Meeting agenda

1. New developments in translation to Alloy
2. Design choices in the semantics of behavioral Clafer
    - Meaning of current structural(cross-tree) constraints
    - Default behavior of subclafers: mutable vs immutable
3. Design choices in the concrete syntax

Translation to Alloy

We reconsidered translation to Alloy. Issues using Amir's vanilla library:

- Need for global state
- Issues with identity when cardinality is more than 1
- Not compatible with current compiler Alloy output
- Logical expression require use of library functions, instead of using Alloy operators
- Suffers from state explosion

New solution

New solution is still similar and based on Bounded Model Checking with Alloy paper[1]. Instead of global state we introduce local state concept:

1. Define discrete Time ordered using util/ordering module.
2. Since Time set is finite we add a loop relation from last Time instance to any other one.
3. Each mutable field relation gets additional Time column.
4. Define behavioral constraints using LTL. LTL encoding over Time is presented in the paper.
5. Traces are modeled according to the ordering of Time atoms. A snapshot in a trace is assembly of immutable values and projection of mutable values at specific Time instance.

[1]Alcino Cunha. "Bounded Model Checking of Temporal Formulas with Alloy". In: *CoRR* abs/1207.2746 (2012).

Meaning of current cross-tree constraints

Current cross-tree constraints may have two different semantics in behavioral Clafer.

Restricts the first state

- Similar to LTL/CTL
- Often meant to restrict all states, so models will need to be altered with global modalities

Restrict globally

- Easy to restrict all states in the trace
- Different semantics from LTL/CTL, so temporal constraints need new concrete syntax
- Otherwise hard to reason about initial states

Subclafer mutability

It can be difficult to implicitly imply which subclafers are mutable. Therefore we need some kind of assumption about default mutability and concrete syntax to express opposite.

- Should top level clafers be immutable?
- Should we imply that subclafers are immutable or mutable by default?

All fields are immutable by default

```
PM
  heart -> Heart
CaseHandler
[mutable] current -> Case
```

All fields are mutable by default

```
Person
  name: String
  [immutable name]
Person
  age: int
```

Design choices for concrete syntax

We need to embed LTL into Behavioral Clafer.Why? With LTL embeded into Behavioral Clafer we will be able to formally and concisely specify useful properties , e.g.

- safety properties (something "bad" will not occur)
- liveness properties (something "good" will happen)
- fairness properties

LTL in Clafer

Embedding LTL into Clafer means:

- design a way to specify LTL formulas e.g. $LTL_Formula, [LTL_Formula] etc.
- unary and binary operators

LTL unary operators:

- X - Next
- G - Globally
- F - Finally

LTL binary operators:

- U - Until
- R - Release
- W - WeakUntil

LTL in Clafer

A U B = "A has to hold at least until B, which holds at the current or a future position"
A R B = "B has to be true until and including the point where A first becomes true. If A never becomes true, B must remain true forever."
Keywords for embedding LTL into Clafer:

$$X \quad | \quad U \quad | \quad G \quad | \quad R \quad | \quad F \quad | \quad W$$

Example

```
abstract Person
  name -> string
  xor gender
    male
    female
  age : integer
  xor mood
    sad
    happy
  xor status
    alive
    dead
  [$LTL_Formula mood.sad -> F(mood.happy)
  $LTL_Formula status.alive -> F(status.dead)
  ]
```