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/ouput 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 inducable class of problems, through **syntactic** constraints, i.e. linear recursion as sole kind of recursion (hypothese language)

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

Sub Functions Functions **neither** defined as **target functions nor** in the **background knowledge**, but automaticallly 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 inducable class of problems, through **syntactic** constraints, i.e. linear recursion as sole kind of recursion (hypothese language)

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

Sub Functions Functions **neither** defined as **target functions nor** in the **background knowledge**, but automaticallly 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 inducable class of problems, through **syntactic** constraints, i.e. linear recursion as sole kind of recursion (hypothese language)

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

Sub Functions Functions **neither** defined as **target functions nor** in the **background knowledge**, but automaticallly 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 inducable class of problems, through **syntactic** constraints, i.e. linear recursion as sole kind of recursion (hypothese language)

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

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

Different Approaches

	analytic	*generate & test*	
		systematic	*evolutionary*
logic	DIALOGS-II	FOIL/FFOIL, GOLEM	
functional	THESYS, IGOR I, IGOR II	MAGIC-HASKELLER	ADATE

Inductive Logic Programming (ILP)

Inductive Functional Programming (IFP)

Different Approaches

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

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

	analytic	*generate & test*	
		systematic	*evolutionary*
logic	DIALOGS-II	FOIL/FFOIL, GOLEM	
functional	THESYS, IGOR I, IGOR II	MAGIC-HASKELLER	ADATE

Inductive Logic Programming (ILP)

Inductive Functional Programming (IFP)

- Based *Term Rewriting* or *Combinatory Logic* / λ*-calculus*
- primary objective is program learning

Different Approaches

	analytic	*generate & test*	
		systematic	*evolutionary*
logic	DIALOGS-II	FOIL/FFOIL, GOLEM	
functional	THESYS, IGOR I, IGOR II	MAGIC-HASKELLER	ADATE

Analytic

Generate & Test

Different Approaches

	analytic	*generate & test*	
		systematic	*evolutionary*
logic	DIALOGS-II	FOIL/FFOIL, GOLEM	
functional	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

	analytic	*generate & test*	
		systematic	*evolutionary*
logic	DIALOGS-II	FOIL/FFOIL, GOLEM	
functional	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

	analytic	*generate & test*	
		systematic	*evolutionary*
logic	DIALOGS-II	FOIL/FFOIL, GOLEM	
functional	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

	analytic	*generate & test*	
		systematic	*evolutionary*
logic	DIALOGS-II	FOIL/FFOIL, GOLEM	
functional	THESYS, IGOR I, IGOR II	MAGIC-HASKELLER	ADATE

large diversity of underlying theoretical concepts and requirements

$\Longrightarrow$

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”) perspecitve** 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 Pr**oblem **D**efinition **L**anguage)

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”) perspecitve** 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** **P**roblem **D**efinition **L**anguage)

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") perspecitve** 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 P**roblem **D**efinition **L**anguage)

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”) perspecitve** 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 Pro**blem **D**efinition **L**anguage)

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”) perspecitve** 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** **Pr**oblem **D**efinition **L**anguage)

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”) perspecitve** 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 P**roblem **D**efinition **L**anguage)

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") perspecitve** 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 P**roblem **D**efinition **L**anguage)

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	$\bullet$	$\{\cdot\}$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\emptyset$	global search, g 'n t
FLIP	$\bullet$	$\bullet$	$\bullet$	$\emptyset$	$\circ$	$\circ,\emptyset$	$\bullet$	$\emptyset$	sequential covering
FFOIL	c	$\bullet$	$\supset$	$\emptyset$	$\circ$	$\circ,\emptyset$	$\circ$	$\emptyset$	sequential covering
GOLEM	$\bullet$	$\{\cdot\}$	$\bullet$	$\emptyset$	$\circ$	$\circ$	$\bullet$	$\emptyset$	sequential covering
IGOR I	$\bullet$	$\{\cdot\}$	$\emptyset$	$\bullet$	$\circ$	$\emptyset$	$\emptyset$	$\emptyset$	2-step, global search
IGOR II	$\bullet$	$\bullet$	$\bullet$	$\bullet$	$\circ$	$\emptyset$	$\circ$	$\emptyset$	global search
MAGH.	$\bullet$	$\{\cdot\}$	$\bullet$	$\emptyset$	$\bullet$	$\bullet$	$\bullet$	$\circ$	breadth first, g 'n t

$\bullet$	unrestricted / conditional rules	$\circ$	restricted / unconditional rules
$\{\cdot\}$	singleton set	$\emptyset$	empty set
c	constants	$\supset$	built in predicates

Empirical Results

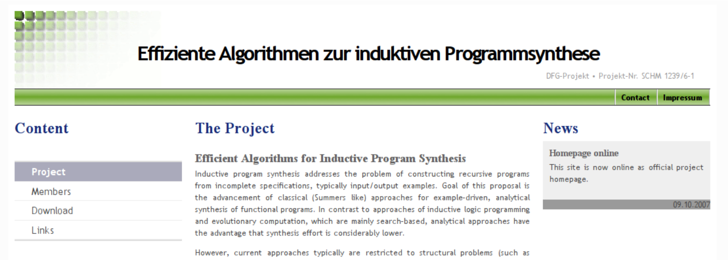
	isort	*reverse*	*weave*	*shiftr*	*mult/add*	*allodds*
ADATE	70.0	78.0	80.0	18.81	—	214.87
FLIP	×	—	$134.24^{\perp}$	$448.55^{\perp}$	×	×
FFOIL	×	—	$0.4^{\perp}$	$< 0.1^{\perp}$	$8.1^{\perp}$	$0.1^{\perp}$
GOLEM	0.714	—	$0.66^{\perp}$	0.298	—	$0.016^{\perp}$
IGOR II	0.105	0.103	0.200	0.127	⊙	⊙
MAGH.	0.01	0.08	⊙	157.32	—	×

	lasts	*last*	*member*	*odd/even*	*multlast*
ADATE	822.0	0.2	2.0	—	4.3
FLIP	×	0.020	17.868	0.130	$448.90^{\perp}$
FFOIL	$0.7^{\perp}$	0.1	$0.1^{\perp}$	$< 0.1^{\perp}$	< 0.1
GOLEM	1.062	< 0.001	0.033	—	< 0.001
IGOR II	5.695	0.007	0.152	0.019	0.023
MAGH.	19.43	0.01	⊙	—	0.30

— not tested × stack overflow ⊙ timeout ⊥ wrong

all runtimes in seconds

Our Project



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

- Publications
- Downloads
- Links

inductive-programming.org

http://www.inductive-programming.org

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

Thank you
for your attention!

estion

Questions?

Questions?

Questions?

Questions?

Questions?

Questions?

Questions?

Questions?

Questions?

CCS in a nutshell

- given a set of function symbols Σ and a set of variables $\mathcal{X}$
- terms over Σ 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 $F(p_1, \ldots, p_n)$ 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 $\equiv \{v_1 = u_1, \ldots, v_n = u_n\}$ 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 Σ and a set of variables $\mathcal{X}$
- terms over Σ 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 $F(p_1, \ldots, p_n)$ with $F \in \mathcal{F}$ and $p_i \in \mathcal{T}_\mathcal{C}(\mathcal{X})$
- conditional rewrite rules $lhs \rightarrow rhs \Leftarrow cond$ where
  $cond \equiv \{v_1 = u_1, \ldots, v_n = u_n\}$ 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 Σ and a set of variables $\mathcal{X}$
- terms over Σ 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 $F(p_1, \ldots, p_n)$ with $F \in \mathcal{F}$ and $p_i \in \mathcal{T}_\mathcal{C}(\mathcal{X})$
- conditional rewrite rules $lhs \rightarrow rhs \Leftarrow cond$ where
  $cond \equiv \{v_1 = u_1, \ldots, v_n = u_n\}$ 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 Σ and a set of variables $\mathcal{X}$
- terms over Σ 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 $F(p_1, \ldots, p_n)$ with $F \in \mathcal{F}$ and $p_i \in \mathcal{T}_\mathcal{C}(\mathcal{X})$
- conditional rewrite rules $lhs \rightarrow rhs \Leftarrow cond$ where
  $cond \equiv \{v_1 = u_1, \ldots, v_n = u_n\}$ 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 Σ and a set of variables $\mathcal{X}$
- terms over Σ 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 $F(p_1, \ldots, p_n)$ with $F \in \mathcal{F}$ and $p_i \in \mathcal{T}_\mathcal{C}(\mathcal{X})$
- conditional rewrite rules *lhs* $\rightarrow$ *rhs* $\Leftarrow$ *cond* where
  $cond \equiv \{v_1 = u_1, \ldots, v_n = u_n\}$ 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 Σ and a set of variables $\mathcal{X}$
- terms over Σ 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 $F(p_1, \ldots, p_n)$ with $F \in \mathcal{F}$ and $p_i \in \mathcal{T}_\mathcal{C}(\mathcal{X})$
- conditional rewrite rules $lhs \rightarrow rhs \Leftarrow cond$ where
  $cond \equiv \{v_1 = u_1, \ldots, v_n = u_n\}$ 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 $F(t_1, \ldots, t_n) \to r$
- E^- negative evidence as inequalities $F(t_1, \ldots, t_n) \to r$
- BK background knowledge
  $F(t_1, \ldots, t_n) \to r \Leftarrow \{v_1 = u_1, \ldots, v_n = u_n\}$

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

ADATE

$\mathcal{C}$	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	$l, v \in \{F(i_1, \ldots, i_n) \vert i_i \in \mathcal{X}_1, F \in \mathcal{F}\}$ $r, u \in \mathcal{T}_\Sigma(\mathcal{X})$
pref. bias	*foil gain*
search str.	sequential covering

GOLEM

$\mathcal{C}$	$\cup\{\mathit{true}, \mathit{false}\}$
$\mathcal{F}_T$	singleton
$\mathcal{F}_B$	unrestricted
$\mathcal{F}_I$	$\emptyset$
E^+	unconditional
E^-	unconditional
BK	unrestricted
$\mathcal{X}_2$	$\emptyset$
restr. bias	$l, v \in \{F(i_1, \ldots, i_n) \mid i_i \in \mathcal{T}_\Sigma(\mathcal{X}), F \in \mathcal{F}\}$ $r, u \in \mathcal{T}_\Sigma(\mathcal{X})$
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*, $rhs = F(\ldots), F \notin \mathcal{F}_I$, 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