kind of recursion (hypothes language)

**Background Knowledge** already **implemented functions**, which can be used for synthesis, i.e. `append` and `partition` for `quicksort`

**Sub Functions** Functions **neither** defined as **target functions** nor in the **background knowledge**, but automatically introduced as auxiliary functions by the IP algorithm

# Key Concepts

**Preference Bias** **criteria** to choose among (**semantically** different!) candidate solutions, i.e syntactic size, number of case distinctions, runtime (search strategy).

**Restriction Bias** **Restricts** the inducible class of problems, through **syntactic** constraints, i.e. linear recursion as sole kind of recursion (hypothes language)

**Background Knowledge** already **implemented functions**, which can be used for synthesis, i.e. `append` and `partition` for `quicksort`

**Sub Functions** Functions **neither** defined as **target functions** nor in the **background knowledge**, but automatically introduced as auxiliary functions by the IP algorithm

# Key Concepts

**Preference Bias** **criteria** to choose among (**semantically** different!) candidate solutions, i.e syntactic size, number of case distinctions, runtime (search strategy).

**Restriction Bias** **Restricts** the inducible class of problems, through **syntactic** constraints, i.e. linear recursion as sole kind of recursion (hypothes language)

**Background Knowledge** already **implemented functions**, which can be used for synthesis, i.e. `append` and `partition` for `quicksort`

**Sub Functions** Functions **neither** defined as **target functions** nor in the **background knowledge**, but automatically introduced as auxiliary functions by the IP algorithm

# Different Approaches

	<i>analytic</i>	<i>generate &amp; test</i>	
		<i>systematic</i>	<i>evolutionary</i>
<i>logic</i>	DIALOGS-II	FOIL/FFOIL, GOLEM	
<i>functional</i>	THESYS, IGOR I, IGOR II	MAGIC- HASKELLER	ADATE

Inductive Logic Programming (ILP)

Inductive Functional Programming (IFP)

# Different Approaches

	<i>analytic</i>	<i>generate &amp; test</i>	
		<i>systematic</i>	<i>evolutionary</i>
<i>logic</i>	DIALOGS-II	FOIL/FFOIL, GOLEM	
<i>functional</i>	THESYS, IGOR I, IGOR II	MAGIC- HASKELLER	A DATE

## Inductive Logic Programming (ILP)

- ILP is machine learning with representation and inference based on *Computational Logic* (PROLOG).
- IP as special case of ILP.

## Inductive Functional Programming (IFP)



# Different Approaches

	<i>analytic</i>	<i>generate &amp; test</i>	
		<i>systematic</i>	<i>evolutionary</i>
<i>logic</i>	DIALOGS-II	FOIL/FFOIL, GOLEM	
<i>functional</i>	THESYS, IGOR I, IGOR II	MAGIC- HASKELLER	ADATE

## Inductive Logic Programming (ILP)

## Inductive Functional Programming (IFP)

- Based *Term Rewriting* or *Combinatory Logic* /  $\lambda$ -calculus
- primary objective is program learning

# Different Approaches

	<i>analytic</i>	<i>generate &amp; test</i>	
		<i>systematic</i>	<i>evolutionary</i>
<i>logic</i>	DIALOGS-II	FOIL/FFOIL, GOLEM	
<i>functional</i>	THESYS, IGOR I, IGOR II	MAGIC- HASKELLER	ADATE

Analytic

Generate & Test

# Different Approaches

	<i>analytic</i>	<i>generate &amp; test</i>	
		<i>systematic</i>	<i>evolutionary</i>
<i>logic</i>	DIALOGS-II	FOIL/FFOIL, GOLEM	
<i>functional</i>	THESYS, IGOR I, IGOR II	MAGIC- HASKELLER	ADATE

## Analytic

- different **inputs are “sub problems” of each other**
- so their output is included in other outputs as subterms
- analyze I/Os and **fold regularities into a recursive definition**

## Generate & Test

# Different Approaches

	<i>analytic</i>	<i>generate &amp; test</i>	
		<i>systematic</i>	<i>evolutionary</i>
<i>logic</i>	DIALOGS-II	FOIL/FFOIL, GOLEM	
<i>functional</i>	THESYS, IGOR I, IGOR II	MAGIC- HASKELLER	ADATE

## Analytic

## Generate & Test (1): systematic

- **enumerate** all correct programs systematically
- constraints limit search space (type information, library, modes)
- I/Os are only used as **filter**

# Different Approaches

	<i>analytic</i>	<i>generate &amp; test</i>	
		<i>systematic</i>	<i>evolutionary</i>
<i>logic</i>	DIALOGS-II	FOIL/FFOIL, GOLEM	
<i>functional</i>	THESYS, IGOR I, IGOR II	MAGIC- HASKELLER	ADATE

## Analytic

## Generate & Test (2): evolutionary heuristic

- use **genetic coperators** to traverse search space
- **fitness function** maps programs to numeric space
- evaluated program attributes are e.g. runtime, program size, etc.

# Different Approaches

	<i>analytic</i>	<i>generate &amp; test</i>	
		<i>systematic</i>	<i>evolutionary</i>
<i>logic</i>	DIALOGS-II	FOIL/FFOIL, GOLEM	
<i>functional</i>	THESYS, IGOR I, IGOR II	MAGIC- HASKELLER	ADATE

large diversity of underlying theoretical concepts and requirements



hard to compare and evaluate

# Need for Unifying Framework

**Provide system independent syntax and operational semantics**

## Benefits

- + consistent **representation** of different target languages
- + gives a **unifying** (“normalised”) **perspective** on IP systems
- + helps **identifying** system specific **strength and weaknesses**
- + provide a **transparent evaluation and comparison** of IP systems
- + basis for a **general IP algorithm**
- + means for an abstract problem definition language  
(**IP Problem Definition Language**)

## Conditional Constructor (Rewrite) Systems (CCS)

# Need for Unifying Framework

**Provide system independent syntax and operational semantics**

## Benefits

- + consistent **representation** of different target languages
- + gives a **unifying (“normalised”) perspective** on IP systems
- + helps **identifying** system specific **strength and weaknesses**
- + provide a **transparent evaluation and comparison** of IP systems
- + basis for a **general IP algorithm**
- + means for an abstract problem definition language  
(**IP Problem Definition Language**)

## Conditional Constructor (Rewrite) Systems (CCS)



# Need for Unifying Framework

**Provide system independent syntax and operational semantics**

## Benefits

- + consistent **representation** of different target languages
- + gives a **unifying (“normalised”) perspective** on IP systems
- + helps **identifying** system specific **strength and weaknesses**
- + provide a **transparent evaluation and comparison** of IP systems
- + basis for a **general IP algorithm**
- + means for an abstract problem definition language  
(**IP Problem Definition Language**)

## Conditional Constructor (Rewrite) Systems (CCS)

# Need for Unifying Framework

**Provide system independent syntax and operational semantics**

## Benefits

- + consistent **representation** of different target languages
- + gives a **unifying** (“**normalised**”) **perspective** on IP systems
- + helps **identifying** system specific **strength and weaknesses**
- + provide a **transparent evaluation and comparison** of IP systems
- + basis for a **general IP algorithm**
- + means for an abstract problem definition language  
(**IP Problem Definition Language**)

## Conditional Constructor (Rewrite) Systems (CCS)

# Need for Unifying Framework

**Provide system independent syntax and operational semantics**

## Benefits

- + consistent **representation** of different target languages
- + gives a **unifying** (“**normalised**”) **perspective** on IP systems
- + helps **identifying** system specific **strength and weaknesses**
- + provide a **transparent evaluation and comparison** of IP systems
- + basis for a **general IP algorithm**
- + means for an abstract problem definition language  
(**IP Problem Definition Language**)

## Conditional Constructor (Rewrite) Systems (CCS)

# Need for Unifying Framework

**Provide system independent syntax and operational semantics**

## Benefits

- + consistent **representation** of different target languages
- + gives a **unifying** (“**normalised**”) **perspective** on IP systems
- + helps **identifying** system specific **strength and weaknesses**
- + provide a **transparent evaluation and comparison** of IP systems
- + basis for a **general IP algorithm**
- + means for an abstract problem definition language  
(**IP Problem Definition Language**)

## Conditional Constructor (Rewrite) Systems (CCS)

# Need for Unifying Framework

**Provide system independent syntax and operational semantics**

## Benefits

- + consistent **representation** of different target languages
- + gives a **unifying** (“**normalised**”) **perspective** on IP systems
- + helps **identifying** system specific **strength and weaknesses**
- + provide a **transparent evaluation and comparison** of IP systems
- + basis for a **general IP algorithm**
- + means for an abstract problem definition language  
(**IP Problem Definition Language**)

## Conditional Constructor (Rewrite) Systems (CCS)

# The paper at one glance

	$\mathcal{C}$	$\mathcal{F}_T$	$\mathcal{F}_B$	$\mathcal{F}_I$	$E^+$	$E^-$	$BK$	$\mathcal{X}_2$	search strategy
ADATE	●	{·}	●	●	●	●	●	∅	global search, g 'n t
FLIP	●	●	●	∅	○	○, ∅	●	∅	sequential covering
FFOIL	$\mathcal{C}$	●	⊃	∅	○	○, ∅	○	∅	sequential covering
GOLEM	●	{·}	●	∅	○	○	●	∅	sequential covering
IGOR I	●	{·}	∅	●	○	∅	∅	∅	2-step, global search
IGOR II	●	●	●	●	○	∅	○	∅	global search
MAGH.	●	{·}	●	∅	●	●	●	○	breadth first, g 'n t

●	unrestricted / conditional rules	○	restricted / unconditional rules
{·}	singleton set	∅	empty set
$\mathcal{C}$	constants	⊃	built in predicates

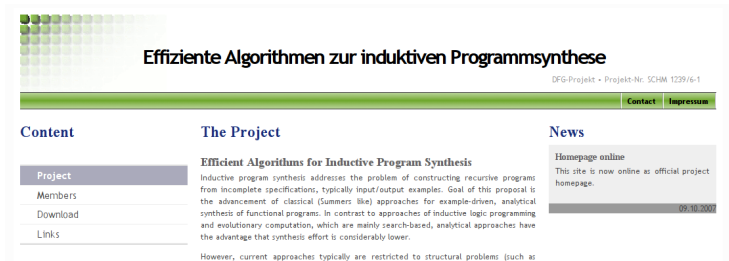
# Empirical Results

	<i>isort</i>	<i>reverse</i>	<i>weave</i>	<i>shiftr</i>	<i>mult/add</i>	<i>allodds</i>
ADATE	70.0	78.0	80.0	18.81	—	214.87
FLIP	×	—	134.24 <sup>⊥</sup>	448.55 <sup>⊥</sup>	×	×
FFOIL	×	—	0.4 <sup>⊥</sup>	< 0.1 <sup>⊥</sup>	8.1 <sup>⊥</sup>	0.1 <sup>⊥</sup>
GOLEM	0.714	—	0.66 <sup>⊥</sup>	0.298	—	0.016 <sup>⊥</sup>
<b>IGOR II</b>	<b>0.105</b>	<b>0.103</b>	<b>0.200</b>	<b>0.127</b>	<b>⊙</b>	<b>⊙</b>
MAGH.	0.01	0.08	⊙	157.32	—	×

	<i>lasts</i>	<i>last</i>	<i>member</i>	<i>odd/even</i>	<i>multlast</i>
ADATE	822.0	0.2	2.0	—	4.3
FLIP	×	0.020	17.868	0.130	448.90 <sup>⊥</sup>
FFOIL	0.7 <sup>⊥</sup>	0.1	0.1 <sup>⊥</sup>	< 0.1 <sup>⊥</sup>	< 0.1
GOLEM	1.062	< 0.001	0.033	—	< 0.001
<b>IGOR II</b>	<b>5.695</b>	<b>0.007</b>	<b>0.152</b>	<b>0.019</b>	<b>0.023</b>
MAGH.	19.43	0.01	⊙	—	0.30

— not tested × stack overflow ⊙ timeout ⊥ wrong  
all runtimes in seconds

# Our Project



The screenshot shows a website with a green and white color scheme. At the top left is a decorative graphic of green circles. The main title is 'Effiziente Algorithmen zur induktiven Programmsynthese' in a bold, black font. Below the title, on the right, is the text 'DFG-Projekt • Projekt-Nr. SCHM 1239/6-1'. A green horizontal bar contains two buttons: 'Contact' and 'Impressum'. The page is divided into three columns. The left column is titled 'Content' and contains a vertical list of links: 'Project' (highlighted), 'Members', 'Download', and 'Links'. The middle column is titled 'The Project' and contains the heading 'Efficient Algorithms for Inductive Program Synthesis' followed by a paragraph of text. The right column is titled 'News' and contains the heading 'Homepage online' followed by a short announcement and the date '09.10.2007'.

**Effiziente Algorithmen zur induktiven Programmsynthese**  
DFG-Projekt • Projekt-Nr. SCHM 1239/6-1

**Contact** **Impressum**

**Content**

- Project
- Members
- Download
- Links

**The Project**

**Efficient Algorithms for Inductive Program Synthesis**  
Inductive program synthesis addresses the problem of constructing recursive programs from incomplete specifications, typically input/output examples. Goal of this proposal is the advancement of classical (Summers like) approaches for example-driven, analytical synthesis of functional programs. In contrast to approaches of inductive logic programming and evolutionary computation, which are mainly search-based, analytical approaches have the advantage that synthesis effort is considerably lower.

However, current approaches typically are restricted to structural problems (such as

**News**

**Homepage online**  
This site is now online as official project homepage.  
09.10.2007

<http://www.cogsys.wiai.uni-bamberg.de/effalip/>

- Publications
- Downloads
- Links



# inductive-programming.org

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,  $f(0)=1$   
4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418,  $f(1)=1$   
317811, 514229, **inductive-programming.org**, 798089, 1258686, 2019077, 3194037, 5014907, 7778742, 12075216,  $f(x)=f(x-1)+$   
227465, 1493035, 227465, 1493035, 3542229, 542229, 841313, 1274691, 196418, 30354229, 46368, 75025, 121393, 196418,  $f(x-2)$   
165580141, 267914296, 433494437, 701408733, 1134903170, 18  
271180329, 4201913073, 6507528058, 10000473049, 15599067903

Main Introduction People Resources Events Links News Impressum Contact Mailing List

## Content

Main  
Introduction  
People

## Welcome

Welcome to *inductive-programming.org*, the online platform of the Inductive Programming community. At the workshop for "Approaches and Applications of Inductive Programming" at ECML 2007 in Warsaw, its participants agreed in the need for an online Inductive programming portal to promote the research field of IP at large and create a contact point for everybody interested in IP in particular.

## News

### New IP Logo

We are proud to present our new IP logo. It depicts the stylised acronym

<http://www.inductive-programming.org>

- Introduction to IP
- Systems' overview
- Repository with benchmark problems
- IP related publications
- Mailing list
- ...

Thank you  
for your attention!

Questions?  
Questions?  
Questions?  
Questions?  
Questions?  
Questions?  
Questions?

# CCS in a nutshell

- given a set of function symbols  $\Sigma$  and a set of variables  $\mathcal{X}$
- terms over  $\Sigma$  and  $\mathcal{X}$  denoted as  $\mathcal{T}_\Sigma(\mathcal{X})$
- *constructors*  $\mathcal{C}$  and *defined function symbols*  $\mathcal{F}$   
 $\Sigma = \mathcal{F} \cup \mathcal{C}, \mathcal{F} \cap \mathcal{C} = \emptyset$
- programs are sets of rewrite rules  $lhs \rightarrow rhs$
- $lhs$  is of the form  $F(p_1, \dots, p_n)$  with  $F \in \mathcal{F}$  and  $p_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- conditional rewrite rules  $lhs \rightarrow rhs \Leftarrow cond$  where  
 $cond \equiv \{v_1 = u_1, \dots, v_n = u_n\}$  and  $v_i, u_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- rewriting binds free variables in  $v_i$ , modelling variable declaration, `let`- and `case`-expressions
- higher-order context with  $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$  and abstraction operator  $[-]$

# CCS in a nutshell

- given a set of function symbols  $\Sigma$  and a set of variables  $\mathcal{X}$
- terms over  $\Sigma$  and  $\mathcal{X}$  denoted as  $\mathcal{T}_\Sigma(\mathcal{X})$
- *constructors*  $\mathcal{C}$  and *defined function symbols*  $\mathcal{F}$   
 $\Sigma = \mathcal{F} \cup \mathcal{C}, \mathcal{F} \cap \mathcal{C} = \emptyset$
- programs are sets of rewrite rules  $lhs \rightarrow rhs$
- $lhs$  is of the form  $F(p_1, \dots, p_n)$  with  $F \in \mathcal{F}$  and  $p_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- conditional rewrite rules  $lhs \rightarrow rhs \Leftarrow cond$  where  
 $cond \equiv \{v_1 = u_1, \dots, v_n = u_n\}$  and  $v_i, u_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- rewriting binds free variables in  $v_i$ , modelling variable declaration, `let`- and `case`-expressions
- higher-order context with  $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$  and abstraction operator  $[-]$

# CCS in a nutshell

- given a set of function symbols  $\Sigma$  and a set of variables  $\mathcal{X}$
- terms over  $\Sigma$  and  $\mathcal{X}$  denoted as  $\mathcal{T}_\Sigma(\mathcal{X})$
- *constructors*  $\mathcal{C}$  and *defined function symbols*  $\mathcal{F}$   
 $\Sigma = \mathcal{F} \cup \mathcal{C}, \mathcal{F} \cap \mathcal{C} = \emptyset$
- programs are sets of rewrite rules  $lhs \rightarrow rhs$ 
  - $lhs$  is of the form  $F(p_1, \dots, p_n)$  with  $F \in \mathcal{F}$  and  $p_i \in \mathcal{T}_\mathcal{C}(\mathcal{X})$
  - conditional rewrite rules  $lhs \rightarrow rhs \Leftarrow cond$  where  $cond \equiv \{v_1 = u_1, \dots, v_n = u_n\}$  and  $v_i, u_i \in \mathcal{T}_\Sigma(\mathcal{X})$
  - rewriting binds free variables in  $v_i$ , modelling variable declaration, `let`- and `case`-expressions
  - higher-order context with  $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$  and abstraction operator  $[-]$

# CCS in a nutshell

- given a set of function symbols  $\Sigma$  and a set of variables  $\mathcal{X}$
- terms over  $\Sigma$  and  $\mathcal{X}$  denoted as  $\mathcal{T}_\Sigma(\mathcal{X})$
- *constructors*  $\mathcal{C}$  and *defined function symbols*  $\mathcal{F}$   
 $\Sigma = \mathcal{F} \cup \mathcal{C}, \mathcal{F} \cap \mathcal{C} = \emptyset$
- programs are sets of rewrite rules  $lhs \rightarrow rhs$
- $lhs$  is of the form  $F(p_1, \dots, p_n)$  with  $F \in \mathcal{F}$  and  $p_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- conditional rewrite rules  $lhs \rightarrow rhs \Leftarrow cond$  where  
 $cond \equiv \{v_1 = u_1, \dots, v_n = u_n\}$  and  $v_i, u_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- rewriting binds free variables in  $v_i$ , modelling variable declaration, `let`- and `case`-expressions
- higher-order context with  $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$  and abstraction operator  $[-]$

# CCS in a nutshell

- given a set of function symbols  $\Sigma$  and a set of variables  $\mathcal{X}$
- terms over  $\Sigma$  and  $\mathcal{X}$  denoted as  $\mathcal{T}_\Sigma(\mathcal{X})$
- *constructors*  $\mathcal{C}$  and *defined function symbols*  $\mathcal{F}$   
 $\Sigma = \mathcal{F} \cup \mathcal{C}, \mathcal{F} \cap \mathcal{C} = \emptyset$
- programs are sets of rewrite rules  $lhs \rightarrow rhs$
- $lhs$  is of the form  $F(p_1, \dots, p_n)$  with  $F \in \mathcal{F}$  and  $p_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- conditional rewrite rules  $lhs \rightarrow rhs \Leftarrow cond$  where  
 $cond \equiv \{v_1 = u_1, \dots, v_n = u_n\}$  and  $v_i, u_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- rewriting binds free variables in  $v_i$ , modelling variable declaration, `let-` and `case-expressions`
- higher-order context with  $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$  and abstraction operator  $[-]$



# CCS in a nutshell

- given a set of function symbols  $\Sigma$  and a set of variables  $\mathcal{X}$
- terms over  $\Sigma$  and  $\mathcal{X}$  denoted as  $\mathcal{T}_\Sigma(\mathcal{X})$
- *constructors*  $\mathcal{C}$  and *defined function symbols*  $\mathcal{F}$   
 $\Sigma = \mathcal{F} \cup \mathcal{C}, \mathcal{F} \cap \mathcal{C} = \emptyset$
- programs are sets of rewrite rules  $lhs \rightarrow rhs$
- $lhs$  is of the form  $F(p_1, \dots, p_n)$  with  $F \in \mathcal{F}$  and  $p_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- conditional rewrite rules  $lhs \rightarrow rhs \Leftarrow cond$  where  
 $cond \equiv \{v_1 = u_1, \dots, v_n = u_n\}$  and  $v_i, u_i \in \mathcal{T}_\Sigma(\mathcal{X})$
- rewriting binds free variables in  $v_i$ , modelling variable declaration, `let`- and `case`-expressions
- higher-order context with  $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$  and abstraction operator  $[-]$

# Target Languages in the CCS Framework

## CCS

```
multlast ([])      -> []
multlast ([A])    -> [A]
multlast ([A,B|C]) -> [D,D|E]
                  <= [D|E] = multlast ([B|C])
```

## Haskell

```
multlast []      = []
multlast [A]     = [A]
multlast [A,B|C] =
    let [D|E] = multlast ([B|C]) in [D,D|E]
```

## Prolog

```
multlast([], []).
multlast([A], [A]).
multlast([A,B|C], [D,D|E]) :-
    multlast([B|C], [D|E]).
```

# The IP task in CCS

function symbols  $\mathcal{F} = \mathcal{F}_T \cup \mathcal{F}_B \cup \mathcal{F}_I$

user defined rules  $R = E^+ \cup E^- \cup BK$

restriction bias ( $lhs, rhs, u, v \subseteq \mathcal{T}_\Sigma(\mathcal{X})$ )

preference bias ( $\preceq$ )

IP Task

# The IP task in CCS

function symbols  $\mathcal{F} = \mathcal{F}_T \cup \mathcal{F}_B \cup \mathcal{F}_I$

$\mathcal{F}_T$  function symbols of *target functions*

$\mathcal{F}_B$  function symbols of *background knowledge*

$\mathcal{F}_I$  pool of function symbols for inventing *sub functions*

user defined rules  $R = E^+ \cup E^- \cup BK$

restriction bias ( $lhs, rhs, u, v \subseteq \mathcal{T}_\Sigma(\mathcal{X})$ )

preference bias ( $\preceq$ )

IP Task

# The IP task in CCS

function symbols  $\mathcal{F} = \mathcal{F}_T \cup \mathcal{F}_B \cup \mathcal{F}_I$

user defined rules  $R = E^+ \cup E^- \cup BK$

$E^+$  positive evidence  $F(t_1, \dots, t_n) \rightarrow r$

$E^-$  negative evidence as inequalities  $F(t_1, \dots, t_n) \rightarrow r$

$BK$  background knowledge

$F(t_1, \dots, t_n) \rightarrow r \Leftarrow \{v_1 = u_1, \dots, v_n = u_n\}$

restriction bias ( $lhs, rhs, u, v \subseteq \mathcal{T}_\Sigma(\mathcal{X})$ )

preference bias ( $\preceq$ )

IP Task

# The IP task in CCS

function symbols  $\mathcal{F} = \mathcal{F}_T \cup \mathcal{F}_B \cup \mathcal{F}_I$

user defined rules  $R = E^+ \cup E^- \cup BK$

restriction bias ( $lhs, rhs, u, v \subseteq \mathcal{T}_\Sigma(\mathcal{X})$ )

Allow only a subset of  $\mathcal{T}_\Sigma(\mathcal{X})$  for lhss, rhss, and conditions

preference bias ( $\preceq$ )

IP Task

# The IP task in CCS

function symbols  $\mathcal{F} = \mathcal{F}_T \cup \mathcal{F}_B \cup \mathcal{F}_I$

user defined rules  $R = E^+ \cup E^- \cup BK$

restriction bias  $(lhs, rhs, u, v \subseteq \mathcal{T}_\Sigma(\mathcal{X}))$

preference bias  $(\preceq)$

Partial ordering on terms, lhss, rhss, conditions, rules, and programs

IP Task

# The IP task in CCS

function symbols  $\mathcal{F} = \mathcal{F}_T \cup \mathcal{F}_B \cup \mathcal{F}_I$

user defined rules  $R = E^+ \cup E^- \cup BK$

restriction bias ( $lhs, rhs, u, v \subseteq \mathcal{T}_\Sigma(\mathcal{X})$ )

preference bias ( $\preceq$ )

## IP Task

Find a set of rules  $R_T$  s.t.

$$R_T \cup BK \models E^+$$

$$R_T \cup BK \not\models E^-$$

and  $R_T$  is optimal w.r.t. restriction and preference bias.



# Higher-Order Rewriting

```
map ([u] Z (u), nil)           -> nil
map ([u] Z (u), cons (X, Y))  -> cons (Z (X), map ([u] Z (u), Y))
```

more Terese p 612

# A DATE

$\mathcal{C}$  unrestricted

$\mathcal{F}_T$  singleton

$\mathcal{F}_B$  unrestricted

$\mathcal{F}_I$   $\emptyset$

$E^+$  unrestricted

$E^-$  unrestricted

$BK$  unrestricted

$\mathcal{X}_2$   $\emptyset$

restr. bias subset of SML

pref. bias user defined fitness function

search str. global search, generate and test

# FLIP

$\mathcal{C}$  unrestricted

$\mathcal{F}_T$  unrestricted

$\mathcal{F}_B$  unrestricted

$\mathcal{F}_I$   $\emptyset$

$E^+$  unconditional

$E^-$  unconditional (may be empty)

$BK$  unrestricted

$\mathcal{X}_2$   $\emptyset$

**restr.bias**  $lhs$  is a consistent (w.r.t. evidence) but restricted (no new variables on  $rhs$  least general generalisation of two positive examples  $rhs$  is derived via inverse narrowing from two  $lhss$ )

**pref. bias** minimum discription length and coverage

**search str.** heuristic search with sequential covering

# FFOIL

$\mathcal{C}$  constants, including  $\{true, false\}$

$\mathcal{F}_T$  singleton

$\mathcal{F}_B \cup \{=, \neq, <, \leq, >, \geq, \neg\}$

$\mathcal{F}_I \emptyset$

$E^+$  unconditional

$E^-$  unconditional (may be empty)

$BK$  unconditional

$\mathcal{X}_2 \emptyset$

restr. bias  $l, v \in \{F(i_1, \dots, i_n) \mid i_j \in \mathcal{X}_1, F \in \mathcal{F}\}$   
 $r, u \in \mathcal{T}_\Sigma(\mathcal{X})$

pref. bias *foil gain*

search str. sequential covering

# GOLEM

$\mathcal{C} \cup \{true, false\}$

$\mathcal{F}_T$  singleton

$\mathcal{F}_B$  unrestricted

$\mathcal{F}_I \emptyset$

$E^+$  unconditional

$E^-$  unconditional

$BK$  unrestricted

$\mathcal{X}_2 \emptyset$

**restr. bias**  $l, v \in \{F(i_1, \dots, i_n) \mid i_j \in \mathcal{T}_\Sigma(\mathcal{X}), F \in \mathcal{F}\}$   
 $r, u \in \mathcal{T}_\Sigma(\mathcal{X})$

**pref. bias** clause with highest coverage in a lattice of least general generalisations relative to  $BK$  of randomly picked examples

**search str.** sequential covering

# IGOR II

$\mathcal{C}$  unrestricted

$\mathcal{F}_T$  unrestricted

$\mathcal{F}_B$  unrestricted

$\mathcal{F}_I$  domain of invented function equals domain of calling function (no variable invention)

$E^+$  unconditional

$E^-$   $\emptyset$

$BK$  unconditional

$\mathcal{X}_2$   $\emptyset$

**restr. bias** non-overlapping *lhss*,  $rhs = F(\dots)$ ,  $F \notin \mathcal{F}_I$ , conditions model only `let`-expressions

**pref. bias** fewer case distinctions, most specific patterns, fewer recursive calls or calls to  $BK$

**search str.** best first

# MAGICHASKELLER

$\mathcal{C}$  unrestricted

$\mathcal{F}_T$  singleton

$\mathcal{F}_B$  unrestricted

$\mathcal{F}_I$   $\emptyset$

$E^+$  unrestricted

$E^-$  unrestricted

$BK$  unrestricted

$\mathcal{X}_2$  only via paramorphisms from  $BK$

restr. bias type constraints, composition of functions from  $BK$

pref. bias smallest w.r.t.  $BK$

search str. breadth first, generate and test