Nonlinear Counterfactuals in Isabelle/HOL

Johannes Hauschild

December 12, 2023

Abstract

This thesis investigates the semantics of Lewis' counterfactual operators, as well as their improvements by Finkbeiner and Siber. Moving on, we give a further improvement.

To this end we formalise the counterfactual operators under investigation in a logic CFL defined over world dependent Kripke structures. Problematic assumptions about these Kripke structures are necessary for Lewis' operators to obtain their (likely) intended semantics. We validate that Finkbeiner and Sibers operators enable a relaxation of these assumptions. Furthermore, we define counterfactual operators which again allow for a further relaxation.

Finally, we analyse the relationship between CFL and CTL* using a translation function. Results show that using the translation function, CTL* and CFL are not comparable in terms of their expressive power.

Our findings are backed by a formalisation in Isabelle/HOL.

1 Structure and Sets of Assumptions

```
theory StructureAndAssumptions
  imports Main
begin
```

This structure is meant to implement a Kripke structure close to these employed by Baier and Katoen [BK08]. We dropped the set of initial states, allowing every state to be an initial state.

```
locale world_dependent_kripke_structure =
  fixes
    — Assigning a set of atomic propositions to each world.
    labeling :: <'i ⇒ 'ap set> and
    — w1 ≤w w2 is enconding the notion "w1 at least as similar to w as is w2". A similar relation was
defined by Lewis [Lew73] as well as Finkbeiner and Siber [FS23].
    accessibility :: <'i ⇒ 'i ⇒ 'i ⇒ bool> ("_ ≤<_> _" [70, 70, 70] 80)
  assumes
    reflexive [intro]: <w1 ≤<w> w1>

locale preordered_counterfactual_structure = world_dependent_kripke_structure labeling accessibility
  for
    labeling :: <'i ⇒ 'ap set> and
    accessibility :: "'i ⇒ 'i ⇒ 'i ⇒ bool" ("_ ≤<_> _" [70, 70, 70] 80) +
  assumes
    — Lewis' [Lew73] as well as Finkbeiner and Sibers [FS23] structure are transitive:
    transitive [intro]: <⟦w1 ≤<w> w2; w2 ≤<w> w3⟧ ⟹ w1 ≤<w> w3>  and
    — We assume that any two worlds which are comparable in respect to a world are also accessible
from that world:
    meaningful_acessibility [intro]: <⟦w1 ≤<w> w2; w1 ≠ w2⟧ ⟹ w ≤<w> w1 ∧ w ≤<w> w2>
```

```
locale finkbeiner_siber_structure = preordered_counterfactual_structure labeling accessibility
  for
    labeling :: <'i ⇒ 'ap set> and
    accessibility :: "'i ⇒ 'i ⇒ 'i ⇒ bool" ("_ ≤<_> _" [70, 70, 70] 80) +
  assumes
    total_accessibility: <∀ w'. w ≤<w> w'>

locale lewisian_structure =  preordered_counterfactual_structure labeling accessibility
  for
    labeling :: <'i ⇒ 'ap set> and
    accessibility :: "'i ⇒ 'i ⇒ 'i ⇒ bool" ("_ ≤<_> _" [70, 70, 70] 80) +
  assumes
    linearity: <⟦w ≤<w> w1; w≤<w> w2⟧ ⟹ w1 ≤<w> w2 ∨ w2 ≤<w> w1>

locale total_accessible_lewisian_structure = lewisian_structure + finkbeiner_siber_structure
```

2 Formula equivalences

```
lemma (in preordered_counterfactual_structure) not_phi_would_phi_false:
  shows
    <∃w1. w ≤<w> w1 ∧ w1 ∈ UNIV - φ ∧ (∀w2. w2 ≤<w> w1 ⟶ w2 ∈ φ) ⟹ False>
  by blast
```

The following lemmata are dedicated to prove formula equivalences for later use.

```
lemma (in lewisian_structure) non_vacuous_woulds_equal:
  assumes
    <∃ wz. w ≤<w> wz  ∧ wz ∈ φ>
  shows
    <(∀ w1. (w ≤<w> w1 ∧ w1 ∈ φ) ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧
    (∀ w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - φ ∪ ψ))) ⟷ (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧
    (∀ w2. w2 ≤<w> w1 ⟶ w2 ∈ UNIV - φ ∪ ψ))>
proof
  assume <∀w1. w ≤<w> w1 ∧ w1 ∈ φ ⟶ (∃w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧
          (∀w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - φ ∪ ψ))>
  thus <∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀w2. w2 ≤<w> w1 ⟶ w2 ∈ UNIV - φ ∪ ψ)>
    by (metis assms meaningful_acessibility)
next
  assume <∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀w2. w2 ≤<w> w1 ⟶ w2 ∈ UNIV - φ ∪ ψ)>
  thus <∀w1. w ≤<w> w1 ∧ w1 ∈ φ ⟶ (∃w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧
       (∀w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - φ ∪ ψ))>
    by (metis linearity transitive)
qed

lemma (in preordered_counterfactual_structure) phi_stays_in_set:
  shows <∃w2. w2 ≤<w> w1 ∧ w2 ∈ (φ ∪ ψ) ∧ (∀w3. w3 ≤<w> w2 ⟶ w3 ∉ (φ ∪ ψ) ∨ w3 ∈ (UNIV
- ψ))
        ⟹ (∃w2. w2 ≤<w> w1 ∧ w2 ∈ (φ ∪ ψ) ∧ (∀w3. w3 ≤<w> w2 ⟶  w3 ∈ (UNIV - ψ)))>
  by auto

lemma (in preordered_counterfactual_structure) no_element_in_psi:
  shows <(∃w2. w2 ≤<w> w1 ∧ w2 ∈ (φ ∪ ψ) ∧ (∀w3. w3 ≤<w> w2 ⟶  w3 ∈ (UNIV - ψ))) ⟹

  (∃w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧ (∀w3. w3 ≤<w> w2 ⟶  w3 ∈ (UNIV - ψ)))> by blast

lemma (in preordered_counterfactual_structure) possible_phi:
  shows <(∃w1. w ≤<w> w1 ∧ w1 ∈ (φ ∪ ψ) ∧
```

```
  (∃w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧ (∀w3. w3 ≤<w> w2 ⟶ w3 ∉ ψ))) ⟹ (∃w1. w ≤<w> w1 ∧ w1
∈ φ)>
  by (metis meaningful_acessibility)

lemma (in preordered_counterfactual_structure) subset_generalized:
  shows<(∀ w1. w ≤<w> w1 ∧ w1 ∈ ψ ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧
        (∀ w3. w3 ≤<w> w2 ⟶ w3 ∉ ψ))) ⟶ (∀ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ⟶
      (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∪ ψ ∧ (∀ w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - (φ ∪ ψ) ∪ (UNIV
- ψ))))>
  by (metis DiffI UNIV_I Un_iff local.reflexive local.transitive meaningful_acessibility)

lemma (in preordered_counterfactual_structure) simplify_general_strong_would:
  shows <(∀ w1. w ≤<w> w1 ∧ w1 ∈ (φ ∪ ψ) ⟶
  (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ (φ ∪ ψ) ∧ (∀ w3. w3 ≤<w> w2 ⟶ w3 ∉ ψ))) ⟹
  (∀w1. w ≤<w> w1 ∧ w1 ∈ (φ ∪ ψ) ⟶ (∃w2. w2 ≤<w> w1 ∧ w2 ∈ φ))>
  by auto

lemma (in preordered_counterfactual_structure) simplify_general_might_to_at_least_as_pos:
  shows
    <¬(∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ) ∨
    (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∪ ψ ⟶
    (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ (φ ∪ ψ) ∩ φ))) ⟹
    ¬(∃w1. w ≤<w> w1 ∧ w1 ∈ ψ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ∧
    (∀w2. w2 ≤<w> w1 ∧ w2 ∈ ψ ⟶ (∃w3. w3 ≤<w> w2 ∧ w3 ∈ φ)))>
  by auto

lemma (in preordered_counterfactual_structure) extend_at_least_as_pos_to_general_might:
  shows
    <¬(∃ w1. w ≤<w> w1 ∧ w1 ∈ ψ) ∨ (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ∧ (∀ w2. w2 ≤<w> w1
∧ w2 ∈ ψ
    ⟶ (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ φ))) ⟹ ¬(∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ) ∨
    ((∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∪ ψ ⟶
    (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ (φ ∪ ψ) ∩ φ))))>
  by (auto, metis meaningful_acessibility)

lemma (in lewisian_structure) general_would_comprehension_is_would_comprehension:
  shows
    <(∀ w1. (w ≤<w> w1 ∧ w1 ∈ φ) ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧
     (∀ w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - φ ∪ ψ))) ⟷ (∀ w1. (w ≤<w> w1) ⟶ w1 ∉ φ) ∨

     (∃ w1. (w ≤<w> w1) ∧ w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ⟶ w2 ∈ UNIV - φ ∪ ψ))>
proof
  assume <(∀w1. w ≤<w> w1 ∧ w1 ∈ φ ⟶ (∃w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧
          (∀w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - φ ∪ ψ)))>
  thus <(∀w1. w ≤<w> w1 ⟶ w1 ∉ φ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∧
        (∀w2. w2 ≤<w> w1 ⟶ w2 ∈ UNIV - φ ∪ ψ))>
    by (metis meaningful_acessibility)
next
  assume <(∀w1. w ≤<w> w1 ⟶ w1 ∉ φ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∧
          (∀w2. w2 ≤<w> w1 ⟶ w2 ∈ UNIV - φ ∪ ψ))>
  thus <(∀w1. w ≤<w> w1 ∧ w1 ∈ φ ⟶ (∃w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧
        (∀w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - φ ∪ ψ)))>
    by (metis linearity transitive)
qed

lemma (in preordered_counterfactual_structure) phi_or_psi_to_phi:
  <¬(∃ w1. w ≤<w> w1 ∧ w1 ∈ ψ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ∧
```

```
  (∀w2. w2 ≤<w> w1 ∧ w2 ∈ ψ ⟶ (∃w3. w3 ≤<w> w2 ∧ w3 ∈ φ))) ⟷
  (∄w1. w ≤<w> w1 ∧ w1 ∈ ψ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀w2. w2 ≤<w> w1 ∧ w2 ∈ ψ ⟶

   (∃w3. w3 ≤<w> w2 ∧ w3 ∈ φ)))>
  by (auto, metis reflexive transitive meaningful_acessibility)

end
```

3 Definition of Lewis Counterfactual Operators

```
theory LewisOperators
  imports StructureAndAssumptions
begin
```

In this theory, we provide definitions for Lewis' counterfactual operators [Lew73] and compare these to their semantics, as intended by Lewis. The counterfactual operators in this theory and the following theories have been shallowly embedded [Ben17].

```
context preordered_counterfactual_structure begin
```

3.1 Lewis' central operators

```
definition would ::
  <'i set ⇒ 'i set ⇒ 'i set> (<_ □→L _> [70, 70] 100)
  where
    <φ □→L ψ ≡ {w. (∀ w1. (w ≤<w> w1) ⟶ w1 ∉ φ) ∨
    (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ⟶ w2 ∈ UNIV - φ ∪ ψ))}>

abbreviation (in preordered_counterfactual_structure) might ::
  <'i set ⇒ 'i set ⇒ 'i set> (infixr<◇→L>100)
  where
    <φ ◇→L ψ ≡ UNIV - (φ □→L(UNIV - ψ))>
```

3.2 Necessity and Possibility

```
abbreviation (in preordered_counterfactual_structure) necessary ::
  <'i set ⇒ 'i set> (<□ _>  [70] 80)
  where
    <□ φ ≡ (UNIV - φ) □→L φ>
```

— For the 'possible' operator, definition and check against semantics have been swapped, in order to obtain just one 'possible' operator, independent of any of the counterfactual operators.

```
abbreviation (in preordered_counterfactual_structure) possible ::
  <'i set ⇒ 'i set> (<◇ _>  [70] 80)
  where
    <◇ φ ≡ {w. ∃ w1. w ≤<w> w1 ∧ w1 ∈ φ}>
```

3.3 Four more of Lewis' operators

```
abbreviation  (in preordered_counterfactual_structure) strong_would ::
  <'i set ⇒ 'i set ⇒ 'i set> (infixr<□⇒L>100)
  where
    <φ □⇒L ψ ≡ (◇φ) ∩ (φ □→L ψ)>

abbreviation (in preordered_counterfactual_structure) weak_might ::
  <'i set ⇒ 'i set ⇒ 'i set> (infixr<◇⇒L>100)
  where
```

```
    <φ ◊⇒L ψ ≡ UNIV - (φ □⇒L (UNIV - ψ))>

abbreviation (in preordered_counterfactual_structure) more_possible ::
  <'i set ⇒ 'i set ⇒ 'i set> (infixr<≺L>100)
  where
    <φ ≺L ψ ≡ (φ ∪ ψ) □⇒L (UNIV - ψ)>

abbreviation (in preordered_counterfactual_structure) at_least_as_possible ::
  <'i set ⇒  'i set ⇒ 'i set> (infixr<≼L>100)
  where
    <φ ≼L ψ ≡ (φ ∪ ψ) ◊⇒L φ>
end
```

3.4 Validation of Lewis' operators

```
lemma (in lewisian_structure) would_instatiation:
  assumes
    <φ = {W3, W1}> and
    <ψ = {W1}> and
    <W ≤<W> W1> and
    <W ≤<W> W3> and
    <W ≤<W> W2> and
    <W2 ≤<W> W3> and
    <¬W3 ≤<W> W1> and
    <W1 ≤<W> W3> and
    <W ≠ W3 ∧ W ≠ W1 ∧ W3 ≠ W1>
  shows
    <W ∈ φ □→L ψ>
  using assms unfolding would_def by blast

context preordered_counterfactual_structure begin
```

In order to ensure that our implementation of the structure, as well as the definition of Lewis' operators within are valid, we perform a comparison for each operator between its abbreviation and a *set comprehension* depicting its intended semantics.

```
lemma necessary_follows_definition:
  shows
    <w ∈ □ φ ⟷ w ∈ {w. ∀w1. w ≤<w> w1 ⟶ w1 ∈ φ}>
  using would_def by auto

lemma possible_follows_defintion:
  shows
    <w ∈ ◊ φ ⟷ w ∈ UNIV - (φ □→L{})>
  using would_def by auto

lemma might_follows_definition:
  shows
    <w ∈ φ ◊→L ψ ⟷ w ∈  {w. (∃ w1. w ≤<w> w1  ∧ w1 ∈ φ) ∧
     (∀ w1. w ≤<w> w1 ⟶ w1 ∉ φ ∨ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∩ ψ))}>
proof
  assume <w ∈ UNIV - φ □→L(UNIV - ψ)>
  thus <w ∈ {w. (∃w1. w ≤<w> w1 ∧ w1 ∈ φ) ∧
        (∀ w1. w ≤<w> w1 ⟶ w1 ∉ φ ∨ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∩ ψ))}>
    by (simp add: would_def preordered_counterfactual_structure_axioms)
next
  assume <w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ) ∧
          (∀ w1. w ≤<w> w1 ⟶ w1 ∉ φ ∨ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∩ ψ))}>
```

```
  thus <w ∈ UNIV - φ □→L(UNIV - ψ) >
    using would_def by auto
qed

lemma strong_would_follows_definition:
  shows
    <w ∈ φ □⇒L ψ ⟷ w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧
     (∀ w2. w2 ≤<w> w1 ⟶ (w2 ∈ UNIV - φ ∪ ψ)))}>
  using would_def by auto

lemma weak_might_follows_definition:
  shows
    <w ∈ φ ◊⇒L ψ ⟷ w ∈ {w. ¬(∃ w1. w ≤<w> w1 ∧ w1 ∈ φ) ∨
     (∀ w1. w ≤<w> w1 ∧ w1 ∈ φ ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∩ ψ))}>
  using would_def by auto

lemma more_possible_follows_definition:
  shows
    <w ∈ φ ≺L ψ ⟷ w ∈ {w. ∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ⟶ w2 ∉ ψ)}>
  unfolding would_def by blast

lemma at_least_as_possible_follows_definition:
  shows
    <w ∈ φ ⪯L ψ ⟷ w ∈ {w. ∀ w1. w ≤<w> w1 ∧ w1 ∈ ψ ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ)}>
  unfolding would_def by blast

end
end
theory FinkbeinerSiberOperators
  imports
    LewisOperators
begin
```

4 Formalisation and Validation of Finkbeiner and Sibers Operators

In this theory, we provide definitions for Finkbeiner and Sibers counterfactual operators [FS23] and compare their definitions to their semantics as stated by Finkbeiner and Siber.

4.1 Finkbeiner and Sibers adapted Counterfactual Operators

```
definition (in preordered_counterfactual_structure) universal_would ::
  <'i set ⇒ 'i set ⇒ 'i set> (<_ □→FS _> [70, 70] 100)
  where
    <φ  □→FS ψ ≡ {w. (∀ w1. w1 ∉ φ) ∨
     (∀ w1. w1 ∈ φ ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧ (∀ w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV -
φ ∪ ψ)))}>

abbreviation (in preordered_counterfactual_structure) existential_might ::
  <'i set ⇒ 'i set ⇒ 'i set> (<_ ◊→FS _> [70, 70] 100)
  where
    <φ ◊→FS ψ ≡ UNIV - (φ □→FS (UNIV - ψ))>
```

4.2 Validation of the Formalisation of Finkbeiner and Sibers Operators

```
lemma (in preordered_counterfactual_structure) existential_might_follows_definition :
  shows
    <w ∈ φ ◊→FS ψ ⟷ w ∈ {w. ∃ w1. w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ⟶
     (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ φ ∩ ψ))}>
proof
  assume <w ∈ UNIV - (φ □→FS (UNIV - ψ))>
  thus <w ∈ {w. ∃ w1. w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ⟶ (∃ w3. w3 ≤<w> w2 ∧ w3
∈ φ ∩ ψ))}>
    unfolding universal_would_def using DiffD2 by auto
next
  assume <w ∈ {w. ∃ w1. w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ⟶
          (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ φ ∩ ψ))}>
  thus  <w ∈ UNIV - (φ □→FS (UNIV - ψ))>
    unfolding universal_would_def using preordered_counterfactual_structure_axioms by auto
qed
```

4.3 Equivalence to Lewis' Operators under Total Accessibility and Linearity

As shown by Finkbeiner and Siber [FS23], 'Universal Would' and 'Existential Might' are equivalent to Lewis' version of these operators, assuming total accessibility and linearity.

```
lemma (in total_accessible_lewisian_structure) would_equal_to_universal_would:
  shows <w ∈ φ □→L ψ ⟷ w ∈ φ □→FS ψ>
  using reflexive transitive total_accessibility linearity unfolding would_def universal_would_def

  by (auto, metis)

lemma (in  total_accessible_lewisian_structure) might_equal_to_existential_might:
  shows
    <w ∈ φ ◊→L ψ ⟷ w ∈ φ ◊→FS ψ>
  by (simp add: would_equal_to_universal_would)

end
theory GeneralOperators
  imports
    FinkbeinerSiberOperators
begin
```

5 A Reinterpretation of the Counterfactual Operators in the Base Structure

Finkbeiner and Siber [FS23] suggest to drop the assumption that every world is accessible from the actual world by not including inaccessible worlds in `accessibility`. We modify the counterfactual operators to depict their intended semantics in structures without total accessibility.

5.1 Defining the Operators

```
definition (in preordered_counterfactual_structure) general_would ::
  <'i set ⇒ 'i set ⇒ 'i set> (<_ □→ _> [70, 70] 100)
  where
    <φ □→ ψ ≡ {w. (∀ w1. w ≤<w> w1 ⟶  w1 ∉ φ) ∨ (∀ w1. w ≤<w> w1 ∧ w1 ∈ φ ⟶
     (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧ (∀ w3. w3 ≤<w> w2 ⟶ (w3 ∈ UNIV - φ ∪ ψ))))}>
```

```
abbreviation (in preordered_counterfactual_structure) general_might ::
  <'i set ⇒ 'i set ⇒ 'i set> (<_ ◇→ _> [70, 70] 100)
  where
    <φ ◇→ ψ ≡ UNIV - (φ □→ (UNIV - ψ))>

abbreviation  (in preordered_counterfactual_structure) general_strong_would ::
  <'i set ⇒ 'i set ⇒ 'i set> (<_ □⇒ _> [70, 70] 100)
  where
    <φ □⇒ ψ ≡ (◇φ) ∩ (φ □→ ψ)>

abbreviation  (in preordered_counterfactual_structure) general_weak_might ::
  <'i set ⇒ 'i set ⇒ 'i set> (<_ ◇⇒ _> [70, 70] 100)
  where
    <φ ◇⇒ ψ ≡ UNIV - (φ □⇒ (UNIV - ψ))>

abbreviation (in preordered_counterfactual_structure) general_more_possible ::
  <'i set ⇒ 'i set ⇒ 'i set> (infixr<≺> 100)
  where
    <φ ≺ ψ ≡ (φ ∪ ψ) □⇒ (UNIV - ψ)>

abbreviation (in preordered_counterfactual_structure) general_at_least_as_possible ::
  <'i set ⇒  'i set ⇒ 'i set> (infixr<≼> 100)
  where
    <φ ≼ ψ ≡ (φ ∪ ψ) ◇⇒ φ>
```

5.2 Validating the Definitions

```
lemma (in preordered_counterfactual_structure) general_would_instatiation:
  assumes
    <φ = {W3, W1}> and
    <ψ = {W1}> and
    <W ≤<W> W1> and
    <W ≤<W> W3> and
    <W ≤<W> W2> and
    <W2 ≤<W> W3> and
    <¬W3 ≤<W> W1> and
    <W1 ≤<W> W3> and
    <W ≠ W3 ∧ W ≠ W1 ∧ W3 ≠ W1>
  shows
    <W ∈ φ □→ ψ>
  using assms unfolding general_would_def by blast

lemma (in preordered_counterfactual_structure) possible_eq_to_def_by_gen_would:
  shows <w ∈ ◇ φ ⟷ w ∈ UNIV - (φ □→ {})>
  unfolding general_would_def by fastforce

lemma (in preordered_counterfactual_structure) general_might_follows_definition:
  shows <w ∈ φ ◇→ ψ ⟷ w ∈ {w. ∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧
         (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ⟶ (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ φ ∩ ψ))}>
  unfolding general_would_def using Int_iff UNIV_I Un_iff mem_Collect_eq
    preordered_counterfactual_structure_axioms by auto

lemma (in preordered_counterfactual_structure) general_strong_would_follows_definition:
  shows <w ∈ φ □⇒ ψ ⟷ w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ) ∧
        (∀ w1. (w ≤<w> w1 ∧ w1 ∈ φ) ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧
        (∀ w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - φ ∪ ψ)))}>
  using general_would_def preordered_counterfactual_structure_axioms by auto
```

```
lemma (in preordered_counterfactual_structure) general_weak_might_follows_definition:
  shows <w ∈ φ ◊⇒ ψ ⟷ w ∈ {w. ¬(∃ w1. w ≤<w> w1 ∧ w1 ∈ φ) ∨
         (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ⟶
         (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ φ ∩ ψ)))}>
proof
  assume <w ∈ UNIV - ((◊ φ) ∩ (φ □→ (UNIV - ψ)))>
  thus <w ∈ {w. ¬(∃ w1. w ≤<w> w1 ∧ w1 ∈ φ) ∨ (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧
  (∀w2. w2 ≤<w> w1 ∧ w2 ∈ φ ⟶ (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ φ ∩ ψ)))}>
    using general_would_def preordered_counterfactual_structure_axioms by auto
next
  assume <w ∈ {w. ¬(∃ w1. w ≤<w> w1 ∧ w1 ∈ φ) ∨ (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧
          (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ⟶ (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ φ ∩ ψ)))}>
  thus <w ∈ UNIV - ((◊ φ) ∩ (φ □→ (UNIV - ψ)))>
    using general_would_def preordered_counterfactual_structure_axioms by auto
qed
```

While we are able to deliver *set comprehensions* capturing the semantics for 'General More Possible' as well as 'General At least As Possible', the difference from the *set comprehensions* for their Lewisian counterparts suggests that their semantics are strongly tied to the linearity assumption.

```
lemma (in preordered_counterfactual_structure) general_more_possible_follows_definition:
  shows <w ∈ φ ≺ ψ ⟷ w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ) ∧ (∀ w1. w ≤<w> w1 ∧ w1 ∈
ψ
  ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧ (∀ w3. w3 ≤<w> w2 ⟶ w3 ∉ ψ)))}>
proof
  assume <w ∈ (◊ (φ ∪ ψ)) ∩ ((φ ∪ ψ) □→ (UNIV - ψ))>
  hence <w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ) ∧ (∀ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ⟶
  (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∪ ψ ∧ (∀ w3. w3 ≤<w> w2 ⟶ w3 ∉ φ ∪ ψ ∨ w3 ∈ UNIV - ψ)))}>
    using general_would_def preordered_counterfactual_structure_axioms by auto
  hence <w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ) ∧ (∀ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ⟶
  (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ (φ ∪ ψ) ∧ (∀ w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - ψ)))}>
      using phi_stays_in_set by auto
  hence <w ∈ {w. (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ) ∧ (∀ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ⟶
  (∃w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧ (∀w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - ψ)))}>
      using no_element_in_psi by auto
  hence <w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ) ∧ (∀w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ⟶
  (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧ (∀ w3. w3 ≤<w> w2 ⟶ w3 ∉ ψ)))}> by auto
  thus <w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ) ∧ (∀ w1. w ≤<w> w1 ∧ w1 ∈ ψ ⟶
  (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧ (∀ w3. w3 ≤<w> w2 ⟶ w3 ∉ ψ)))}> using possible_phi by
blast
next
  assume <w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ) ∧
          (∀ w1. w ≤<w> w1 ∧ w1 ∈ ψ ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧
          (∀ w3. w3 ≤<w> w2 ⟶ w3 ∉ ψ)))}>
  hence <w ∈ {w. (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ) ∧
        (∀ w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ⟶ (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∪ ψ ∧
        (∀ w3. w3 ≤<w> w2 ⟶ w3 ∈ UNIV - (φ ∪ ψ) ∪ (UNIV - ψ))))}>
      using subset_generalized by blast
    thus <w ∈ (◊ (φ ∪ ψ)) ∩ ((φ ∪ ψ) □→ (UNIV - ψ))>
      using general_strong_would_follows_definition by presburger
qed

lemma (in preordered_counterfactual_structure) weak_might_def_to_set_comprehension:
  shows <w ∈ UNIV - ((◊ φ) ∩ (φ □→ (UNIV - ψ))) ⟷ w ∈ {w. ¬(∃ w1. w ≤<w> w1 ∧ w1 ∈
φ) ∨
         (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ⟶
```

```
        (∃w3. w3 ≤<w> w2 ∧ w3 ∈ φ ∩ ψ))}>
  using general_weak_might_follows_definition by presburger

lemma (in preordered_counterfactual_structure) general_at_least_as_possible_follows_definition:
  shows
    <w ∈ φ ≼ ψ ⟷ w ∈ {w. ¬(∃ w1. w ≤<w> w1 ∧ w1 ∈ ψ) ∨
    (∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ ψ ⟶ (∃ w3. w3 ≤<w> w2 ∧ w3
∈ φ)))}>
proof
  assume <w ∈ UNIV - (◇ (φ ∪ ψ)) ∩ ((φ ∪ ψ) □→ (UNIV - φ))>
  hence <w ∈ {w. ¬(∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ) ∨
  (∃w1. w ≤<w> w1 ∧ w1 ∈ (φ ∪ ψ) ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∪ ψ ⟶
  (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ (φ ∪ ψ) ∩ φ)))}> using weak_might_def_to_set_comprehension by
meson
  hence <w ∈ {w. ¬(∃ w1. w ≤<w> w1 ∧ w1 ∈ ψ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ∧
  (∀w2. w2 ≤<w> w1 ∧ w2 ∈ ψ ⟶ (∃w3. w3 ≤<w> w2 ∧ w3 ∈ φ)))}>
    using simplify_general_might_to_at_least_as_pos by auto
  thus <w ∈ {w. (∄w1. w ≤<w> w1 ∧ w1 ∈ ψ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∧
        (∀w2. w2 ≤<w> w1 ∧ w2 ∈ ψ ⟶ (∃w3. w3 ≤<w> w2 ∧ w3 ∈ φ)))}>
    using phi_or_psi_to_phi by force
next
  assume <w ∈ {w. ¬(∃w1. w ≤<w> w1 ∧ w1 ∈ ψ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∧
  (∀w2. w2 ≤<w> w1 ∧ w2 ∈ ψ ⟶ (∃w3. w3 ≤<w> w2 ∧ w3 ∈ φ)))}>
  hence <w ∈ {w. ¬(∃w1. w ≤<w> w1 ∧ w1 ∈ ψ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ ∧
  (∀w2. w2 ≤<w> w1 ∧ w2 ∈ ψ ⟶ (∃w3. w3 ≤<w> w2 ∧ w3 ∈ φ)))}> using phi_or_psi_to_phi
by force
  hence <w ∈ {w. ¬(∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ) ∨ (∃w1. w ≤<w> w1 ∧ w1 ∈ φ ∪ ψ
  ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∪ ψ ⟶ (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ (φ ∪ ψ) ∩ φ)))}>
    using extend_at_least_as_pos_to_general_might by auto
  thus <w ∈ UNIV - ((◇ (φ ∪ ψ)) ∩ ((φ ∪ ψ) □→ (UNIV - φ)))>
    using weak_might_def_to_set_comprehension by presburger
qed
```

5.3 Comparison to Lewis Operators

Finally, we are able to show that the generalised operators meet their Lewisian counterparts under the linearity assumption. The same holds for Finkbeiner and Sibers operators under the total accessibility assumption. For any of the Lewisian operators (except 'At Least As Possible') it can be shown by counterexample, that they miss their (likely) intended semantics being evaluated under the base assumptions. The same holds for Finkbeiner and Sibers operators.

```
lemma (in lewisian_structure) would_equivalent_to_general_would:
  shows
    <w ∈ φ □→L ψ ⟷ w ∈ φ □→ ψ>
  unfolding would_def general_would_def using general_would_comprehension_is_would_comprehension

  by blast
```

— This lemma resembles Finkbeiners and Sibers findings regarding the difference between 'Would' and 'Universal Would'

```
lemma (in preordered_counterfactual_structure) would_not_wide_enough:
  assumes
    <φ = {W1, W2}> and
    <ψ = {W2}> and
    <W ≤<W> W2> and
    <W ≤<W> W1> and
    <¬ W1 ≤<W> W2> and
```

```
    <¬ W2 ≤<W> W1> and
    <W ≠ W1 ∧ W ≠ W2 ∧ W1 ≠ W2>
  shows
    <W ∉ φ □→ ψ ∧ W ∈ φ □→L ψ>
proof
    have <(∀ w1. W ≤<W> w1 ⟶ w1 ∉ φ) ∨ (∀ w1. W ≤<W> w1 ∧ w1 ∈ φ ⟶
    (∃ w2. w2 ≤<W> w1 ∧ w2 ∈ φ ∧ (∀ w3. w3 ≤<W> w2 ⟶ w3 ∈ UNIV - φ ∪ ψ))) ⟹ False>
      using assms(1) assms(2) assms(4) assms(6) assms(7) by blast
    thus <W ∉ φ □→ ψ> using general_would_def by force
next
    have <(∀ w1. W ≤<W> w1 ⟶ w1 ∉ φ) ∨
    (∃ w1. W ≤<W> w1 ∧ w1 ∈ φ ∧ (∀ w2. w2 ≤<W> w1 ⟶ w2 ∈ UNIV - φ ∪ ψ))>
      using assms(1) assms(2) assms(3) assms(5) by blast
    thus <W ∈ φ □→L ψ> by (simp add: would_def)
qed

lemma (in lewisian_structure) might_equivalent_to_general_might:
  shows
    <w ∈ φ ◊→L ψ ⟷ w ∈ φ ◊→ ψ>
  by (simp add: would_equivalent_to_general_would)
```

— This lemma resembles Finkbeiners and Sibers findings regarding the difference between 'Might' and 'Existential Might'

```
lemma (in preordered_counterfactual_structure) might_not_narrow_enough:
  assumes
    <φ = {W1, W2}> and
    <ψ = {W1}> and
    <W ≤<W> W2> and
    <W ≤<W> W1> and
    <¬ W1 ≤<W> W2> and
    <¬ W2 ≤<W> W1> and
    <W ≠ W1 ∧ W ≠ W2 ∧ W1 ≠ W2>
  shows
    <W ∉ φ ◊→L ψ ∧ W ∈ φ ◊→ ψ>
proof
  show <W ∉ UNIV - φ □→L(UNIV - ψ)>
    unfolding would_def using assms(1) assms(2) assms(3) assms(5) by blast
next
  have  <W ∈ {w. ∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ⟶
         (∃ w3. w3 ≤<w> w2 ∧ w3 ∈ φ ∩ ψ))}> using assms(1) assms(2) assms(4) assms(6) by
blast
  thus <W ∈ UNIV - φ □→ (UNIV - ψ)> using general_might_follows_definition by presburger
qed

lemma (in lewisian_structure) strong_would_equivalent_to_general_strong_would:
  shows
    <w ∈ φ □⇒L ψ ⟷ w ∈ φ □⇒ ψ>
  by (simp add: would_equivalent_to_general_would)

lemma (in preordered_counterfactual_structure) strong_would_not_wide_enough:
  assumes
    <φ = {W1, W2}> and
    <ψ = {W2, W}> and
    <W ≤<W> W2> and
    <W ≤<W> W1> and
    <¬ W1 ≤<W> W2> and
    <¬ W2 ≤<W> W1> and
```

```
    ‹W ≠ W1 ∧ W ≠ W2 ∧ W1 ≠ W2›
  shows
    ‹W ∉ φ □⇒ ψ ∧ W ∈ φ □⇒L ψ›
proof
  show ‹W ∈ (◇ φ) ∩ (φ □→L ψ)›
    using assms would_def by auto
next
  show ‹W ∉ (◇ φ) ∩ (φ □→ ψ)›
    using assms general_would_def preordered_counterfactual_structure_axioms by auto
qed

lemma (in lewisian_structure) weak_might_equivalent_to_general_weak_might:
  shows
    ‹w ∈ φ ◇⇒L ψ ⟷ w ∈ φ ◇⇒ ψ›
  by (simp add: would_equivalent_to_general_would)

lemma (in preordered_counterfactual_structure) weak_might_not_narrow_enough:
  assumes
    ‹φ = {W1, W2}› and
    ‹ψ = {W1}› and
    ‹W ≤<W> W2› and
    ‹W ≤<W> W1› and
    ‹¬ W1 ≤<W> W2› and
    ‹¬ W2 ≤<W> W1› and
    ‹W ≠ W1 ∧ W ≠ W2 ∧ W1 ≠ W2›
  shows
    ‹W ∉ φ ◇⇒L ψ ∧ W ∈ φ ◇⇒ ψ›
proof
  show ‹W ∉ UNIV - (◇ φ) ∩ (φ □→L (UNIV - ψ))›
    unfolding would_def using assms(1) assms(2) assms(3) assms(5) by blast
  show ‹W ∈ UNIV - (◇ φ) ∩ (φ □→ (UNIV - ψ))›
    using assms might_not_narrow_enough by auto
qed

lemma (in lewisian_structure) more_possible_equivalent_to_general_more_possible:
  shows
    ‹w ∈ φ ≺L ψ ⟷ w ∈ φ ≺ ψ›
  using strong_would_equivalent_to_general_strong_would by presburger

lemma (in preordered_counterfactual_structure) more_possible_not_wide_enough:
  assumes
    ‹φ = {W2}› and
    ‹ψ = {W1}› and
    ‹W ≤<W> W2› and
    ‹W ≤<W> W1› and
    ‹¬ W1 ≤<W> W2› and
    ‹¬ W2 ≤<W> W1› and
    ‹W ≠ W1 ∧ W ≠ W2 ∧ W1 ≠ W2›
  shows ‹W ∈ φ ≺L ψ ∧ W ∉ φ ≺ ψ›
proof
  have ‹W ∈ {w. ∃ w1. w ≤<w> w1 ∧ w1 ∈ φ ∧ (∀ w2. w2 ≤<w> w1 ⟶ w2 ∉ ψ)}›
      using assms(1) assms(2) assms(3) assms(5) by blast
  thus ‹W ∈ (◇ (φ ∪ ψ)) ∩ ((φ ∪ ψ) □→L (UNIV - ψ))›
      using assms might_not_narrow_enough preordered_counterfactual_structure_axioms by auto
next
  show ‹W ∉ (◇ (φ ∪ ψ)) ∩ ((φ ∪ ψ) □→ (UNIV - ψ))›
    using assms might_not_narrow_enough by auto
```

```
qed

lemma (in lewisian_structure) at_least_as_possible_equivalent_to__general_at_least_as_possible:
  shows
    <w ∈ φ ≼ ψ ⟷ w ∈ φ ≼_L ψ>
  using weak_might_equivalent_to_general_weak_might by presburger

lemma (in preordered_counterfactual_structure) at_least_as_possible_not_narrow_enough:
  assumes
    <φ = {W2}> and
    <ψ = {W1, W2}> and
    <W ≤<W> W2> and
    <W ≤<W> W1> and
    <¬ W1 ≤<W> W2> and
    <¬ W2 ≤<W> W1> and
    <W ≠ W1 ∧ W ≠ W2 ∧ W1 ≠ W2>
  shows
    <W ∈ φ ≼ ψ ∧ W ∉ φ ≼_L ψ>
  using assms general_at_least_as_possible_follows_definition unfolding would_def by blast
```

5.4 Comparison to Finkbeiner and Sibers Operators

```
lemma (in finkbeiner_siber_structure) universal_would_equivalent_to_general_would:
  shows
    <w ∈ φ □→_FS ψ ⟷ w ∈ φ □→ ψ>
  by (simp add: general_would_def universal_would_def preordered_counterfactual_structure_axioms

      total_accessibility)

lemma (in preordered_counterfactual_structure) universal_would_considering_inaccessible_worlds:
  assumes
    <φ = {W1}>  and
    <ψ = {}> and
    <¬ W ≤<W> W1>
  shows
    <W ∉ φ □→_FS ψ ∧ W ∈ φ □→ ψ>
  using assms unfolding universal_would_def general_would_def by blast

lemma (in finkbeiner_siber_structure) existential_might_equivalent_to_general_might:
  shows
    <w ∈ φ ◇→_FS ψ ⟷ w ∈ φ ◇→ ψ>
  by (simp add: universal_would_equivalent_to_general_would)

lemma (in preordered_counterfactual_structure) existential_might_considering_inaccessible_worlds:
  assumes
    <φ = {W1}>  and
    <ψ = {W1}> and
    <¬ W ≤<W> W1>
  shows
    <W ∈ φ ◇→_FS ψ ∧ W ∉ φ ◇→ ψ>
  by (simp add: assms general_would_def universal_would_def)

end
theory CTLStar

imports
  GeneralOperators
```

```
  Main "HOL-Library.Omega_Words_Fun"

begin
```

6 Relation between Expressive Power of CTL* and Counterfactual Operators

This theory gives a comparison between the expressive power of CTL* and the expressive power of the 'General Would' operator. To this end, we prove that CTL* can not distinguish bisimilar processes.

```
context world_dependent_kripke_structure begin
```

The path definition is taken from the Isabelle/HOL Tutorial [NPW02]. Usage of ω-words is inspired by Sickert [Sic16]. To model the omega words, we use the `word` datatype from the standard library of Isabelle/HOL [BNO+23].

```
definition is_path :: <'i ⇒ 'i word ⇒ bool>
  where <is_path w π ≡  π 0 = w ∧ (∀ n. π n ≤<π n> π (Suc n))>
```

Syntax and Semantics of the logic CTL* as in [BK08], concrete implementation inspired by [Sic16].

```
datatype 'a state_formula =
  Prop_state 'a ("prop'(_')")
  | True_state  ("true")
  | And_state <'a state_formula> <'a state_formula> (<_ && _ > [82,82] 81)
  | Neg_state <'a state_formula> (<~~ _ > [100])
  | Exists_state <'a path_formula> (<EE _>)
  and 'a path_formula =
  Is <'a state_formula>
  | Neg_path <'a path_formula> (<neg _ > [100])
  | And_path <'a path_formula> <'a path_formula> (<_ and _> [82,82] 81)
  | Next_path <'a path_formula> (<X _>)
  | Until_path <'a path_formula> <'a path_formula> (<_ UU _> [82,82] 81)
```

— Truth conditions resemble [Sic16], except for `EE` and `Is`.

```
primrec
  mc_state_formula :: <'i ⇒ 'ap state_formula ⇒ bool> (<_ ⊨s _> [82,82] 81)  and
  mc_path_formula :: <'i word ⇒ 'ap path_formula ⇒ bool> (<_ ⊨p _> [82,82] 81)
  where
  <w ⊨s prop(a) = (a ∈ (labeling w))>
| <w ⊨s true = True>
| <w ⊨s (~~ φ) = (¬(w ⊨s φ))>
| <w ⊨s (φ && ψ) = ((w ⊨s φ) ∧ (w ⊨s ψ))>
| <w ⊨s (EE φ) = (∃ π. is_path w π ∧ π ⊨p φ)>
| <π ⊨p(Is φ) = (π 0) ⊨s φ>
| <π ⊨p neg φ = (¬π ⊨p φ)>
| <π ⊨p (φ and ψ) = (π ⊨p φ ∧ π ⊨p ψ)>
| <π ⊨p(X φ) = ((suffix 1 π) ⊨p φ)>
| <π ⊨p(φ UU ψ) = (∃i. ((suffix i π) ⊨p ψ) ∧ (∀j<i. ((suffix j π) ⊨p φ)))>
```

— Bisimulation definition inspired by Baier and Katoen [BK08] and by Pohlmann [Poh21].

```
definition bisimulation :: <('i ⇒ 'i ⇒ bool) ⇒ bool>
  where <bisimulation R ≡ ∀ w v. R w v ⟶
    (labeling w = labeling v) ∧
    (∀ w'. w ≤<w> w' ⟶ (∃ v'. v ≤<v> v' ∧ R w' v')) ∧
    (∀ v'. v ≤<v> v' ⟶ (∃ w'. w ≤<w> w' ∧ R w' v'))>
```

— Definition taken from [Poh21].

```
definition bisimilar :: <'i ⇒ 'i ⇒ bool> (<_ ↔ _> [70, 70] 70)
  where <w ↔ v ≡ ∃ R. bisimulation R ∧ R w v>
```

— Lemma and its proof also taken from [Poh21].

```
lemma bisim_sym:
  assumes <p ↔ q>
  shows <q ↔ p>
  using assms unfolding bisimilar_def
proof
  fix R
  assume <bisimulation R ∧ R p q>
  let ?R' = <λ a b. R b a>
  have <bisimulation ?R' ∧ ?R' q p> using bisimulation_def <bisimulation R ∧ R p q> by presburger
  thus <∃R. bisimulation R ∧ R q p> by auto
qed
```

For the following lemma, we assume that the path lifting lemma (Lemma 7.5 in [BK08]) holds. It states that for any two worlds w_1, w_2 if w_1 is bisimilar to w_2, then for a path π_1 departing from w_1, there exists a path π_2 departing from w_2, such that for all $n \in \mathbb{N}$ it holds that the world at the nth position in π_1 is bisimilar to the world at the nth position in π_2. Proving it is out of scope for this thesis.

```
lemma existential_path_non_dist:
  assumes
    <⋀ π1 π2. (∀i. π1 i ↔ π2 i) ⟶ π1 ⊨ₚ x = π2 ⊨ₚ x> and
    path_lifting: <⋀ w v π1. ⟦bisimilar w v; is_path w π1⟧ ⟹
      (∃ π2. is_path v π2 ∧ (∀ i. π1 i ↔ π2 i))>
  shows <w ↔ v ⟹  w ⊨ₛ (EE x) ⟹ v ⊨ₛ (EE x)>
proof -
  assume bisim: <w ↔ v>
  assume <w ⊨ₛ EE x>
  then obtain π1 where pi1_path: <is_path w π1 ∧ (π1  ⊨ₚ x)>
    using mc_state_formula.simps(5) by blast
  then obtain π2 where stepwise_bisim_pi2_path: <is_path v π2 ∧ (∀ i. (∀ i. π1 i ↔ π2 i))>
    using bisim assms path_lifting by blast
  hence <π1 ⊨ₚ x = π2 ⊨ₚ x> using assms by blast
  thus <v ⊨ₛ (EE x)> using stepwise_bisim_pi2_path pi1_path by auto
qed
```

This is an Isabelle implementation of the proof that bisimulation is finer than CTL*-equivalence, as shown by Baier and Katoen [BK08] in Lemma 7.26.

```
lemma bisimulation_finer_than_ctls:
  fixes
    Φ :: <'ap state_formula> and
    φ :: <'ap path_formula>
  assumes
    path_lifting: <⋀ w v π1. ⟦bisimilar w v; is_path w π1⟧ ⟹
      (∃ π2. is_path v π2 ∧ (∀ i. π1 i ↔ π2 i))>
  shows
    <⋀ w v. w ↔ v ⟶ (w  ⊨ₛ Φ ⟷ v ⊨ₛ Φ)>
    and <⋀ π1 π2. (∀ i. π1 i ↔ π2 i) ⟶ π1 ⊨ₚ φ ⟷ π2 ⊨ₚ φ>
proof (induct Φ and φ)
  case (Prop_state x)
  then show ?case using bisimilar_def bisimulation_def by auto
next
  case True_state
```

```
  then show ?case by simp
next
  case (And_state x1 x2)
  then show ?case by force
next
  case (Neg_state x)
  then show ?case using mc_state_formula.simps(3) by blast
next
  case (Exists_state x)
  show ?case using bisim_sym path_lifting
    using Exists_state existential_path_non_dist by blast
next
  case (Is x)
  then show ?case using mc_path_formula.simps(1) by blast
next
  case (Neg_path x)
  then show ?case using mc_path_formula.simps(2) by blast
next
  case (And_path x1 x2)
  then show ?case by simp
next
  thm mc_path_formula.simps(4)
  case (Next_path x)
  then show ?case by force
next
  case (Until_path x1 x2)
  then show ?case by (metis mc_path_formula.simps(5) suffix_nth)
qed

end
```

6.1 Example Counterfactual Structure containing two 'General Would' distinguishable Worlds

```
datatype world = W_true | W_false | W1 | W2 | W3
datatype ap = B | F

fun cf_accessibility :: <world ⇒ world ⇒ world ⇒ bool> ("_ ≲<_> _" [70, 70, 70] 80)
  where
    <cf_accessibility W_true _ W_true = True>
  | <cf_accessibility W_false _ W_false = True>
  | <cf_accessibility W1 _ W1 = True>
  | <cf_accessibility W2 _ W2 = True>
  | <cf_accessibility W3 _ W3 = True>
  | <cf_accessibility W_true W_true W1 = True>
  | <cf_accessibility W_true W_true W2 = True>
  | <cf_accessibility W_false W_false W1 = True>
  | <cf_accessibility W_false W_false W2 = True>
  | <cf_accessibility W_false W_false W3 = True>
  | <cf_accessibility W1 W_true W2 = True>
  | <cf_accessibility W1 W_false W2 = True>
  | <cf_accessibility W1 W1 W2 = True>
  | <cf_accessibility W_true _ _ = False>
  | <cf_accessibility W_false _ _ = False>
  | <cf_accessibility W1 _ _ = False>
  | <cf_accessibility W2 _ _ = False>
  | <cf_accessibility W3 _ _ = False>
```

```
primrec atomic_truth :: <world ⇒ ap set> (<ℒ _> [80])
  where
    <atomic_truth W_true = {}>
  | <atomic_truth W_false = {}>
  | <atomic_truth W1 = {B, F}>
  | <atomic_truth W2 = {B}>
  | <atomic_truth W3 = {B}>

locale general_would_distinguishing_wt_wf =
  preordered_counterfactual_structure <atomic_truth> <cf_accessibility>

begin

notation general_would (<_ □→ _> [70, 70] 100)

lemma (in preordered_counterfactual_structure) simplify_general_would:
  shows <{w. (∀ w1. (w ≤<w> w1 ∧ w1 ∈ φ) ⟶
  (∃ w2. w2 ≤<w> w1 ∧ w2 ∈ φ ∧ (∀ w3. w3 ≤<w> w2 ⟶ (w3 ∈ UNIV - φ ∪ ψ))))} = φ □→
ψ>
  using general_would_def by auto

lemma phi_semantics:
  shows <{w. B ∈ ℒ w} = {W1, W2, W3}>
proof
  show <{w. B ∈ ℒ w} ⊆ {W1, W2, W3}>
  proof (rule ccontr)
    assume <¬ {w. B ∈ ℒ w} ⊆ {W1, W2, W3}>
    hence <W_true ∈ {w. B ∈ ℒ w} ∨ W_false ∈ {w. B ∈ ℒ w}>
      by (auto, metis atomic_truth.simps(1) atomic_truth.simps(2) empty_iff world.exhaust)
    thus <False> by simp
  qed
next
  show <{W1, W2, W3} ⊆ {w. B ∈ ℒ w}> by simp
qed

lemma psi_semantics:
  shows <{w. F ∈ ℒ w} = {W1}>
proof
  show <{w. F ∈  ℒ w} ⊆ {W1}>
  proof (rule ccontr)
    assume <¬ {w. F ∈ ℒ w} ⊆ {W1}>
    hence <W_true ∈ {w. F ∈ ℒ w} ∨ W_false∈ {w. F ∈ ℒ w} ∨ W2 ∈ {w. F ∈ ℒ w} ∨ W3 ∈ {w.
F ∈ ℒ w}>
      by (auto, metis ap.simps(2) atomic_truth.simps(1) atomic_truth.simps(2) atomic_truth.simps(4)

          atomic_truth.simps(5) empty_iff singleton_iff world.exhaust)
    thus <False> by simp
  qed
next
  show <{W1} ⊆ {w. F ∈ ℒ w}> by simp
qed
```

The following two lemmata show that W_{true} and W_{false} are distinguishable by 'General Would'.

```
lemma W_true_in_gen_would:
  shows <W_true ∈ {w. B ∈ ℒ w} □→ {w. F ∈ ℒ w}>
```

```
proof -
  have <(∀ w1. (W_true ≲<W_true> w1 ∧ w1 ∈ {W1,W2,W3}) ⟶
        (∃ w2. w2 ≲<W_true> w1 ∧ w2 ∈ {W1,W2,W3} ∧ (∀ w3. w3 ≲<W_true> w2 ⟶
        (w3 ∈ UNIV - {W1,W2,W3} ∪ {W1}))))>
    by (metis (no_types, opaque_lifting) Diff_iff cf_accessibility.simps(11)
        cf_accessibility.simps(23) cf_accessibility.simps(44) emptyE insertE insertI1 insertI2

        insert_is_Un iso_tuple_UNIV_I local.reflexive meaningful_acessibility sup.commute)
  hence <W_true ∈ {w. (∀ w1. (w ≲<w> w1 ∧ w1 ∈ {W1,W2,W3}) ⟶
  (∃ w2. w2 ≲<w> w1 ∧ w2 ∈ {W1,W2,W3} ∧ (∀ w3. w3 ≲<w> w2 ⟶ (w3 ∈ UNIV - {W1,W2,W3} ∪
{W1})))))}>
    by blast
  thus <W_true ∈ {w. B ∈ 𝓛 w}  □→ {w. F ∈ 𝓛 w}>
    using simplify_general_would phi_semantics psi_semantics by metis
qed

lemma W_false_not_in_gen_would:
  shows <W_false ∉ {w. B ∈ 𝓛 w} □→ {w. F ∈ 𝓛 w}>
proof -
  have  <¬(∀ w1. (W_false ≲<W_false> w1 ∧ w1 ∈ {W1,W2,W3}) ⟶
  (∃ w2. w2 ≲<W_false> w1 ∧ w2 ∈ {W1,W2,W3} ∧ (∀ w3. w3 ≲<W_false> w2 ⟶
  (w3 ∈ UNIV - {W1,W2,W3} ∪ {W1}))))>
    by (metis Diff_iff Un_iff cf_accessibility.simps(10) cf_accessibility.simps(41) insertCI

        local.reflexive singleton_iff)
  hence <W_false ∉ {w. (∀ w1. (w ≲<w> w1 ∧ w1 ∈ {W1,W2,W3}) ⟶
  (∃ w2. w2 ≲<w> w1 ∧ w2 ∈ {W1,W2,W3} ∧ (∀ w3. w3 ≲<w> w2 ⟶ (w3 ∈ UNIV - {W1,W2,W3} ∪
{W1})))))}>
    by blast
  thus <W_false ∉ {w. B ∈ 𝓛 w}  □→ {w. F ∈ 𝓛 w}>
    using simplify_general_would phi_semantics psi_semantics by metis
qed

end
```

6.2 W_{true} and W_{false} are not distinguishable by CTL*

In this part, we proof that CTL* can not distinguish between W_{true} and W_{false}. To this end, we examine a version of the `cf_accessiblity` function, translated for CTL*.

```
fun ctls_accessibility :: <world ⇒ world ⇒ world ⇒ bool> ("_ ≲<_> _" [70, 70, 70] 80)

  where
    <ctls_accessibility W_true _ W_true = True>
  | <ctls_accessibility W_false _ W_false = True>
  | <ctls_accessibility W1 _ W1 = True>
  | <ctls_accessibility W2 _ W2 = True>
  | <ctls_accessibility W3 _ W3 = True>
  | <ctls_accessibility W_true _ W1 = True>
  | <ctls_accessibility W_true _ W2 = True>
  | <ctls_accessibility W_false _ W1 = True>
  | <ctls_accessibility W_false _ W2 = True>
  | <ctls_accessibility W_false _ W3 = True>
  | <ctls_accessibility W1 _ W2 = True>
  | <ctls_accessibility W_true _ _ = False>
  | <ctls_accessibility W_false _ _ = False>
  | <ctls_accessibility W1 _ _ = False>
  | <ctls_accessibility W2 _ _ = False>
```

```
  | <ctls_accessibility W3 _ _ = False>

locale ctls_not_distinguishing_wt_wf =
  world_dependent_kripke_structure <atomic_truth> <ctls_accessibility>

begin

notation mc_state_formula (<_ ⊨ₛ _> [82,82] 81)
notation bisimilar (<_ ↔ _> [70, 70] 70)

fun bisim_r :: <world ⇒ world ⇒ bool>
  where
    <bisim_r W_true W_true = True>
  | <bisim_r W_false W_false = True>
  | <bisim_r W1 W1 = True>
  | <bisim_r W2 W2 = True>
  | <bisim_r W3 W3 = True>
  | <bisim_r W2 W3 = True>
  | <bisim_r W3 W2 = True>
  | <bisim_r W_true W_false = True>
  | <bisim_r W_false W_true = True>
  | <bisim_r W_true _ = False>
  | <bisim_r W_false _ = False>
  | <bisim_r W1 _ = False>
  | <bisim_r W2 _ = False>
  | <bisim_r W3 _ = False>

lemma bism_r_has_eq_labels:
  shows <∀w v. bisim_r w v ⟶ ℒ w = ℒ v>
proof -
  have <ℒ W_true = ℒ W_false> using atomic_truth.simps by blast
  moreover have <ℒ W2 = ℒ W3> using atomic_truth.simps by blast
  ultimately show Atom_truths_sim: <∀w v. bisim_r w v ⟶ ℒ w = ℒ v>
    using atomic_truth.simps apply auto using bisim_r.elims(2) apply force
    using bisim_r.elims(2) by force
qed
```

A block of lemmata proving properties of bisimulation function for later use.

```
lemma w_true_reaches_w1_w2_w_true:
  shows <W_true ≲<W_true> w ⟶ w = W1 ∨ w = W2 ∨ w = W_true>
  using ctls_accessibility.elims(2) by blast

lemma w_false_reaches_w1_w2_w3_w_false:
  shows <W_false ≲<W_false> w ⟶ w = W1 ∨ w = W2 ∨ w = W3 ∨ w = W_false>
  using ctls_accessibility.elims(2) by blast

lemma w1_reaches_w1_w2:
  shows <W1 ≲<W1> w ⟶ w = W1 ∨ w = W2>
  using ctls_accessibility.elims(2) by blast

lemma w2_reaches_w2:
  shows <W2 ≲<W2> w ⟶ w = W2>
  using ctls_accessibility.elims(2) by blast

lemma w3_reaches_w3:
  shows <W3 ≲<W3> w ⟶ w = W3>
  using ctls_accessibility.elims(2) by blast
```

```
lemma bisim_r_contains_lhs:
  shows <bisim_r w v ⟶ (w = W1) ∨ (w = W2) ∨ (w = W3) ∨ (w = W_false) ∨ (w = W_true)>
  using bisim_r.simps world.exhaust by blast

lemma bisim_r_contains_rhs:
  shows <bisim_r w v ⟶ (v = W1) ∨ (v = W2) ∨ (v = W3) ∨ (v = W_false) ∨ (v = W_true)>
  using bisim_r.simps world.exhaust by blast

lemma bisim_constraints_meant_for_bisim_worlds_lhs:
  shows <∀ v. P W1 v  ∧ P W2 v ∧ P W3 v ∧ P W_false v ∧ P W_true v ⟹
         ∀ w v. bisim_r w v ⟶ P w v>
  using bisim_r_contains_lhs by blast

lemma bisim_constraints_meant_for_bisim_worlds_rhs:
  shows <∀ w. P w W1  ∧ P w W2 ∧ P w W3 ∧ P w W_false ∧ P w W_true ⟹
         ∀ w v. bisim_r w v ⟶ P w v>
  using bisim_r_contains_rhs by blast

lemma bisim_r_is_bisimulation_forth:
  shows <(bisim_r w v) ⟶ (∀w'. w ≲<w> w' ⟶ (∃v'. v ≲<v> v' ∧ bisim_r w' v'))>
proof
  assume <bisim_r w v>
  show <∀w'. w ≲<w> w' ⟶ (∃v'. v ≲<v> v' ∧ bisim_r w' v')>
  proof (cases w)
    case W_true
    then show ?thesis
      using <bisim_r w v> bisim_r.simps(10) bisim_r.simps(11) bisim_r.simps(12) bisim_r.simps(3)

        bisim_r.simps(4) bisim_r_contains_rhs ctls_accessibility.simps(8)
        ctls_accessibility.simps(9) w_true_reaches_w1_w2_w_true by blast
  next
    case W_false
    then show ?thesis
      using <bisim_r w v> bisim_r.simps(13) bisim_r.simps(14) bisim_r.simps(15) bisim_r.simps(3)

        bisim_r.simps(4) bisim_r.simps(5) bisim_r.simps(7) bisim_r_contains_rhs
        ctls_accessibility.simps(6) ctls_accessibility.simps(7) w_false_reaches_w1_w2_w3_w_false

        by blast
  next
    case W1
    then show ?thesis
      using <bisim_r w v> bisim_r.simps(16) bisim_r.simps(17) bisim_r.simps(19) bisim_r.simps(4)

        bisim_r_contains_rhs w1_reaches_w1_w2 by blast
  next
    case W2
    then show ?thesis
      using <bisim_r w v> w2_reaches_w2 by blast
  next
    case W3
    then show ?thesis using <bisim_r w v> w3_reaches_w3 by blast
  qed
qed
```

```
lemma bisim_r_is_bisimulation_back:
  shows <(bisim_r w v) ⟶ (∀v'. v ≲<v> v' ⟶ (∃w'. w ≲<w> w' ∧ bisim_r w' v'))>
proof
  assume <bisim_r w v>
  then show <∀v'. v ≲<v> v' ⟶ (∃w'. w ≲<w> w' ∧ bisim_r w' v')>
  proof (cases w)
    case W_true
    then show ?thesis
      by (metis bisim_r.simps(1) bisim_r.simps(3) bisim_r.simps(4) bisim_r.simps(6) bisim_r.simps(8)

          ctls_accessibility.simps(6) ctls_accessibility.simps(7) local.reflexive world.exhaust)

  next
    case W_false
    then show ?thesis
      by (metis bisim_r.simps(2) bisim_r.simps(3) bisim_r.simps(5) bisim_r.simps(7) bisim_r.simps(9)

          ctls_accessibility.simps(10) ctls_accessibility.simps(8) local.reflexive world.exhaust)
  next
    case W1
    then show ?thesis
      using <bisim_r w v> bisim_r.simps(16) bisim_r.simps(17) bisim_r.simps(18) bisim_r.simps(19)

        bisim_r.simps(4) bisim_r_contains_rhs w1_reaches_w1_w2 by blast
  next
    case W2
    then show ?thesis
      using <bisim_r w v> bisim_r.simps(20) bisim_r.simps(21) bisim_r.simps(22)
        ctls_accessibility.elims(2) by blast
  next
    case W3
    then show ?thesis
      by (metis <bisim_r w v> bisim_r.simps(23) bisim_r.simps(24) bisim_r.simps(25) bisim_r.simps(6)

          local.reflexive w2_reaches_w2 w3_reaches_w3 world.exhaust)
  qed
qed

lemma bism_r_is_bisimulation:
  shows <bisimulation bisim_r>
  unfolding bisimulation_def
  using bisim_r_is_bisimulation_back bisim_r_is_bisimulation_forth bism_r_has_eq_labels by
blast
```

Knowing that W_{true} and W_{false} are bisimilar, we can show that there is no distinguishing CTL* formula for them.

```
lemma w_true_w_false_not_ctl_star_distinguishable:
  fixes
    Φ :: <ap state_formula>
  assumes
    path_lifting: <⋀ w v π1. ⟦bisimilar w v; is_path w π1⟧ ⟹
      (∃ π2. is_path v π2 ∧ (∀ i. π1 i ↔ π2 i))>
  shows
    <(W_true ⊨ₛ Φ ⟷  W_false ⊨ₛ Φ)>
proof -
  have <bisim_r W_true W_false> by simp
  hence <W_true ↔ W_false> using bism_r_is_bisimulation bisimilar_def by force
```

```
  thus ?thesis using path_lifting bisimulation_finer_than_ctls by auto
qed

end

end
```

References

[Ben17] Christoph Benzmüller. Universal reasoning, rational argumentation and human-machine interaction. *CoRR*, abs/1703.09620, 2017. URL: http://arxiv.org/abs/1703.09620.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 01 2008.

[BNO+23] Gertrud Bauer, Tobias Nipkow, David von Oheimb, Lawrence C Paulson, Thomas M Rasmussen, Christophe Tabacznyj, and Markus Wenzel. The supplemental Isabelle/HOL library. https://isabelle.in.tum.de/library/HOL/Library/document.pdf, 2023.

[FS23] Bernd Finkbeiner and Julian Siber. Counterfactuals modulo temporal logics. In Ruzica Piskac and Andrei Voronkov, editors, *Proceedings of 24th international conference on logic for programming, artificial intelligence and reasoning*, volume 94 of *EPiC series in computing*, pages 181–204. EasyChair, 2023. ISSN: 2398-7340. URL: https://easychair.org/publications/paper/sWZw, doi:10.29007/qtw7.

[Lew73] David K. Lewis. *Counterfactuals*. Blackwell, Oxford, 1973.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[Poh21] Max Pohlmann. *Reducing strong reactive bisimilarity to strong bisimilarity*. Bachelorarbeit, Technische Universität Berlin, June 2021. URL: https://maxpohlmann.github.io/Reducing-Reactive-to-Strong-Bisimilarity/thesis.pdf.

[Sic16] Salomon Sickert. Linear temporal logic. *Archive of Formal Proofs*, March 2016. https://isa-afp.org/entries/LTL.html, Formal proof development.