examples

April 29, 2021

1 Learning arguments in case models

Here we replicate the examples from *Bart Verheij, 2017. Proof with and without probabilities.*

Moreover, for each case model presented in the paper, we also add a section where we learn the arguments.

1.1 Algorithms

We use two algorithms for learning the arguments:

1. Enumerate and test and filter ('naive search').

   This algorithm creates all possible arguments, tests whether they are conclusive, presumptively valid, or coherent. Then it does some postprocessing:

   - Arguments are removed if the premise is more specific than the premise of another argument with the same conclusion;
   - except if they form an exception to some other argument, that is, if there is some argument with a less specific premise and the opposite conclusion.

2. Structured search ('pruned search').

   This follows the idea behind the Apriori algorithm. A frequent itemset has to consist of frequent subsets. Similarly, the premise of a coherent argument has to consist of subsets that form a coherent argument together with the same conclusion.

   The algorithm uses this in the following way in order to search for the premises for a given conclusion:

   - In the first two iterations, it considers the tautological premise, and the premises consisting of single facts.
   - For each later iteration, it will consider all premises from the previous iteration that turned out to form coherent arguments. It will merge all pairs of these premises where only a single fact is different. This results in larger premises that are potentially coherent and will then be checked whether they really are coherent, and presumptively valid, and conclusive.

Both algorithms generate arguments for a given conclusion of a single fact. The generated arguments are then merged together if they have the same premises.

The first algorithm turns out to work well on very small examples only; for the core example of the paper, only the second algorithm generates the arguments efficiently. For our examples, both

algorithms always create the same arguments.

[1]:
```
from definitions import *
```

1.2 Presumption of innocence

(p. 137-138)

"Let `inn` denote that a suspect is innocent, and `gui` that he is guilty. Then the argument `(inn, ¬gui)` is properly presumptive, since `inn` $\not\models$ `gui`. The argument `(inn` $\wedge$ `¬gui, ¬gui)` is non-presumptive, since `inn` $\wedge$ `¬gui` $\models$ gui."

[2]:
```
assert Argument.fromStr('inn -> gui').is_properly_presumptive
assert not Argument.fromStr('inn, ¬gui -> ¬gui').is_properly_presumptive
```

"Presumptive validity and defeasibility are illustrated using a case model. Consider the case model with two cases `inn` $\wedge$ `¬gui` and `¬inn` $\wedge$ `gui` $\wedge$ `evi` with the first case preferred to the second (…). Here evi denotes evidence for the suspect's guilt."

[3]:
```
case_model = CaseModel.fromStr([
    (1, 'inn, ¬gui'),
    (0, '¬inn, gui, evi')
])
```

"Then the properly presumptive argument `(inn, ¬gui)` is presumptively valid with respect to this case model since the conclusion `¬gui` follows in the case `inn` $\wedge$ `¬gui` that is a preferred case of the premise `inn`. The argument is conclusive since there are no other cases implying `inn`."

[4]:
```
argument = Argument.fromStr('inn -> ¬gui')
assert argument.is_presumptively_valid_in(case_model)
assert argument.is_conclusive_in(case_model)
```

"The argument `([], inn)`—in fact a presumption now that its premises are tautologous—is presumptively valid since `inn` follows in the preferred case `inn` $\wedge$ `¬gui`. This shows that the example represents what is called the presumption of innocence, when there is no evidence. This argument is properly defeasible since in the other case of the argument's premises the conclusion does not follow."

[5]:
```
presumption_of_innocence = Argument.fromStr('-> inn')
assert presumption_of_innocence.is_a_presumption
assert presumption_of_innocence.is_presumptively_valid_in(case_model)
assert presumption_of_innocence.is_properly_defeasible_in(case_model)
```

"In fact, the argument `(evi, inn)` is not coherent since there is no case in which both `evi` and `inn` follow."

[6]:
```
assert not Argument.fromStr('evi -> inn').is_coherent_in(case_model)
```

"The argument `(evi, gui)` is presumptively valid, even conclusive."

[7]:
```
argument = Argument.fromStr('evi -> gui')
assert argument.is_presumptively_valid_in(case_model)
assert argument.is_conclusive_in(case_model)
```

”Continuing the example of the case model (…), we find the following. The circumstances `evi` defeat the presumptively valid argument (⊤, `inn`) since `(evi, inn)` is not presumptively valid. In fact, these circumstances are excluding since `(evi, inn)` is not coherent. The circumstances are also rebutting since the argument for the opposite conclusion `(evi, inn)` is presumptively valid.

[8]:
```
circumstances = [Fact.fromStr('evi')]
assert presumption_of_innocence.is_defeated_by_in(circumstances, case_model)
assert presumption_of_innocence.is_excluded_by_in(circumstances, case_model)
assert presumption_of_innocence.is_rebutted_by_in(circumstances, case_model)
```

1.2.1 Learning arguments

[9]:
```
Theory.learn_with_naive_search(case_model).print()
print('---')
Theory.learn_with_pruned_search(case_model).print()
```

```
evi   gui <- ¬inn
evi   ¬inn <- gui
gui   ¬inn <- evi
inn <- ¬gui
¬gui <- inn
inn   ¬gui <~
---
evi   gui <- ¬inn
evi   ¬inn <- gui
gui   ¬inn <- evi
inn <- ¬gui
¬gui <- inn
inn   ¬gui <~
```

1.3 Lying witness

“In the cases, there is a witness testimony (`wit`) that the suspect was at the crime scene (`sus`). In Case 1, the witness was not misguided (`¬mis`), in Case 2 he was. In Case 1, the suspect was indeed at the crime scene; in Case 2, the witness was misguided and it is unspecified whether the suspect was at the crime scene or not. In the case model, Case 1 is preferred to Case 2 (…), representing that witnesses are usually not misguided.” (p. 139)

[10]:
```
case_model = CaseModel.fromStr([
    (1, 'sus, ¬mis, wit'),
    (0, 'mis, wit')
])
```

“Since Case 1 is a preferred case of `wit`, the argument `(wit, sus)` is presumptively valid: the

witness testimony provides a presumptively valid argument for the suspect having been at the crime scene. The argument's conclusion can be strengthened to include that the witness was not misguided. Formally, this is expressed by saying that (wit, sus ∧ ¬mis) is a presumptively valid argument." (p. 139)

[11]:
```
assert Argument.fromStr('wit -> sus').is_presumptively_valid_in(case_model)
assert Argument.fromStr('wit -> sus, ¬mis').
 ↪is_presumptively_valid_in(case_model)
```

"When the witness was misguided after all (mis), there are circumstances defeating the argument (wit, sus). This can be seen by considering that Case 2 is the only case in which wit ∧ mis follows, hence is preferred. Since sus does not follow in Case 2, the argument (wit ∧ mis, sus) is not presumptively valid. The misguidedness is not rebutting, hence undercutting since (wit ∧ mis, ¬sus) is not presumptively valid. The misguidedness is excluding since the argument (wit ∧ mis, sus) is not even coherent." (p. 139)

[12]:
```
argument = Argument.fromStr('wit -> sus')
circumstances = [Fact.fromStr('mis')]
assert argument.is_defeated_by_in(circumstances, case_model)
assert argument.is_undercut_by_in(circumstances, case_model)
assert argument.is_excluded_by_in(circumstances, case_model)
```

1.3.1 Learning arguments

[13]:
```
Theory.learn_with_naive_search(case_model).print()
print('---')
Theory.learn_with_pruned_search(case_model).print()
```

```
sus <- ¬mis
wit <-
¬mis <- sus
sus   ¬mis <~
---
sus <- ¬mis
wit <-
¬mis <- sus
sus   ¬mis <~
```

1.4 Chaining arguments

"Arguments can typically be chained, namely when the conclusion of one is a premise of another. For instance when there is evidence (evi) that a suspect is guilty of a crime (gui), the suspect's guilt can be the basis of punishing the suspect (pun). For both steps there are typical defeating circumstances. The step from the evidence to guilt is blocked when there is a solid alibi (ali), and the step from guilt to punishing is blocked when there are grounds of justification (jus), such as force majeure. (…) In the case model, Case 1 is preferred to Case 2 and Case 3, modeling that the evidence typically leads to guilt and punishing, unless there are grounds for justification (Case 2) or there is an alibi (Case 3). Cases 2 and 3 are preferentially equivalent." (p. 139-140)

[14]:
```
case_model = CaseModel.fromStr([
    (1, 'pun, gui, evi'),
    (0, '¬pun, gui, evi, jus'),
    (0, '¬gui, evi, ali')
])
```

”In this case model, the following arguments are presumptively valid:

- Argument 1 (presumptively valid): (evi, gui)
- Argument 2 (presumptively valid): (gui, pun)
- Argument 3 (presumptively valid): (evi, gui ∧ pun)”

(p. 140)

[15]:
```
argument1 = Argument.fromStr('evi -> gui')
argument2 = Argument.fromStr('gui -> pun')
argument3 = Argument.fromStr('evi -> gui, pun')

assert argument1.is_presumptively_valid_in(case_model)
assert argument2.is_presumptively_valid_in(case_model)
assert argument3.is_presumptively_valid_in(case_model)
```

”The following arguments are not presumptively valid in this case model:

- Argument 4 (not presumptively valid): (evi ∧ ali, gui)
- Argument 5 (not presumptively valid): (gui ∧ jus, pun)”

(p. 141)

[16]:
```
argument4 = Argument.fromStr('evi, ali -> gui')
argument5 = Argument.fromStr('gui, jus -> pun')
assert not argument4.is_presumptively_valid_in(case_model)
assert not argument5.is_presumptively_valid_in(case_model)
```

“This shows that Arguments 1 and 2 are defeated by circumstances ali and jus, respectively:” (p. 141)

[17]:
```
assert argument1.is_defeated_by_in([Fact.fromStr('ali')], case_model)
assert argument2.is_defeated_by_in([Fact.fromStr('jus')], case_model)
```

”As expected, chaining the arguments fails under both of these defeating circumstances, as shown by the fact that these two arguments are not presumptively valid:

- Argument 6 (not presumptively valid): (evi ∧ ali, gui ∧ pun)
- Argument 7 (not presumptively valid): (gui ∧ jus, gui ∧ pun)”

(p. 141)

[18]:
```
argument6 = Argument.fromStr('evi, ali -> gui, pun')
argument7 = Argument.fromStr('gui, jus -> gui, pun')
assert not argument4.is_presumptively_valid_in(case_model)
```

```
assert not argument5.is_presumptively_valid_in(case_model)
```

"But the first step of the chain—the step to guilt—can be made when there are grounds for justification. Formally, this can be seen by the presumptive validity of this argument:

- Argument 8 (presumptively valid): `(evi ∧ jus, gui)`"

(p. 141)

[19]:
```
argument8 = Argument.fromStr('evi, jus -> gui')
assert argument8.is_presumptively_valid_in(case_model)
```

1.4.1 Learning arguments

[20]:
```
Theory.learn_with_naive_search(case_model).print()
print('---')
Theory.learn_with_pruned_search(case_model).print()
```

```
ali <- ¬gui
evi <-
gui <- pun
gui   jus <- ¬pun
gui   ¬pun <- jus
¬gui <- ali
gui   pun <~
---
ali <- ¬gui
evi <-
gui <- pun
gui   jus <- ¬pun
gui   ¬pun <- jus
¬gui <- ali
gui   pun <~
```

1.5 DNA evidence

"We discuss an example, adapting our earlier treatment of the presumption of innocence. Consider a crime case where two pieces of evidence are found, one after another. In combination, they are considered to prove the suspect's guilt beyond a reasonable doubt. For instance, one piece of evidence is a witness who claims to have seen the suspect committing the crime (`evi`), and a second piece of evidence is DNA evidence matching the suspect's profile (`evi'`). The issue is whether the suspect is innocent (`inn`) or guilty (`gui`). Consider now a case model with four cases:

- Case 1: `inn ∧ ¬gui ∧ ¬evi`
- Case 2: `¬inn ∧ gui ∧ evi ∧ ¬evi'`
- Case 3: `inn ∧ ¬gui ∧ evi ∧ ¬evi'`
- Case 4: `¬inn ∧ gui ∧ evi ∧ evi'`

Case 1 expresses the situation when no evidence has been found, hence the suspect is considered innocent and not guilty. In order to express that by default there is no evidence concerning

someone's guilt, this case has highest preference. Cases 2 and 3 express the situation that the first piece of evidence is found. Case 2 expresses guilt, Case 3 innocence, still considered a possibility given only the first piece of evidence. In order to express that `evi` makes the suspect's guilt more plausible than his innocence, Case 2 has higher preference than Case 3. Case 4 represents the situation that both pieces of evidence are available, proving guilt. It has lowest preference. Summarizing the preference relation we have:

Case 1 > Case 2 > Case 3 > Case 4"

(p. 145-146)

[21]:
```
case_model = CaseModel.fromStr([
    (3, 'inn, ¬gui, ¬evi'),
    (2, "¬inn, gui, evi, ¬evi'"),
    (1, "inn, ¬gui, evi, ¬evi'"),
    (0, "¬inn, gui, evi, evi'"),
])
```

table 1 and 2

[22]:
```
argument1 = Argument.fromStr('-> inn')
argument2 = Argument.fromStr('-> gui')
argument3 = Argument.fromStr('evi -> inn')
argument4 = Argument.fromStr('evi -> gui')
argument5 = Argument.fromStr("evi, evi' -> inn")
argument6 = Argument.fromStr("evi, evi' -> gui")

assert argument1.is_coherent_in(case_model)
assert not argument1.is_conclusive_in(case_model)
assert argument1.is_presumptively_valid_in(case_model)

assert argument2.is_coherent_in(case_model)
assert not argument2.is_conclusive_in(case_model)
assert not argument2.is_presumptively_valid_in(case_model)

assert argument3.is_coherent_in(case_model)
assert not argument3.is_conclusive_in(case_model)
assert not argument3.is_presumptively_valid_in(case_model)

assert argument4.is_coherent_in(case_model)
assert not argument4.is_conclusive_in(case_model)
assert argument4.is_presumptively_valid_in(case_model)

assert not argument5.is_coherent_in(case_model)
assert not argument5.is_conclusive_in(case_model)
assert not argument5.is_presumptively_valid_in(case_model)

assert argument6.is_coherent_in(case_model)
assert argument6.is_conclusive_in(case_model)
```

```
assert argument6.is_presumptively_valid_in(case_model)
```

1.5.1 Learning arguments

[23]:
```
Theory.learn_with_naive_search(case_model).print()
print('---')
Theory.learn_with_pruned_search(case_model).print()
```

```
evi <- ¬evi'
evi   gui <- ¬inn
evi   gui   ¬inn <- evi'
evi   ¬inn <- gui
inn <- ¬gui
inn   ¬gui <- ¬evi
¬evi' <- evi   inn
¬evi' <- evi   ¬gui
¬gui <- inn
gui   ¬evi'   ¬inn <~ evi
gui   ¬inn <~ evi   ¬evi'
gui   ¬inn <~ ¬evi'
inn   ¬evi   ¬gui <~
¬evi' <~ gui
¬evi' <~ ¬inn
---
evi <- ¬evi'
evi   gui <- ¬inn
evi   gui   ¬inn <- evi'
evi   ¬inn <- gui
inn <- ¬evi'   ¬gui
inn <- ¬gui
inn   ¬evi' <- evi   ¬gui
inn   ¬gui <- ¬evi
¬evi'   ¬gui <- evi   inn
¬gui <- inn
¬gui <- inn   ¬evi'
gui   ¬evi'   ¬inn <~ evi
gui   ¬inn <~ ¬evi'
inn   ¬evi   ¬gui <~
¬evi' <~ gui
¬evi' <~ ¬inn
```

1.6 Alfred Hitchcock's "To catch a thief"

"During the investigation, gradually a case model has been developed representing the arguments discussed in the example. (…) First the properties of the four main hypotheses are accumulated (…):" (p. 149)

[24]:
```
hypothesis1 = Case.fromStr(None, 'rob')
hypothesis2 = Case.fromStr(None, '¬rob, fou')
hypothesis3 = Case.fromStr(None, '¬rob, ¬fou, dau, jwl')
hypothesis4 = Case.fromStr(None, '¬rob, ¬fou, ¬dau, ¬jwl')
```

"Then these are conjoined with the maximally specific accumulated evidence that provide a coherent argument for them:" (p. 149)

[25]:
```
evidence1 = Case.fromStr(None, 'res, esc')
evidence2 = Case.fromStr(None, 'res, esc, fgt')
evidence3 = Case.fromStr(None, 'res, esc, fgt, pro, cau, con, fin')
evidence4 = Case.fromStr(None, 'res, esc, fgt, pro, cau, con')
```

"Cases 5–7 complete the case model. Case 5 is the hypothetical case that Robie is not the thief, that there is resemblance, and the Robie does not escape. In Case 6, Robie and Foussard are not the thieves, and there is no fight. In Case 7, Robie, Foussard and his daughter are not the thieves, and she is not caught in the act. Note that the cases are consistent and mutually exclusive."

[26]:
```
case1 = Case(4, hypothesis1.facts + evidence1.facts)
case2 = Case(3, hypothesis2.facts + evidence2.facts)
case3 = Case(1, hypothesis3.facts + evidence3.facts)
case4 = Case(0, hypothesis4.facts + evidence4.facts)
case5 = Case.fromStr(3, '¬rob, res, ¬esc')
case6 = Case.fromStr(2, '¬rob, ¬fou, res, esc, ¬fgt')
case7 = Case.fromStr(0, '¬rob, ¬fou, ¬dau, res, esc, fgt, pro, ¬cau')
case_model = CaseModel(
    [case1, case2, case3, case4, case5, case6, case7])
```

"(…) the argument from the evidential premises `res` ∧ `esc` to the hypothesis `rob` is presumptively valid in this case model since Case 1 is the only case implying the case made by the argument. It is not conclusive since also the argument from these same premises to `¬rob` is coherent." (p. 150)

[27]:
```
argument1 = Argument.fromStr('res, esc -> rob')
assert argument1.is_presumptively_valid_in(case_model)
assert not argument1.is_conclusive_in(case_model)
```

"The latter argument is not presumptively valid since all `¬rob`-cases implying the premises (Cases 2–7) have lower preference than Case 1." (p. 150)

[28]:
```
argument2 = Argument.fromStr('res, esc -> ¬rob')
assert not argument2.is_presumptively_valid_in(case_model)
```

"The argument from `res` ∧ `esc` ∧ `fgt` to `rob` is incoherent as there is no case in which the premises and the conclusion follow." (p. 150)

[29]:
```
argument3 = Argument.fromStr('res, esc, fgt -> rob')
assert not argument3.is_coherent_in(case_model)
```

"Also arguments that do not start from evidential premises can be evaluated. For instance, the

argument from the premise (not itself evidence) dau to jwl is conclusive since in the only case implying the premises (Case 3) the conclusion follows." (p. 150)

[30]:
```
argument4 = Argument.fromStr('dau -> jwl')
assert argument4.is_conclusive_in(case_model)
```

"Finally we find the conclusive argument from premises res ∧ esc ∧ fgt ∧ pro ∧ cau ∧ con ∧ jwl to conclusion ¬rob ∧ ¬fou ∧ dau ∧ jwl (only Case 3 implies the premises), hence also to dau." (p. 150)

[31]:
```
argument5 = Argument.fromStr('res, esc, fgt, pro, cau, con, jwl -> ¬rob, ¬fou,␣
↪dau, jwl')
assert argument5.is_conclusive_in(case_model)
```

1.6.1 Learning arguments

We cannot learn the arguments naively, because the case model is too big.

[32]:
```
# Theory.learn_with_naive_search(case_model).print()
print('---')
Theory.learn_with_pruned_search(case_model).print()
```

```
---
cau  con  dau  esc  fgt  fin  pro  ¬fou  ¬rob <- jwl
cau  con  dau  esc  fgt  jwl  pro  ¬fou  ¬rob <- fin
cau  con  esc  fgt  fin  jwl  pro  ¬fou  ¬rob <- dau
cau  con  esc  fgt  pro  ¬dau  ¬fou  ¬rob <- ¬jwl
cau  esc  fgt  pro  ¬fou  ¬rob <- con
con  esc  fgt  pro  ¬fou  ¬rob <- cau
esc <- rob
esc  fgt  pro  ¬dau  ¬fou  ¬rob <- ¬cau
esc  fgt  pro  ¬fou  ¬rob <- ¬dau
esc  fgt  ¬fou  ¬rob <- pro
esc  fgt  ¬rob <- fou
esc  ¬fou  ¬rob <- ¬fgt
esc  ¬rob <- fgt
esc  ¬rob <- ¬fou
fgt <- cau  esc  ¬fou  ¬rob
fgt <- cau  ¬fou
fgt <- con  esc  ¬fou  ¬rob
fgt <- con  ¬fou
fgt <- dau  esc  ¬fou  ¬rob
fgt <- dau  ¬fou
fgt <- esc  fin  ¬fou  ¬rob
fgt <- esc  jwl  ¬fou  ¬rob
fgt <- esc  pro  ¬fou  ¬rob
fgt <- esc  ¬cau  ¬fou  ¬rob
fgt <- esc  ¬dau  ¬fou  ¬rob
```

```
fgt <- esc  ¬fou  ¬jwl  ¬rob
fgt <- fin  ¬fou
fgt <- jwl  ¬fou
fgt <- pro  ¬fou
fgt <- ¬cau  ¬fou
fgt <- ¬dau  ¬fou
fgt <- ¬fou  ¬jwl
pro <- fgt  ¬fou
res <-
¬dau <- cau  ¬jwl
¬dau <- con  ¬jwl
¬dau <- fgt  ¬cau  ¬fou
¬dau <- fgt  ¬fou  ¬jwl
¬dau <- pro  ¬cau
¬dau <- pro  ¬jwl
¬fou <- cau  esc  ¬rob
¬fou <- cau  fgt
¬fou <- con  esc  ¬rob
¬fou <- con  fgt
¬fou <- dau  esc  ¬rob
¬fou <- dau  fgt
¬fou <- esc  fin  ¬rob
¬fou <- esc  jwl  ¬rob
¬fou <- esc  pro  ¬rob
¬fou <- esc  ¬cau  ¬rob
¬fou <- esc  ¬dau  ¬rob
¬fou <- esc  ¬fgt  ¬rob
¬fou <- esc  ¬jwl  ¬rob
¬fou <- fgt  fin
¬fou <- fgt  jwl
¬fou <- fgt  pro
¬fou <- fgt  ¬cau
¬fou <- fgt  ¬dau
¬fou <- fgt  ¬jwl
¬jwl <- cau  fgt  ¬dau  ¬fou
¬jwl <- cau  pro  ¬dau
¬jwl <- cau  ¬dau
¬jwl <- con  fgt  ¬dau  ¬fou
¬jwl <- con  pro  ¬dau
¬jwl <- con  ¬dau
¬rob <- ¬esc
cau  con  dau  fin  jwl <~ fgt  ¬fou
cau  con  dau  fin  jwl <~ pro
dau  fin  jwl <~ cau
dau  fin  jwl <~ con
esc  rob <~
fgt  fou <~ esc  ¬rob
fou <~ fgt
```

```
¬fgt <~ esc  ¬fou  ¬rob
¬fgt <~ ¬fou
```

1.7 Dutch law of unlawful acts

This example is from *Bart Verheij, 2018. Arguments for good artificial intelligence. Inaugural lecture, Groningen.*

[33]:
```
case_model = CaseModel.fromStr([
    (2, '¬dmg'),
    (2, '¬dut, dmg, ¬unl, ¬vrt, ¬vst, ¬vun'),
    (2, '¬dut, dmg, unl, ¬imp, ¬ift, ¬ila, ¬ico'),
    (2, '¬dut, dmg, unl, imp, ¬cau'),
    (1, 'dut, dmg, unl, imp, cau, vrt, ¬vst, ¬vun, ift, ¬ila, ¬ico, ¬jus, prp'),
    (1, 'dut, dmg, unl, imp, cau, vrt, ¬vst, ¬vun, ¬ift, ila, ¬ico, ¬jus, prp'),
    (1, 'dut, dmg, unl, imp, cau, vrt, ¬vst, ¬vun, ¬ift, ¬ila, ico, ¬jus, prp'),
    (1, 'dut, dmg, unl, imp, cau, ¬vrt, vst, ¬vun, ift, ¬ila, ¬ico, ¬jus'),
    (1, 'dut, dmg, unl, imp, cau, ¬vrt, vst, ¬vun, ¬ift, ila, ¬ico, ¬jus'),
    (1, 'dut, dmg, unl, imp, cau, ¬vrt, vst, ¬vun, ¬ift, ¬ila, ico, ¬jus'),
    (1, 'dut, dmg, unl, imp, cau, ¬vrt, ¬vst, vun, ift, ¬ila, ¬ico, ¬jus'),
    (1, 'dut, dmg, unl, imp, cau, ¬vrt, ¬vst, vun, ¬ift, ila, ¬ico, ¬jus'),
    (1, 'dut, dmg, unl, imp, cau, ¬vrt, ¬vst, vun, ¬ift, ¬ila, ico, ¬jus'),
    (0, '¬dut, dmg, ¬unl, vrt, ¬vst, jus'),
    (0, '¬dut, dmg, ¬unl, ¬vrt, vst, jus'),
    (0, '¬dut, dmg, unl, imp, cau, vst, ¬prp')
])
```

[34]:
```
# Theory.learn_with_naive_search(case_model).print()
print('---')
Theory.learn_with_pruned_search(case_model, log=True).print()
```

```
---
learning imp …
learning ¬imp …
learning ift …
learning ¬ift …
learning unl …
learning ¬unl …
learning ila …
learning ¬ila …
learning ico …
learning ¬ico …
learning vrt …
learning ¬vrt …
learning vun …
learning ¬vun …
learning prp …
learning ¬prp …
```

```
learning cau …
learning ¬cau …
learning dmg …
learning ¬dmg …
learning jus …
learning ¬jus …
learning dut …
learning ¬dut …
learning vst …
learning ¬vst …
cau <- dut  imp
cau <- imp  vst
cau <- imp  ¬prp
cau  dmg  dut  imp  unl <- ¬jus
cau  dmg  dut  imp  unl  vrt  ¬jus  ¬vst  ¬vun <- prp
cau  dmg  dut  imp  unl  ¬ico  ¬ift  ¬jus <- ila
cau  dmg  dut  imp  unl  ¬ico  ¬ila  ¬jus <- ift
cau  dmg  dut  imp  unl  ¬ift  ¬ila  ¬jus <- ico
cau  dmg  dut  imp  unl  ¬jus  ¬vrt  ¬vst <- vun
cau  dmg  imp  unl  vst  ¬dut <- ¬prp
cau  dmg  imp  unl  ¬jus <- dut
cau  dut <- ico  imp
cau  dut <- ift  imp
cau  dut <- ila  imp
cau  dut <- imp  prp
cau  dut <- imp  vun
cau  dut <- imp  ¬jus
cau  dut  imp  prp  unl  ¬jus <- vrt  ¬vun
cau  dut  imp  prp  ¬jus  ¬vun <- unl  vrt
cau  dut  imp  prp  ¬jus  ¬vun <- vrt  ¬ico
cau  dut  imp  prp  ¬jus  ¬vun <- vrt  ¬ift
cau  dut  imp  prp  ¬jus  ¬vun <- vrt  ¬ila
cau  dut  imp  unl  ¬jus <- ¬ico  ¬vrt
cau  dut  imp  unl  ¬jus <- ¬ico  ¬vst
cau  dut  imp  unl  ¬jus <- ¬ico  ¬vun
cau  dut  imp  unl  ¬jus <- ¬ift  ¬vrt
cau  dut  imp  unl  ¬jus <- ¬ift  ¬vst
cau  dut  imp  unl  ¬jus <- ¬ift  ¬vun
cau  dut  imp  unl  ¬jus <- ¬ila  ¬vrt
cau  dut  imp  unl  ¬jus <- ¬ila  ¬vst
cau  dut  imp  unl  ¬jus <- ¬ila  ¬vun
cau  dut  imp  unl  ¬jus  ¬vrt <- vst  ¬vun
cau  dut  imp  ¬jus <- unl  ¬vrt
cau  dut  imp  ¬jus <- unl  ¬vst
cau  dut  imp  ¬jus <- unl  ¬vun
cau  dut  imp  ¬jus  ¬vrt  ¬vun <- vst  ¬ico
cau  dut  imp  ¬jus  ¬vrt  ¬vun <- vst  ¬ift
cau  dut  imp  ¬jus  ¬vrt  ¬vun <- vst  ¬ila
```

```
cau  dut  prp  ¬jus  ¬vun <- imp  vrt
cau  dut  unl  ¬jus <- imp  ¬vrt
cau  dut  unl  ¬jus <- imp  ¬vst
cau  dut  unl  ¬jus <- imp  ¬vun
cau  dut  ¬jus <- imp  ¬ico
cau  dut  ¬jus <- imp  ¬ift
cau  dut  ¬jus <- imp  ¬ila
cau  imp <- unl  vst
dmg <- unl
dmg <- vst
dmg <- ¬dut
dmg <- ¬vrt
dmg <- ¬vst
dmg <- ¬vun
dmg  imp  unl <- cau
dmg  imp  unl  ¬dut <- ¬cau
dmg  unl <- imp
dmg  unl <- ¬ico
dmg  unl <- ¬ift
dmg  unl <- ¬ila
dmg  unl  ¬dut  ¬ico  ¬ift  ¬ila <- ¬imp
dmg  ¬dut <- ¬unl
dmg  ¬dut  ¬unl <- jus
dmg  ¬vst <- vrt
dut <- cau  dmg  vrt
dut <- cau  dmg  ¬ico
dut <- cau  dmg  ¬ift
dut <- cau  dmg  ¬ila
dut <- cau  dmg  ¬vrt
dut <- cau  dmg  ¬vst
dut <- cau  dmg  ¬vun
dut <- cau  unl  ¬ico
dut <- cau  unl  ¬ift
dut <- cau  unl  ¬ila
dut <- dmg  ico
dut <- dmg  ift
dut <- dmg  ila
dut <- dmg  imp  vrt
dut <- dmg  imp  ¬ico
dut <- dmg  imp  ¬ift
dut <- dmg  imp  ¬ila
dut <- dmg  imp  ¬vrt
dut <- dmg  imp  ¬vst
dut <- dmg  imp  ¬vun
dut <- dmg  prp
dut <- dmg  unl  vrt
dut <- dmg  unl  ¬vrt
dut <- dmg  unl  ¬vst
```

```
dut <- dmg  unl  ¬vun
dut <- dmg  vrt  ¬ico
dut <- dmg  vrt  ¬ift
dut <- dmg  vrt  ¬ila
dut <- dmg  vrt  ¬vun
dut <- dmg  vst  ¬ico
dut <- dmg  vst  ¬ift
dut <- dmg  vst  ¬ila
dut <- dmg  vst  ¬vun
dut <- dmg  vun
dut <- dmg  ¬ico  ¬vrt
dut <- dmg  ¬ico  ¬vst
dut <- dmg  ¬ico  ¬vun
dut <- dmg  ¬ift  ¬vrt
dut <- dmg  ¬ift  ¬vst
dut <- dmg  ¬ift  ¬vun
dut <- dmg  ¬ila  ¬vrt
dut <- dmg  ¬ila  ¬vst
dut <- dmg  ¬ila  ¬vun
dut <- dmg  ¬jus
dut <- ico  unl
dut <- ift  unl
dut <- ila  unl
dut <- imp  unl  ¬ico
dut <- imp  unl  ¬ift
dut <- imp  unl  ¬ila
dut <- prp  unl
dut <- unl  vst  ¬ico
dut <- unl  vst  ¬ift
dut <- unl  vst  ¬ila
dut <- unl  vun
dut <- unl  ¬jus
dut  imp <- ico  ¬ift
dut  imp <- ico  ¬ila
dut  imp <- ift  ¬ico
dut  imp <- ift  ¬ila
dut  imp <- ila  ¬ico
dut  imp <- ila  ¬ift
dut  imp <- prp  ¬ico
dut  imp <- prp  ¬ift
dut  imp <- prp  ¬ila
dut  imp <- vun  ¬ico
dut  imp <- vun  ¬ift
dut  imp <- vun  ¬ila
dut  imp <- ¬ico  ¬jus
dut  imp <- ¬ift  ¬jus
dut  imp <- ¬ila  ¬jus
dut  imp  ¬jus <- cau  ¬ico
```

```
dut  imp  ¬jus <- cau  ¬ift
dut  imp  ¬jus <- cau  ¬ila
dut  prp  ¬jus  ¬vun <- cau  vrt
dut  unl <- ico  ¬vrt
dut  unl <- ico  ¬vst
dut  unl <- ico  ¬vun
dut  unl <- ift  ¬vrt
dut  unl <- ift  ¬vst
dut  unl <- ift  ¬vun
dut  unl <- ila  ¬vrt
dut  unl <- ila  ¬vst
dut  unl <- ila  ¬vun
dut  unl <- vrt  ¬vst  ¬vun
dut  unl <- vst  ¬vrt  ¬vun
dut  unl <- vun  ¬vrt
dut  unl <- vun  ¬vst
dut  unl <- ¬jus  ¬vrt
dut  unl <- ¬jus  ¬vst
dut  unl <- ¬jus  ¬vun
dut  unl  vrt <- prp  ¬vst
dut  unl  vrt <- prp  ¬vun
dut  unl  ¬jus <- cau  ¬vrt
dut  unl  ¬jus <- cau  ¬vst
dut  unl  ¬jus <- cau  ¬vun
ico <- cau  ¬ift  ¬ila
ico <- dut  ¬ift  ¬ila
ico <- imp  ¬ift  ¬ila
ico <- prp  ¬ift  ¬ila
ico <- vrt  ¬ift  ¬ila
ico <- vst  ¬ift  ¬ila
ico <- vun  ¬ift  ¬ila
ico <- ¬ift  ¬ila  ¬jus
ico <- ¬ift  ¬ila  ¬vrt
ico <- ¬ift  ¬ila  ¬vst
ico <- ¬ift  ¬ila  ¬vun
ift <- cau  ¬ico  ¬ila
ift <- dut  ¬ico  ¬ila
ift <- imp  ¬ico  ¬ila
ift <- prp  ¬ico  ¬ila
ift <- vrt  ¬ico  ¬ila
ift <- vst  ¬ico  ¬ila
ift <- vun  ¬ico  ¬ila
ift <- ¬ico  ¬ila  ¬jus
ift <- ¬ico  ¬ila  ¬vrt
ift <- ¬ico  ¬ila  ¬vst
ift <- ¬ico  ¬ila  ¬vun
ila <- cau  ¬ico  ¬ift
ila <- dut  ¬ico  ¬ift
```

```
ila <- imp  ¬ico  ¬ift
ila <- prp  ¬ico  ¬ift
ila <- vrt  ¬ico  ¬ift
ila <- vst  ¬ico  ¬ift
ila <- vun  ¬ico  ¬ift
ila <- ¬ico  ¬ift  ¬jus
ila <- ¬ico  ¬ift  ¬vrt
ila <- ¬ico  ¬ift  ¬vst
ila <- ¬ico  ¬ift  ¬vun
imp <- dut  ¬ico
imp <- dut  ¬ift
imp <- dut  ¬ila
jus  ¬dut <- vrt  ¬unl
jus  ¬dut  ¬vrt <- vst  ¬unl
jus  ¬unl <- vrt  ¬dut
jus  ¬unl <- vst  ¬dut  ¬vrt
prp  vrt <- cau  ¬vst  ¬vun
prp  vrt <- dut  ¬vst  ¬vun
prp  vrt <- ico  ¬vst  ¬vun
prp  vrt <- ift  ¬vst  ¬vun
prp  vrt <- ila  ¬vst  ¬vun
prp  vrt <- imp  ¬vst  ¬vun
prp  vrt <- unl  ¬vst  ¬vun
prp  vrt <- ¬ico  ¬vst  ¬vun
prp  vrt <- ¬ift  ¬vst  ¬vun
prp  vrt <- ¬ila  ¬vst  ¬vun
prp  vrt <- ¬jus  ¬vst  ¬vun
prp  ¬vun <- dut  vrt
prp  ¬vun <- ico  vrt
prp  ¬vun <- ift  vrt
prp  ¬vun <- ila  vrt
prp  ¬vun <- vrt  ¬jus
unl <- dut  ¬vrt
unl <- dut  ¬vst
unl <- dut  ¬vun
vrt <- jus  ¬unl  ¬vst
vrt <- jus  ¬vst
vst <- cau  ¬vrt  ¬vun
vst <- dut  ¬vrt  ¬vun
vst <- ico  ¬vrt  ¬vun
vst <- ift  ¬vrt  ¬vun
vst <- ila  ¬vrt  ¬vun
vst <- imp  ¬vrt  ¬vun
vst <- jus  ¬unl  ¬vrt
vst <- jus  ¬vrt
vst <- unl  ¬vrt  ¬vun
vst <- ¬ico  ¬vrt  ¬vun
vst <- ¬ift  ¬vrt  ¬vun
```

```
vst <- ¬ila  ¬vrt  ¬vun
vst <- ¬jus  ¬vrt  ¬vun
vst  ¬prp <- cau  ¬dut
vun <- cau  ¬vrt  ¬vst
vun <- dut  ¬vrt  ¬vst
vun <- ico  ¬vrt  ¬vst
vun <- ift  ¬vrt  ¬vst
vun <- ila  ¬vrt  ¬vst
vun <- imp  ¬vrt  ¬vst
vun <- unl  ¬vrt  ¬vst
vun <- ¬ico  ¬vrt  ¬vst
vun <- ¬ift  ¬vrt  ¬vst
vun <- ¬ila  ¬vrt  ¬vst
vun <- ¬jus  ¬vrt  ¬vst
¬dut <- cau  dmg  ¬prp
¬dut <- cau  imp  ¬prp
¬dut <- cau  unl  ¬prp
¬dut <- cau  ¬prp
¬dut <- dmg  jus  vrt
¬dut <- dmg  jus  vst
¬dut <- dmg  vrt  ¬unl
¬dut <- dmg  vst  ¬prp
¬dut <- dmg  vst  ¬unl
¬dut <- imp  vst  ¬prp
¬dut <- unl  vst  ¬prp
¬dut <- vrt  ¬unl  ¬vst
¬dut <- vst  ¬prp
¬dut <- vst  ¬unl  ¬vrt
¬dut  ¬imp <- ¬ico  ¬ift  ¬ila
¬dut  ¬unl <- jus  vrt
¬dut  ¬unl <- jus  vrt  ¬vst
¬dut  ¬unl <- jus  vst  ¬vrt
¬dut  ¬unl <- ¬vrt  ¬vst  ¬vun
¬dut  ¬unl  ¬vrt <- jus  vst
¬ico  ¬ift  ¬imp <- ¬dut  ¬ila
¬ico  ¬ila  ¬imp <- ¬dut  ¬ift
¬ift  ¬ila  ¬imp <- ¬dut  ¬ico
¬prp <- imp  vst  ¬dut
¬prp <- unl  vst  ¬dut
¬unl <- dmg  vrt  ¬dut  ¬vst
¬unl <- dmg  vst  ¬dut  ¬vrt
¬unl <- vrt  ¬dut  ¬vst
¬unl <- ¬dut  ¬vrt
¬unl <- ¬dut  ¬vst
¬unl  ¬vrt  ¬vst <- ¬dut  ¬vun
¬vrt  ¬vst <- ¬unl  ¬vun
¬vrt  ¬vun <- dut  vst
¬vrt  ¬vun <- ico  vst
```

```
¬vrt  ¬vun <- ift  vst
¬vrt  ¬vun <- ila  vst
¬vrt  ¬vun <- vst  ¬jus
¬vun <- cau  vst  ¬vrt
¬vun <- imp  vst  ¬vrt
¬vun <- unl  vst  ¬vrt
¬vun <- ¬dut  ¬vrt  ¬vst
¬vun <- ¬unl  ¬vrt  ¬vst
cau  dut  imp  prp  unl  ¬jus  ¬vun <~ vrt
cau  dut  imp  unl  ¬jus  ¬vrt  ¬vun <~ vst
dut <~ cau  dmg
dut <~ cau  imp
dut <~ cau  unl
dut <~ dmg  vrt
dut <~ dmg  vst
dut <~ imp  vst
dut <~ unl  vst
dut  unl <~ vrt  ¬vst
dut  unl <~ vst  ¬vrt
dut  ¬jus <~ cau
¬cau  ¬dut <~ imp
¬dut <~ dmg
¬dut <~ unl
¬dut  ¬ico  ¬ift  ¬imp <~ ¬ila
¬dut  ¬ico  ¬ila  ¬imp <~ ¬ift
¬dut  ¬ift  ¬ila  ¬imp <~ ¬ico
¬dut  ¬unl  ¬vrt  ¬vst <~ ¬vun
¬dut  ¬unl  ¬vrt  ¬vun <~ ¬vst
¬dut  ¬unl  ¬vst  ¬vun <~ ¬vrt
¬vrt  ¬vst  ¬vun <~ ¬unl
```