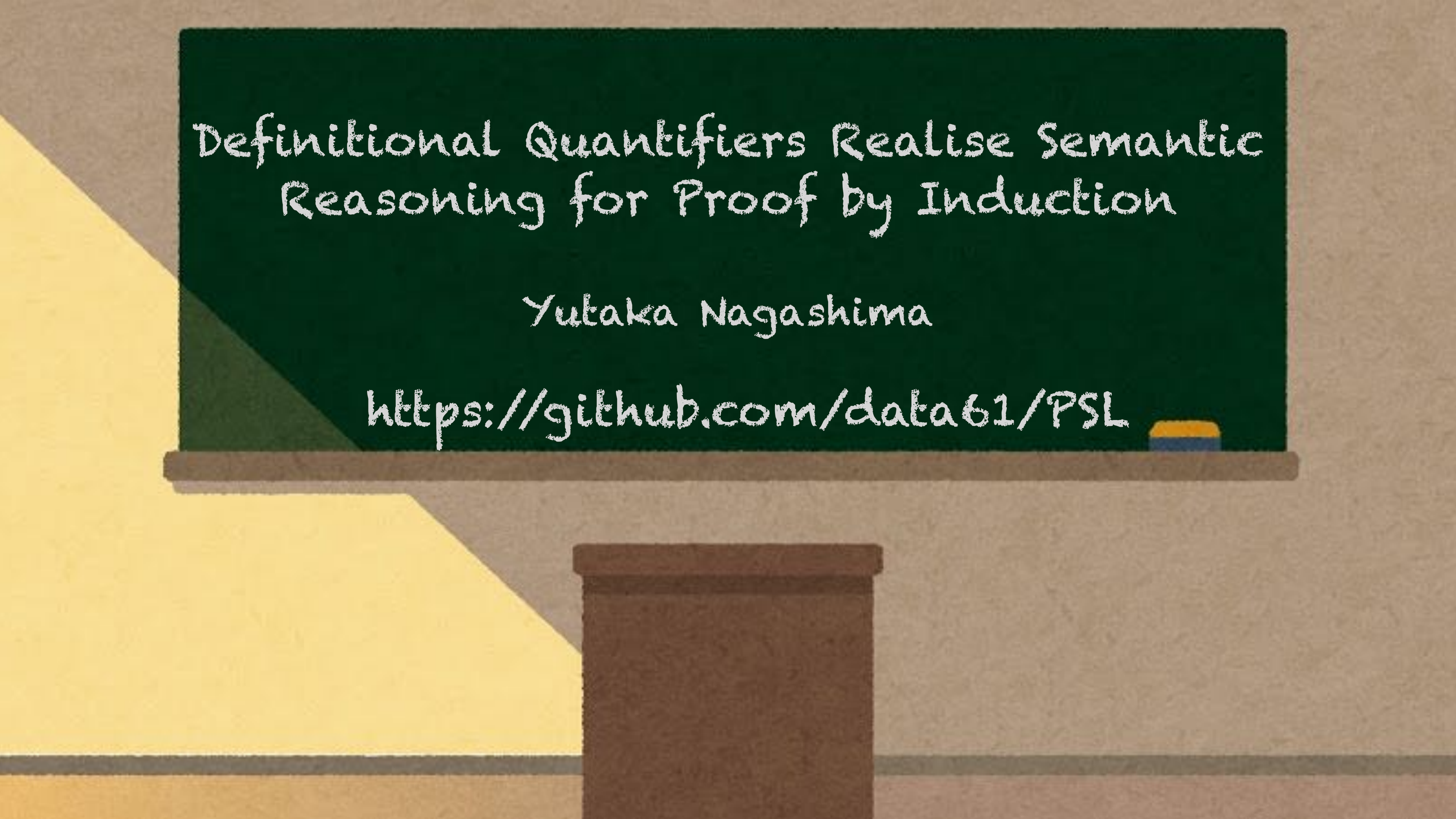
Definitional Quantifiers Realise Semantic Reasoning for Proof by Induction

Yutaka Nagashima

https://github.com/data61/PSL



Definitional Quantifiers Realise Semantic Reasoning for Proof by Induction

Yutaka Nagashima

https://github.com/data61/PSL

Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 125 (2005) 5–43
www.elsevier.com/locate/entcs

Strategic Issues, Problems and Challenges in Inductive Theorem Proving

Bernhard Gramlich[1]

Fakultät für Informatik, TU Wien
Favoritenstr. 9 – E185/2, A–1040 Wien, Austria

Abstract

(Automated) *Inductive Theorem Proving* (ITP) is a challenging field in automated reasoning and theorem proving. Typically, *(Automated) Theorem Proving* (TP) refers to methods, techniques and tools for automatically proving *general* (most often first-order) theorems. Nowadays, the field of TP has reached a certain degree of maturity and powerful TP systems are widely available and used. The situation with ITP is strikingly different, in the sense that proving inductive theorems in an essentially automatic way still is a very challenging task, even for the most advanced existing ITP systems. Both in general TP and in ITP, strategies for guiding the proof search process are of fundamental importance, in automated as well as in interactive or mixed settings. In the paper we will analyze and discuss the most important strategic and proof search issues in ITP, compare ITP with TP, and argue why ITP is in a sense much more challenging. More generally, we will systematically isolate, investigate and classify the main problems and challenges in ITP w.r.t. automation, on different levels and from different points of views. Finally, based on this analysis we will present some theses about the state of the art in the field, possible criteria for what could be considered as *substantial progress*, and promising lines of research for the future, towards (more) automated ITP.

Keywords: Inductive theorem proving, automated theorem proving, automation, interaction, strategies, proof search control, challenges.

1 Background and Overview

Theorem proving tasks are ubiquitous in computer science, e.g., in fields like program specification, transformation and verification. Most often the given, generated or resulting proof tasks are not *general* ones (in the sense, that validity in **all** models of the underlying logic specification is to be established),

[1] Email: gramlich@logic.at www: http://www.logic.at/staff/gramlich/

doi:10.1016/j.entcs.2005.01.006

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 125 (2005) 5–43
www.elsevier.com/locate/entcs

Strategic Issues, Problems and Challenges in Inductive Theorem Proving

Bernhard Gramlich[1]

Fakultät für Informatik, TU Wien
Favoritenstr. 9 – E185/2, A–1040 Wien, Austria

Abstract

(Automated) *Inductive Theorem Proving* (ITP) is a challenging field in automated reasoning and theorem proving. Typically, *(Automated) Theorem Proving* (TP) refers to methods, techniques and tools for automatically proving *general* (most often first-order) theorems. Nowadays, the field of TP has reached a certain degree of maturity and powerful TP systems are widely available and used. The situation with ITP is strikingly different, in the sense that proving inductive theorems in an essentially automatic way still is a very challenging task, even for the most advanced existing ITP systems. Both in general TP and in ITP, strategies for guiding the proof search process are of fundamental importance, in automated as well as in interactive or mixed settings. In the paper we will analyze and discuss the most important strategic and proof search issues in ITP, compare ITP with TP, and argue why ITP is in a sense much more challenging. More generally, we will systematically isolate, investigate and classify the main problems and challenges in ITP w.r.t. automation, on different levels and from different points of views. Finally, based on this analysis we will present some theses about the state of the art in the
be considered as *substantial progress*, and promising lines of
automated ITP.

Keywords: Inductive theorem proving, automated theorem
strategies, proof search control, challenges.

1 Background and Overview

Theorem proving tasks are ubiquitous in compu
program specification, transformation and verifi
generated or resulting proof tasks are not *gener*
lidity in **all** models of the underlying logic spec

[1] Email: gramlich@logic.at www: http:/

1571-0661/$ – see front matter © 2005 Elsevier B.V. All rights reser
doi:10.1016/j.entcs.2005.01.006

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

5 Some Theses

Here are a few theses about (automated) ITP for which there is no formal proof, but only some evidence: Theses:

- **In the near future, ITP will only be successful for**
  - **very specialized domains (e.g., with fixed axiomatizations),**
  - **for very restricted classes of conjectures.**
- **ITP will continue to be a very challenging engineering process.**

Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 125 (2005) 5–43
www.elsevier.com/locate/entcs

Strategic Issues, Problems and Challenges in Inductive Theorem Proving

Bernhard Gramlich[1]

Fakultät für Informatik, TU Wien
Favoritenstr. 9 – E185/2, A–1040 Wien, Austria

Abstract

(Automated) *Inductive Theorem Proving* (ITP) is a challenging field in automated reasoning and theorem proving. Typically, *(Automated) Theorem Proving* (TP) refers to methods, techniques and tools for automatically proving *general* (most often first-order) theorems. Nowadays, the field of TP has reached a certain degree of maturity and powerful TP systems are widely available and used. The situation with ITP is strikingly different, in the sense that proving inductive theorems in an essentially automatic way still is a very challenging task, even for the most advanced existing ITP systems. Both in general TP and in ITP, strategies for guiding the proof search process are of fundamental importance, in automated as well as in interactive or mixed settings. In the paper we will analyze and discuss the most important strategic and proof search issues in ITP, compare ITP with TP, and argue why ITP is in a sense much more challenging. More generally, we will systematically isolate, investigate and classify the main problems and challenges in ITP w.r.t. automation, on different levels and from different points of views. Finally, based on this analysis we will present some theses about the state of the art in the
be considered as *substantial progress*, and promising lines of
automated ITP.

Keywords: Inductive theorem proving, automated theorem
strategies, proof search control, challenges.

1 Background and Overview

Theorem proving tasks are ubiquitous in compu
program specification, transformation and verifi
generated or resulting proof tasks are not *gener*
lidity in **all** models of the underlying logic spec

[1] Email: gramlich@logic.at www: http:/

1571-0661/$ – see front matter © 2005 Elsevier B.V. All rights reser
doi:10.1016/j.entcs.2005.01.006

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

5 Some Theses

Here are a few theses about (automated) ITP for which there is no formal proof, but only some evidence: Theses:

- **In the near future, ITP will only be successful for**
  - **very specialized domains (e.g., with fixed axiomatizations),**
  - **for very restricted classes of conjectures.**
- **ITP will continue to be a very challenging engineering process.**

DEUTSCH ENGLISH

Please note our measures concerning Coronavirus / Covid 19

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik

About Dagstuhl | Program | Publications | Library | dblp

You are here: Program » Seminar Calendar » Seminar Homepage

- Seminar Calendar
  - All Events
- Dagstuhl Seminars
- Dagstuhl Perspectives
- GI-Dagstuhl Seminars
- Summer Schools
- Events
- Research Guests
- Expenses
- Planning your visit

https://www.dagstuhl.de/9530

July 24 – 28 , 1995, Dagstuhl Seminar 9530

Automation of Proof by Mathematical Induction

Organizer

R.S. Boyer, A. Bundy, D. Kapur, Ch. Walther

For support, please contact

Dagstuhl Service Team

Documents

Dagstuhl-Seminar-Report 122

Documentation

In the series Dagstuhl Reports each Dagstuhl Seminar and Dagstuhl Perspectives Workshop is documented. The seminar organizers, in cooperation with the collector, prepare a report that includes contributions from the participants' talks together with a summary of the seminar.

Download overview leaflet (PDF).

Publications

Furthermore, a comprehensive peer-reviewed collection of research papers can be published in the series Dagstuhl Follow-Ups.

Dagstuhl's Impact

Please inform us when a publication was published as a result from your seminar. These publications are listed in the category Dagstuhl's Impact and are presented on a special shelf on the ground floor of the library.

Leibniz Gemeinschaft

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

which there is no formal

ssful for
l axiomatizations),

g engineering process.

DEUTSCH ENGLISH

Please note our measures concerning Coronavirus / Covid 19

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik

About Dagstuhl | Program | Publications | Library | dblp

You are here: Program » Seminar Calendar » Seminar Homepage

- Seminar Calendar
  - All Events
- Dagstuhl Seminars
- Dagstuhl Perspectives
- GI-Dagstuhl Seminars
- Summer Schools
- Events

https://www.dagstuhl.de/9530

July 24 – 28 , 1995, Dagstuhl Seminar 9530

Automation of Proof by Mathematical Induction

ther

Documentation

In the series Dagstuhl Reports each Dagstuhl Seminar and Dagstuhl Perspectives Workshop is documented. The seminar organizers, in cooperation with the collector, prepare a report that includes contributions from the participants' talks together with a summary of the seminar.

Download overview leaflet (PDF).

Publications

Furthermore, a comprehensive peer-reviewed collection of research papers can be published in the series Dagstuhl Follow-Ups.

Dagstuhl's Impact

Please inform us when a publication was published as a result from your seminar. These publications are listed in the category Dagstuhl's Impact and are presented on a special shelf on the ground floor of the library.

oss Dagstuhl - LZI GmbH

Workshop on Automation of Proofs by Mathematical Induction

Held in Connection with
The 17th International Conference on Automated Deduction
June 17-20, 2000
Pittsburgh, Pennsylvania, U.S.A.

Home
Call for papers
Challenges

CALL FOR PAPERS

Proof by Mathematical Induction presents the Automated Deduction community with some very challenging research problems. The aim of this one day workshop is to create an informal forum in which hot-topics and emerging techniques can be presented and discussed.

The workshop will feature invited talks and contributed presentations with ample time for discussion regarding topics like:

- Inductive theorem proving and formal methods,

Prof. Bernhard Gramlich
https://www.logic.at/staff/gramlich/

which there is no formal

ssful for
l axiomatizations),

engineering process.

DEUTSCH ENGLISH

Please note our measures
concerning Coronavirus / Covid 19

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik

About Dagstuhl | Program

You are here: Program » Seminar Calendar » Semin

Seminar Calendar
All Events
Dagstuhl Seminars
Dagstuhl Perspectives
GI-Dagstuhl Seminars
Summer Schools
Events

https://www.dagstuhl.de/9530

July 24 – 28 , 1995, Dagstuhl Seminar 9530

Automation of Proof by Ma
Induction

ther

oss Dagstuhl

Workshop on Automation of Proofs by Mathematical Induction

Held in Connection with
The 17th International Conference on Automated Deduction
June 17-20, 2000
Pittsburgh, Pennsylvania, U.S.A.

Home
Call for papers
Challenges

CALL FOR PAPERS

Proof by Mathematical Induction presents the Automated Deduction community with some very challenging research problems. The aim of this one day workshop is to create an informal forum in which hot-topics and emerging techniques can be presented and discussed.

The workshop will feature invited talks and contributed presentations with ample time for discussion regarding topics like:

- Inductive theorem proving and formal methods,

Workshop on inductive theorem proving
Saturday 23rd & Sunday 24th November 2013
Imperial College London

Department of Computing,
180 Queen's Gate
Room 217/218

This workshop will be very informal and simply aim to give researchers interested in inductive theorem a chance to meet and exchange ideas. We hope for a lot of interaction between participants and an opportunity to actually try out some of the available theorem provers.

Saturday 23/11

9.30 - 10.00

Welcome and introduction. Participants briefly say a few words on what their interests are and what they hope to get from the workshop.

10.00 - 10.45 Demo of the Dafny system Rustan Leino

10.45 - 11.30 Using Dafny in Teaching, Sophia Drossopoulou

11.30- 12.00 Zeno - a retrospective look, Will Sonnex

12.00 - 13.30 Lunch

13.30 -14.15 HipSpec: Automating Inductive Proofs using Theory Exploration, Dan Rosen

nhard Gramlich
ogic.at/staff/gramlich/

ere is no formal

izations),

ing process.

DEUTSCH ENGLISH

Please note our measures
concerning Coronavirus / Covid 19

SCHLO
Leibniz-Zen

About Dagstuhl

You are here: Program » Se

https://www.dagstuhl.de

July 24 – 28 , 1995, Dags

Automation of
Induction

- Seminar Calendar
- All Events
- Dagstuhl Seminars
- Dagstuhl Perspectives
- GI-Dagstuhl Seminars
- Summer Schools
- Events

The Second Workshop on Automated Inductive Theorem-proving (WAIT?), 20-21 March 2015, Chalmers University, Gothenburg, Sweden

Inductive theorem proving is a topic of growing interest in the automated reasoning community. Following our successful first workshop in London 2013, we would like to invite you to a second workshop on inductive theorem proving. The workshop will be very informal and aims to give researchers interested in the topic a chance to meet, exchange ideas and perhaps also try out some of the available theorem provers. We would like to invite talks featuring demos and tutorials of inductive theorem provers, challenge problems, new directions of research or anything else of interest to the inductive theorem proving community.

If you have any questions please contact Nick Smallbone (nicsma@chalmers.se) or Moa Johansson (moa.johansson@chalmers.se).

Hope to see you in Gothenburg!

— The HipSpec crew
(Moa Johansson, Dan Rosén, Nick Smallbone, Koen Claessen)

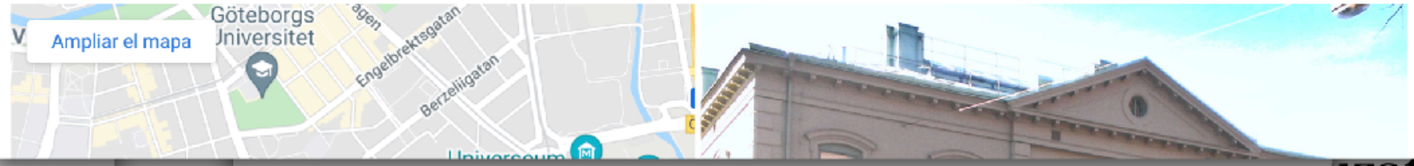
Registration

The workshop will be **free of charge** but please register by filling in the form here. If you want to contribute a talk or demo, please submit a title and a short abstract when you register.

We'll provide coffee but you'll have to pay for lunch/dinner yourself.

Location and programme

The workshop will take place on **Friday 20 March - Saturday 21 March** at Chalmersvillan, Gibraltargatan 1A, Gothenburg, in the room upstairs:



Workshop on Automation of Proofs by Mathematical Induction

Held in Connection with
The 17th International Conference on Automated Deduc
June 17-20, 2000
Pittsburgh, Pennsylvania, U.S.A.

Home
Call for papers
Challenges

CALL FOR PAPERS

Proof by Mathematical Induction presents the Automated Deduction community with some very challenging research problems. The aim of this one day workshop is to create an informal forum in which hot-topics and emerging techniques can be presented and discussed.

The workshop will feature invited talks and contributed presentations with ample time for discussion regarding topics like:

- Inductive theorem proving and formal methods,

10.45 - 11.30 Using Dafny in Teaching, Sophia Drossopoulou

11.30- 12.00 Zeno - a retrospective look, Will Sonnex

12.00 - 13.30 Lunch

13.30 -14.15 HipSpec: Automating Inductive Proofs using Theory Exploration, Dan Rosen

izations),

ing process.

DEUTSCH ENGLISH

Please note our measures
concerning Coronavirus / Covid 19

SCHLO
Leibniz-Zen

About Dagstuhl

You are here: Program » Se

https://www.dagstuhl.de

July 24 – 28 , 1995, Dags

Automation of
Induction

Seminar Calendar
All Events
Dagstuhl Seminars
Dagstuhl Perspectives
GI-Dagstuhl Seminars
Summer Schools
Events

The Second Workshop on Automated Inductive Theorem-proving (WAIT?), 20-21 March 2015, Chalmers University, Gothenburg, Sweden

Inductive theorem proving is a topic of growing interest in the automated reasoning community. Following our successful first workshop in London 2013, we would like to invite you to a second workshop on inductive theorem proving. The workshop will be very informal and aims to give researchers interested in the topic a chance to meet, exchange ideas and perhaps also try out some of the available theorem provers. We would like to invite talks featuring demos and tutorials of inductive theorem provers, challenge problems, new directions of research or anything else of interest to the inductive theorem proving community.

If you have any questions please contact Nick Smallbone (nicsma@chalmers.se) or Moa Johansson (moa.johansson@chalmers.se).

Hope to see you in Gothenburg!

— The HipSpec crew
(Moa Johansson, Dan Rosén, Nick Smallbone, Koen Claessen)

Registration

. If you want to contribute a talk or demo, please submit a title and a short

lmersvillan, Gibraltargatan 1A, Gothenburg, in the room upstairs:

Workshop on Automation of Pro Mathematical Indu

CADE SEVENTEEN

Held in Connection with
The 17th International Conference on
June 17-20, 2000
Pittsburgh, Pennsylvania, U.S.A.

Home
Call for papers
Challenges

CALL FOR

Proof by Mathematical Induction prese
community with some very challenging
this one day workshop is to create an i
and emerging techniques can be presen

The workshop will feature invited talks
with ample time for discussion regardi

- Inductive theorem proving and f

WAIT Workshop:
The Third Workshop on Automated Inductive Theorem-Proving
November 17-18, 2016

Seminar room Zemanek,
Vienna University of Technology,
Favoritenstrasse 9-11, Building (Stiege) 3, Ground floor
1040 Vienna, Austria

Aim:
Inductive theorem proving is a topic of growing interest in the automated reasoning community. The Third Workshop on Automated Inductive Theorem-Proving - WAIT focuses on all relevant aspects of inductive reasoning.

The workshop is an informal event and aims to give researchers interested in the topic a chance to meet, exchange ideas and have an active platform for discussions.

We invite talks featuring demos and tutorials of inductive theorem provers.

izations),

ing process.

Dan Rosen

DEUTSCH ENGLISH

Please note our measures
concerning Coronavirus / Covid 19

SCHLO
Leibniz-Zen

About Dagstuhl

You are here: Program » Se

https://www.dagstuhl.de

July 24 – 28 , 1995, Dags

Automation of
Induction

Seminar Calendar
All Events
Dagstuhl Seminars
Dagstuhl Perspectives
GI-Dagstuhl Seminars
Summer Schools
Events

The Second Workshop on Automated Inductive Theorem-proving (WAIT?), 20-21 March 2015, Chalmers University, Gothenburg, Sweden

Inductive theorem proving is a topic of growing interest in the automated reasoning community. Following our successful first workshop in London 2013, we would like to invite you to a second workshop on inductive theorem proving. The workshop will be very informal and aims to give researchers interested in the topic a chance to meet, exchange ideas and perhaps also try out some of the available theorem provers. We would like to invite talks featuring demos and tutorials of inductive theorem provers, challenge problems, new directions of research or anything else of interest to the inductive theorem proving community.

If you have any questions please contact Nick Smallbone (nicsma@chalmers.se) or Moa Johansson (moa.johansson@chalmers.se).

Hope to see you in Gothenburg!

— The HipSpec crew
(Moa Johansson, Dan Rosén, Nick Smallbone, Koen Claessen)

Registration

. If you want to contribute a talk or demo, please submit a title and a short

Workshop on Automation of Pro Mathematical Indu

Held in Connection with
The 17th International Conference on
June 17-20, 2000
Pittsburgh, Pennsylvania, U.S.A.

Home
Call for papers
Challenges

CALL FOR

Proof by Mathematical Induction prese
community with some very challenging
this one day workshop is to create an i
and emerging techniques can be presen

The workshop will feature invited talks
with ample time for discussion regardi

- Inductive theorem proving and f

WAIT Workshop:
The Third Workshop on Automated Inductive Theorem-Proving

November 17-18, 2016

Seminar room Zemanek,
Vienna University of Techn
Favoritenstrasse 9-11, Build
1040 Vienna, Austria

Aim:
Inductive theorem proving
reasoning community. The
Proving - WAIT focuses on

The workshop is an informa
the topic a chance to meet,
discussions.

We invite talks featuring de

WAIT 2018 Aim Venue Program Participants Organizers

Fourth International Workshop on Automated (Co)inductive Theorem P

28–29 June 2018, Amsterdam, The Netherlands
Co-located with **Matryoshka 2018**

DEUTSCH ENGLISH

Please note our measures
concerning Coronavirus / Covid 19

SCHLO
Leibniz-Zen

About Dagstuhl

You are here: Program » Se

https://www.dagstuhl.de

July 24 – 28 , 1995, Dags

Automation of
Induction

Seminar Calendar
All Events
Dagstuhl Seminars
Dagstuhl Perspectives
GI-Dagstuhl Seminars
Summer Schools
Events

The Second Workshop on Automated Inductive Theorem-proving (WAIT?), 20-21 March 2015, Chalmers University, Gothenburg, Sweden

Inductive theorem proving is a topic of growing interest in the automated reasoning community. Following our successful first workshop in London 2013, we would like to invite you to a second workshop on inductive theorem proving. The workshop will be very informal and aims to give researchers interested in the topic a chance to meet, exchange ideas and perhaps also try out some of the available theorem provers. We would like to invite talks featuring demos and tutorials of inductive theorem provers, challenge problems, new directions of research or anything else of interest to the inductive theorem proving community.

If you have any questions please contact Nick Smallbone (nicsma@chalmers.se) or Moa Johansson (moa.johansson@chalmers.se).

Hope to see you in Gothenburg!

— The HipSpec crew
(Moa Johansson, Dan Rosén, Nick Smallbone, Koen Claessen)

Registration

. If you want to contribute a talk or demo, please submit a title and a short

Workshop on Automation of Pro Mathematical Indu

CADE SEVENTEEN

Held in Connection with
The 17th International Conference on
June 17-20, 2000
Pittsburgh, Pennsylvania, U.S.A.

Home
Call for papers
Challenges

CALL FOR

Proof by Mathematical Induction prese
community with some very challengin
this one day workshop is to create an i
and emerging techniques can be presen

The workshop will feature invited talks
with ample time for discussion regardi

- Inductive theorem proving and f

WAIT Workshop:
The Third Workshop on Automated Inductive Theorem-Proving

November 17-18, 2016

Seminar room Zemanek,
Vienna University of Techn
Favoritenstrasse 9-11, Build
1040 Vienna, Austria

Aim:
Inductive theorem proving
reasoning community. The
Proving - WAIT focuses on

The workshop is an informa
the topic a chance to meet,
discussions.

We invite talks featuring de

WAIT 2018 Aim Venue Program Participants Organizers

(CO) WAIT 2018
WORKSHOP ON AUTOMATED INDUCTIVE THEOREM PROVING

Fourth International Workshop on Automated (Co)inductive Theorem P

28–29 June 2018, Amsterdam, The Netherlands
Co-located with **Matryoshka 2018**

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
  oops
```

```
proof (prove)
goal (1 subgoal):
 1. rev2 xs ys = rev xs @ ys
```

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
apply (induct "xs" arbitrary:ys)
  oops
```

```
proof (prove)
goal (2 subgoals):
 1. ⋀ys. rev2 [] ys = rev [] @ ys
 2. ⋀a xs ys.
       (⋀ys. rev2 xs ys = rev xs @ ys) ⟹
       rev2 (a # xs) ys = rev (a # xs) @ ys
```

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
apply (induct "xs" arbitrary:ys)
  apply auto
```

```
proof (prove)
goal:
No subgoals!
```

Formal Methods in Computer-Aided Design 2020

Smart Induction for Isabelle/HOL (Tool Paper)

Yutaka Nagashima
CIIRC, Czech Technical University in Prague
University of Innsbruck
Email: yutaka.nagashima@cvut.cz

***Abstract*—Proof assistants offer tactics to facilitate inductive proofs; however, deciding what arguments to pass to these tactics still requires human ingenuity. To automate this process, we present `smart_induct` for Isabelle/HOL. Given an inductive problem in any problem domain, `smart_induct` lists promising arguments for the `induct` tactic without relying on a search. Our in-depth evaluation demonstrate that `smart_induct` produces valuable recommendations across problem domains. Currently, `smart_induct` is an interactive tool; however, we expect that `smart_induct` can be used to narrow the search space of automatic inductive provers.**

I. Introduction

Proof by induction lies at the heart of verification of computer programs that involve recursive data-structures, recursion, or iteration [1]. To facilitate proofs by induction, interactive theorem provers, such as Isabelle/HOL [2], Coq [3], and HOL[4], offers tactics. Yet, it requires prover specific expertise to be familiar with such tactics, and human developers have to manually investigate each inductive problem to decide how to apply such tactics.

Unfortunately, the automation of proof by induction is considered as a long standing challenge in computer science, for which Gramlich [1] presented the following conjecture in 2005:

> in the near future, inductive theorem proving will only be successful for very specialised domains for very restricted classes of conjectures. Inductive theorem proving will continue to be a very challenging engineering process [1].

We challenge his conjecture with `smart_induct`, a recommendation tool for proof by induction in Isabelle/HOL. Given an inductive problem in *any* domain, `smart_induct` suggests how one should apply the `induct` tactic to attack that problem.

II. Proof by Induction in Isabelle/HOL

Given the following two simple reverse functions defined in Isabelle/HOL [2], how do you prove their equivalence [5]?

```
primrec rev::"α list => α list" where
  "rev  []     = []"
| "rev (x # xs) = rev xs @ [x]"

fun itrev::"α list => α list => α list"
where
  "itrev  []    ys = ys"
| "itrev (x#xs) ys = itrev xs (x#ys)"

lemma "itrev xs ys = rev xs @ ys"
```

where `#` is the list constructor, and `@` appends two lists. Using the `induct` tactic of Isabelle/HOL, we can prove this inductive problem in multiple ways:

```
lemma prf1: "itrev xs ys = rev xs @ ys"
  apply(induct xs arbitrary: ys) by auto
lemma prf2: "itrev xs ys = rev xs @ ys"
  apply(induct xs ys rule:itrev.induct)
  by auto
```

`prf1` applies structural induction on `xs` while generalising `ys` before applying induction by passing `ys` to the `arbitrary` field. It is worth noting that the `induct` tactic determines the default induction principle in `prf1` from the induction term, `xs`. On the other hand, `prf2` applies functional induction (also known as computation induction) on `itrev` by the induction principle, `itrev.induct`, to the `rule` field.

There are other lesser-known techniques to handle difficult inductive problems using the `induct` tactic, and sometimes users have to develop useful auxiliary lemmas manually; however, for most cases the problem of how to apply induction boils down to the the following three questions:

- On which terms to apply induction?
- Which variables to generalise using the `arbitrary` field?
- Which rule to use for functional induction or rule inversion (as known as rule induction) in the `rule` field?

To answer these questions automatically, we previously developed a proof strategy language, `PSL` [6]. Given an inductive problem, `PSL` produces various combinations of induction arguments for the `induct` tactic and conducts an extensive proof search based on a given strategy. If `PSL` completes a proof search, it identifies the appropriate combination of arguments for the problem and presents the combination to the user; however, when the search space becomes enormous, `PSL` cannot find a proof within a realistic timeout and fails to provide any recommendation, even if `PSL` produces the right combination of induction arguments. For further automation of proof by induction, we need a tool that satisfies the following two criteria:

- The tool suggests right induction arguments without completing a proof search.
- The tool suggests right induction arguments for any inductive problems.

https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_32

Formal Methods in Computer-Aided Design 2020

Smart Induction for Isabelle/HOL (Tool Paper)

Yutaka Nagashima
CIIRC, Czech Technical University in Prague
University of Innsbruck
Email: yutaka.nagashima@cvut.cz

***Abstract*—Proof assistants offer tactics to facilitate inductive proofs; however, deciding what arguments to pass to these tactics still requires human ingenuity. To automate this process, we present `smart_induct` for Isabelle/HOL. Given an inductive problem in any problem domain, `smart_induct` lists promising arguments for the `induct` tactic without relying on a search. Our in-depth evaluation demonstrate that `smart_induct` produces valuable recommendations across problem domains. Currently, `smart_induct` is an interactive tool; however, we expect that `smart_induct` can be used to narrow the search space of automatic inductive provers.**

I. Introduction

Proof by induction lies at the heart of verification of computer programs that involve recursive data-structures, recursion, or iteration [1]. To facilitate proofs by induction, interactive theorem provers, such as Isabelle/HOL [2], Coq [3], and HOL[4], offers tactics. Yet, it requires prover specific expertise to be familiar with such tactics, and human developers have to manually investigate each inductive problem to decide how to apply such tactics.

Unfortunately, the automation of proof by induction is considered as a long standing challenge in computer science, for which Gramlich [1] presented the following conjecture in 2005:

> in the near future, inductive theorem proving will only be successful for very specialised domains for very restricted classes of conjectures. Inductive theorem proving will continue to be a very challenging engineering process [1].

We challenge his conjecture with `smart_induct`, a recommendation tool for proof by induction in Isabelle/HOL. Given an inductive problem in *any* domain, `smart_induct` suggests how one should apply the `induct` tactic to attack that problem.

II. Proof by Induction in Isabelle/HOL

Given the following two simple reverse functions defined in Isabelle/HOL [2], how do you prove their equivalence [5]?

```
primrec rev::"α list => α list" where
  "rev  []     = []"
| "rev (x # xs) = rev xs @ [x]"

fun itrev::"α list => α list => α list"
where
  "itrev  []    ys = ys"
| "itrev (x#xs) ys = itrev xs (x#ys)"
```

```
lemma "itrev xs ys =
```

where # is the list construc
Using the `induct` tactic of I
inductive problem in multiple

```
lemma prf1: "itrev x
  apply(induct xs ar
lemma prf2: "itrev x
  apply(induct xs ys
  by auto
```

`prf1` applies structural induct
before applying induction by
field. It is worth noting that th
default induction principle in
`xs`. On the other hand, `prf`
(also known as computation
induction principle, `itrev.i`

There are other lesser-know
inductive problems using the
users have to develop usefu
however, for most cases the pr
boils down to the the followi

- On which terms to apply
- Which variables to gen
  field?
- Which rule to use for fu
  sion (as known as rule i

To answer these questions
veloped a proof strategy langu
problem, `PSL` produces vari
arguments for the `induct` t
proof search based on a giv
a proof search, it identifies the appropriate combination of arguments for the problem and presents the combination to the user; however, when the search space becomes enormous, `PSL` cannot find a proof within a realistic timeout and fails to provide any recommendation, even if `PSL` produces the right combination of induction arguments. For further automation of proof by induction, we need a tool that satisfies the following two criteria:

- The tool suggests right induction arguments without completing a proof search.
- The tool suggests right induction arguments for any inductive problems.

https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_32

V. Conclusion

We presented `smart_induct`, a recommendation tool for proof by induction in Isabelle/HOL. Our evaluation showed `smart_induct`'s excellent performance in recommending how to apply functional induction and rule inversion and good performance at identifying induction variables for structural induction for various inductive problems across problem domains. This partially refutes Gramlich's bleak conjecture from 2005. However, recommendation of variable generalisation remains as a challenging task.

Formal Methods in Computer-Aided Design 2020

Smart Induction for Isabelle/HOL (Tool Paper)

Yutaka Nagashima
CIIRC, Czech Technical University in Prague
University of Innsbruck
Email: yutaka.nagashima@cvut.cz

***Abstract*—Proof assistants offer tactics to facilitate inductive proofs; however, deciding what arguments to pass to these tactics still requires human ingenuity. To automate this process, we present `smart_induct` for Isabelle/HOL. Given an inductive problem in any problem domain, `smart_induct` lists promising arguments for the `induct` tactic without relying on a search. Our in-depth evaluation demonstrate that `smart_induct` produces valuable recommendations across problem domains. Currently, `smart_induct` is an interactive tool; however, we expect that `smart_induct` can be used to narrow the search space of automatic inductive provers.**

I. Introduction

Proof by induction lies at the heart of verification of computer programs that involve recursive data-structures, recursion, or iteration [1]. To facilitate proofs by induction, interactive theorem provers, such as Isabelle/HOL [2], Coq [3], and HOL[4], offers tactics. Yet, it requires prover specific expertise to be familiar with such tactics, and human developers have to manually investigate each inductive problem to decide how to apply such tactics.

Unfortunately, the automation of proof by induction is considered as a long standing challenge in computer science, for which Gramlich [1] presented the following conjecture in 2005:

> in the near future, inductive theorem proving will only be successful for very specialised domains for very restricted classes of conjectures. Inductive theorem proving will continue to be a very challenging engineering process [1].

We challenge his conjecture with `smart_induct`, a recommendation tool for proof by induction in Isabelle/HOL. Given an inductive problem in *any* domain, `smart_induct` suggests how one should apply the `induct` tactic to attack that problem.

II. Proof by Induction in Isabelle/HOL

Given the following two simple reverse functions defined in Isabelle/HOL [2], how do you prove their equivalence [5]?

```
primrec rev::"α list => α list" where
  "rev  []     = []"
| "rev (x # xs) = rev xs @ [x]"

fun itrev::"α list => α list => α list"
where
  "itrev  []    ys = ys"
| "itrev (x#xs) ys = itrev xs (x#ys)"
```

```
lemma "itrev xs ys =
```

where # is the list construc
Using the `induct` tactic of I
inductive problem in multiple

```
lemma prf1: "itrev x
  apply(induct xs ar
lemma prf2: "itrev x
  apply(induct xs ys
  by auto
```

`prf1` applies structural induct
before applying induction by
field. It is worth noting that th
default induction principle in
`xs`. On the other hand, `prf`
(also known as computation
induction principle, `itrev.i`

There are other lesser-know
inductive problems using the
users have to develop usefu
however, for most cases the pr
boils down to the the followi

- On which terms to apply
- Which variables to gen
  field?
- Which rule to use for fu
  sion (as known as rule i

To answer these questions
veloped a proof strategy langu
problem, `PSL` produces vari
arguments for the `induct` t
proof search based on a giv
a proof search, it identifies the appropriate combination of arguments for the problem and presents the combination to the user; however, when the search space becomes enormous, `PSL` cannot find a proof within a realistic timeout and fails to provide any recommendation, even if `PSL` produces the right combination of induction arguments. For further automation of proof by induction, we need a tool that satisfies the following two criteria:

- The tool suggests right induction arguments without completing a proof search.
- The tool suggests right induction arguments for any inductive problems.

https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_32

V. Conclusion

We presented `smart_induct`, a recommendation tool for proof by induction in Isabelle/HOL. Our evaluation showed `smart_induct`'s excellent performance in recommending how to apply functional induction and rule inversion and good performance at identifying induction variables for structural induction for various inductive problems across problem domains. This partially refutes Gramlich's bleak conjecture from 2005. However, recommendation of variable generalisation remains as a challenging task.

https://www.ijcai.org/proceedings/2021/273

Faster Smarter Proof by Induction in Isabelle/HOL

Yutaka Nagashima
Yale-NUS College, National University of Singapore
University of Innsbruck
Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague
yutaka@yale-nus.edu.sg

Abstract

We present `sem_ind`, a recommendation tool for proof by induction in Isabelle/HOL. Given an inductive problem, `sem_ind` produces candidate arguments for proof by induction, and selects promising ones using heuristics. Our evaluation based on 1,095 inductive problems from 22 source files shows that `sem_ind` improves the accuracy of recommendation from 20.1% to 38.2% for the most promising candidates within 5.0 seconds of timeout compared to its predecessor while decreasing the median value of execution time from 2.79 seconds to 1.06 seconds.

Program 1 Equivalence of two reverse functions

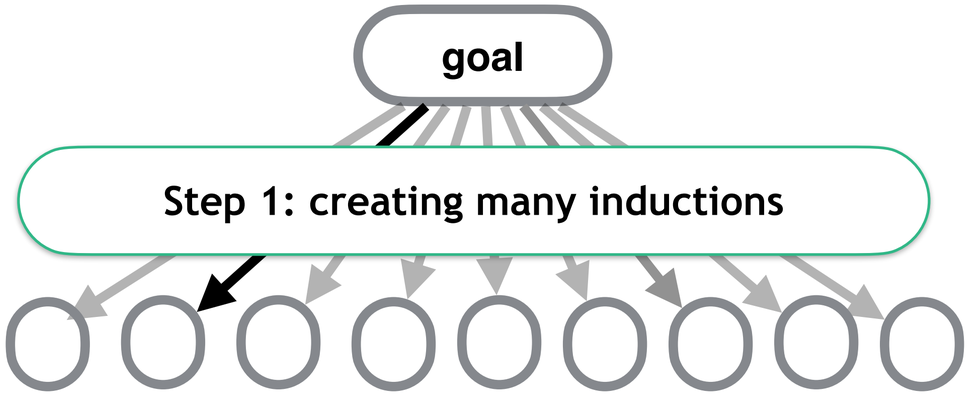
```
@ :: α list ⇒ α list ⇒ α list
  []       @ ys = ys
| (x # xs) @ ys = x # (xs @ ys)

rev1 :: α list ⇒ α list
  rev1 []       = []
| rev1 (x # xs) = rev1 xs @ [x]

rev2 :: α list ⇒ α list ⇒ α list
  rev2 []       ys = ys
| rev2 (x # xs) ys = rev2 xs (x # ys)

theorem rev2 xs ys = rev1 xs @ ys
```

smart induction

(FMCAD2020)

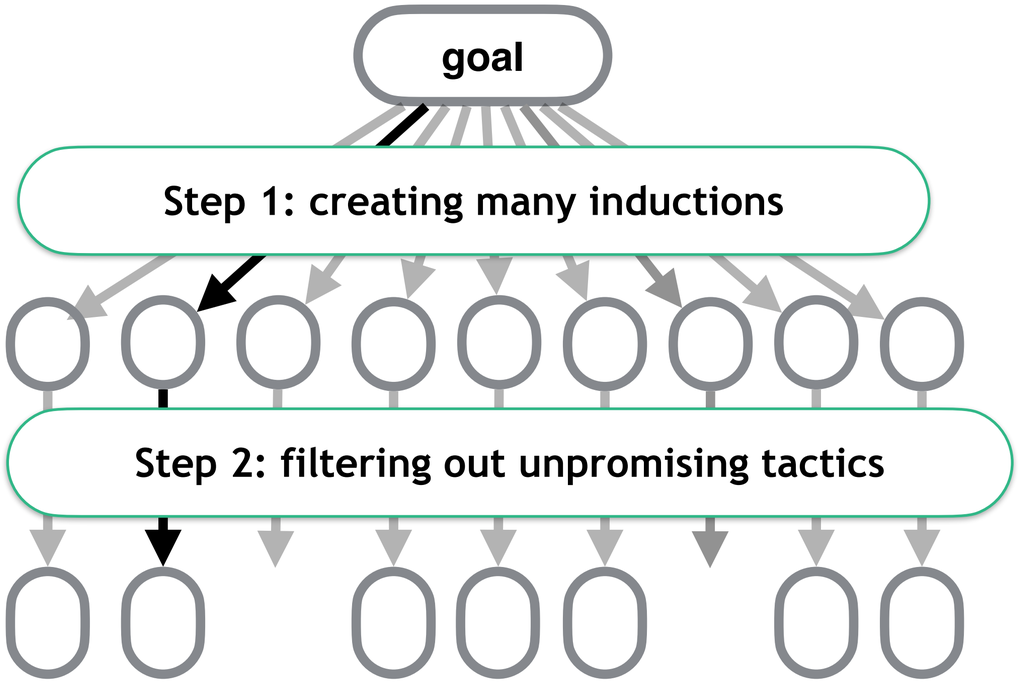
smart induction

(FMCAD2020)



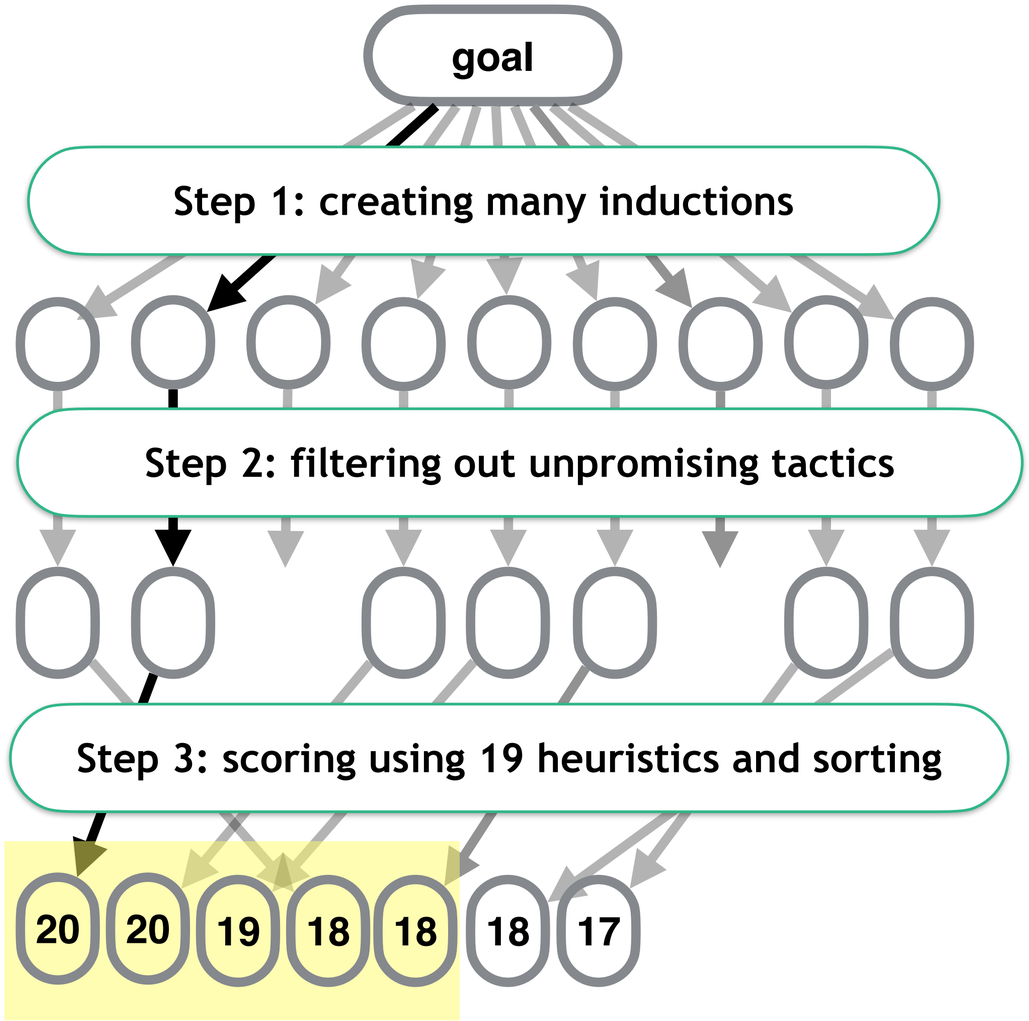
smart induction

(FMCAD2020)



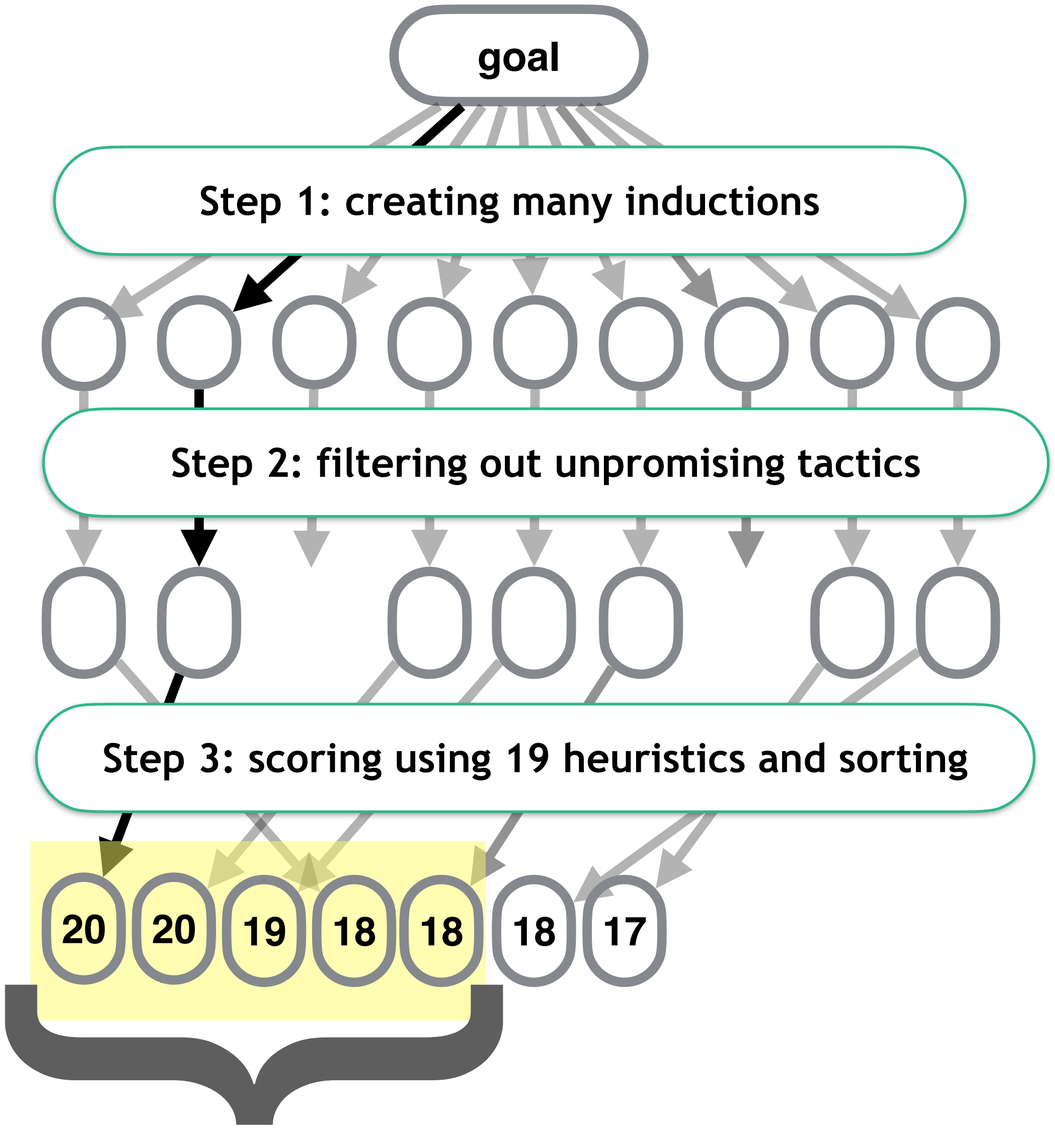
smart induction

(FMCAD2020)

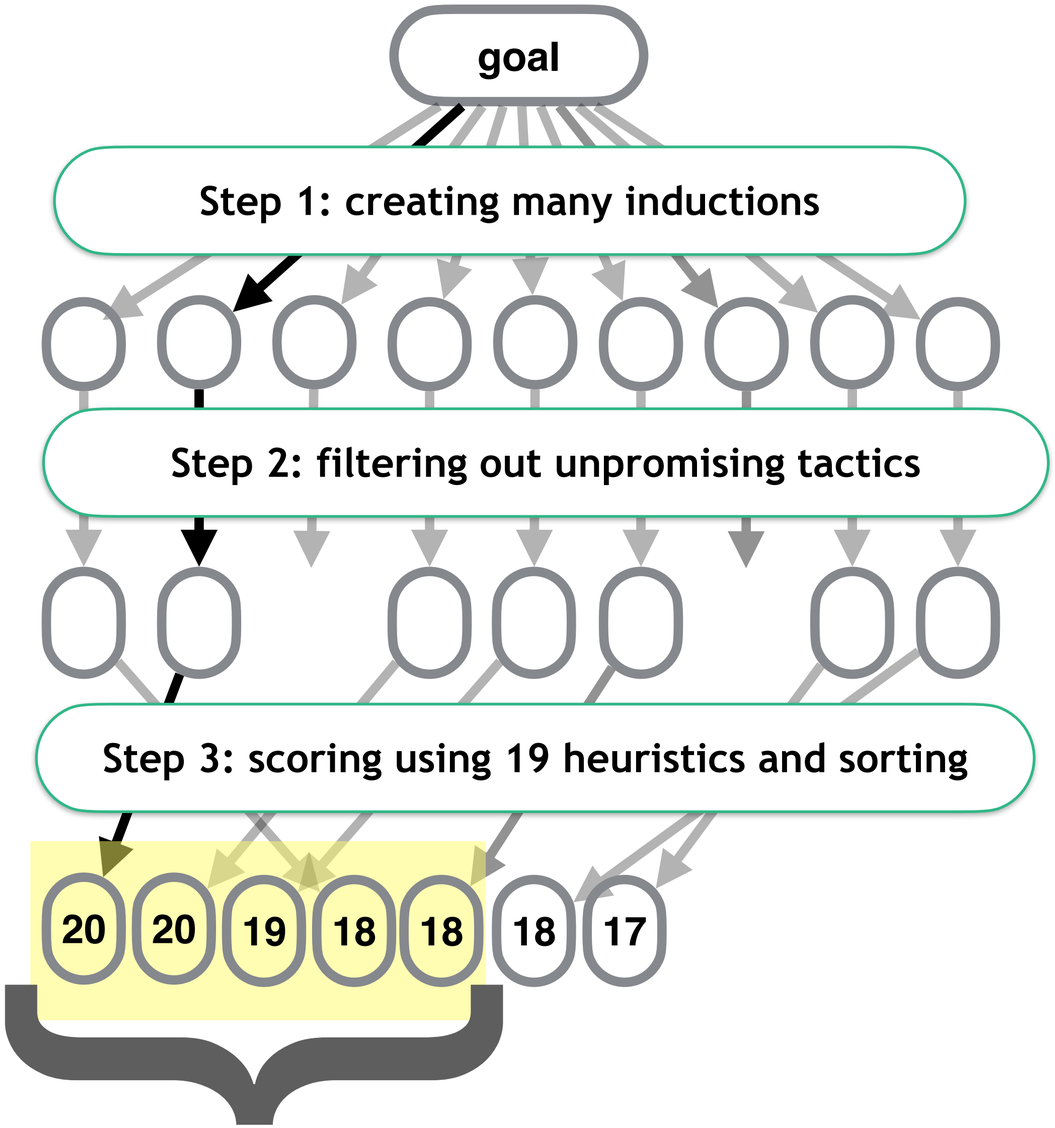


smart induction

(FMCAD2020)

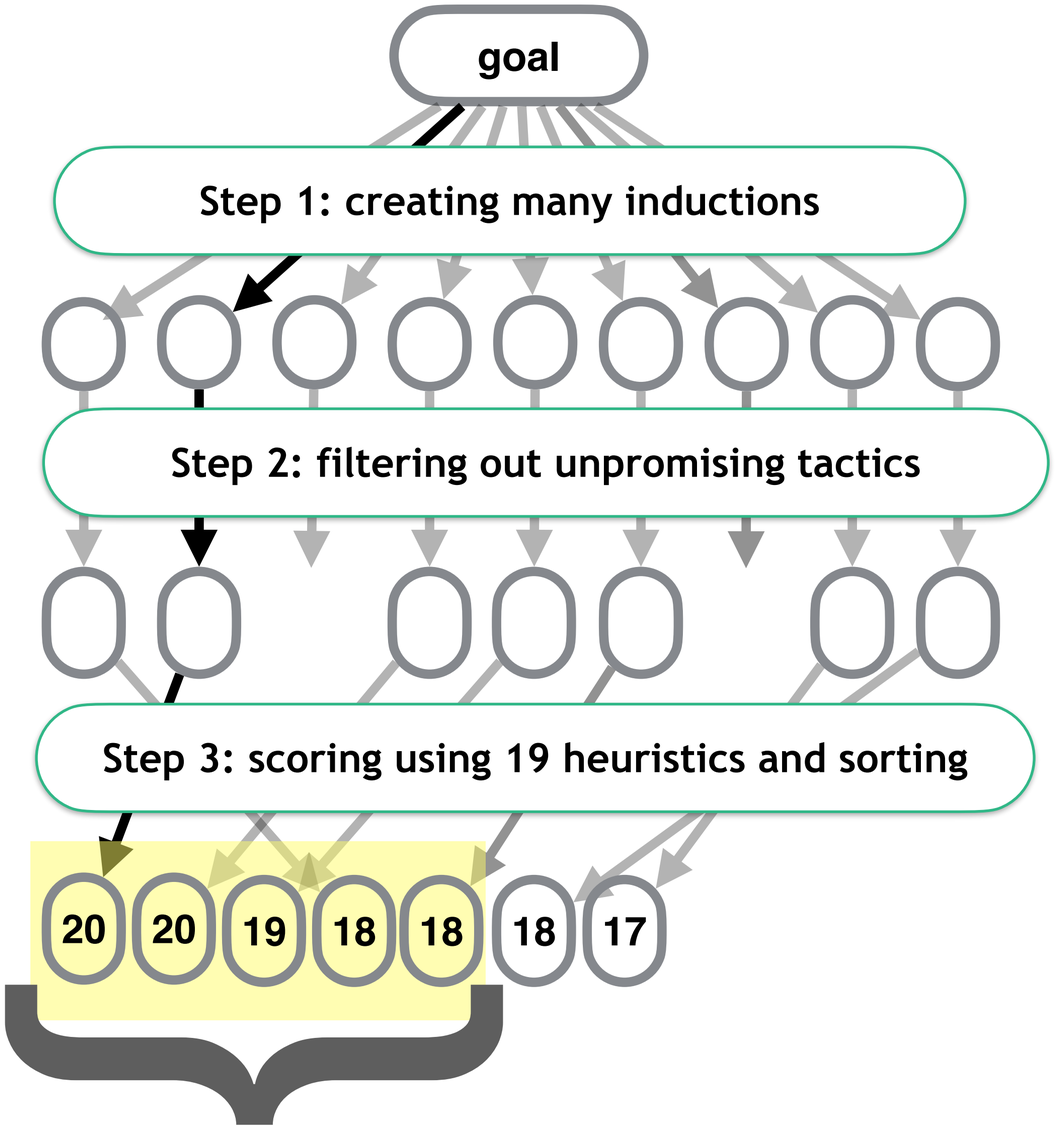


smart induction
(FMCAD2020)

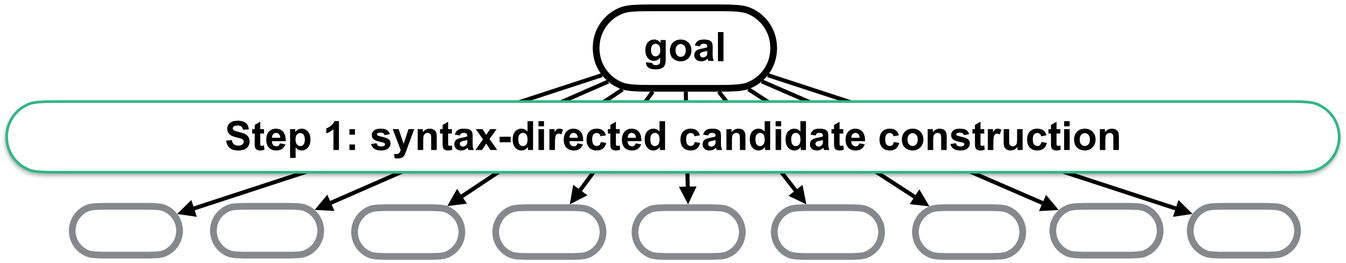


sem_ind
(IJCAI2011)

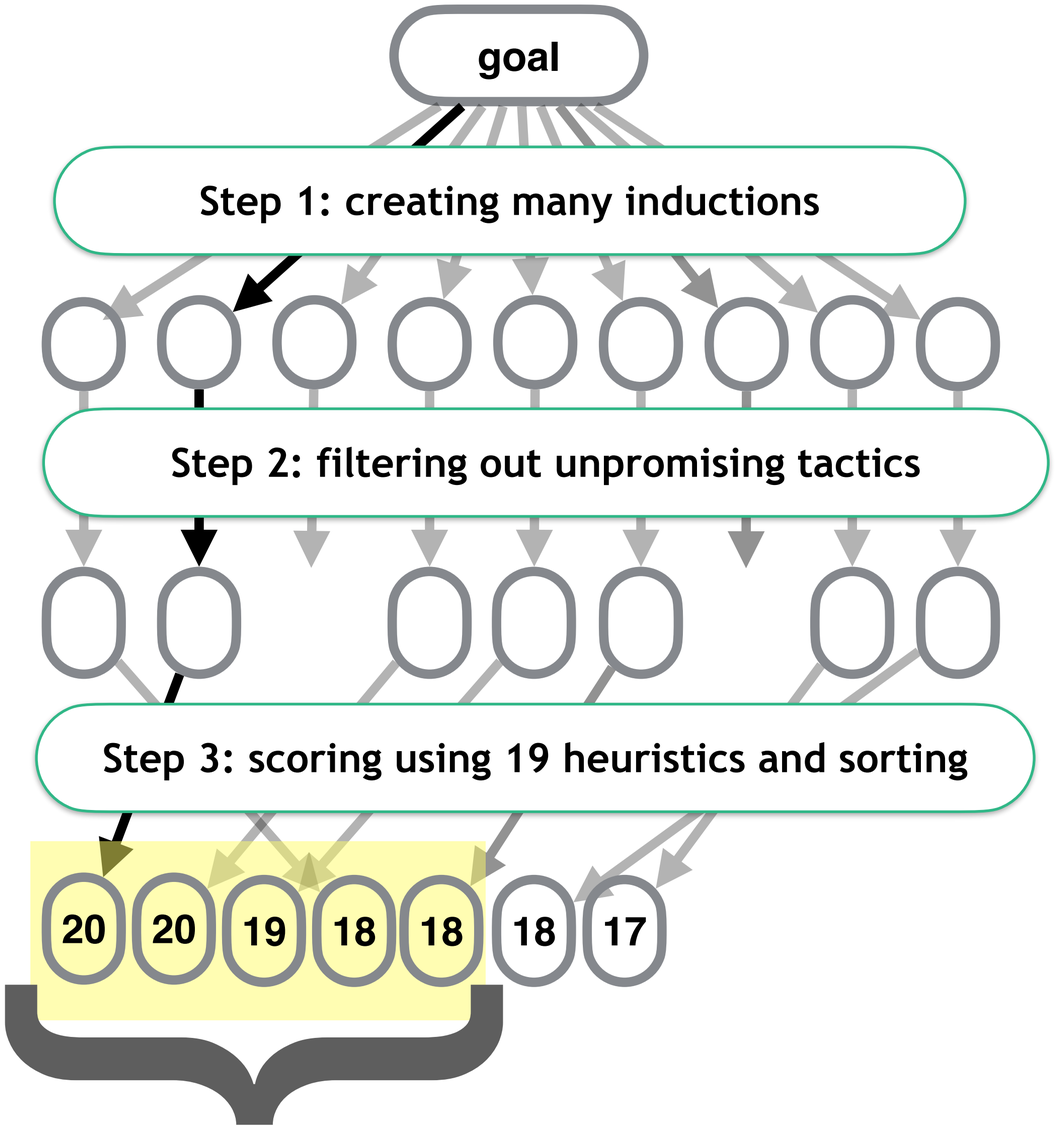
smart induction
(FMCAD2020)



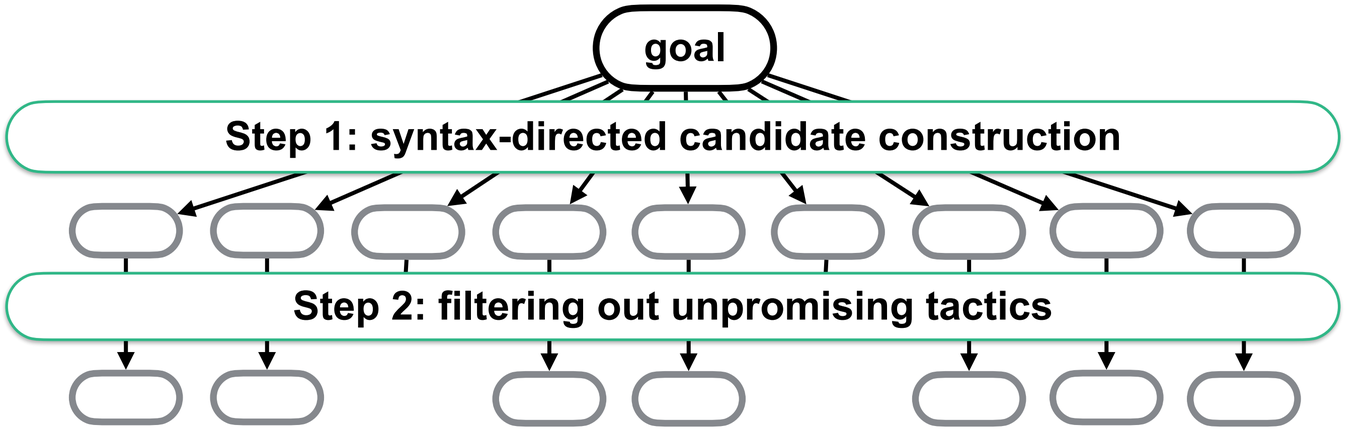
sem_ind
(IJCAI2011)



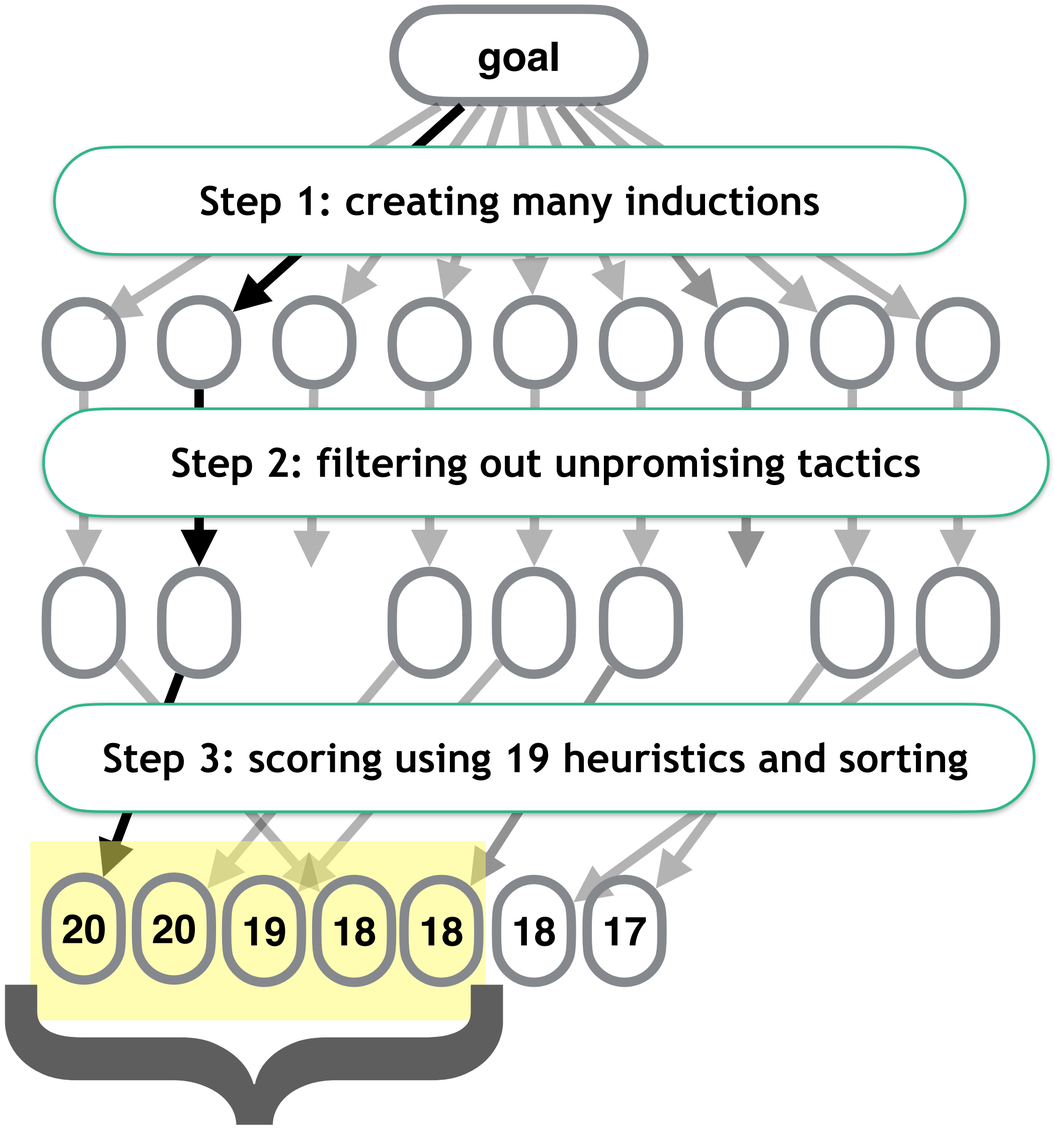
smart induction
(FMCAD2020)



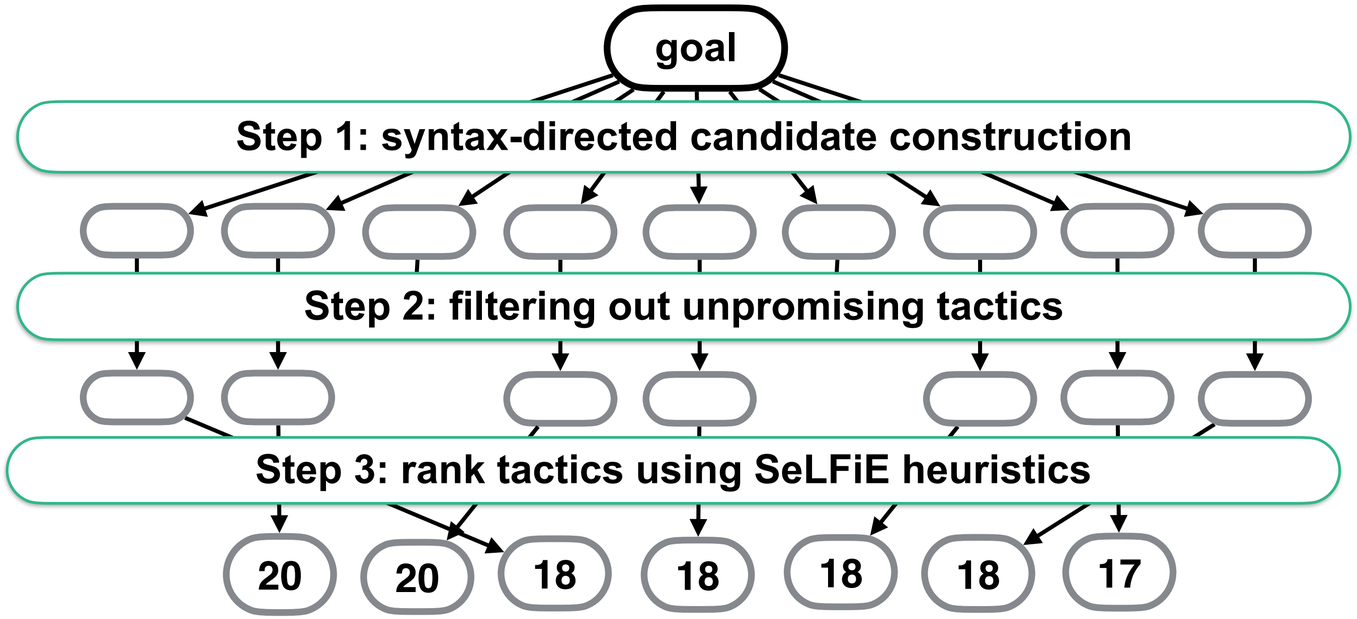
sem_ind
(IJCAI2011)



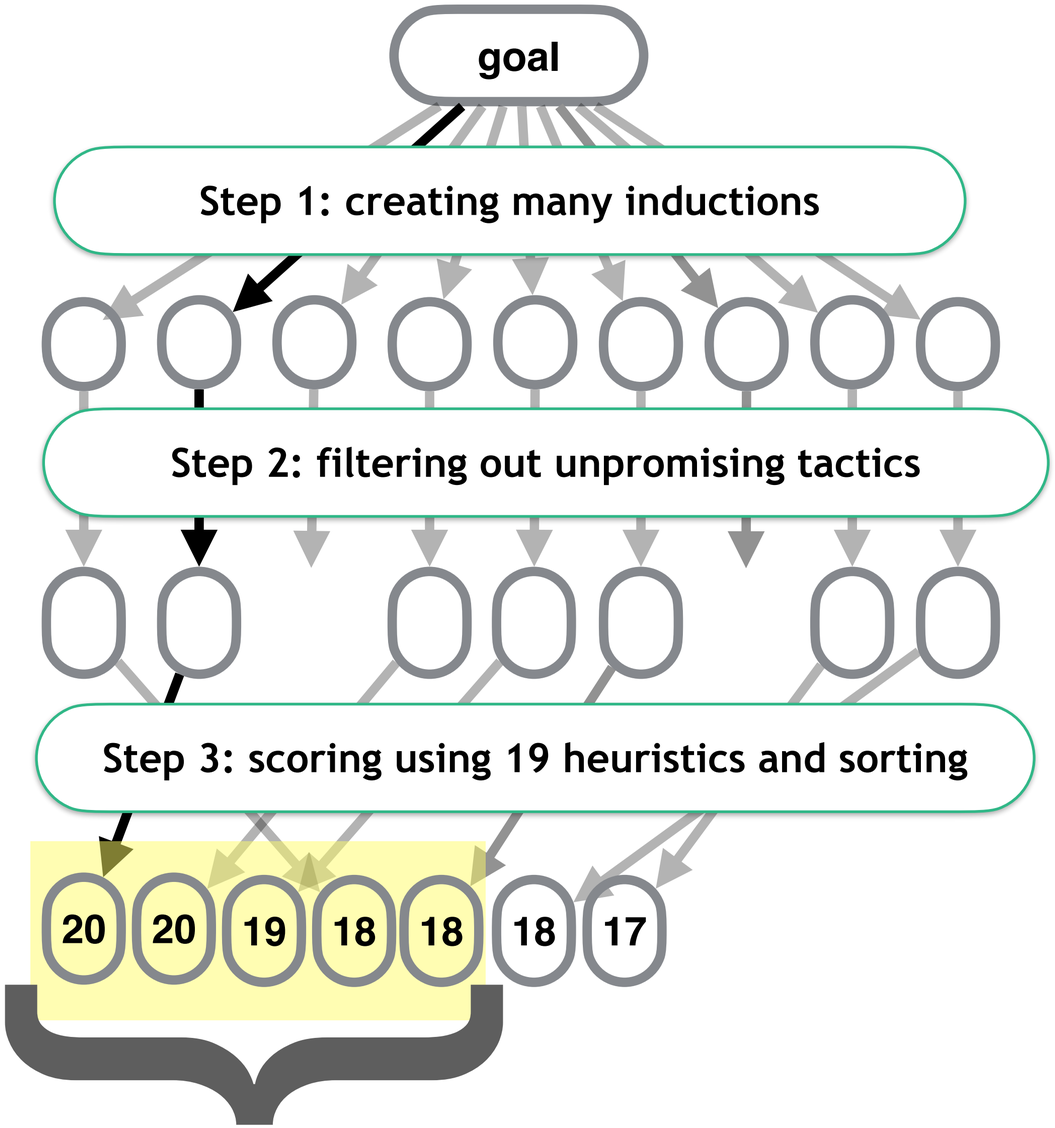
smart induction
(FMCAD2020)



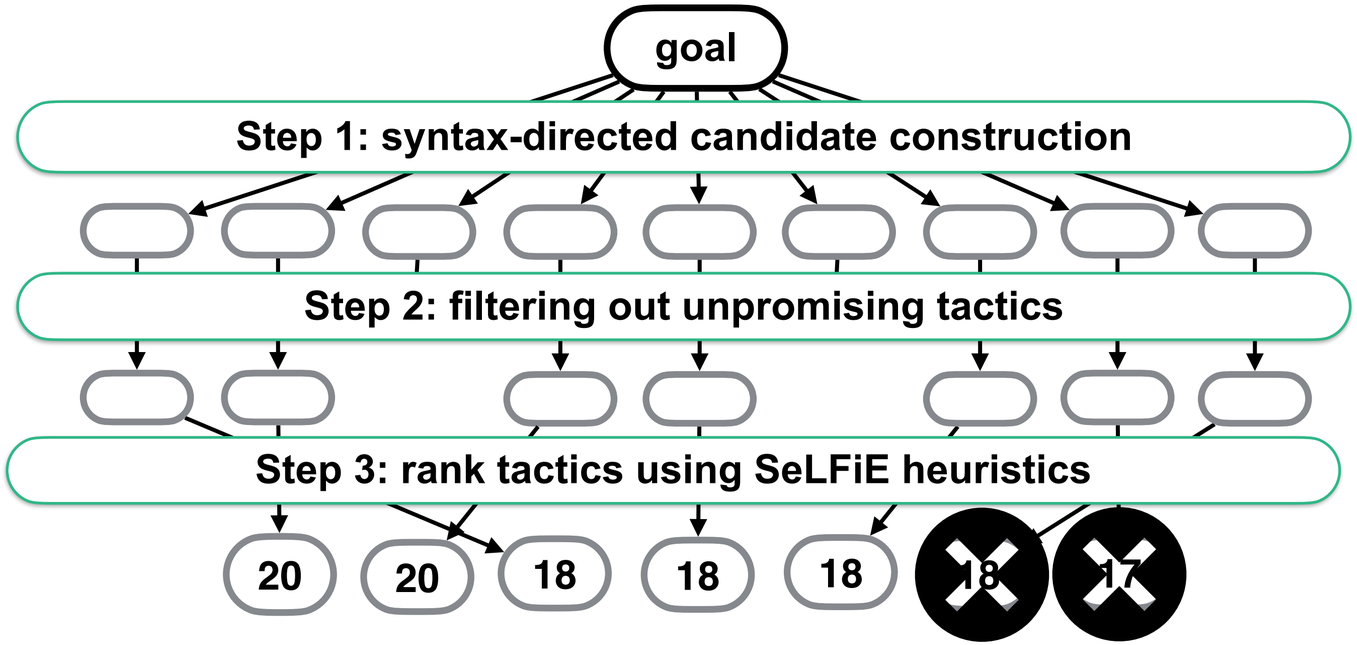
sem_ind
(IJCAI2011)



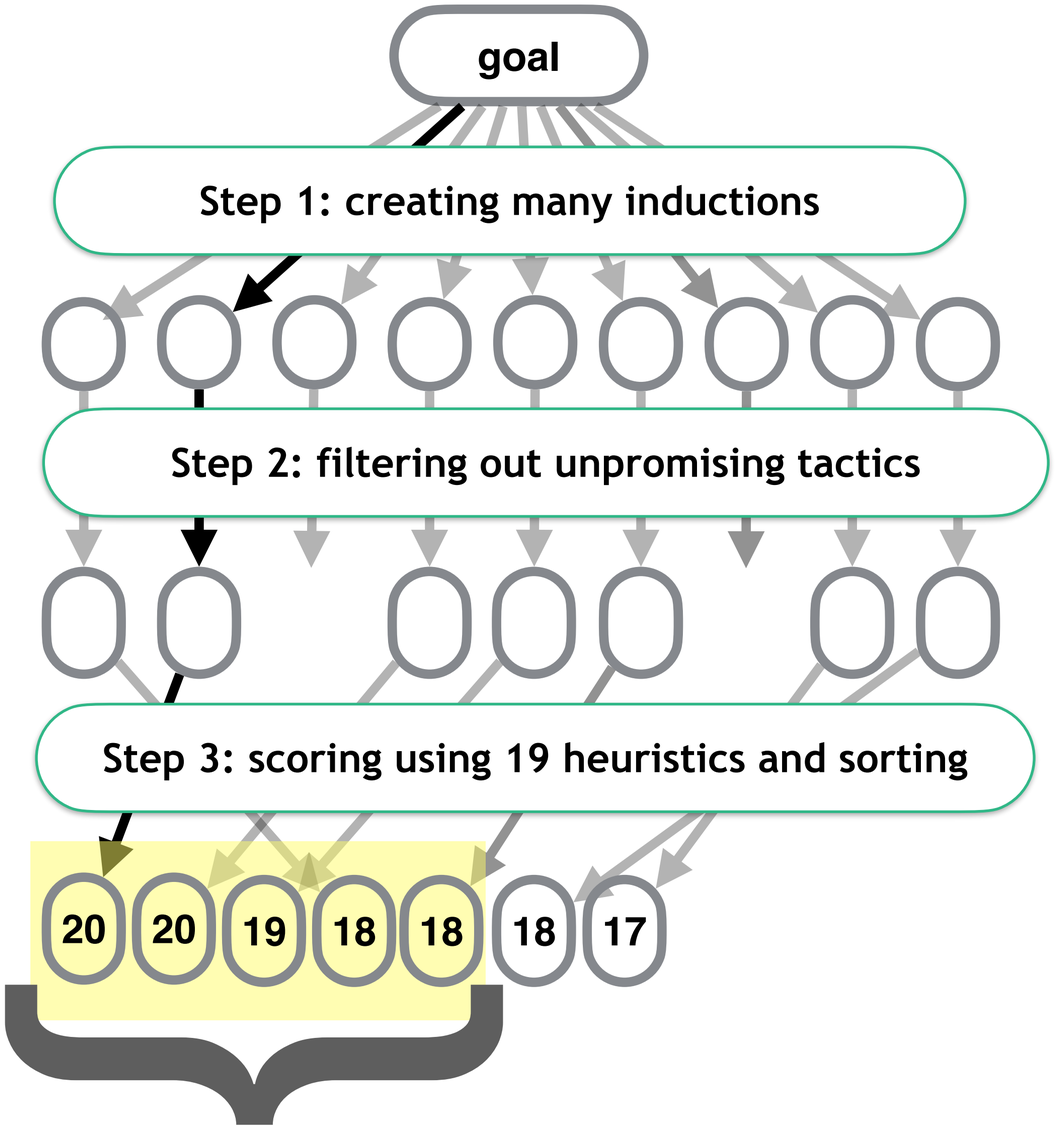
smart induction
(FMCAD2020)



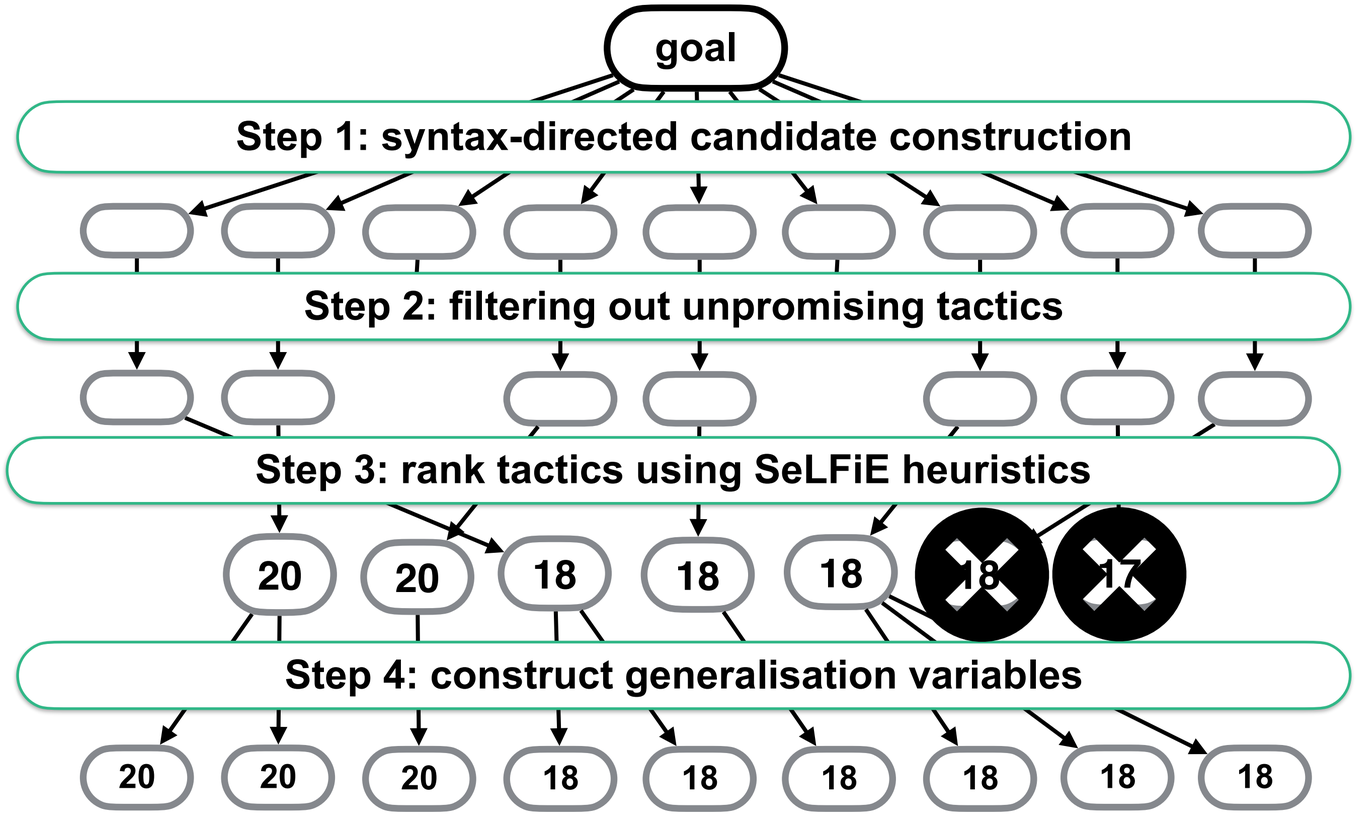
sem_ind
(IJCAI2011)



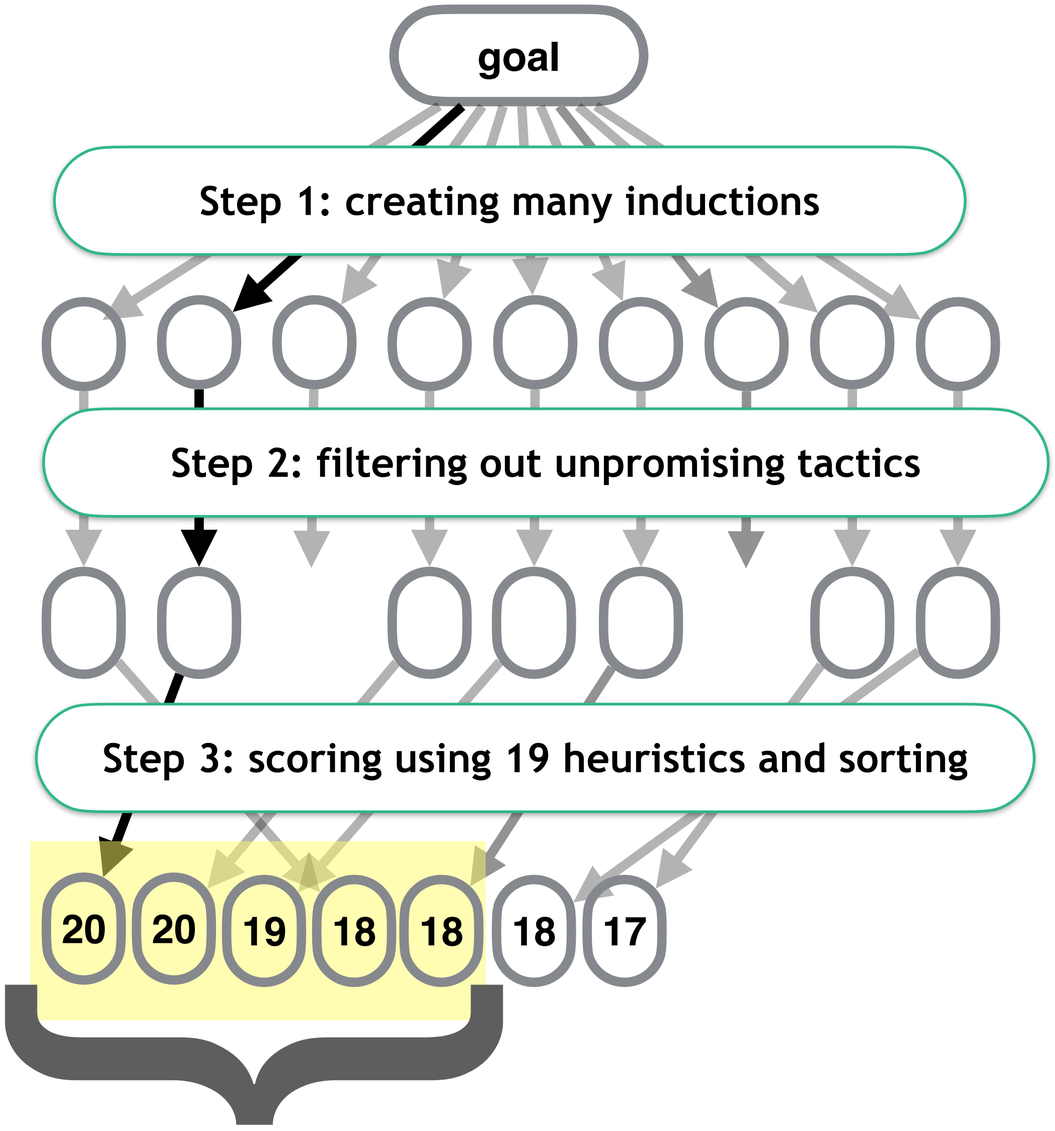
smart induction
(FMCAD2020)



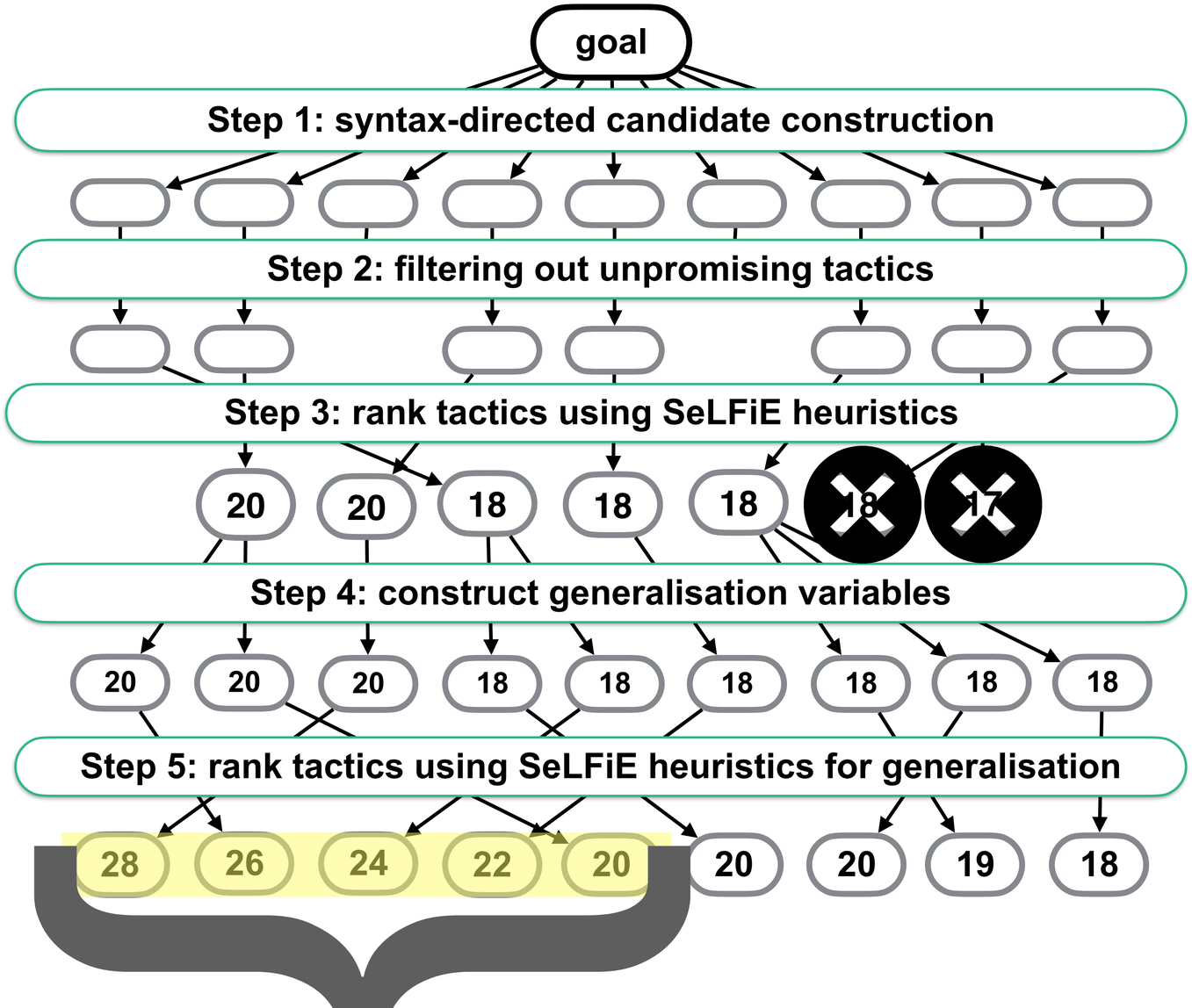
sem_ind
(IJCAI2011)



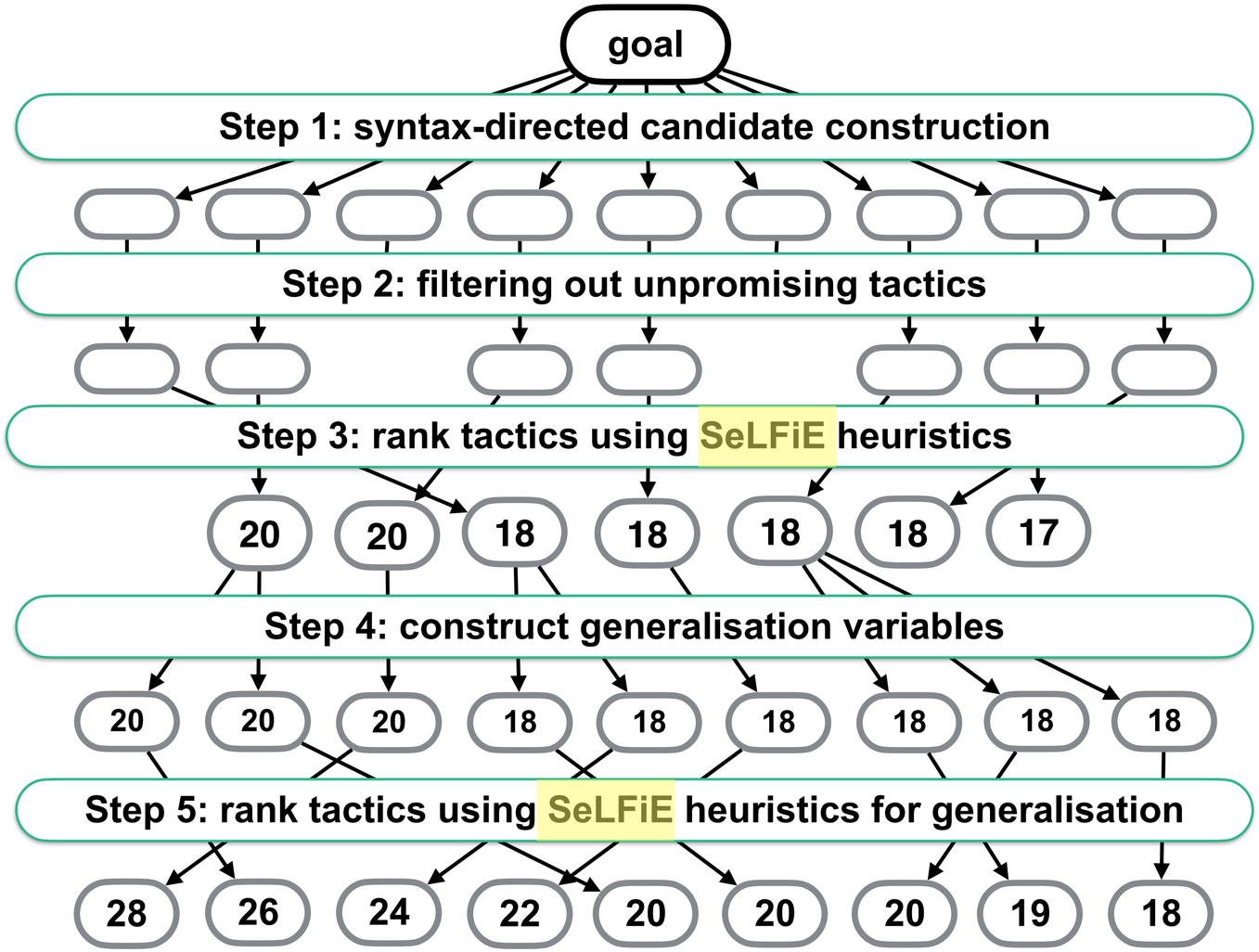
smart induction
(FMCAD2020)



sem_ind
(IJCAI2011)

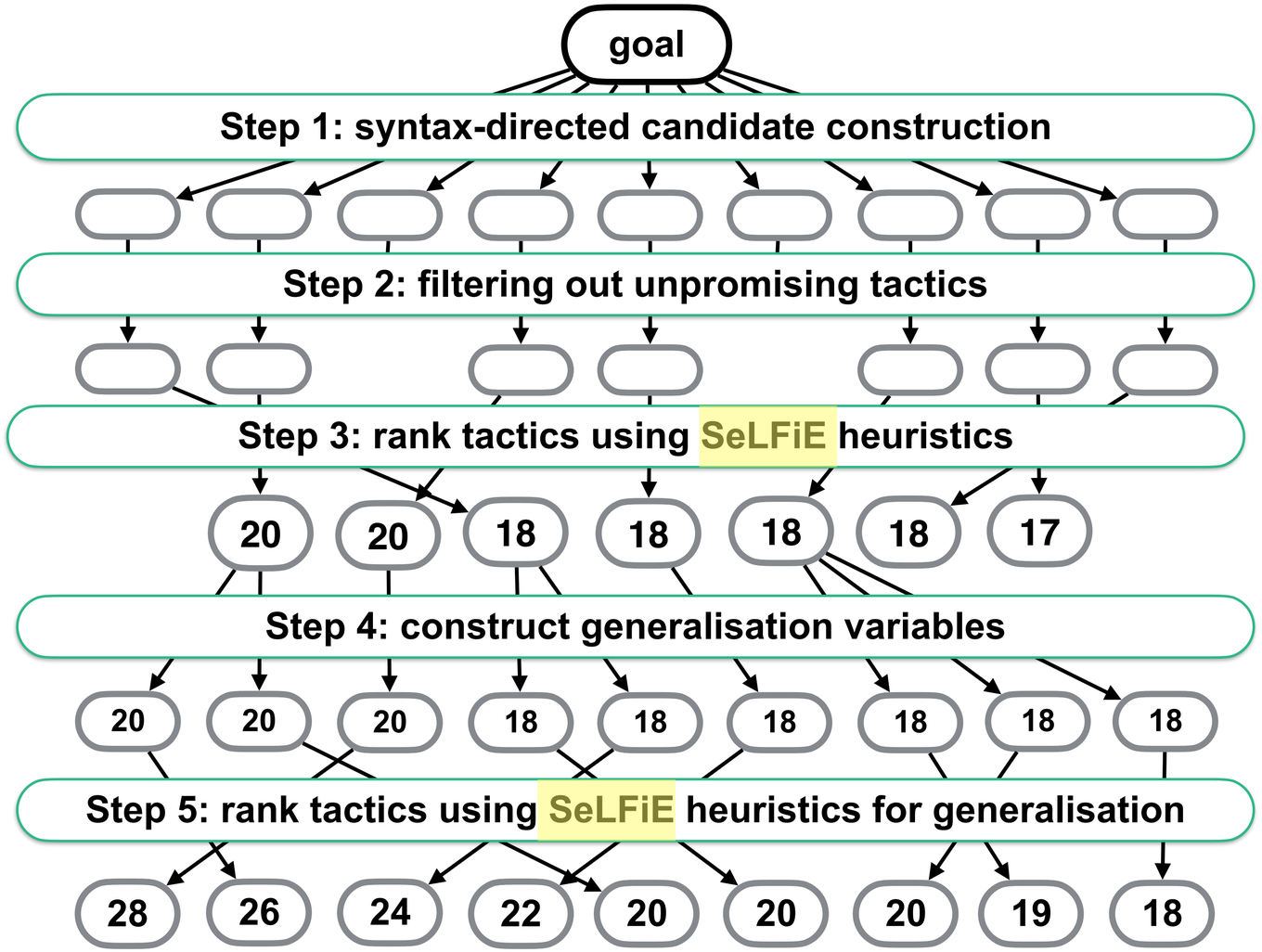


sem_ind
(IJCAI2011)



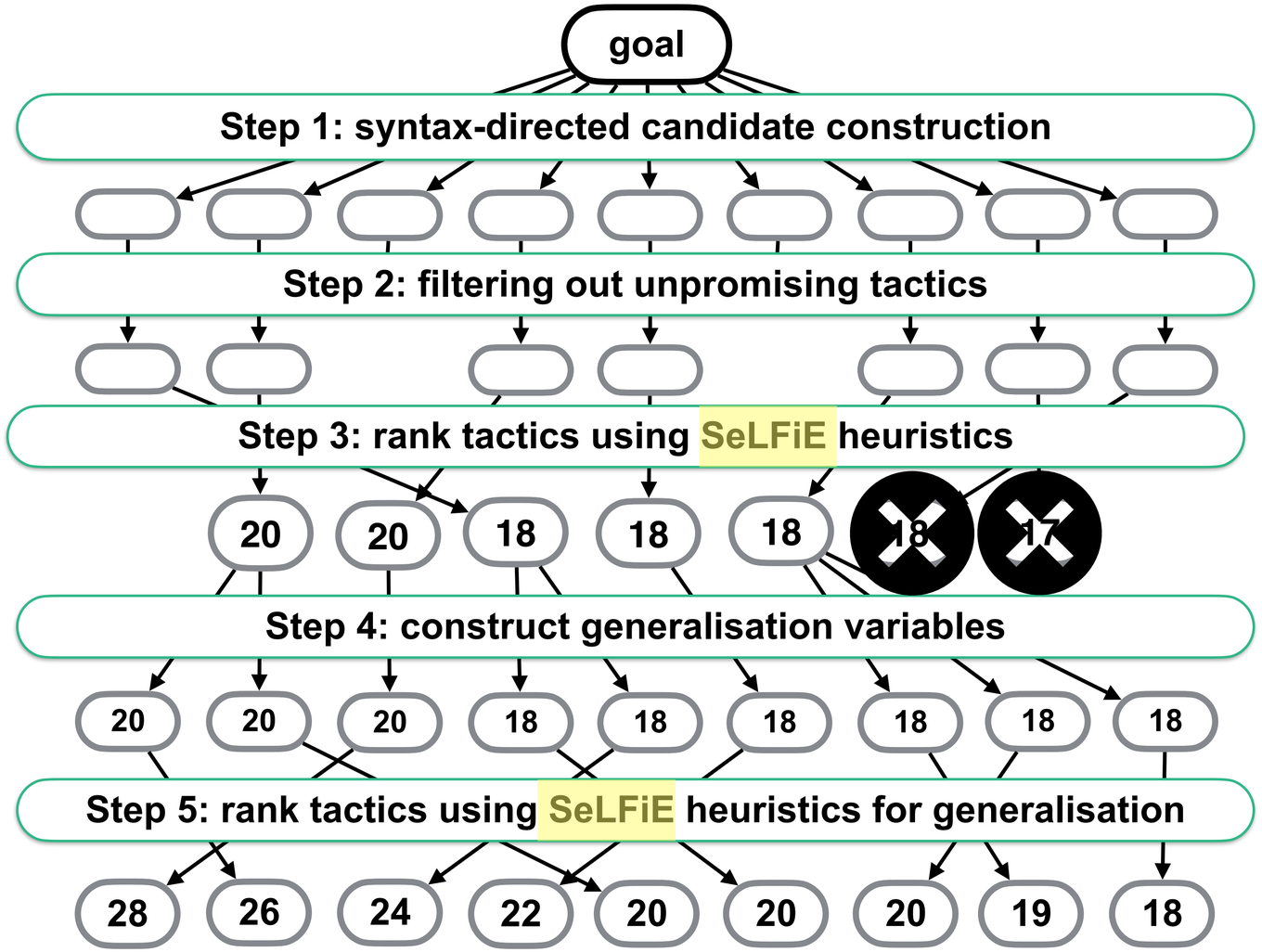
sem_ind
(IJCAI2011)

induction terms



sem_ind
(IJCAI2011)

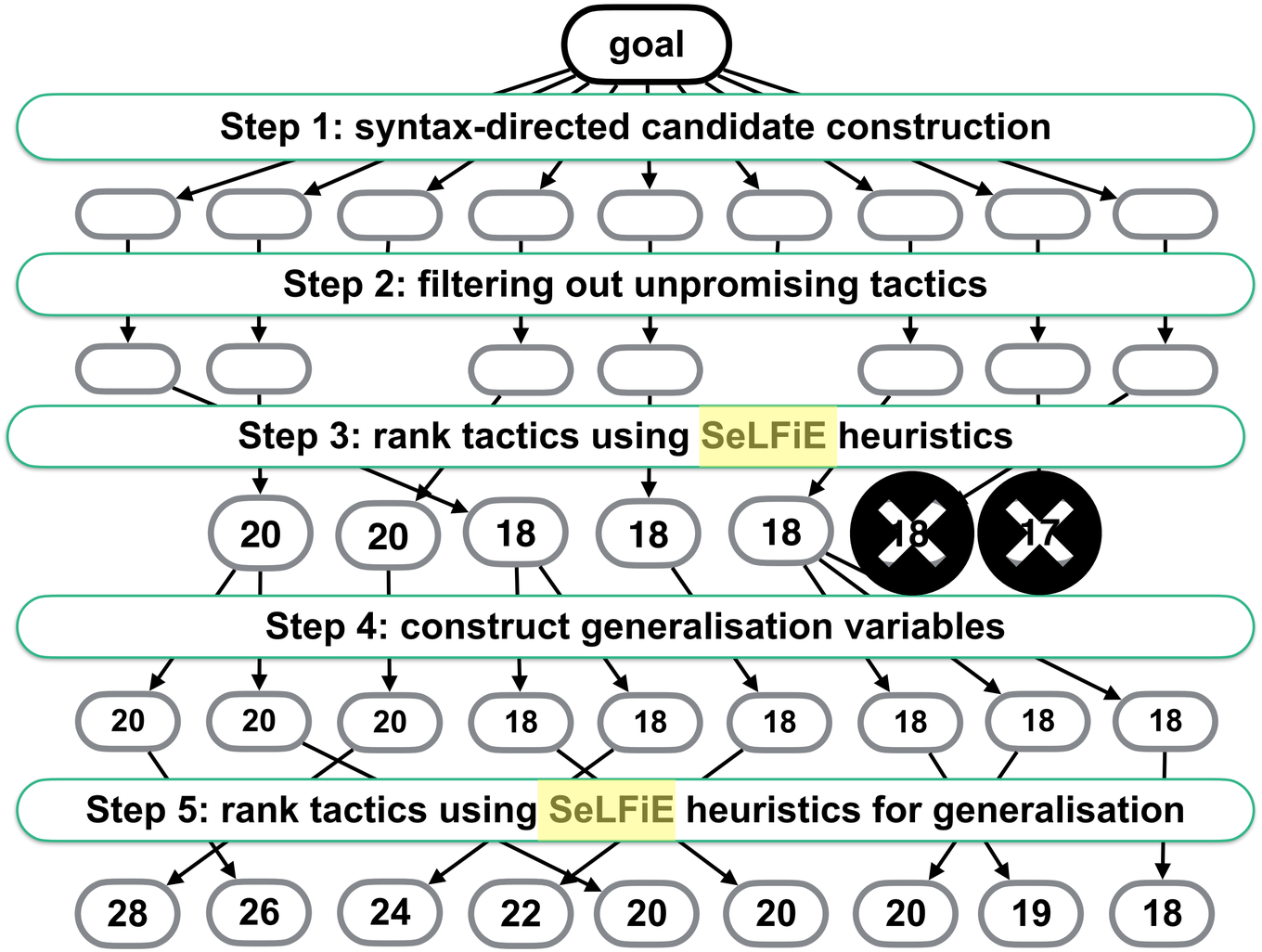
induction terms



sem_ind
(IJCAI2011)

induction terms

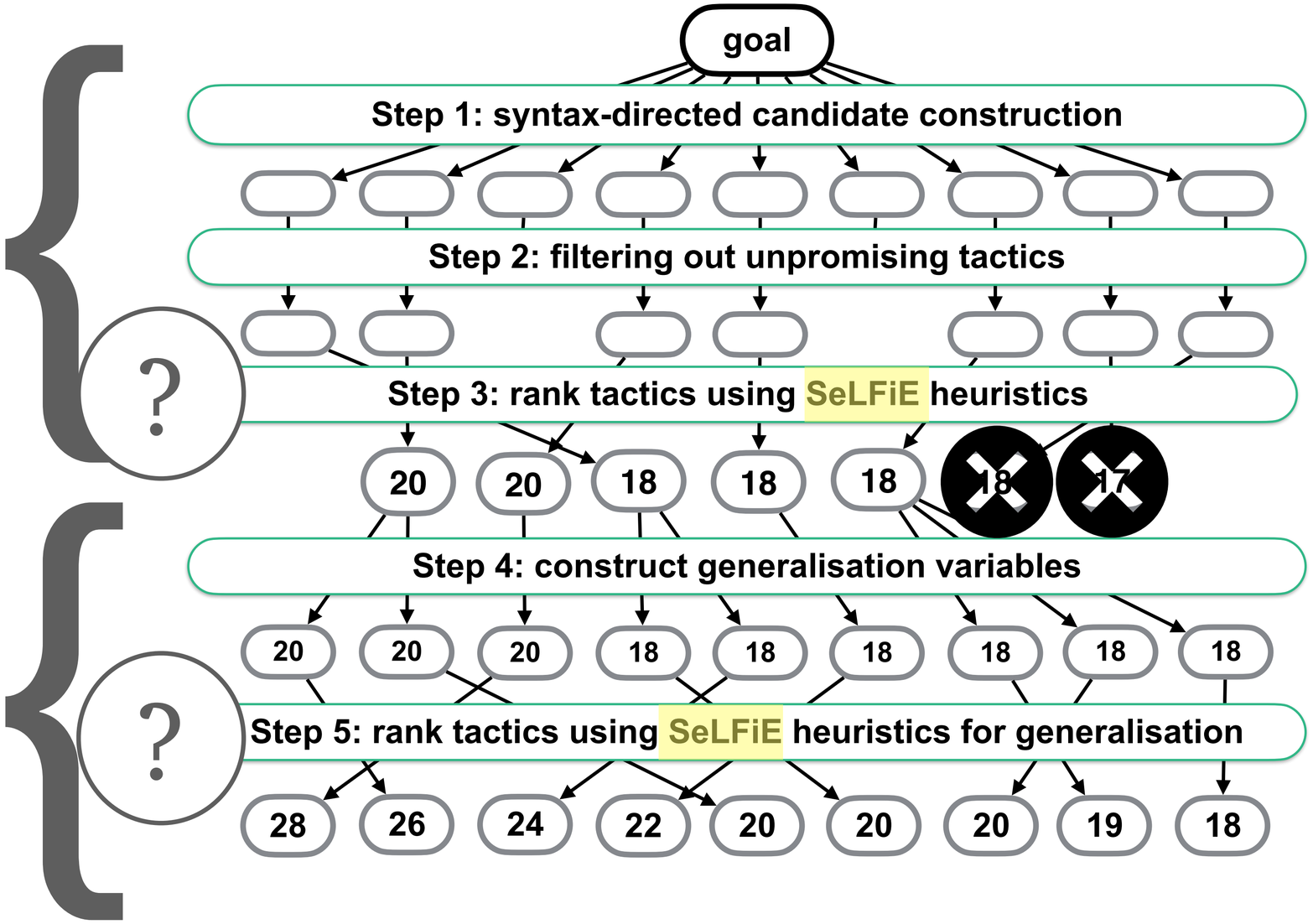
variable generalisation



sem_ind
(IJCAI2011)

induction terms

variable generalisation



Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
      is_or_below_nth_argument_of
        (arb_occ, generalise_nth, f_occ)
     ∧
      ∃_def
        (f_term,
         generalise_nth_argument_of,
         [generalise_nth, f_term])
```

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
       (nth_param_on_rhs, nth_param_on_lhs)
     ∧
      ∃ f_occ_on_rhs : term_occ ∈ f_term.
       is_nth_argument_of
         (nth_param_on_rhs,
          generalise_nth,
          f_occ_on_rhs)
```

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
     is_or                                  f
        (ar                                 cc)
    ∧
     ∃def
        (f_term,
         generalise_nth_argument_of,
         [generalise_nth, f_term])
```

definitional quantifier

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
       (nth_param_on_rhs, nth_param_on_lhs)
     ∧
      ∃ f_occ_on_rhs : term_occ ∈ f_term.
       is_nth_argument_of
          (nth_param_on_rhs,
           generalise_nth,
           f_occ_on_rhs)
```

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
     is_or                                  f
        (ar                                 cc)
    ∧
     ∃_def
        (f_term,
         generalise_nth_argument_of,
         [generalise_nth, f_term])
```

definitional quantifier

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
       (nth_param_on_rhs, nth_param_on_lhs)
     ∧
      ∃ f_occ_on_rhs : term_occ ∈ f_term.
       is_nth_argument_of
          (nth_param_on_rhs,
           generalise_nth,
           f_occ_on_rhs)
```

Question about definitions.

Answer about definitions.

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
     is_or                                  f
        (ar                                 cc)
     ∧
      ∃def
        (f_term,
         generalise_nth_argument_of,
         [generalise_nth, f_term])
```

definitional quantifier

Checks if **generalise_nth_argument_of** returns **True** when applied to *generalise_nth* and *f_term* for some clause defining *f_term*.

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
       (nth_param_on_rhs, nth_param_on_lhs)

                    rhs : term_occ ∈ f_term.
                    argument_of
                    ram_on_rhs,
                   generalise_nth,
                   f_occ_on_rhs)
```

Question about definitions.

Answer about definitions.

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
     is_or                                  f
        (ar                                 cc)
     ∧
      ∃def
        (f_term,
         generalise_nth_argument_of,
         [generalise_nth, f_term])
```

definitional quantifier

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
                    m_on_rhs, nth_param_on_lhs)

                rhs : term_occ ∈ f_term.
               rgument_of
               ram_on_rhs,
    generalise_nth,
    f_occ_on_rhs)
```

Checks if **generalise_nth_argument_of** returns **True** when applied to *generalise_nth* and *f_term* for some clause defining *f_term*.

Question about definitions.

Answer about definitions.

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
     is_or                                  f
        (ar                                 cc)
     ∧
      ∃def
        (f_term,
         generalise_nth_argument_of,
         [generalise_nth, f_term])
```

definitional quantifier

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
                    m_on_rhs, nth_param_on_lhs)

                rhs : term_occ ∈ f_term.
               argument_of
               ram_on_rhs,
     generalise_nth,
     f_occ_on_rhs)
```

Checks if **generalise_nth_argument_of** returns **True** when applied to *generalise_nth* and *f_term* for some clause defining *f_term*.

Question about definitions.

Answer about definitions.

Program 2 for inductive problems

Program 3 for definitions

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
 apply (induct "xs" arbitrary:ys) by auto
```

Program 2 for inductive problems

Program 3 for definitions

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
 apply (induct "xs" arbitrary:ys) by auto
```

}

Program 2

Question about definitions.

Answer about definitions.

Program 2 for inductive problems

Program 3 for definitions

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
 apply (induct "xs" arbitrary:ys) by auto
```

} Program 3

} Program 2

Question about definitions.

Answer about definitions.

Program 2 for inductive problems

Program 3 for definitions

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
 apply (induct "xs" arbitrary:ys) by auto
```

} Program 3

} Program 2

May I generalise *ys*, which appears as the second argument of rev2?

Answer about definitions.

Program 2 for inductive problems

Program 3 for definitions

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
 apply (induct "xs" arbitrary:ys) by auto
```

Program 3

Program 2

May I generalise *ys*, which appears as the second argument of rev2?

Yes. You may do so because the second argument changes from the left-hand side to the right-hand side in the second clause defining rev2.

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
      is_or_below_nth_argument_of
        (arb_occ, generalise_nth, f_occ)
     ∧
      ∃_def
        (f_term,
         generalise_nth_argument_of,
         [generalise_nth, f_term])
```

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
       (nth_param_on_rhs, nth_param_on_lhs)
     ∧
      ∃ f_occ_on_rhs : term_occ ∈ f_term.
       is_nth_argument_of
         (nth_param_on_rhs,
          generalise_nth,
          f_occ_on_rhs)
```

May I generalise *ys*, which appears as the second argument of rev2?

Yes. You may do so because the second argument changes from the left-hand side to the right-hand side in the second clause defining rev2.

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

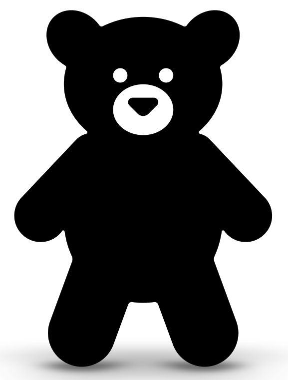
```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
     is_                  _of
      (                   f_occ)
    ∧
     ∃def
      (f_term,
       generalise_nth_argument_of,
       [generalise_nth, f_term])
```

for the second clause

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
   is_nth_argument_of
    (nth_param_on_lhs, generalise_nth,
     lhs_occ)
  ∧
   ∃ nth_param_on_rhs : term_occ.
    ¬ are_of_same_term
     (nth_param_on_rhs, nth_param_on_lhs)
   ∧
    ∃ f_occ_on_rhs : term_occ ∈ f_term.
     is_nth_argument_of
      (nth_param_on_rhs,
       generalise_nth,
       f_occ_on_rhs)
```

May I generalise *ys*, which appears as the second argument of rev2?



Yes. You may do so because the second argument changes from the left-hand side to the right-hand side in the second clause defining rev2.

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
      is_
          (
    ∧
     ∃def
        (f_term,
         generalise_nth_argument_of,
         [generalise_nth, f_term])
```

for the second clause

defining rev2

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
       (nth_param_on_rhs, nth_param_on_lhs)
     ∧
      ∃ f_occ_on_rhs : term_occ ∈ f_term.
       is_nth_argument_of
         (nth_param_on_rhs,
          generalise_nth,
          f_occ_on_rhs)
```

May I generalise *ys*, which appears as the second argument of rev2?

Yes. You may do so because the second argument changes from the left-hand side to the right-hand side in the second clause defining rev2.

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
      is_
          (
      ∧
       ∃def
          (f_term,
           generalise_nth_argument_of,
           [generalise_nth, f_term])
```

for the second clause

defining rev2

second argument

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
       (nth_param_on_rhs, nth_param_on_lhs)
     ∧
      ∃ f_occ_on_rhs : term_occ ∈ f_term.
       is_nth_argument_of
          (nth_param_on_rhs,
           generalise_nth,
           f_occ_on_rhs)
```

May I generalise *ys*, which appears as the second argument of rev2?

Yes. You may do so because the second argument changes from the left-hand side to the right-hand side in the second clause defining rev2.

Program 2 for inductive problems

Program 3 for definitions

Program 2 Syntactic analysis for generalisation in SeLFiE

```
∀ arb_term : term ∈ arbitrary_term.
 ∃ f_term : term.
  ∃ f_occ : term_occ ∈ f_term.
   ∃ arb_occ ∈ arb_term.
    ∃ generalise_nth : number.
      is_
          (
      ∧
       ∃def
          (f_term,
           generalise_nth_argument_of,
           [generalise_nth, f_term])
```

for the second clause

defining rev2

second argument

changes from the left-hand side to the right-hand side

Program 3 Definitional analysis for generalisation in SeLFiE

```
generalise_nth_argument_of :=
λ [generalise_nth, f_term].
 ∃ lhs_occ : term_occ.
  is_left_hand_side (lhs_occ)
 ∧
  ∃ nth_param_on_lhs : term_occ.
    is_nth_argument_of
     (nth_param_on_lhs, generalise_nth,
      lhs_occ)
   ∧
    ∃ nth_param_on_rhs : term_occ.
      ¬ are_of_same_term
       (nth_param_on_rhs, nth_param_on_lhs)
     ∧
      ∃ f_occ_on_rhs : term_occ ∈ f_term.
       is_nth_argument_of
          (nth_param_on_rhs,
           generalise_nth,
           f_occ_on_rhs)
```

May I generalise *ys*, which appears as the second argument of rev2?

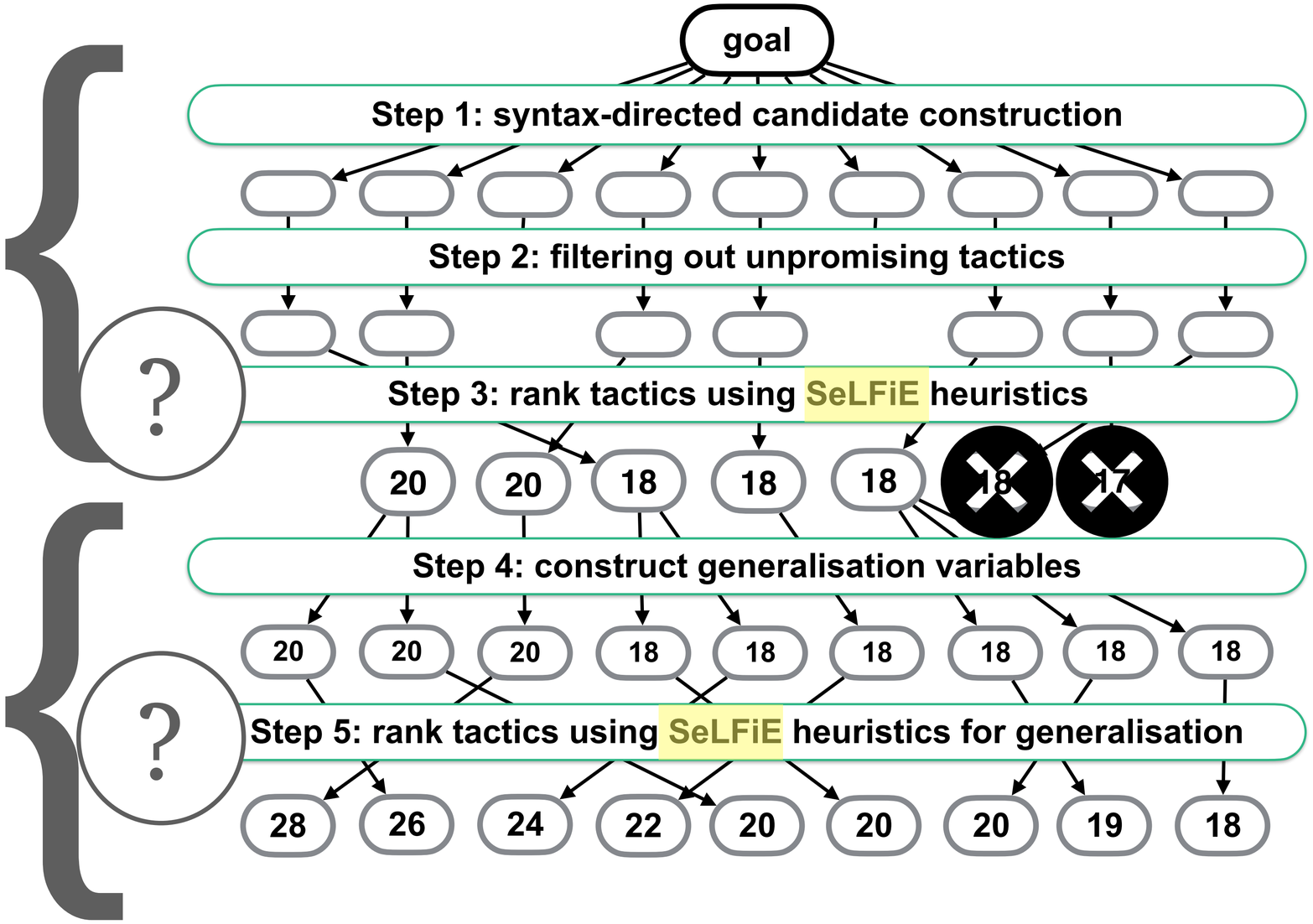
Yes. You may do so because the second argument changes from the left-hand side to the right-hand side in the second clause defining rev2.

Program 2 for inductive problems

Program 3 for definitions

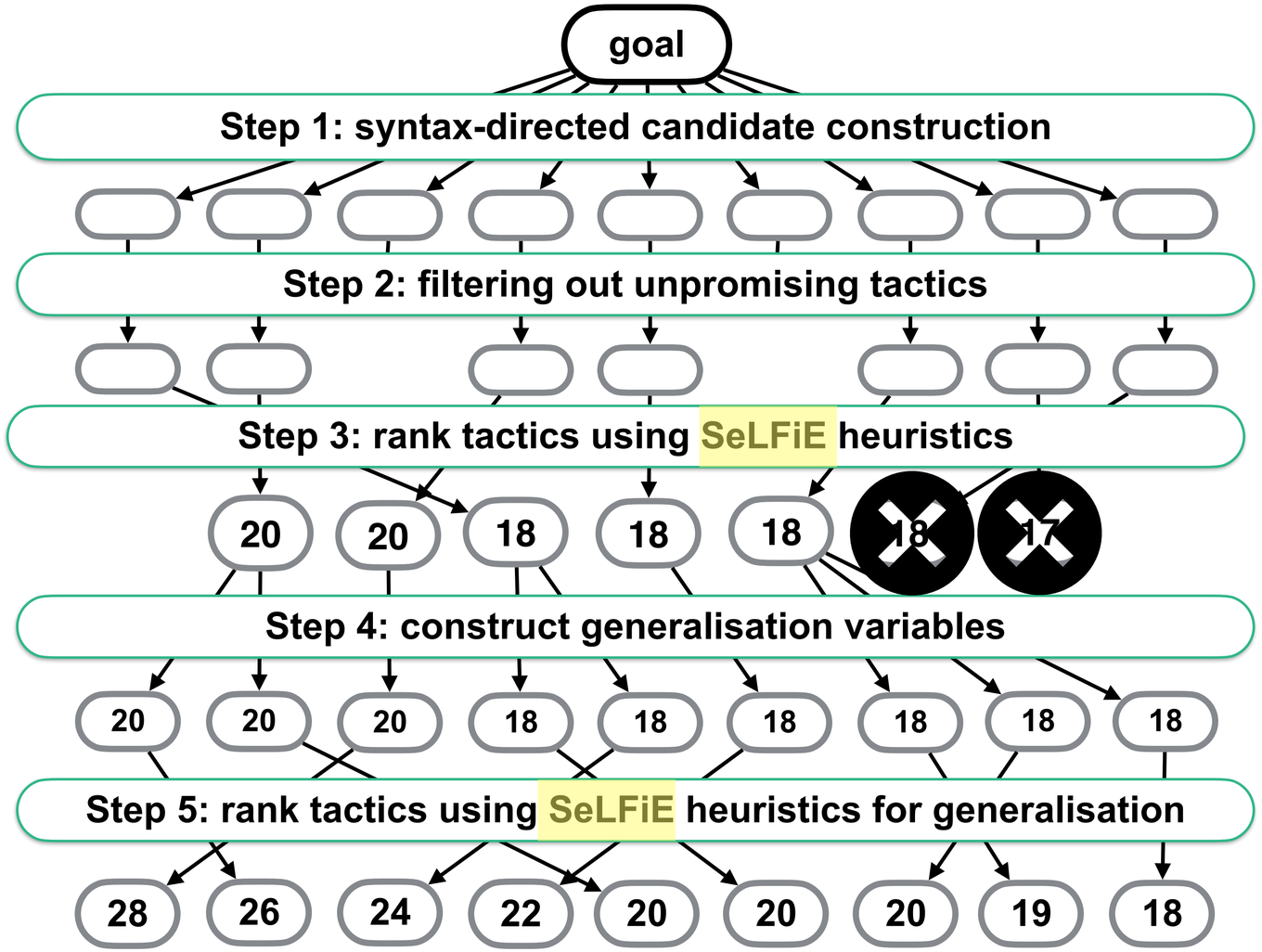
induction terms

variable generalisation



induction terms

variable generalisation



https://www.ijcai.org/proceedings/2021/273

Faster Smarter Proof by Induction in Isabelle/HOL

Yutaka Nagashima
Yale-NUS College, National University of Singapore
University of Innsbruck
Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague
yutaka@yale-nus.edu.sg

Abstract

We present `sem_ind`, a recommendation tool for proof by induction in Isabelle/HOL. Given an inductive problem, `sem_ind` produces candidate arguments for proof by induction, and selects promising ones using heuristics. Our evaluation based on 1,095 inductive problems from 22 source files shows that `sem_ind` improves the accuracy of recommendation from 20.1% to 38.2% for the most promising candidates within 5.0 seconds of timeout compared to its predecessor while decreasing the median value of execution time from 2.79 seconds to 1.06 seconds.

Program 1 Equivalence of two reverse functions

```
@ :: α list ⇒ α list ⇒ α list
  []       @ ys = ys
| (x # xs) @ ys = x # (xs @ ys)

rev1 :: α list ⇒ α list
  rev1 []       = []
| rev1 (x # xs) = rev1 xs @ [x]

rev2 :: α list ⇒ α list ⇒ α list
  rev2 []       ys = ys
| rev2 (x # xs) ys = rev2 xs (x # ys)
```

①

https://www.ijcai.org/proceedings/2021/273

Faster Smarter Proof by Induction in Isabelle/HOL

Yutaka Nagashima

Yale-NUS College, National University of Singapore
University of Innsbruck
Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague
yutaka@yale-nus.edu.sg

Abstract

We present `sem_ind`, a recommendation tool for proof by induction in Isabelle/HOL. Given an inductive problem, `sem_ind` produces candidate arguments for proof by induction, and selects promising ones using heuristics. Our evaluation based on 1,095 inductive problems from 22 source files shows that `sem_ind` improves the accuracy of recommendation from 20.1% to 38.2% for the most promising candidates within 5.0 seconds of timeout compared to its predecessor while decreasing the median value of execution time from 2.79 seconds to 1.06 seconds.

Program 1 Equivalence of two reverse functions

```
@ :: α list ⇒ α list ⇒ α list
  []       @ ys = ys
| (x # xs) @ ys = x # (xs @ ys)

rev1 :: α list ⇒ α list
  rev1 []       = []
| rev1 (x # xs) = rev1 xs @ [x]

rev2 :: α list ⇒ α list ⇒ α list
  rev2 []       ys = ys
| rev2 (x # xs) ys = rev2 xs (x # ys)
```

① ② https://www.ijcai.org/proceedings/2021/273

Faster Smarter Proof by Induction in Isabelle/HOL

Yutaka Nagashima

Yale-NUS College, National University of Singapore
University of Innsbruck
Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague
yutaka@yale-nus.edu.sg

Abstract

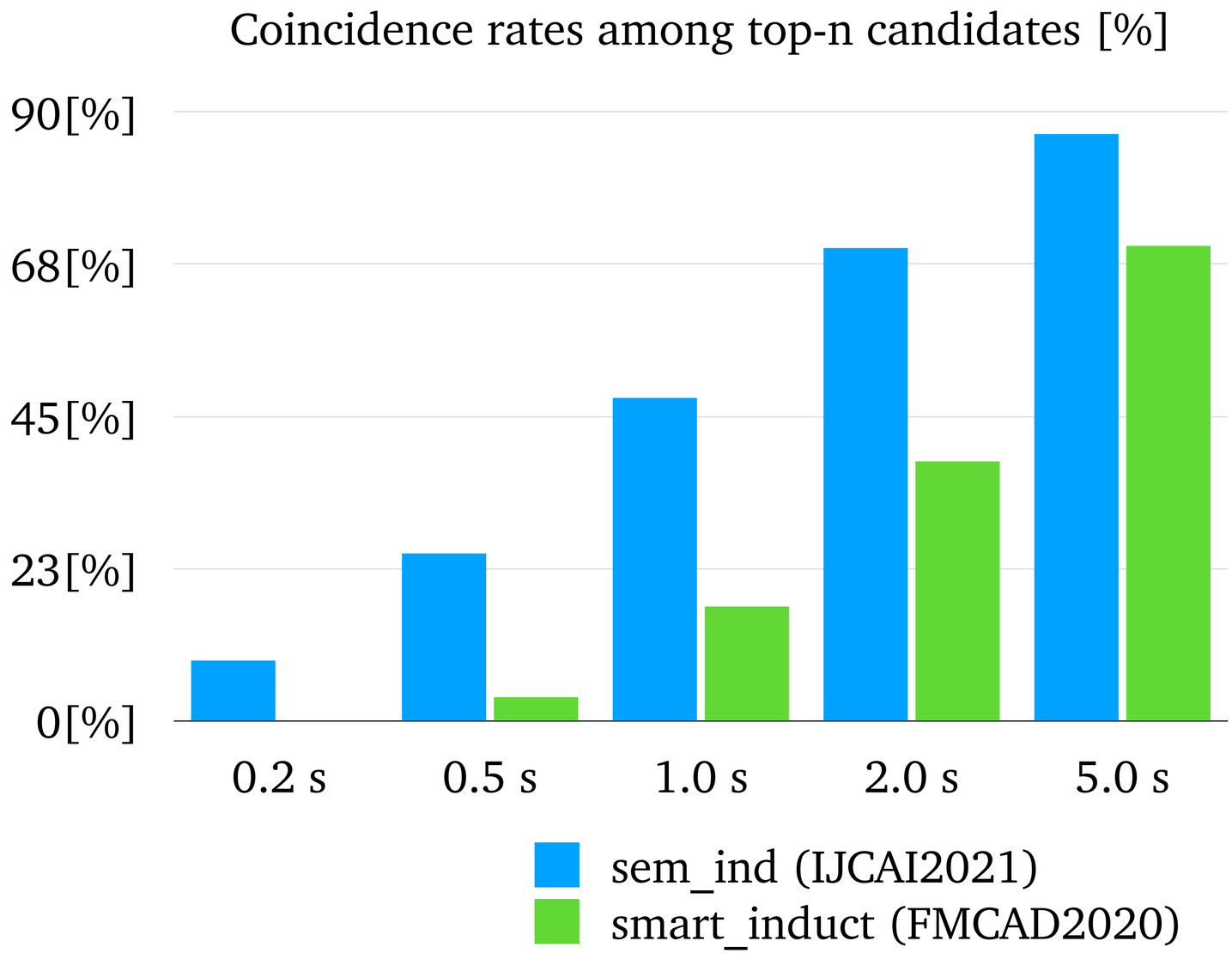
We present `sem_ind`, a recommendation tool for proof by induction in Isabelle/HOL. Given an inductive problem, `sem_ind` produces candidate arguments for proof by induction, and selects promising ones using heuristics. Our evaluation based on 1,095 inductive problems from 22 source files shows that `sem_ind` improves the accuracy of recommendation from 20.1% to 38.2% for the most promising candidates within 5.0 seconds of timeout compared to its predecessor while decreasing the median value of execution time from 2.79 seconds to 1.06 seconds.

Program 1 Equivalence of two reverse functions

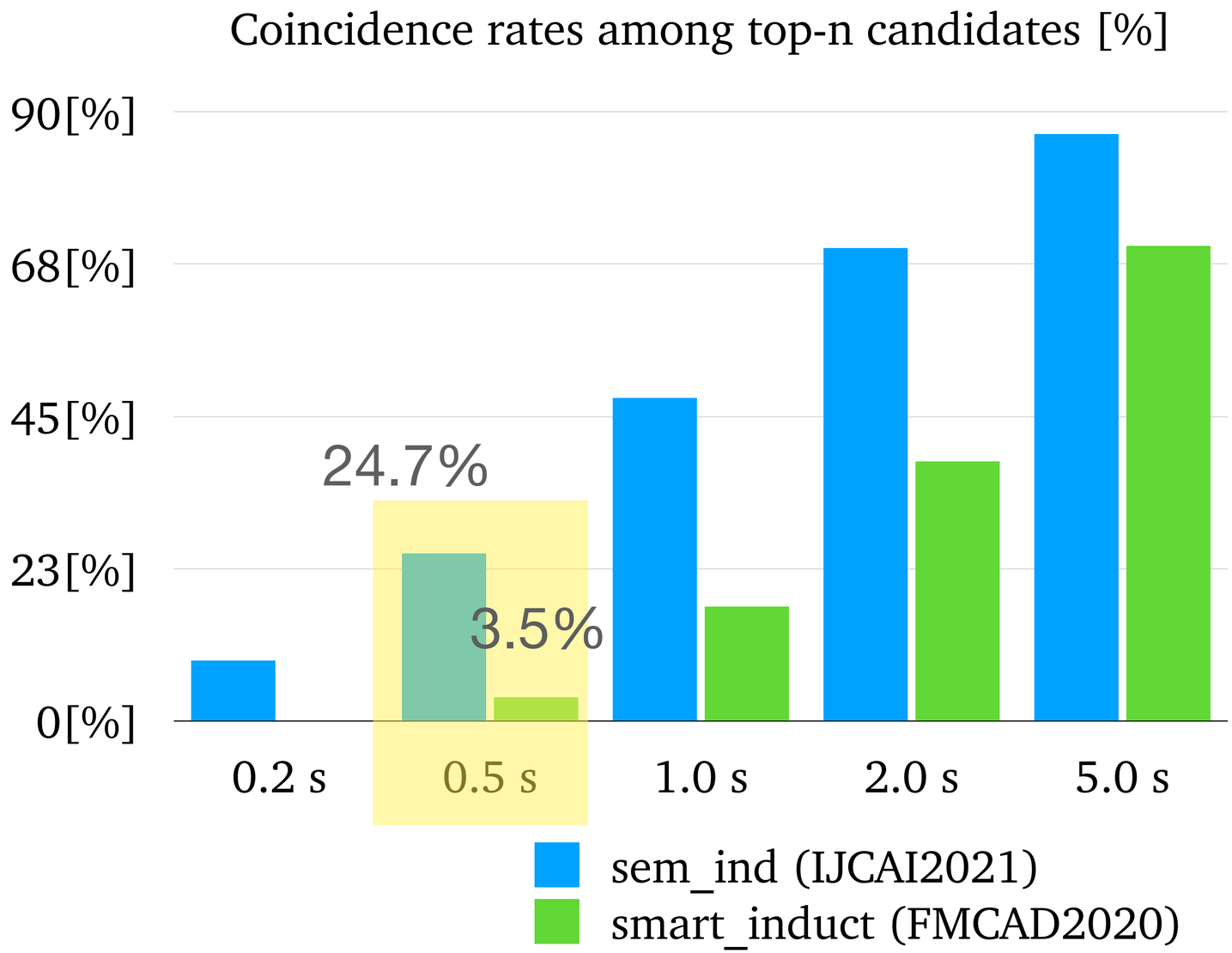
```
@ :: α list ⇒ α list ⇒ α list
  []       @ ys = ys
| (x # xs) @ ys = x # (xs @ ys)

rev1 :: α list ⇒ α list
  rev1 []       = []
| rev1 (x # xs) = rev1 xs @ [x]

rev2 :: α list ⇒ α list ⇒ α list
  rev2 []       ys = ys
| rev2 (x # xs) ys = rev2 xs (x # ys)
```

ⓛ Faster Proof by Induction in Isabelle/HOL

① Faster Proof by Induction in Isabelle/HOL

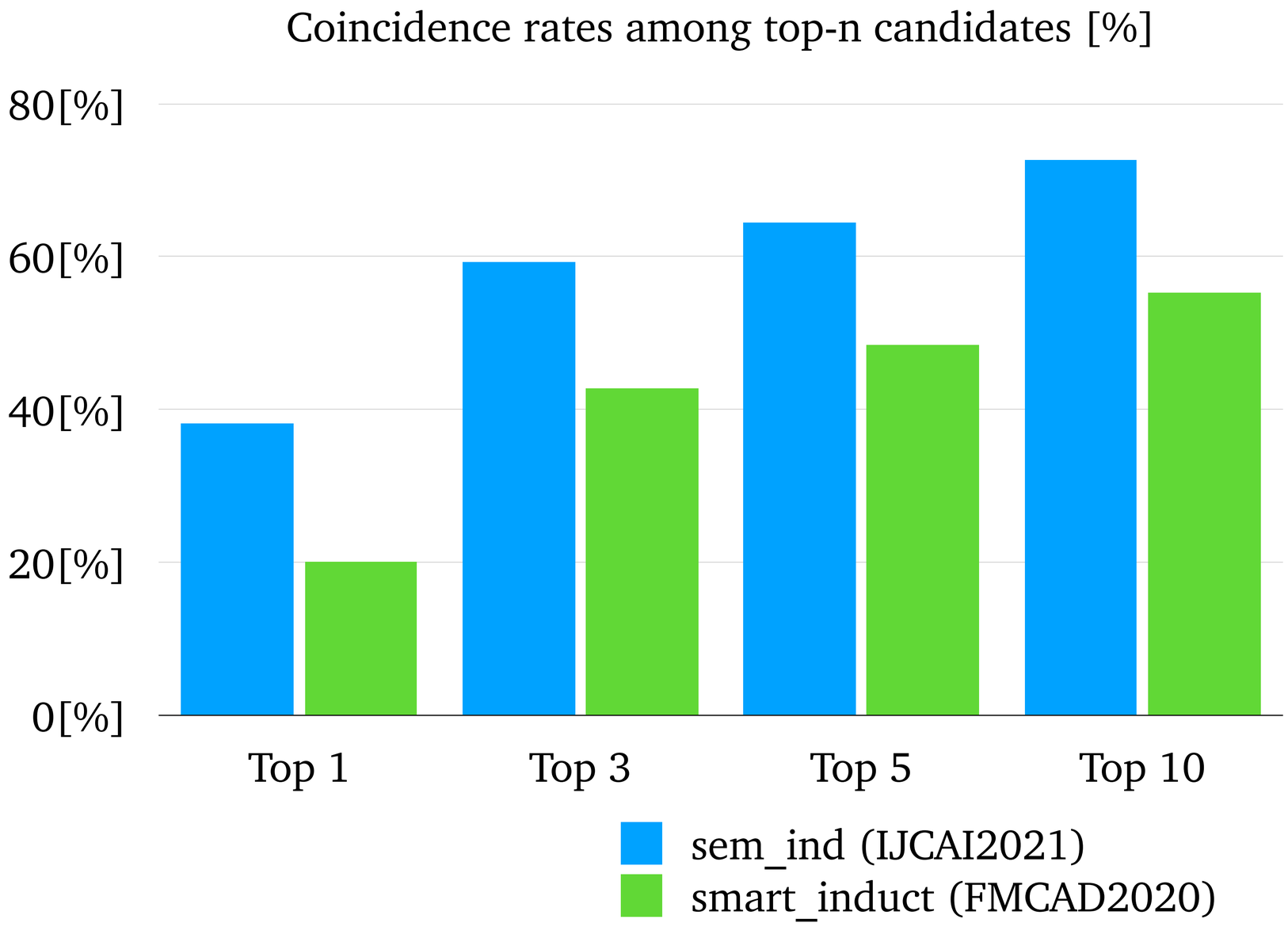


① Faster Proof by Induction in Isabelle/HOL

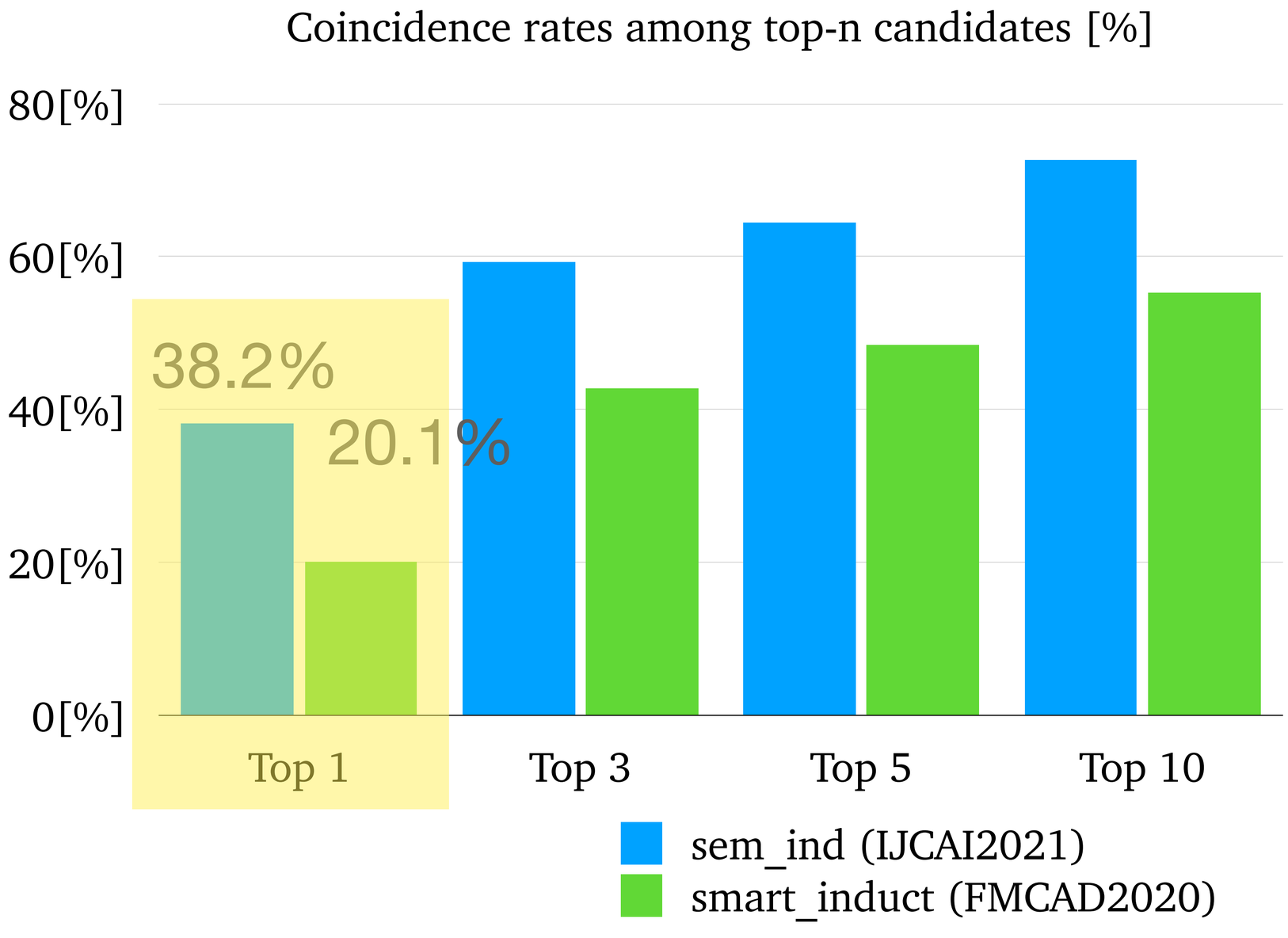


② Smarter Proof by Induction in Isabelle/HOL

② Smarter Proof by Induction in Isabelle/HOL



② Smarter Proof by Induction in Isabelle/HOL



```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
```

```
proof (prove)
goal (1 subgoal):
 1. rev2 xs ys = rev xs @ ys
```

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
  sem_ind
```

```
1st candidate is apply (induct "xs" arbitrary:ys)
  (* The score is 103.0 out of 104.0. *)
2nd candidate is apply (induct "xs")
  (* The score is 102.0 out of 104.0. *)
3th candidate is apply (induct "xs" "ys" rule:IJCAI.rev2.induct)
  (* The score is 100.0 out of 104.0. *)
4th candidate is apply (induct "xs" "ys" rule:List.list_induct2')
  (* The score is 100.0 out of 104.0. *)
```

```
primrec rev1::"'a list ⇒ 'a list" where
  "rev1  []       = []"
| "rev1 (x # xs) = rev1 xs @ [x]"

fun rev2:: "'a list ⇒ 'a list ⇒ 'a list" where
  "rev2  []       ys = ys"
| "rev2 (x # xs) ys = rev2 xs (x # ys)"

lemma "rev2 xs ys = rev xs @ ys"
  sem_ind
  apply (induct "xs" arbitrary:ys)
```

```
proof (prove)
goal (2 subgoals):
 1. ⋀ys. rev2 [] ys = rev [] @ ys
 2. ⋀a xs ys.
       (⋀ys. rev2 xs ys = rev xs @ ys) ⟹
       rev2 (a # xs) ys = rev (a # xs) @ ys
```

old

Program 4 Automatic inductive prover without SeLFiE

```
Auto_Solve = Thens[Auto, Solved]
PSL_WO_SeLFiE =
Ors[Auto_Solve,
    PThenOne[Dynamic (Induct), Auto_Solve]
    PThenOne[Dynamic(Induct), Thens[Auto, RepeatN(Hammer), Solved]]]
```

new

Program 5 Automatic inductive prover with SeLFiE

```
PSL_W_SeLFiE =
Ors[Auto_Solve,
    PThenOne[Semantic_Induct, Auto_Solve]
    PThenOne[Semantic_Induct, Thens[Auto, RepeatN(Hammer), Solved]]]
```

old

Program 4 Automatic inductive prover without SeLFiE

```
Auto_Solve = Thens[Auto, Solved]
PSL_WO_SeLFiE =
Ors[Auto_Solve,
    PThenOne[Dynamic (Induct), Auto_Solve]
    PThenOne[Dynamic(Induct), Thens[Auto, RepeatN(Hammer), Solved]]]
```

new

Program 5 Automatic inductive prover with SeLFiE

```
PSL_W_SeLFiE =
Ors[Auto_Solve,
    PThenOne[Semantic_Induct, Auto_Solve]
    PThenOne[Semantic_Induct, Thens[Auto, RepeatN(Hammer), Solved]]]
```

timeouts	Program 5	Program 4
0.3[s]	11.0%	1.2%
1.0[s]	25.6%	1.7%
3.0[s]	28.2%	21.9%
10.0[s]	34.9%	28.0%
30.0[s]	45.8%	38.3%

(a) Success rates

speedup [times]	occurrence
$x < 1.0$	3 (2.4%)
$1.0 \leq x < 5.0$	64 (50.8%)
$5.0 \leq x < 10.0$	44 (34.9%)
$10.0 \leq x < 15.0$	9 (7.1%)
$15.0 \leq x <$	6 (4.8%)

(b) Speedup of execution time

old

Program 4 Automatic inductive prover without SeLFiE

```
Auto_Solve = Thens[Auto, Solved]
PSL_WO_SeLFiE =
Ors[Auto_Solve,
    PThenOne[Dynamic (Induct), Auto_Solve]
    PThenOne[Dynamic(Induct), Thens[Auto, RepeatN(Hammer), Solved]]]
```

new

Program 5 Automatic inductive prover with SeLFiE

```
PSL_W_SeLFiE =
Ors[Auto_Solve,
    PThenOne[Semantic_Induct, Auto_Solve]
    PThenOne[Semantic_Induct, Thens[Auto, RepeatN(Hammer), Solved]]]
```

timeouts	Program 5	Program 4
0.3[s]	11.0%	1.2%
1.0[s]	25.6%	1.7%
3.0[s]	28.2%	21.9%
10.0[s]	34.9%	28.0%
30.0[s]	45.8%	38.3%

(a) Success rates

speedup [times]	occurrence
$x < 1.0$	3 (2.4%)
$1.0 \leq x < 5.0$	64 (50.8%)
$5.0 \leq x < 10.0$	44 (34.9%)
$10.0 \leq x < 15.0$	9 (7.1%)
$15.0 \leq x <$	6 (4.8%)

(b) Speedup of execution time

Definitional Quantifiers Realise Semantic Reasoning for Proof by Induction

Yutaka Nagashima

https://github.com/data61/PSL

Definitional Quantifiers Realise Semantic Reasoning for Proof by Induction

Yutaka Nagashima

https://github.com/data61/PSL