7 Supplementary Material

7.1 Methodology: Details

Algorithm 3 is_counterfactual: checks if a state is a counterfactual/goal state

Require: State $s \in S$, Set of Causal *rules* C, Set of Decision *rules* Q

if s satisfies **ALL** rules in C **AND** s satisfies **NO** rules in Q **then**
    Return $TRUE$.
else
    Return $FALSE$.
end if

***is_ counterfactual*: Checks for Goal State/ Counterfactual State** The function *is_ counterfactual* is our algorithmic implementation of checking if a state $s \in G$ from definition 8. In Algorithm 3, we specify the pseudo-code for a function *is_ counterfactual* which takes as arguments a state $s \in S$, a set of causal rules C, and a set of Decision rules Q. The function checks if a state $s \in S$ is a counterfactual/goal state. By definition *is_ counterfactual* is $TRUE$ for state s that is causally consistent with all $c \in C$ and **does not** agree with the any decision rules $q \in Q$.

$$is\_counterfactual(s, C, Q) = TRUE \mid s\ agrees\ with\ C;\ \ s\ disagrees\ with\ Q; \tag{11}$$

Intervene: Transition Function for moving from the current state to the new state The function *intervene* is our algorithmic implementation of the transition function δ from definition 6. In Algorithm 4, we specify the pseudo-code for a function *intervene*, which takes as arguments an Initial State i that is causally consistent, a set of Causal Rules C, and a set of actions A. It is called by *find_ path* in line 4 of algorithm 1.

The function *intervene* acts as a transition function that inputs a list *visited_ states* containing the current state s as the last element, and returns the new state s' by appending s' to *visited_ states*. The new state s' is what the current state s traverses. Additionally, the function *intervene* ensures that no states are revisited. In traversing from s to s', if there are a series of intermediate states that are **not** causally consistent, it is also included in *visited_ states*, thereby depicting how to traverse from one causally consistent state to another.

Update In Algorithm 5, we specify the pseudo-code for a function *update*, that given a state s, list *actions_ taken*, list *visited_ states*and given an action a, appends a to *actions_ taken*. It also appends the list *actions_ taken* as well as the

Algorithm 4 intervene: reach a causally consistent state from a causally consistent current state

Require: Causal *rules* C, List *visited_states*, List *actions_taken*, Actions $a \in A$:

- Causal Action: s gets altered to a causally consistent new state $s' = a(s)$. OR
- Direct Action: new state $s' = a(s)$ is obtained by altering 1 feature value of s.

Set $(s, actions\_taken)$ = *pop(visited_states)*
Try to select an action $a \in A$ ensuring *not_member(a(s),visited_states)* and *not_member(a,actions_taken)* are $TRUE$
if a exists **then**
  Set $(s, actions\_taken), visited\_states = update(s, visited\_states, actions\_taken, a)$
else
  //Backtracking
  if *visited_states* is empty **then**
    EXIT with Failure
  end if
  Set $(s, actions\_taken)$ = *pop(visited_states)*
end if
Set $(s, actions\_taken), visited\_states =$
  $make\_consistent(s, actions\_taken, visited\_states, C, A)$
Append $(s, actions\_taken)$ to *visited_states*
Return *visited_states.*

Algorithm 5 update: Updates the list *actions_taken* with the planned action. Then updates the current state.

Require: State s, List *visited_states*, List *actions_taken*, Action $a \in A$:

- Causal Action: s gets altered to a causally consistent new state $s' = a(s)$. OR
- Direct Action: new state $s' = a(s)$ is obtained by altering 1 feature value of s.

Append a to *actions_taken.*
Append $(s, actions\_taken)$ to *visited_states.*
Set $s = a(s)$.
return $(s, [\,])$, *visited_states*

new resultant state resulting from the action $a(s)$ to the list *visited_states*. The list *actions_taken* is used to track all the actions attempted from the current state to avoid repeating them. The function *update* is called by both functions *intervene* and *make_consistent.*

Candidate Path Given the CFG (S_C, S_Q, I, δ) constructed from a run of algorithm 1 and the return value that we refer to as r such that r is a list (algorithm succeeds). r' is the resultant list obtained from removing all elements containing states $s' \notin S_C$. We can construct the corresponding candidate path as follows: r'_i represents the i th element of list r'. The candidate path is the sequence of states $s_0, ..., s_{m-1}$, where m is the length of list r'. s_i is the state corresponding to r'_i for $0 \leq i < m$.

7.2 Proofs

Theorem 1. *Soundness Theorem*
Given a CFG $\mathbb{X} = (S_C, S_Q, I, \delta)$, constructed from a run of algorithm 1 and a corresponding candidate path P, P is a solution path for $\mathbb{X}$.

Proof. *Let G be a goal set for $\mathbb{X}$. By definition 11, $P = s_0, ..., s_m$, where $m \geq 0$. By definition 9 we must show P has the following properties.*

1) $s_0 = I$
2) $s_m \in G$
3) $s_j \in S_C$ $for\ all\ j \in \{0, ..., m\}$
4) $s_0, ..., s_{m-1} \notin G$
5) $s_{i+1} \in \delta(s_i)$ $for\ i \in \{0, ..., m-1\}$

1) By definition 4, i is causally consistent and cannot be removed from the candidate path. Hence I must be in the candidate path and is the first state as per line 2 in algorithm 1. Therefore s_0 must be i.
2) The while loop in algorithm 1 ends if and only if $is\_counterfactual(s, C, Q)$ is True. From theorem 2, $is\_counterfactual(s, C, Q)$ is True only for the goal state. Hence $s_m \in G$.
3) By definition 11 of the candidate path, all states $s_j \in S_C$ $for\ all\ j \in \{0, ..., m\}$.
4) By theorem 4, we have proved the claim $s_0, ..., s_{m-1} \notin G$.
5) By theorem 3, we have proved the claim $s_{i+1} \in \delta(s_i)$ $for\ i \in \{0, ..., m-1\}$. Hence we proved the candidate path P (definition 11) is a solution path (definition 9).

Theorem 2. *Given a CFG $\mathbb{X} = (S_C, S_Q, I, \delta)$, constructed from a run of algorithm 1, with goal set G, and $s \in S_C$; $is\_counterfactual(s, C, Q)$ will be $TRUE$ if and only if $s \in G$.*

Proof. *By the definition of the goal set G we have*

$$G = \{s \in S_C | s \notin S_Q\} \tag{12}$$

For $is\_counterfactual$ which takes as input the state s, the set of causal rules C and the set of decision rules Q (Algorithm 3), we see that by from line 1 in algorithm 3, it returns $TRUE$ if it satisfied all rules in C and no rules in Q.

By the Definition 3, $s \in S_Q$ if and only if it satisfies a rule in Q. Therefore, $is\_counterfactual(s, C, Q)$ is $TRUE$ if and only if $s \notin S_Q$ and since $s \in S_C$ and $s \notin S_Q$, then $s \in G$.

Theorem 3. *Given a CFG $\mathbb{X} = (S_C, S_Q, I, \delta)$, constructed from a run of algorithm 1 and a corresponding candidate path $P = s_0, ..., s_m$; $s_{i+1} \in \delta(s_i)$ $for\ i \in \{0, ..., m-1\}$*

Proof. This property can be proven by induction on the length of the list visited_lists obtained from Algorithm 1, 2, and 4.
Base Case*: The list visited_lists from algorithm 1 has length of 1, i.e., [s_0]. The property $s_{i+1} \in \delta(s_i)$ for i $\in \{0, ..., m-1\}$ is trivially true as there is no s_{-1}.*
Inductive Hypotheses*: We have a list [$s_0, ..., s_{n-1}$] of length n generated from 0 or more iteration of running the function intervene (algorithm 4), and it satisfies the claim $s_{i+1} \in \delta(s_i)$ for i $\in \{0, ..., n-1\}$*

Inductive Step*: If we have a list [$s_0, ..., s_{n-1}$] of length n and we wish to get element s_n obtained through running another iteration of function intervene (algorithm 4). Since [$s_0, ..., s_{n-1}$] is of length n by the inductive hypothesis, it satisfies the property, and it is sufficient to show $s_n \in \delta(s_{n-1})$ where $s_{i+1} \in \delta(s_i)$ for i $\in \{0, ..., n-1\}$.*

The list visited_lists from algorithm 1 has length of n. Going from s_{n-1} to s_n involves calling the function intervene (Algorithm 4) which in turn calls the function make_consistent (Algorithm 2).

Function make_consistent (Algorithm 2) takes as input the state s, the list of actions taken actions_taken, the list of visited states visited_states, the set of causal rules C and the set of possible actions A. It returns visited_states with the new causally consistent states as the last element. From line 1, if we pass as input a causally consistent state, then function make_consistent does nothing. On the other hand, if we pass a causally inconsistent state, it takes actions to reach a new state. Upon checking if the action taken results in a new state that is causally consistent from the while loop in line 1, it returns the new state. Hence, we have shown that the moment a causally consistent state is encountered in function make_consistent, it does not add any new state.

Function intervene (Algorithm 4) takes as input the list of visited states visited_states which contains the current state as the last element, the set of causal rules C and the set of possible actions A. It returns visited_states with the new causally consistent states as the last element. It calls the make_consistent function. For the function intervene, in line 1 it obtains the current state (in this case s_{n-1}) from the list visited_states. It is seen in line 2 that an action a is taken:

1) Case 1: If a causal action is taken, then upon entering the the function make_consistent (Algorithm 2), it will not do anything as causal actions by definition result in causally consistent states.

2) Case 2: If a direct action is taken, then the new state that may or may not be causally consistent is appended to visited_states. The call to the function make_consistent will append one or more states with only the final state appended being causally consistent.

Hence we have shown that the moment a causally consistent state is appended in function intervene, it does not add any new state. This causally consistent

Dataset	# of Features Used	# of Counterfactuals
Adult	6	112
Cars	4	78
German	7	240

Table 2: Table showing a Number of Counterfactuals produce by the *is_ counterfactual* function given all possible states.

state is s_n. In both cases $s_n = \sigma(s_{n-1})$ as defined in Definition 10 and this $s_n \in \delta(s_{n-1})$.

Theorem 4. *Given a CFG problem $\mathbb{X} = (S_C, S_Q, I, \delta)$, constructed from a run of algorithm 1, with goal set G and a corresponding candidate path $P = s_0, ..., s_m$ with $m \geq 0$, $s_0, ..., s_{m-1} \notin G$.*

Proof. This property can be proven by induction on the length of the list visited_ lists obtained from Algorithm 1, 2, and 4.
***Base Case**: visited_ lists has length of 1. Therefore the property $P = s_0, ..., s_m$ with $m \geq 0$, $s_0, ..., s_{m-1} \notin G$ is trivially true as state s_j for $j < 0$ does not exist.*

***Inductive Hypotheses**: We have a list $[s_0, ..., s_{n-1}]$ of length n generated from 0 or more iteration of running the function intervene (Algorithm 4), and it satisfies the claim $s_0, ..., s_{n-2} \notin G$.*

***Inductive Step**: Suppose we have a list $[s_0, ..., s_{n-1}]$ of length n and we wish to append the n+1 th element (state s_n) by calling the function intervene, and we wish to show that that the resultant list satisfies the claim $s_0, ..., s_{n-1} \notin G$. The first n-1 elements $(s_0, ..., s_{n-2})$ are not in G as per the inductive hypothesis.*

*From line 3 in the function find_ path (Algorithm 1), we see that to call the function intervene another time, the current state (in this case s_{n-1}) **cannot** be a counterfactual, by theorem 2. Hence $s_{n-1} \notin G$*

Therefore by induction the claim $s_0, ..., s_{n-1} \notin G$ holds.

7.3 Experimental Setup

Counterfactuals Algorithm 3 *is_ counterfactual* returns `True` for all states consistent with causal rules C that disagree with the decision rules Q. Given our state space S, from *is_ counterfactual*, we obtain all states that are realistic counterfactuals. Table 2 shows the number of counterfactuals that we obtain using *is_ counterfactual*.

Dataset: Cars The Car Evaluation dataset provides information on car purchasing acceptability. We relabelled the Car Evaluation dataset to *'acceptable'* and *'unacceptable'* in order to generate the counterfactuals. We applied the *CoGS* methodology to rules generated by the FOLD-SE algorithm. These rules indicate whether a car is *acceptable* to buy or should be *rejected*, with the undesired outcome being rejection.

For the Car Evaluation dataset, table 1) shows a path from initial *rejection* to changes that make the car *acceptable* for purchase.

We run the FOLD-SE algorithm to produce the following rules:

```
label(X,'negative') :- persons(X,'2').

label(X,'negative') :- safety(X,'low').

label(X,'negative') :- buying(X,'vhigh'), maint(X,'vhigh').

label(X,'negative') :- not buying(X,'low'), not buying(X,'med'),
                       maint(X,'vhigh').

label(X,'negative') :- buying(X,'vhigh'), maint(X,'high').
```

The rules described above indicate if the purchase of a car was rejected .

1. Accuracy: 93.9%
2. Precision: 100%
3. Recall: 91.3%

2) Features and Feature Values used:

– Feature: persons
– Feature: safety
– Feature: buying
– Feature: maint

Dataset: Adult The Adult dataset includes demographic information with labels indicating income ('$=<\ \$50k/year$' or '$>\ \$50k/year$'). We applied the *CoGS* methodology to rules generated by the FOLD-SE algorithm on the Adult dataset [5]. These rules indicate whether someone makes '=<$50k/year'.

Additionally, causal rules are also learnt and verified (for example, if feature **'Marital Status'** has value *'Never Married'*, then feature **'Relationship'** should (not) have the value *'husband'* or *'wife'*. We learn the rules to verify these assumptions on cause-effect dependencies.

The goal of *CoGS* is to find a path to a counterfactual instance where a person makes '>$50k/year'.

For the Adult dataset, Table 1 shows a path from making '=<$50k/year' to '>$50k/year'.

We run the FOLD-SE algorithm to produce the following decision making rules:

```
label(X,'<=50K') :- not marital_status(X,'Married-civ-spouse'),
```

```
                       capital_gain(X,N1), N1=<6849.0.

label(X,'<=50K') :- marital_status(X,'Married-civ-spouse'),
                       capital_gain(X,N1), N1=<5013.0,
                       education_num(X,N2), N2=<12.0.
```

1. Accuracy: 84.5%
2. Precision: 86.5%
3. Recall: 94.6%

2) FOLD-SE gives Causal rules for the 'marital_status' feature having value 'never_married':

```
marital_status(X,'Never-married') :- not relationship(X,'Husband'),
                             not relationship(X,'Wife'), age(X,N1),
N1=<29.0.
```

1. Accuracy: 86.4%
2. Precision: 89.2%
3. Recall: 76.4%

3) FOLD-SE gives Causal rules for the 'marital_status' feature having value 'Married-civ-spouse':

```
marital_status(X,'Married-civ-spouse') :- relationship(X,'Husband').

marital_status(X,'Married-civ-spouse') :- relationship(X,'Wife').
```

1. Accuracy: 99.1%
2. Precision: 99.9%
3. Recall: 98.2%

4) For values of the feature 'marital_status' that are not 'Married-civ-spouse' or 'never_married' which we shall call 'neither', a user defined rule is used:

```
marital_status(X,neither) :- not relationship(X,'Husband'),
                       not relationship(X,'Wife').
```

5) FOLD-SE gives Causal rules for the 'relationship' feature having value 'husband':

```
relationship(X,'Husband') :- sex(X,'Male') ,
                       age(X,N1), not(N1=<27.0).
```

1. Accuracy: 82.3%
2. Precision: 71.3%
3. Recall: 93.2%

5) For the ‘relationship’ feature value of ‘wife’, a user defined rule is used:

```
relationship(X,‘Wife’) :- sex(X,‘Female’).
```

6)Features Used in Generating the counterfactual path:

- Feature: marital_status
- Feature: relationship
- Feature: sex
- capital_gain
- education_num
- age

Dataset: German We run the FOLD-SE algorithm to produce the following decision making rules:

```
label(X,‘good’):-checking_account_status(X,‘no_checking_account’)

label(X,‘good’):-not checking_account_status(X,‘no_checking_account’),
                 not credit_history(X,‘all_dues_atbank_cleared’),
                 duration_months(X,N1), N1=<21.0,
                 credit_amount(X,N2), not(N2=<428.0),
                 not ab1(X,‘True’).

ab1(X,‘True’):-property(X,‘car or other’),
                 credit_amount(X,N2), N2=<1345.0.
```

1. Accuracy: 77%
2. Precision: 83%
3. Recall: 84.2%

2) FOLD-SE gives Causal rules for the ‘present_employment_since’ feature having value ‘employed’ where employed is the placeholder for all feature values that are **not** equal to the feature value ‘unemployed’:

```
present_employment_since(X,‘employed’) :-
                 not job(X,‘unemployed/unskilled-non_resident’).
```

1. Accuracy: 95%
2. Precision: 96.4%
3. Recall: 98.4%

3) For values of the feature ‘present_employment_since’ that are ‘unemployed’, a user defined rule is used

```
present_employment_since(X,‘unemployed’) :-
                 job(X,‘unemployed/unskilled-non_resident’).
```

6)Features Used in Generating the counterfactual path:

- checking_account_status

- credit_history
- property
- duration_months
- credit_amount
- present_employment_since
- job