Extra (Non Required, No Submission) Homework

1. Quarks

Here are the axioms of the U2 algebra. It differs from the U1 algebra by self interaction: A#A in U1 corresponds to A ∩ A in U2, and A ∩ A = A.

<table>
<thead>
<tr>
<th>+ identity</th>
<th>A ⊕ 0 = A</th>
<th>∩ inert</th>
<th>A ∩ 0 = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ commute</td>
<td>A ⊕ B = B ⊕ A</td>
<td>∩ commute</td>
<td>A ∩ B = B ∩ A</td>
</tr>
<tr>
<td>+ associate</td>
<td>A ⊕ (B ⊕ C) = (A ⊕ B) ⊕ C</td>
<td>∩ associate</td>
<td>A ∩ (B ∩ C) = (A ∩ B) ∩ C</td>
</tr>
<tr>
<td>× product</td>
<td>A × B = (A ∩ B) ⊕ A ⊕ B</td>
<td>distributivity</td>
<td>A ∩ (B ⊕ C) = A ∩ B ⊕ A ∩ C</td>
</tr>
<tr>
<td>+ involution</td>
<td>A ⊕ A = 0</td>
<td>∩ idempotent</td>
<td>A ∩ A = A</td>
</tr>
</tbody>
</table>

Prove that x distributes over ∩:

\[ A x (B ∩ C) = (A x B) ∩ (A x C) \]

2. Layers and FeatureHouse Collaboration Diagrams

Consider the following generic component of 2 classes (ListContainer and ListIterator) and one mixin (ListElem) written in pseudo-Java:
There are three layers/features: BasicList, ListIter, and DeleteFlag in this component. BasicList creates a singly-linked list of E elements that provides no iterator support and no ability to delete. DeleteFlag adds boolean del (delete) flag to every element of type E to indicate if the element on the list has been deleted. ListIter adds iterator support to these lists. The order in which layers/features are added is:

- BasicList then
- ListIter then
- DeleteFlag

Note that this order causes element removal via iterators to be vacuous (or a no-op) unless DeleteFlag is also present. Your tasks are:

```java
class ListContainer<E> {
    ListElem<E> head;
    ListContainer() { head = null; }

    void insert(ListElem<E> e) {
        e.next = head;
        head = e;
        e.del = false;
    }

    void delete(ListElem<E> e) {
        e.del = true;
    }

    ListIterator<E> iterator() {
        return ListIterator<E>(this);
    }
}

class ListIterator<E> implements Iterator<E> {
    ListElem<E> current;

    ListIterator(ListContainer<E> c) {
        current = c.head;
    }

    boolean hasNext() { return e.next!=null; }

    ListElem<E> next() { return e.next; }

    void remove() {
        e.del = true;
    }
}

class ListElem<E> extends E {
    boolean del;
    ListElem<E> next;

    ListElem<E>(E elem) {
        elem.del = false;
        elem.next = null;
    }
```
1. Decompose these classes/mixins into the 3 layer/features above
2. Show their FeatureHouse collaboration diagram when all 3 layers/features are composed (click here for a definition of such diagrams)
3. Explain the AHEAD vector representation of each layer/feature and the composition of these 3 layers as an equation.

3. Safe Composition

Here are two lines of code:

```java
    ListIterator<E> iterator() {
        return ListIterator(this);
    }
```

Let $\pi$ be the function that takes a line of code OR a program identifier and returns the feature predicate that defines when that line of code or that identifier appears the text of an SPL program. What safe composition constraints would be checked in these 2 lines of code?
Extra (Non Required, No Submission) Homework

1. Quarks

Here are the axioms of the U2 algebra. It differs from the U1 algebra by self interaction: A#A in U1 corresponds to A ∩ A in U2, and A ∩ A = A.

<table>
<thead>
<tr>
<th></th>
<th>A ⊕ 0 = A</th>
<th>∩ inert</th>
<th>A ∩ 0 = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ identity</td>
<td>A ⊕ 0 = A</td>
<td>∩ inert</td>
<td>A ∩ 0 = 0</td>
</tr>
<tr>
<td>+ commute</td>
<td>A ⊕ B = B ⊕ A</td>
<td>∩ commute</td>
<td>A ∩ B = B ∩ A</td>
</tr>
<tr>
<td>+ associate</td>
<td>A ⊕ (B ⊕ C) = (A ⊕ B) ⊕ C</td>
<td>∩ associate</td>
<td>A ∩ (B ∩ C) = (A ∩ B) ∩ C</td>
</tr>
<tr>
<td>× product</td>
<td>A × B = (A ∩ B) ⊕ A ⊕ B</td>
<td>distributivity</td>
<td>A ∩ (B ⊕ C) = A ∩ B ⊕ A ∩ C</td>
</tr>
<tr>
<td>+ involution</td>
<td>A ⊕ A = 0</td>
<td>∩ idempotent</td>
<td>A ∩ A = A</td>
</tr>
</tbody>
</table>

Prove that ∩ distributes over x in U2:

(A × B) ∩ C = (A ∩ C) × (B ∩ C)

Proof, where + is the same as (+) circle-plus:

(A × B) ∩ C
= (A∩B + A + B) ∩ C  // defn of x
= A∩B∩C + A∩C + B∩C  // ∩ distributes over +
= A∩B∩C∩C + A∩C + B∩C  // ∩ idempotent
= A∩C∩B∩C + A∩C + B∩C  // ∩ commute
= (A∩C) x (B∩C)  // defn of x

2. Layers and FeatureHouse Collaboration Diagrams

Consider the following generic component of 2 classes (ListContainer and ListIterator) and one mixin (ListElem) written in pseudo-Java:
There are three layers/features: BasicList, ListIter, and DeleteFlag in this component. BasicList creates a singly-linked list of E elements that provides no iterator support and no ability to delete. DeleteFlag adds boolean del (delete) flag to every element of type E to indicate if the element on the list has been deleted. ListIter adds iterator support to these lists. The order in which layers/features are added is:

- BasicList then
- ListIter then
- DeleteFlag

Note that this order causes element removal via iterators to be vacuous (or a no-op) unless DeleteFlag is also present. Your tasks are:

1. Decompose these classes/mixins into the 3 layer/features above using FeatureHouse encoding
2. Show their FeatureHouse collaboration diagram when all 3 layers/features are composed (click here for a definition of such diagrams)
3. Explain the AHEAD vector representation of each layer-feature and the composition of these 3 layers as an equation.

Answers for 1 and 2:
Answer for 3:

DeleteFlag + ListIter + basicList
= \[ \Delta \text{ListContainer}_{\text{DeleteFlag}} + \Delta \text{ListIter}_{\text{DeleteFlag}} + \Delta \text{ListElem}_{\text{DeleteFlag}} \] + \[ \Delta \text{ListContainer}_{\text{ListIter}}, \Delta \text{ListIter}_{\text{ListIter}}, 0 \] + \[ \text{ListContainer}_{\text{basicList}}, 0, \text{ListElem}_{\text{basicList}} \]
= \[ \Delta \text{ListContainer}_{\text{DeleteFlag}} + \Delta \text{ListContainer}_{\text{ListIter}} + \text{ListIter}_{\text{ListIter}} + 0, \Delta \text{ListIter}_{\text{DeleteFlag}} + \text{ListIter}_{\text{ListIter}} + 0, \Delta \text{ListElem}_{\text{DeleteFlag}} + 0 + \text{ListElem}_{\text{basicList}} \]

3. Safe Composition

Here are two lines of code:

\[ \ell_1: \text{ListIter}<E> \text{ iterator()} \{ \]
\[ \ell_2: \quad \text{return ListIter}(this); \]

Let \( \pi \) be the function that takes a line of code OR a program identifier and returns the feature predicate that defines when that line of code or that identifier appears the text of an SPL program. What safe composition constraints would be checked in these 2 lines of code?

Ans:
\[ \pi(\ell_1) \text{ implies } \pi(\text{ListIter}) \]
\[ \pi(\ell_2) \text{ implies } \pi(\text{ListIter}) \]