Directed Model Checking
– Theorem Proving –

Stefan Edelkamp
1 Overview

- FOL
- Automated Inference
- Horn Clauses
- Forward and Backward Chaining
- Resolution, Unification
- FOL to Clause Form
- Resolution as Search
- Directed Automated Theorem Proving
2 First-Order Logic

User defined primitives:

- Constant symbols: *Mary*, 3

- Function symbols: $father-of(Mary) = John, color-of(Sky) = Blue$

- Predicate symbols: $greater(5, 3), green(Grass), color(Grass, Green)$

FOL primitives:

- Variable symbols. E.g., $x, y$

- Connectives: not ($\neg$), and ($\land$), or ($\lor$), implies ($\Rightarrow$), if and only if ($\iff$)

- Quantifiers: Universal ($\forall$) and Existential ($\exists$)
Quantification

Universal quantification: \((\forall x)P(x)\) means \(P\) holds for all values of \(x\) in domain associated with that variable.

\((\forall x)\text{dolphin}(x) \Rightarrow \text{mammal}(x)\)

Existential quantification: \((\exists x)P(x)\) means \(P\) holds for some value of \(x\) in the domain associated with that variable \((\exists x)\text{mammal}(x) \land \text{lays-eggs}(x)\)

- Universal quantifiers usually used with \(\Rightarrow\) to form if-then rules
  
  \((\forall x)\text{student}(x) \Rightarrow \text{smart}(x)\)

Rarely universal quantification to make blanket statements:

\((\forall x)\text{student}(x) \land \text{smart}(x)\) means that everyone in the world is a student and is smart
Quantification (ctd.)

- Existential quantifiers usually used with $\land$ to specify list of properties or facts about an individual: $\left( \exists x \right) student(x) \land smart(x)$ means there is a student who is smart.

Common mistake: $\left( \exists x \right) student(x) \Rightarrow smart(x)$

- Switching order of universal (existential) quantifiers does not change meaning: $(\forall x)(\forall y)P(x, y) \iff (\forall y)(\forall x)P(x, y)$

- Switching order of universals and existentials does change meaning:

Everyone likes someone: $(\forall x)(\exists y) likes(x, y)$

Someone is liked by everyone: $(\exists y)(\forall x) likes(x, y)$
Sentences

... built up from terms and atoms:

*Term*: constant symbol, a variable symbol, or a function of \( n \) terms

For example, \( x \) and \( f(x_1, ..., x_n) \) are terms, where each \( x_i \) is a term

*Atom*: either predicate of \( n \) terms, or, if \( P \) and \( Q \) atoms, then \( \neg P, P \lor Q, P \land Q, P \Rightarrow Q, P \iff Q \) atoms

*Sentence* atom, or, if \( P \) sentence and \( x \) variable, then \( (\forall x)P \) and \( (\exists x)P \) sentences

*Well-formed formula (wff)*: sentence containing no free variables. E.g., \( (\forall x)P(x, y) \) has \( x \) bound as a universally quantified variable, but \( y \) is free
Inference Rules for FOL

Inference rules for PL: Modus Ponens, etc.

Inference rules for use with quantifiers:

- **Universal Elimination:** If $(\forall x)P(x)$ is true, then $P(c)$ for $c$ constant in domain of $x$

- **Existential Introduction:** If $P(c)$ is true, then $(\exists x)P(x)$ is inferred

- **Existential Elimination:** From $(\exists x)P(x)$ infer $P(c)$, with $c$ brand new.

- **Paramodulation:** Given $(P_1 \lor \ldots \lor P_N)$ and $(t = s \lor Q_1 \lor \ldots \lor Q_M)$ where each $P_i$ and $Q_i$ is literal and $P_j$ contains term $t$, derive $(P_1 \lor \ldots \lor P_{j-1} \lor P_j[s] \lor P_{j+1} \lor \ldots \lor P_N \lor Q_1 \lor \ldots \lor Q_M)$, where $P_j[s]$ means single occurrence of term $t$ is replaced by term $s$ in $P_j$
Generalized Modus Ponens (GMP)

Example: from \( P(c), Q(c), \) and \((\forall x)(P(x) \land Q(x)) \Rightarrow R(x)\), derive \( R(c)\)

In general: given atomic sentences \( P_1, P_2, \ldots, P_N \), and implication sentence \((Q_1 \land Q_2 \land \ldots \land Q_N) \Rightarrow R\), where \( Q_1, \ldots, Q_N \) and \( R \) are atomic sentences, and \( subst(\Theta, P_i) = subst(\Theta, Q_i) \) for \( i = 1, \ldots, N \), derive new sentence: \( subst(\Theta, R)\)

\( subst(\Theta, \alpha) \): result of applying set of substitutions defined by \( \Theta \) to the sentence \( \alpha \)

Substitution list \( \Theta = v_1/t_1, v_2/t_2, \ldots, v_n/t_n \) replace symbol \( v_i \) by term \( t_i \) (left-to-right)

Example: \( subst(x/IceCream, y/Ziggy, eats(y, x)) = eats(Ziggy, IceCream) \)
Automated Inference for FOL

... harder than using PL; variables can take on potentially infinite number of possible values from their domain.

⇒ potentially infinite number of ways to apply Universal-Elimination rule of inference

Gödel’s Completeness Theorem: FOL entailment is semi-decidable:

- if a sentence is true given a set of axioms, there is a procedure to determine this
- if the sentence is false, there is no guarantee that a procedure will ever terminate

Truth Table: not complete for FOL, truth table size may be infinite

Natural Deduction: complete but not practical; branching factor too large

Generalized Modus Ponens, not complete for FOL, but complete for Horn clauses
Horn Clauses

... sentence of form $(\forall x)(P_1(x) \land P_2(x) \land \ldots \land P_n(x)) \Rightarrow Q(x)$ where there are 0 or more $P_i$'s; $P_i$'s and $Q$ positive (i.e., un-negated) literals

Horn clauses represent subset of set of sentences representable in FOL

$P(a) \lor Q(a)$ not a Horn clause

*Natural deduction using GMP*: complete for KBs containing only Horn clauses

- Proofs start with the given axioms/premises in KB
- deriving new sentences using GMP until the goal/query sentence is derived

This defines a *forward chaining* inference procedure because it moves forward from the KB to the goal
Example

$KB =$ All cats like fish, cats eat everything they like, and Ziggy is a cat

1. $\forall x \text{cat}(x) \Rightarrow \text{likes}(x, \text{Fish})$

2. $(\forall x)(\forall y)(\text{cat}(x) \land \text{likes}(x, y)) \Rightarrow \text{eats}(x, y)$

3. $\text{cat}(Ziggy)$

Goal query: Does Ziggy eat fish?

Proof:
Use GMP with (1) and (3) to derive $\text{likes}(Ziggy, \text{Fish})$
Use GMP with (3), (4) and (2) to derive $\text{eats}(Ziggy, \text{Fish}) \Rightarrow \text{Ziggy eats fish}$. 
GMP Backward-Chaining Deduction

... complete for KBs containing only Horn clauses

- proof starts with goal query; find implications that allows us to prove it

- prove each of antecedents in implication; continue backwards until we get to axioms, known to be true.

*Example*: Does Ziggy eat fish? Goal \( \text{eats}(\text{Ziggy}, \text{Fish}) \)

*Proof*: Goal not known, but there is Horn clause (2) that has consequent matching the goal, new sub-goals \( \text{cat}(\text{Ziggy}) \) and \( \text{likes}(\text{Ziggy}, \text{Fish}) \), corresponding to the LHS of (2) \( \text{cat}(\text{Ziggy}) \) matches axiom (3), \( \text{likes}(\text{Ziggy}, \text{Fish}) \) matches the RHS of (1), so try and prove \( \text{cat}(\text{Ziggy}) \) \( \text{cat}(\text{Ziggy}) \) matches axiom (3) solving the sub-goal \( \Rightarrow \) Ziggy eats fish
Resolution Rule for PL:

From sentences $P_1 \lor P_2 \lor \ldots \lor P_n$ and $\neg P_1 \lor Q_2 \lor \ldots \lor Q_m$ derive resolvent $P_2 \lor \ldots \lor P_n \lor Q_2 \lor \ldots \lor Q_m$

Examples:

From $P$ and $\neg P \lor Q$, derive $Q$ (Modus Ponens)

From $(\neg P \lor Q)$ and $(\neg Q \lor R)$, derive $\neg P \lor R$

From $P$ and $\neg P$, derive False

From $(P \lor Q)$ and $(\neg P \lor \neg Q)$, derive True
Resolution Rule for FOL

Given sentences $P_1 \lor \ldots \lor P_n$ and $Q_1 \lor \ldots \lor Q_m$ where each $P_i$ and $Q_i$ is a literal

if $P_j$ and $\neg Q_k$:

- *unify* with substitution list $\Theta$

- derive the resolvent sentence

$\text{subst}(\Theta, P_1 \lor \ldots \lor P_{j-1} \lor P_{j+1} \lor \ldots \lor P_n \lor Q_1 \lor \ldots \lor Q_{k-1} \lor Q_{k+1} \lor \ldots \lor Q_m)$

**Example:** From clauses $P(x, f(a)) \lor P(x, f(y)) \lor Q(y)$ and $\neg P(z, f(a)) \lor \neg Q(z)$, derive resolvent clause

$P(z, f(y)) \lor Q(y) \lor \neg Q(z)$ using $\Theta = x/z$
Unification

*pattern matching* procedure

- takes two atomic sentences, called literals, as input,

- returns *failure* if no match and substitution list, $\Theta$, if match

- $\text{unify}(p, q) = \Theta$ means $\text{subst}(\Theta, p) = \text{subst}(\Theta, q)$ for two atomic sentences $p$ and $q$

- $\Theta$ called the *most general unifier* (mgu)

- variables are implicitly universally quantified

- to make literals match, replace variables by terms
Examples:

\[ \text{parents}(x, \text{father}(x), \text{mother}(\text{Bill})), \text{parents}(\text{Bill}, \text{father}(\text{Bill}), y) \]
\{x/\text{Bill}, y/\text{mother}(\text{Bill})\} \]

\[ \text{parents}(x, \text{father}(x), \text{mother}(\text{Bill})), \text{parents}(\text{Bill}, \text{father}(y), z) \]
\{x/\text{Bill}, y/\text{Bill}, z/\text{mother}(\text{Bill})\} \]

\[ \text{parents}(x, \text{father}(x), \text{mother}(\text{Jane})), \text{parents}(\text{Bill}, \text{father}(y), \text{mother}(y)) \]
failure

- unify: linear time algorithm that returns most general unifier (not unique)

- variable can never be replaced by a term containing that variable, \(x/f(x)\) is illegal

- occurs check before making recursive call

Resolution Rule for PL:
Pseudo Code

Procedure Unify($p, q, \theta$)
    Scan $p$ and $q$ left-to-right . . .
    . . . find first corresponding terms where $p$ and $q$ disagree
    if (no disagreement) return $(\theta, success)$
    $r \leftarrow$ term in $p$, where disagreement first occurs
    $s \leftarrow$ term in $q$, where disagreement first occurs
    if variable($r$)
        $\theta \leftarrow \theta \cup \{r/s\}$
        unify($\text{subst}(\theta, p)$, $\text{subst}(\theta, q)$, $\theta$)
    else if variable($s$)
        $\theta \leftarrow \theta \cup \{s/r\}$
        Unify($\text{subst}(\theta, p)$, $\text{subst}(\theta, q)$, $\theta$)
    else return failure

Resolution Rule for PL: 16
Resolution (Refutation) Procedure ctd.

**Task:** Given consistent set of axioms $KB$ and goal sentence $Q$, show that $KB \models Q$, i.e., every interpretation $I$ that satisfies $KB$, satisfies $Q$

Any interpretation either satisfies $Q$ or $\neg Q \Rightarrow KB \cup \neg Q$ is unsatisfiable

$$(KB \models Q) \iff (KB \land \neg Q \models False)$$

**Gain:** If $KB \cup \neg Q$ is unsatisfiable, then some finite subset is unsatisfiable

- used to establish that a given sentence $Q$ is entailed by KB; cannot to be used to generate all logical consequences of a set sentences

- used to prove that $Q$ is *not* entailed by KB; won’t always give answer, since entailment only semi-decidable

Resolution Rule for PL:
Pseudo-Code

Procedure $Resolution$-$Refutation$

Input: $KB$ set of consistent, true FOL sentences

$Q$ goal sentence that we want to derive

Output: $success$ if $KB \models Q$, and $failure$ otherwise

$$KB \leftarrow KB \cup \{\neg Q\}$$

while $\neg false \notin KB$

Select $S_1, S_2 \in KB$ containing literals that unify

if (none) return failure

resolvent $\leftarrow$ resolution-rule($S_1, S_2$)

$KB \leftarrow KB \cup \{\text{resolvent}\}$

return $success$
Problems Yet to Be Addressed

Resolution rule only applicable with sentences of the form $P_1 \lor P_2 \lor \ldots \lor P_n$,

- where each $P_i$ is a negated or un-negated predicate; contains functions, constants, and universally quantified variables,

How to convert every FOL sentence into this form?

How to pick which pair of sentences to resolve?

How to pick which pair of literals, one from each sentence, to unify?

Every FOL sentence can be converted to a logically equivalent sentence that is in a normal form called clause form.
9 Steps to Clause Form

1. Eliminate all $\iff$ connectives by replacing $(P \iff Q)$ by the equivalent $((P \implies Q) \land (Q \implies P))$

2. Eliminate all $\implies$ connectives by replacing $(P \implies Q)$ by $(\neg P \lor Q)$

3. Reduce scope of each negation to a single predicate by applying equivalences such as $\neg\neg P$ to $P$; $\neg(P \lor Q)$ to $\neg P \land \neg Q$; $\neg(P \land Q)$ to $\neg P \lor \neg Q$; $\neg(\forall x) P$ to $(\exists x) \neg P$, and $\neg(\exists x) P$ to $(\forall x) \neg P$

4. Standardize variables: rename all variables s.t. each quantifier has its own unique variable name; convert $(\forall x) P(x)$ to $(\forall y) P(y)$ if there is another place where variable $x$ is already used.

5. Eliminate existential quantification by introducing Skolem functions

Resolution Rule for PL:
Skolem functions

More generally, if the existential quantifier is within the scope of a universal quantified variable, then introduce a Skolem function that depend on the universally quantified variable.

For example, $(\forall x)(\exists y)P(x, y)$ is converted to $(\forall x)P(x, f(x))$.

$f$ is called a Skolem function, and must be a brand new function name that does not occur in any other sentence in the entire KB.

Example: $(\forall x)(\exists y)\text{loves}(x, y)$ is converted to $(\forall x)\text{loves}(x, f(x))$ where in this case $f(x)$ specifies the person that $x$ loves.

If we knew that everyone loved their mother, then $f$ could stand for the mother-of function.
9 Step Compilation (ctd.)

6. Remove universal quantification symbols by first moving them all to the left end and making the scope of each the entire sentence, and then just dropping the ”prefix” part. For example, convert $(\forall x)P(x)$ to $P(x)$.

7. Distribute and over or to get a conjunction of disjunctions called conjunctive normal form. Convert $(P \land Q) \lor R$ to $(P \lor R) \land (Q \lor R)$, and convert $(P \lor Q) \lor R$ to $(P \lor Q \lor R)$

8. Split each conjunct into a separate clause, which is just a disjunction of negated and un-negated predicates, called literals

9. Standardize variables apart again so that each clause contains variable names that do not occur in any other clause

Resolution Rule for PL:
Example

\((\forall x)(P(x) \Rightarrow ((\forall y)(P(y) \Rightarrow P(f(x, y))) \land \neg(\forall y)(Q(x, y) \Rightarrow P(y))))\)

1. Eliminate \(\iff\): Nothing to do here.

2. Eliminate \(\Rightarrow\)
\((\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x, y))) \land \neg(\forall y)(\neg Q(x, y) \lor P(y))))\)

3. Reduce scope of negation
\((\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x, y))) \land (\exists y)(Q(x, y) \land \neg P(y))))\)

4. Standardize variables
\((\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x, y))) \land (\exists z)(Q(x, z) \land \neg P(z))))\)

5. Eliminate existential quantification
\((\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x, y))) \land (Q(x, g(x)) \land \neg P(g(x))))\))

Resolution Rule for PL:

23
Example (ctd.)

6. Drop universal quantification symbols
\((\neg P(x) \lor ((\neg P(y) \lor P(f(x, y))) \land (Q(x, g(x)) \land \neg P(g(x))))))\)

7. Convert to conjunction of disjunctions
\((\neg P(x) \lor \neg P(y) \lor P(f(x, y))) \land (\neg P(x) \lor Q(x, g(x))) \land (\neg P(x) \lor \neg P(g(x)))\)

8. Create separate clauses \((\neg P(x) \lor \neg P(y) \lor P(f(x, y)), \neg P(x) \lor Q(x, g(x)),\)
\((\neg P(x) \lor \neg P(g(x)))\)

9. Standardize variables \((\neg P(x) \lor \neg P(y) \lor P(f(x, y)), \neg P(z) \lor Q(z, g(z)),\)
\((\neg P(w) \lor \neg P(g(w)))\)

Resolution Rule for PL:
Pseudo-Code Revised

Procedure Resolution-Refutation
Input: $KB$ set of consistent, true FOL sentences
       $Q$ goal sentence that we want to derive
Output: success if $KB \models Q$, and failure otherwise

$KB \leftarrow KB \cup \{\neg Q\}$
$KB \leftarrow \text{clause-form}(KB)$
while ($false \notin KB$)
  Select $S_1, S_2 \in KB$ containing literals that unify
  if (none) return failure
  $\text{resolvent} \leftarrow \text{resolution-rule}(S_1, S_2)$
  $KB \leftarrow KB \cup \{\text{resolvent}\}$
return success

Resolution Rule for PL:
Resolution procedure can be thought of as the bottom-up construction of a search tree, where the leaves are the clauses produced by KB and the negation of the goal.

When a pair of clauses generates a new resolvent clause, add a new node to the tree with arcs directed from the resolvent to the two parent clauses.

The resolution procedure succeeds when a node containing the *False* clause is produced, becoming the root node of the tree.

A strategy is *complete* if its use guarantees that the empty clause (i.e., false) can be derived whenever it is entailed.
Controlling Resolution’s Search

*Breadth-First*: Level 0 clauses are those from the original axioms and the negation of the goal. Level $k$ clauses are the resolvents computed from two clauses, one of which must be from level $k - 1$ and the other from any earlier level.

Compute all level 1 clauses possible, then all possible level 2 clauses, etc. ⇒ Complete, but very inefficient.

*Set-of-Support*: At least one parent clause must be from the negation of the goal or one of the *descendents* of such a goal clause ⇒ Complete (assuming all possible set-of-support clauses are derived).

*Unit Resolution*: At least one parent clause must be a *unit clause*, i.e., a clause containing a single literal ⇒ Not complete, but complete for Horn clause KBs.
6 Directed Automated Theorem Proving

*Top-Down Proof tree*: label of each interior node corresponds to the conclusion, and labels of its children correspond to the premises of an inference step.

Leaves of the proof tree are either axioms or instances of proven theorems.

Proof State: outer fragment of a proof tree: the top-node, representing the goal and all leaves, representing the subgoals of the proof state.

All proven leaves can be discharged, because they are not needed for further considerations. If all subgoals have been solved, the proof is successful.
Heuristics

1. \# internal nodes of the current proof-state.

2. length of a proof state representation.

# trees with $k$ internal nodes and the number of strings with length $k$ are both finite.

$\Rightarrow$ # proof states with fixed heuristic value $k$ is finite.

Basis for the design of guided search algorithms with guaranteed progress.

3. \# open subgoals in the current proof state

consistent, assuming that 1 inference step can close at most 1 open subgoal at a time
Functional A* and IDA*

In functional implementations of heuristic search algorithms, one input parameter is the heuristic function $h$.

Furthermore, the successor generation function $\Gamma$, goal predicate $Goal$ and the initial state $I$ are passed to the algorithms as parameters.

Goal: A* and IDA* in functional programming language like Scheme, Haskell or ML

For A*, $Open$ is represented by a list of triples $(g, f, u)$, sorted by ascending $f$-values.

We omit re-opening of already expanded nodes on shorter generating paths; If $h$ is consistent, i.e. $h(v) - h(u) + 1 \geq 0$ for all $(u, v) \in E$, this is no restriction.
Functional A*

Function A* \((\mathcal{I}, \text{Goal}, h, \Gamma) =\) \(\{\text{Interface}\}\)

let func relax(succs, t, g) = 
  let func f(v) = \((g, g + h(v), v)\) \(\{\text{Local merit computation}\}\)
  \(l \leftarrow (\text{filter Goal succs})\) \(\{\text{Search for terminal state}\}\)
  in if \((l \neq [])\) then \(l\) else 
    \(\text{Open(foldr (insert,(map f succs), t))}\) \(\{\text{Insert successors calling . . .}\}\)
  end 

and 
  func Open [] = [] \(\{\text{Recursive definition}\}\)
  Open ((g, f, u) :: t) = relax \((\Gamma(u), t, g + 1)\) \(\{\text{Call subroutine}\}\)
  in 
  relax \((\Gamma(\mathcal{I}), [], 0)\) \(\{\text{End second subroutine}\}\)
  end

and 
  \(\text{Open} (\text{foldr (insert,(map f succs), t))}\) \(\{\text{Insert successors calling . . .}\}\)
  \(\text{Open} (((g, f, u) :: t)) = \text{relax} \((\Gamma(u), t, g + 1)\)\) \(\{\text{Call subroutine}\}\)
  \text{relax} \((\Gamma(\mathcal{I}), [], 0)\) \(\{\text{Initial call}\}\)
Function IDA*  

\[ \text{Function IDA* } (I, \text{Goal}, h, \Gamma) = \]

\[ \text{let func depth } (U, U', [],) = \]

\[ \text{depth } (U', \infty, [(h(I), \Gamma(I))]) \]

\[ \text{depth } (U, U', (f, []) :: t) = \text{depth } (U, U', t) \]

\[ \text{depth } (U, U', (f, \text{succs}) :: t) = \]

\[ \text{if } (f > U) \text{ then depth } (U, \min(U', f), t) \]

\[ \text{else let } v \leftarrow \text{head succs}; \text{succs'} \leftarrow \text{tail succs} \]

\[ \text{l } \leftarrow \text{(filter Goal succs)} \]

\[ \text{in if } (l \neq []) \text{ then l else } \]

\[ \text{depth } (U, U', (f + h(v) - h(u) + 1, \Gamma(v)) :: (f, \text{succs'}) :: t) \]

\[ \text{end} \]

\[ \text{in depth } (0, h(I), []) \]

\[ \text{end} \]
Remarks

Different to Imperative A*: in Functional A*, \textit{insert} implements dictionary updates within the set of horizon nodes.

If the state is already contained in the priority queue, no insertion takes place.

\textit{Closed} supplied as additional parameter: in A* to \textit{relax}, and in IDA* to \textit{depth}.

In A* \textit{Closed} initialized at start, while in IDA* re-initialized in each iteration.

Instead of \((\text{map } f \text{ succs})\) visited states are first eliminated by \((\text{map } f \text{ eliminate}(\text{Closed, succs}))\).

\textit{Closed} implemented through lists, balanced trees, or low level hash tables.
Isabelle

... interactive and tactical theorem prover, supporting forward and backward proofs.

(forward proof: axioms and already proven theorems are combined to gain new theorems; backward proof: one starts with the theorem to prove, which is step by step reduced to new subgoals)

With a tactic, basic inference steps are combined to larger case-sensitive and proof-searching rules using axioms, memorized theorems or assumptions.

For increasing performance, tableau theorem provers have been integrated into Isabelle, but their inference is not generic for all object logics.

The inference process is hidden in \textit{auto/blast} tactic.

Logics formulated in Isabelle's own meta logic
Selected subset of FOL inference rules in Isabelle notation:

<table>
<thead>
<tr>
<th>Name</th>
<th>Inference Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>disjI1</td>
<td>(?P \Rightarrow ?P</td>
</tr>
<tr>
<td>disjI2</td>
<td>(?Q \Rightarrow ?P</td>
</tr>
<tr>
<td>impl</td>
<td>(?P \Rightarrow ?Q) \Rightarrow ?P \rightarrow ?Q</td>
</tr>
<tr>
<td>conjunct1</td>
<td>(?P &amp; &amp; ?Q \Rightarrow ?P)</td>
</tr>
<tr>
<td>allI</td>
<td>((!!x.?P(x)) \Rightarrow \forall x.?P(x))</td>
</tr>
<tr>
<td>exE</td>
<td>[[\exists x.?P(x); !!x.?P(x) \Rightarrow ?R]] \Rightarrow ?R</td>
</tr>
<tr>
<td>mp</td>
<td>[[?P \rightarrow ?Q; ?P]] \Rightarrow ?Q</td>
</tr>
<tr>
<td>spec</td>
<td>(\forall x.?P x \Rightarrow ?P?x)</td>
</tr>
</tbody>
</table>

where “⇒” denotes meta logic implication, while “→” denotes object level implication, and ? prefixed characters are meta logic variables.
Search

... only documented heuristic search algorithm in *Isabelle* is greedy best-first search (BF).

This greedy strategy, which always expands the state with minimal evaluation function value, is attracted by local minima and not optimal.

Even worse, the implementation of greedy best-first heuristic search in *Isabelle* is not complete even for finite graphs.

In contrast, DFS and BFS are complete (DFS uses global memory to store already proven subgoals and BFS omits all pruning of duplicate states)

We added Functional A* and Functional IDA* with pruning capabilities.