Declarative Programming

6: reasoning with incomplete information: default reasoning, abduction
Reasoning with incomplete information: overview

Reasoning that leads to conclusions that are plausible, but not guaranteed to be true because not all information is available.

Such reasoning is unsound. Deduction is sound, but only makes implicit information explicit.

**default reasoning**

Assume normal state of affairs, unless there is evidence to the contrary.

"If something is a bird, it flies."

**abduction**

Choose between several explanations that explain an observation.

"I flipped the switch, but the light doesn’t turn on. The bulb might be broken."

**induction**

Generalize a rule from a number of similar observations.

"The sky is full of dark clouds. It will rain."
Default reasoning:
*Tweety is a bird. Normally, birds fly. Therefore, Tweety flies.*

```
bird(tweety).
flies(X) :- bird(X), normal(X).
```

has three models:

- `{bird(tweety)}`
- `{bird(tweety), flies(tweety)}`
- `{bird(tweety), flies(tweety), normal(tweety)}`

*bird(tweety)* is the only logical conclusion of the program because it occurs in every model.

If we want to conclude *flies(tweety)* through deduction, we have to state *normal(tweety)* explicitly. Default reasoning assumes something is normal, unless it is known to be abnormal.
Default reasoning:
A more natural formulation using abnormal/1

bird(tweety).
flies(X) ; abnormal(X) :- bird(X).

has two minimal models:
{bird(tweety), flies(tweety)}
{bird(tweety), abnormal(tweety)}

model 2 is model of the general clause:
abnormal(X) :- bird(X), not(flies(X)).
model 1 is model of the general clause:
flies(X) :- bird(X), not(abnormal(X)).

bird(tweety).
flies(X) :- bird(X), not(abnormal(X)).
ostrich(tweety).
abnormal(X) :- ostrich(X).

tweety no longer flies, he is an ostrich: the default rule (birds fly) is cancelled by the more specific rule (ostriches)
Default reasoning: non-monotonic form of reasoning

new information can invalidate previous conclusions:

```
bird(tweety).
flies(X):-bird(X),not(abnormal(X)).
```

```
bird(tweety).
flies(X):-bird(X),not(abnormal(X)).
ostrich(tweety).
abnormal(X) :- ostrich(X).
```

Not the case for deductive reasoning, which is monotonic in the following sense:

\[ \text{Th} \vdash p \Rightarrow \text{Th} \cup \{q\} \vdash p \]

\[ \text{Closure(Th)} = \{p \mid \text{Th} \vdash p\} \]

\[ \text{Th}_1 \subseteq \text{Th}_2 \Rightarrow \text{Closure(Th}_1) \subseteq \text{Closure(Th}_2) \]
Default reasoning:
without not/1, using a meta-interpreter

problematic: e.g., floundering but also because it has no clear declarative semantics

Distinguish regular rules (without exceptions) from default rules (with exceptions.)
Only apply a default rule when it does not lead to an inconsistency.

default((flies(X) :- bird(X))).
rule((not(flies(X)) :- penguin(X))).
rule((bird(X) :- penguin(X))).
rule((penguin(tweety) :- true)).
rule((bird(opus) :- true)).
Default reasoning:
using a meta-interpreter

E explains F: lists the rules used to prove F

prove(true,E,E) :- !.
prove((A,B),E0,E) :- !,
    prove(A,E0,E1),
    prove(B,E1,E).
prove(A,E0, [rule((A:-B))|E]) :-
    rule((A:-B)),
    prove(B,E0,E).
contradiction(not(A),E) :- !,
    prove(A,E,__).
contradiction(A,E):-
    prove(not(A),E,__).
Default reasoning: using a meta-interpreter, Opus example

default((flies(X) :- bird(X))).
rule((not(flies(X)) :- penguin(X))).
rule((bird(X) :- penguin(X))).
rule((penguin(tweety) :- true)).
rule((bird(opus) :- true)).

?- explain(flies(X),E)
X=opus
E=[default((flies(opus) :- bird(opus))),
    rule((bird(opus) :- true))]

?- explain(not(flies(X)),E)
X=tweety
E=[rule((not(flies(tweety)) :- penguin(tweety))),
   rule((penguin(tweety) :- true))]
Default reasoning: using a meta-interpreter, Dracula example

```prolog
default((not(flies(X)) :- mammal(X))).
default((flies(X) :- bat(X))).
default((not(flies(X)) :- dead(X))).
  rule((mammal(X) :- bat(X))).
  rule((bat(dracula) :- true)).
  rule((dead(dracula) :- true)).
```

?-explain(flies(dracula),E)
E=[default((flies(dracula) :- bat(dracula))),
  rule((bat(dracula) :- true))]

?-explain(not(flies(dracula)),E)
E=[default((not(flies(dracula)) :- mammal(dracula))))
  rule((mammal(dracula) :- bat(dracula))),
  rule((bat(dracula) :- true))]
E=[default((not(flies(dracula)) :- dead(dracula))))
  rule((dead(dracula) :- true))]
```

Dracula flies because bats typically fly.

Dracula doesn’t fly because mammals typically don’t.

Dracula doesn’t fly because dead things typically don’t.
Default reasoning: using a revised meta-interpreter

need a way to cancel particular defaults in certain situations: bats are flying mammals although the default is that mammals do not fly

name associated with default rule

default(mammals_dont_fly(X), (not(flies(X)):-mammal(X))).
default(bats_fly(X), (flies(X):-bat(X))).
default(dead_things_dont_fly(X), (not(flies(X)):-dead(X))).
  rule((mammal(X):-bat(X))).
  rule((bat(dracula):-true)).
  rule((dead(dracula):-true)).
  rule((not(mammals_dont_fly(X)):-bat(X))).
  rule((not(bats_fly(X)):-dead(X))).
Default reasoning: using a revised meta-interpreter

need a way to cancel particular defaults in certain situations: bats are flying mammals although the default is that mammals do not fly

```
default(mammals_dont_fly(X), (not(flies(X)):-mammal(X))).
default(bats_fly(X), (flies(X):-bat(X))).
default(dead_things_dont_fly(X), (not(flies(X)):-dead(X))).
rule((mammal(X):-bat(X))).
rule((bat(dracula):-true)).
rule((dead(dracula):-true)).
rule((not(mammals_dont_fly(X)):-bat(X))).
rule((not(bats_fly(X)):-dead(X))).
```
Default reasoning: using a revised meta-interpreter

explanations keep track of names rather than default rules

explain(A,E0,[default(Name)|E]):-  
default(Name,(A:- B)),  
explain(B,E0,E),  
not(contradiction(Name,E)),  
not(contradiction(A,E)).

default rule is not cancelled in this situation: e.g., do not use default named bats_fly(X) if you can prove not(bats_fly(X))

dracula can not fly after all

?-explain(flies(dracula),E)  
no  
?-explain(not(flies(dracula)),E)  
E= [default(dead_things_dont_fly(dracula)),  
rule((dead(dracula):- true))]
Default reasoning: 
Dracula revisited

default(mammals_dont_fly(X), (not(flies(X)):-mammal(X))).
default(bats_fly(X), (flies(X):-bat(X))).
default(dead_things_dont_fly(X), (not(flies(X)):-dead(X))).
  rule((mammal(X):-bat(X))).
  rule((bat(dracula):-true)).
  rule((dead(dracula):-true)).
  rule((not(mammals_dont_fly(X)):-bat(X))).
  rule((not(bats_fly(X)):-dead(X))).

notflies(X):-mammal(X),not(flying_mammal(X)).
flies(X):-bat(X),not(nonflying_bat(X)).
notflies(X):-dead(X),not(flying_deadthing(X)).
mammal(X):-bat(X).
bat(dracula).
dead(dracula).
flying_mammal(X):-bat(X).
nonflying_bat(X):-dead(X).

typical case is a clause that is only applicable when it does not lead to inconsistencies; applicability can be restricted using clause names

typical case is general clause that negates abnormality predicate
Abduction: given a theory $T$ and an observation $O$, find an explanation $E$ such that $T \cup E \models O$

- $T$ likes(peter,S) :- student_of(S,peter).
  likes(X,Y) :- friend(X,Y).
- $O$ likes(peter,paul)
- $E_1$ \{student_of(paul,peter)}
- $E_2$ \{friend(peter,paul)}

Default reasoning makes assumptions about what is false (e.g., tweety is not an abnormal bird), abduction can also make assumptions about what is true.

Another possibility, but abductive explanations are usually restricted to ground literals with predicates that are undefined in the theory (abducibles).
Abduction: abductive meta-interpreter

Theory $\cup$ Explanation $\models$ Observation

Try to prove Observation from theory, when a literal is encountered that cannot be resolved (an abducible), add it to the Explanation.

```prolog
abduce(0,E):-
    abduce(0,[],E).
abduce(true,E,E) :- !.
abduce((A,B),E0,E) :- !,
    abduce(A,E0,E1),
    abduce(B,E1,E).
abduce(A,E0,E):-
    clause(A,B),
    abduce(B,E0,E).
abduce(A,E,E):-
    element(A,E).
abduce(A,E,[A|E]):-
    not(element(A,E)),
    abducible(A).
abducible(A):-
    not(clause(A,B)).
likes(peter,S) :- student_of(S,peter).
likes(X,Y) :- friend(X,Y).
?-abduce(likes(peter,paul),E)
E = [student_of(paul,peter)];
E = [friend(paul,peter)]
```

A already assumed

A can be assumed if it was not already assumed and it is an abducible.
Abduction: abductive meta-interpreter and negation

flies(X) :- bird(X), not(abnormal(X)).
abnormal(X) :- penguin(X).
bird(X) :- penguin(X).
bird(X) :- sparrow(X).

?-abduce(flies(tweety),E)
E = [not(abnormal(tweety)),penguin(tweety)];
E = [not(abnormal(tweety)),sparrow(tweety)];

Since no clause is found for not(abnormal(tweety)), it is added to the explanation.
Abduction:
first attempt at abduction with negation

extend abduce/3 with negation as failure:

abduce(not(A),E,E):-
    not(abduce(A,E,E)).

do not add negated literals to the explanation:

abducible(A):-
    A \= not(X),
    not(clause(A,B)).

flies(X) :- bird(X), not(abnormal(X)).
abnormal(X) :- penguin(X).
bird(X) :- penguin(X).
bird(X) :- sparrow(X).

?-abduce(flies(tweety),E)
E = [sparrow(tweety)]
Abduction:

first attempt at abduction with negation: FAILED

any explanation of bird(tweety) will also be an explanation of flies1(tweety):

```prolog
flies1(X):- not(abnormal(X)),bird(X).
abnormal(X) :- penguin(X).
bird(X) :- penguin(X).
bird(X) :- sparrow(X).
```

the fact that abnormal(tweety) is to be considered false, is not reflected in the explanation:

```prolog
?- abduce(not(abnormal(tweety)),[],[]), true .
```

assumes the explanation is already complete

reversed order of literals
Abduction:

**final abductive meta-interpreter: abduce/3**

```
abduce(true,E,E) :- !.
abduce((A,B),E0,E) :- !,
    abduce(A,E0,E1),
    abduce(B,E1,E).
abduce(A,E0,E):-
    clause(A,B),
    abduce(B,E0,E).
abduce(A,E,E):-
    element(A,E).
abduce(A,E,[A|E]):-
    not(element(A,E)),
    abducible(A),
    not(abduce_not(A,E,E)).
abduce(not(A),E0,E):-
    not(element(A,E0)),
    abduce_not(A,E0,E).
```

- **A already assumed**
- **A can be assumed if it was not already, it is abducible, E doesn’t explain not(A)**
- **only assume not(A) if A was not already assumed, ensure not(A) is reflected in the explanation**

```
abducible(A):-
    A \= not(X),
    not(clause(A,B)).
```
Abduction:

final abductive meta-interpreter: abduce_not/3

\[
\text{abduce_not}((A,B),E0,E) :\!
\begin{align*}
&!, \\
&\text{abduce_not}(A,E0,E) ; \\
&\text{abduce_not}(B,E0,E).
\end{align*}
\]

\[
\text{abduce_not}(A,E0,E) :\!
\begin{align*}
&\text{setof}(B,\text{clause}(A,B),L), \\
&\text{abduce_not_list}(L,E0,E).
\end{align*}
\]

\[
\text{abduce_not}(A,E,E) :\!
\begin{align*}
&\text{element}(\text{not}(A),E).
\end{align*}
\]

\[
\text{abduce_not}(A,E,[\text{not}(A)|E]) :\!
\begin{align*}
&\text{not}(\text{element}(\text{not}(A),E)), \\
&\text{abducible}(A), \\
&\text{not}(\text{abduce}(A,E,E)).
\end{align*}
\]

\[
\text{abduce_not}(\text{not}(A),E0,E) :\!
\begin{align*}
&\text{not}(\text{element}(\text{not}(A),E0)), \\
&\text{abduce}(A,E0,E).
\end{align*}
\]

\[
\text{abduce_not_list}([],E,E).
\]

\[
\text{abduce_not_list}([B|Bs],E0,E) :\!
\begin{align*}
&\text{abduce_not}(B,E0,E1), \\
&\text{abduce_not_list}(Bs,E1,E).
\end{align*}
\]

**disjunction:** a negation
conjunction can be explained by
explaining A or by explaining B

not(A) is explained by explaining
not(B) for **every** A:-B

not(A) already assumed

**assume not(A) if not already so, A is abducible**
and E does not already explain A

**explain not(not(A)) by explaining A**
Abduction:

**final abductive meta-interpreter: example**

flies(X) :- bird(X), not(abnormal(X)).
flies1(X) :- not(abnormal(X)), bird(X).
abnormal(X) :- penguin(X).
abnormal(X) :- dead(X).
bird(X) :- penguin(X).
bird(X) :- sparrow(X).

?- abduce(flies(tweety),E).
E = [not(penguin(tweety)),
     not(dead(tweety)),
     sparrow(tweety)]

?- abduce(flies1(tweety),E).
E = [sparrow(tweety),
     not(penguin(tweety)),
     not(dead(tweety))]

now abduces as expected
Abduction: diagnostic reasoning

3-bit adder

usually what has to be carried on from previous computation

Theory: system description
Observation: input values, output values
Explanation: diagnosis=hypothesis about which components are faulty

Theory describing normal operation

adder(X,Y,Z,Sum,Carry) :-
xor(X,Y,S),
xor(Z,S,Sum),
and(X,Y,C1),and(Z,S,C2),
or(C1,C2,Carry).

xor(0,0,0). and(0,0,0). or(0,0,0).
xor(0,1,1). and(0,1,0). or(0,1,1).
xor(1,0,1). and(1,0,0). or(1,0,1).
xor(1,1,0). and(1,1,1). or(1,1,1).
Abduction:
**diagnostic reasoning - fault model**

describes how each component can behave in a faulty manner

can be nested: subSystemName-componentName

```prolog
fault(NameComponent=State)

adder(N,X,Y,Z,Sum,Carry):-
xorg(N-xor1,X,Y,S),
xorg(N-xor2,Z,S,Sum),
andg(N-and1,X,Y,C1),
andg(N-and2,X,S,C2),
org(N-or1,C1,C2,Carry).

xorg(N,X,Y,Z) :- xor(X,Y,Z).
xorg(N,0,0,1) :- fault(N=s1).
xorg(N,0,1,0) :- fault(N=s0).
xorg(N,1,0,0) :- fault(N=s0).
xorg(N,1,1,1) :- fault(N=s1).

xandg(N,X,Y,Z) :- and(X,Y,Z).
xandg(N,0,0,1) :- fault(N=s1).
xandg(N,0,1,1) :- fault(N=s1).
xandg(N,1,0,1) :- fault(N=s1).
xandg(N,1,1,0) :- fault(N=s0).

org(N,X,Y,Z) :- or(X,Y,Z).
org(N,0,0,1) :- fault(N=s1).
org(N,0,1,0) :- fault(N=s0).
org(N,1,0,0) :- fault(N=s0).
org(N,1,1,0) :- fault(N=s0).
```
s0: output stuck at 0,
s1: output stuck at 1
Abduction: diagnostic reasoning - diagnoses for faulty adder

diagnosis(Observation,Diagnosis):-
    abduce(Observation,Diagnosis).

?-diagnosis(add(a,0,0,1,0,1),D).
D = [fault(a-or1=s1), fault(a-xor2=s0)];
D = [fault(a-and2=s1), fault(a-xor2=s0)];
D = [fault(a-and1=s1), fault(a-xor2=s0)];
D = [fault(a-and2=s1), fault(a-and1=s1), fault(a-xor2=s0)];
D = [fault(a-or1=s1), fault(a-and2=s0), fault(a-xor1=s1)];
D = [fault(a-and1=s1), fault(a-xor1=s1)];
D = [fault(a-and2=s0), fault(a-and1=s1), fault(a-xor1=s1)];
D = [fault(a-xor1=s1)]

obvious diagnosis: outputs of adder are stuck

adder(N,X,Y,Z,Sum,Carry): both Sum and Carry are wrong

most plausible as only one faulty component accounts for entire fault