Session 4: Linear Time Calculus

Goals of the following exercise sessions

At the end of this exercise session, you are supposed to be able to

- Model a dynamic system in linear time calculus
- Solve simple tasks like simulation and some verification using IDP

1 Books

In order to get you started with the linear time calculus, we will go over the *books* example.

The situation is the following: there are three books (`Book1`, `Book2`, `Book3`) and three persons (`Bob`, `John`, `Mary`). Initially `Bob` owns `Book1`, `John` owns `Book2`, and `Mary` owns `Book3`. Our first goal is to write down a model in the linear time calculus in which books can be passed to one another. To come to this model, one can follow the next steps.

1. Our vocabulary should consists of the following: *types*, *static predicates* and *dynamic predicates*. It's clear that static predicates do not change over time while dynamic do (they thus have a temporal argument). The first thing you should do when modeling a system is determining which types and predicates you will use and write them down.

2. Create a type `Time`, a (partial) function `Next(Time):Time`, and a constant `Start:Time`.

3. Very often we can categorize the *dynamic* predicates as follows: some of then can be seen as *actions* and other as *fluents*. The idea is that actions, e.g., *Give*, *Drop*, ..., are *non-inertial*. This means that their value can change randomly, while fluents, e.g., *Owns*, ..., are *inertial*. This means that their value can only change when some actions happen. Though we do not make an explicit distinction between actions and fluents in LTC, it might make it easier to model a system if you do make this distinction in your mind!

4. For dynamic predicates P, we typically make an extra predicate In_P, describing the initial situation. Enumerate the *complete* initial situation, in a definition or structure, for example

$$\begin{array}{l}\{\\ \quad \forall x : In_P(x) \leftarrow \ldots \\ \quad In_P(7).\\ \}\end{array}$$

5. For each dynamic predicate $P(t, x)$, introduce two new predicates: $C_P(t, x)$ and $C_{\neg P}(t, x)$, and add the following definition (inertia axiom) to the theory basically stating that P holds at the start if and only if In_p holds and holds on a next time point if and only if it either is caused, or it was already true and did not get uncaused.

$$\begin{array}{l}\{\\ \quad \forall x : P(Start, x) \leftarrow In_P(x).\\ \quad \forall x : P(Next(t), x) \leftarrow C_P(t, x)\\ \quad \forall x : P(Next(t), x) \leftarrow P(t, x) \wedge \neg C_{\neg P}(t, x)\\ \}\end{array}$$

   Check that you understand what this means!

6. Now we define the newly introduced predicates $C_P(t, x)$ and $C_{\neg P}(t, x)$.

7. While 'actions' are non-inertial, they usually do not occur completely at random. They are often bound by other types of conditions. E.g preconditions, concurrency axioms,... Express them in first order logic.

8. One last thing to do is fix the interpretation of `Time`, `Start` and `Next`. Typically, you work in a finite interval $0..n$ and thus want $Start$ to be 0, and $Next$ to map t onto $t + 1$. You usually do this in a separate theory (see the skeleton for the books example on Toledo).

9. **Inferences:** You can now perform several inferences. In the skeleton on Toledo, you find two procedures:

   - `findmodels()` searches for models of the entire theory (first it sets $Next$ and $Start$ correctly, next it searches for models of your theory, using model expansion)
   - `simulate()` makes simulations: time point per time point: it starts with enumerating initial situations. It asks you which solution to continue with. This can be useful for *debugging* your theory.
   - **Please make sure you understand these procedures!**

2 Plates

At home you have p number of *plate*s you can use for dinner. If you eat they become dirty and when you wash them, they become *clean* again. Note that a plate can not be used to eat if it is dirty and it is useless to wash a clean plate.

Tasks

1. Write down this model in the Linear Time Calculus. First, make use of the intermediate causal rules, as in the previous example, and then translate! Suggestion:
   - Types: `Plate`
   - Fluents: `clean(Plate)`
   - Actions: `cleanPlate(Plate)`, `usePlate(Plate)`
2. Write down this theory in an IDP file (skeleton on Toledo).
3. Check using the IDP system, whether it is possible that all your plates become dirty. Such a constraint, a goal constraint, cannot be used in the simulation inference. Therefore, you typically write this in a separate theory. That way, you can use your regular theory to simulate and the combination of the two for planning.
4. Suppose you always want to have an extra clean plate in case there is some unexpected guest. Alter your theory to incorporate this constraint.

3 Jugs

You must obtain exactly 4 liters of water using a 5 liter jug, a 3 liter jug and a water pump. Use the IDP system to find the trace that gives you the solution. This is a typical planning problem.

Tasks:

1. Find a representation in Linear Time Calculus (use intermediate causal rules!).
2. Find a solution for the question using IDP.
3. What is the smallest number of steps necessary to solve this?

4 Garage door controller

In this exercise, we are specifying a simple controller for a garage door. A garage is operated by a controller and two remotes both with an “open” and a “close” button. The controller has several state variables, the most important are:

- "position" is a dynamic function taking values in 1,..,5 to represent the position of the port: 1 is closed and 5 is opened.
- "opening_door" represents that the door is opening.
- "closing_door" represents that the door is closing.

As long as "opening_door" is true, the port opens up, one position per time step, until the door is fully open, at which time "opening_door" is switched off. The inverse happens when "closing_door" is true. The two remotes can be used simultaneously and when they issue contradictory orders, opening has priority to closing the garage, to avoid damage to cars. The initial state is that the door is closed and that both "opening_door" and "closing_door" are false.

- Write down this model in the linear time calculus.
- Write the axioms for the guided simulation:
  - $t = 0$: door is closed.
  - $t = 1$: driver A uses one remote to open door.
  - $t = 10$: driver A has left the garage and uses the same remote to close the door
  - $t = 12$: driver B wants to enter the garage and uses the second remote to open door.

  In what state is the controller at t = 13?
- A desired implicit invariant is that "opening_door" and "closing_door" are not true simultaneously. Do you have an intuition as to how you would prove this invariant? There is a weak and a strong way to check it.

5 Nim

Nim is a game for two players. In the initial situation, a set of little disks are on the table in the form of a set of heaps of increasing size, respectively $1, 3, 5, 7, \ldots$ When modeling the game, you may assume that there are only four heaps. In turns, the players take a number of matches away from a heap: at least one and at most as much as there are in the heap. The player that takes the last match loses the game. Use the following vocabulary:

- Type `Time isa int`
- Type `Player` representing the players in the game.
- Type `Matches isa int` is a finite integer type, representing a number of matches.
- Type `Heap` representing the set of heaps in the game.

- Predicate `nb(Time, Heap, Matches)` models the number of matches on a heap at a specific time.
- Predicate `turn(Time, Player)` determines the player on turn.
- Predicate `winner(Time, Player)` determines the player that wins the game.
- Predicate `gameOver(Time)` determines that the game is over, i.e. no move is possible
- Predicate `takes(Time, Player, Heap, Matches)` indicates the action.

Once you have your theory, you can run a number of verification tasks. Do this by creating additional theories representing the questions and checking whether they are satisfiable when merged with your theory.

- Either player has the possibility to win the game.
- In every state, it is the turn of exactly one player.
- Every game comes to an end. (You might need to check this for only three heaps, for computational reasons.)
- The game is not over until all the heaps are empty.
- There is a game that lasts for exactly 16 moves.
- Suppose the first player wants the game to end as quickly as possible. How many moves does he need?
  Hint: use two specifications: one to show that the first player can always make sure that the game ends after n moves, and one to show that this is not the case for $n - 1$.