Formal Methods Homework 2011
Solar Heating System Verification

Lyudmil Angelov
Davide Leonarduzzi
Camillo Lugaresi

2011/06/24

1 Introduction

We decided to perform verification of some properties of our system using Trio2Promela (T2P in the rest of this document). The process involved a number of interesting challenges, mainly stemming from the fact that the current version of T2P is geared towards a different subset of Trio than what we are using. We will document the main adaptations we performed on our initial formulation in order to make it verifiable with T2P and Spin.

2 Physical properties

In our formalization of the system, we chose to use Trio to model both its physical and its logical behavior. Moreover, we attempted to minimize the number of axioms used. As a result, there are some important physical properties which are not directly included as axioms, but follow as theorems from the axioms we did include. Therefore, we would like to use model checking to show that our axioms are indeed sufficient to prove the necessary properties.

2.1 Arithmetic

The published grammar of T2P, as well as the program's parser, allow axioms to contains mathematical expressions involving variables. Unfortunately, we soon discovered that the Promela generator had different plans: whenever

an identifier occurred as an operand to an arithmetic operator, it blithely attempted to parse it as a literal numeric constant, resulting in an exception.

Fortunately, it is still possible to simulate arithmetic over very small domains using logic. However, the axioms involving temperature were too complex to be translated in that way, and so we had to drop them. We were, however, still able to verify some properties involving volumes.

2.2 Volumes

In our modeling of the hydraulic part of the system, we decided that connections between components should match their physical connections in reality. Therefore, the pump, which pumps water from the tank into the collector, is connected to an exit port of the former and to the entry port of the latter; but the circuit is closed by connecting the exit port of the collector to an entry port of the tank, and the flow in this pipe is not driver directly by the pump. Instead, since the tank and the collector start out full [1] and must remain so (leaks are disallowed by an axiom), it follows that the variation of the volume of water in each container must be zero, from which follows that the volume flowing in the collector-tank pipe is the same as that flowing in the tank-pump-collector pipe.

In our original formulation, the volume and temperature deltas calculated on the basis of the current state of the system were added to the current volume and temperature to produce the values of V and T at the following instant. As a result, the volume deltas had to be zero because a non-zero value would have necessarily caused a violation of the maximum volume and no-leaks axioms at the following time instant.

This means that this property cannot be verified by simply checking the LTL property $\Box(t_0 \wedge s \rightarrow delta\_volume = 0)$, because if $delta\_volume \neq 0$, s does not become false until the next cycle. Instead, we need to use the method for verifying metric properties.

We use the LTL property $\Diamond(t_0 \wedge \neg s)$; this states that (for all possible paths) eventually a state is reached where the axioms cannot be satisfied. First we need to ensure that this property is invalid when checking the base trio specification: this tells us that our axioms are not contradictory, that is, it is possible to have at least one history where they are always satisfied. Then we add the negation of the property to be verified to the model, and check the LTL property again with this new axiom. If this time the LTL

[1] while producing the original document, the fact that the constant total_volume should be equal to the sum of the max_volume of the tank and the collector may have been left on the editing floor; it will, however, be made explicit here.

property is valid, it means that negating the property eliminated all possible histories of the system, which means that it holds in all of them.

volume4.trio

```
variables
  pump_volume:[0..1],

  collector_volume:[9..11],
  collector_out_volume:[0..1],
  collector_delta_volume:[-1..1],

  tank_volume:[19..21],
  tank_delta_volume_A:[-1..1],
  tank_delta_volume_B:[-1..1],
  tank_delta_volume:[-2..2],

  tap_volume:[0..1]


constants
  DELTA_LOW = 0, DELTA_MEDIUM = 1, DELTA_HIGH = 2, PUMP_VOLUME=1,
  COLD_SUPPLY_TEMPERATURE=1, TAP_VOLUME=1, COLLECTOR_MAX_VOLUME=10,
  WATER_SPECIFIC_HEAT=1, WATER_DENSITY=1, TANK_MAX_VOLUME=20,
  POLLING_INTERVAL=3, TOTAL_VOLUME=30

axioms

    collector_volume_within_bounds:
      0 < collector_volume & collector_volume <= COLLECTOR_MAX_VOLUME;

    collector_delta_volume_definition:
      (pump_volume = collector_out_volume -> collector_delta_volume = 0) &
      (pump_volume = 1 & collector_out_volume = 0 -> collector_delta_volume = 1) &
      (pump_volume = 0 & collector_out_volume = 1 -> collector_delta_volume = -1);

    collector_volume_change:
      (Past(collector_delta_volume=0,1) -> (
        (Past(collector_volume=9,1) -> collector_volume=9) &
        (Past(collector_volume=10,1) -> collector_volume=10) &
        (Past(collector_volume=11,1) -> collector_volume=11)
      )) &
```

```
  (Past(collector_delta_volume=1,1) -> (
    (Past(collector_volume=9,1) -> collector_volume=10) &
    (Past(collector_volume=10,1) -> collector_volume=11) &
    (Past(collector_volume=11,1) -> collector_volume=12)
  )) &
  (Past(collector_delta_volume=-1,1) -> (
    (Past(collector_volume=9,1) -> collector_volume=8) &
    (Past(collector_volume=10,1) -> collector_volume=9) &
    (Past(collector_volume=11,1) -> collector_volume=10)
  ));


tank_volume_within_bounds:
  0 < tank_volume & tank_volume <= TANK_MAX_VOLUME;

tank_delta_volume_A_def:  // collector_out_volume - pump_volume
  (collector_out_volume = pump_volume -> tank_delta_volume_A = 0) &
  (collector_out_volume > pump_volume -> tank_delta_volume_A = 1) &
  (collector_out_volume < pump_volume -> tank_delta_volume_A = -1);

tank_delta_volume_B_def:  // tap_volume - tap_volume
  (tap_volume = tap_volume -> tank_delta_volume_B = 0) &
  (tap_volume > tap_volume -> tank_delta_volume_B = 1) &
  (tap_volume < tap_volume -> tank_delta_volume_B = -1);

tank_delta_volume_definition:   // tank_delta_volume_A + tank_delta_volume_B
  (tank_delta_volume_A = 0 -> (tank_delta_volume = tank_delta_volume_B)) &
  (tank_delta_volume_A = 1 -> (
    (tank_delta_volume_B = -1 -> tank_delta_volume = 0) &
    (tank_delta_volume_B = 0 -> tank_delta_volume = 1) &
    (tank_delta_volume_B = 1 -> tank_delta_volume = 2)
  )) &
  (tank_delta_volume_A = -1 -> (
    (tank_delta_volume_B = -1 -> tank_delta_volume = -2) &
    (tank_delta_volume_B = 0 -> tank_delta_volume = -1) &
    (tank_delta_volume_B = 1 -> tank_delta_volume = 0)
  ));

tank_volume_change:
  (Past(tank_delta_volume=0,1) -> (
    (Past(tank_volume=19,1) -> tank_volume=19) &
```

```
      (Past(tank_volume=20,1) -> tank_volume=20) &
      (Past(tank_volume=21,1) -> tank_volume=21)
    )) &
    (Past(tank_delta_volume=1,1) -> (
      (Past(tank_volume=19,1) -> tank_volume=20) &
      (Past(tank_volume=20,1) -> tank_volume=21) &
      (Past(tank_volume=21,1) -> tank_volume=22)
    )) &
    (Past(tank_delta_volume=-1,1) -> (
      (Past(tank_volume=19,1) -> tank_volume=18) &
      (Past(tank_volume=20,1) -> tank_volume=19) &
      (Past(tank_volume=21,1) -> tank_volume=20)
    )) &
    (Past(tank_delta_volume=-2,1) -> (
      (Past(tank_volume=19,1) -> tank_volume=17) &
      (Past(tank_volume=20,1) -> tank_volume=18) &
      (Past(tank_volume=21,1) -> tank_volume=19)
    )) &
    (Past(tank_delta_volume=2,1) -> (
      (Past(tank_volume=19,1) -> tank_volume=21) &
      (Past(tank_volume=20,1) -> tank_volume=22) &
      (Past(tank_volume=21,1) -> tank_volume=23)
    ));

  NoLeaks:
    (tank_volume = 17 -> collector_volume = 13) &
    (tank_volume = 18 -> collector_volume = 12) &
    (tank_volume = 19 -> collector_volume = 11) &
    (tank_volume = 20 -> collector_volume = 10) &
    (tank_volume = 21 -> collector_volume = 9) &
    (tank_volume = 22 -> collector_volume = 8) &
    (tank_volume = 23 -> collector_volume = 7);

// PROPERTIES TO CHECK
//
// we want to prove that, when the pump is active, the flow in the tank->pump pipe,
// pump->collector pipe, and collector-> tank pipe is the same. in the original
// formulation this is a consequence of the noleaks axiom, which (assuming
// total_volume = max_tank_volume + max_collector_volume) forces delta_volume=0,
// which forces in_volume = out_volume.
//
```

```
// here we need to do things a bit differently because we don't have arithmetic.
//
// properties to check:
//    collector_delta_volume=0
//    tank_delta_volume=0

//  collector_delta_volume_zero:
//    1=1 -> collector_delta_volume<>0;

//  tank_delta_volume_zero:
//    1=1 -> tank_delta_volume<>0;

// note that we cannot simply write "not( collector_delta_volume = 0 )" because
// that axiom gets silently ignored by t2p (it's simply missing from the generated
// promela code.
```

As mentioned in the comments at the bottom, we ran into a strange problem where T2P would silently ignore axioms of the form $a = b$ or $a \neq b$. The workaround was to prepend them with $1 = 1 \rightarrow$.

3 Logical properties

According to the specification, the controller checks the temperatures periodically and, at that time, starts or stops the pump according to the difference. In our formalization, this behavior is implicit in the fact that the temperature readings are only updated when the update timer fires. We can verify that this property holds, using the same method as before.

control2.trio

```
variables
  pump_start:[0..1], pump_stop:[0..1], pump_on:[0..1], pump_off:[0..1],
  tap_on:[0..1],

  temp_tank:[1..5], temp_collector:[1..5], delta_temp:[0..2],
  pump_volume:[0..1],

  collector_temperature:[1..5],
  tank_temperature:[1..5],

  update:[0..1]
```

```
constants
  DELTA_LOW = 0, DELTA_MEDIUM = 1, DELTA_HIGH = 2, PUMP_VOLUME=1,
  COLD_SUPPLY_TEMPERATURE=1, TAP_VOLUME=1, COLLECTOR_MAX_VOLUME=10,
  WATER_SPECIFIC_HEAT=1, WATER_DENSITY=1, TANK_MAX_VOLUME=20,
  POLLING_INTERVAL=3, TOTAL_VOLUME=30

axioms

  controller_define_delta:
    (temp_collector = 5 -> (
      ((temp_tank <= 3) -> delta_temp = DELTA_HIGH) &
      ((temp_tank = 4) -> delta_temp = DELTA_MEDIUM) &
      ((temp_tank = 5) -> delta_temp = DELTA_LOW)
    )) &
    (temp_collector = 4 -> (
      ((temp_tank = 1 | temp_tank = 2) -> delta_temp = DELTA_HIGH) &
      ((temp_tank = 3) -> delta_temp = DELTA_MEDIUM) &
      ((temp_tank = 4 | temp_tank = 5) -> delta_temp = DELTA_LOW)
    )) &
    (temp_collector = 3 -> (
      ((temp_tank = 1) -> delta_temp = DELTA_HIGH) &
      ((temp_tank = 2) -> delta_temp = DELTA_MEDIUM) &
      ((temp_tank >= 3) -> delta_temp = DELTA_LOW)
    )) &
    (temp_collector = 2 -> (
      ((temp_tank = 1) -> delta_temp = DELTA_MEDIUM) &
      ((temp_tank > 1) -> delta_temp = DELTA_LOW)
    )) &
    (temp_collector = 1 -> delta_temp = DELTA_LOW);

  controller_start:
    UpToNow(pump_off=1) & (delta_temp = DELTA_HIGH) <-> pump_start=1;

  controller_stop:
    UpToNow(pump_on=1) & (delta_temp = DELTA_LOW) <-> pump_stop=1;

  pump_start_off:
    AlwP(pump_start=0) -> pump_off=1;

  pump_when_off:
```

```
  pump_off=1 -> pump_volume = 0;

pump_when_on:
  pump_on=1 -> pump_volume = PUMP_VOLUME;

pump_start:
  UpToNow(pump_off=1) & pump_start=1 -> Until_w(pump_on=1, pump_stop=1);

pump_stop:
  UpToNow(pump_on=1) & pump_stop=1 -> Until_w(pump_off=1, pump_start=1);

pump_state_unique:
  (pump_off=1 & pump_on=0) | (pump_on=1 & pump_off=0);

  periodicity:
    update=1 -> NextTime(update=1, POLLING_INTERVAL);

  sens_tank_update:
    update=1 -> temp_tank = tank_temperature;

  sens_collector_update:
    update=1 -> temp_collector = collector_temperature;

  sens_tank_hold:
    update=0 -> (
      (Past(temp_tank=1, 1) -> temp_tank=1) &
      (Past(temp_tank=2, 1) -> temp_tank=2) &
      (Past(temp_tank=3, 1) -> temp_tank=3) &
      (Past(temp_tank=4, 1) -> temp_tank=4) &
      (Past(temp_tank=5, 1) -> temp_tank=5)
    );

  sens_collector_hold:
    update=0 -> (
      (Past(temp_collector=1, 1) -> temp_collector=1) &
      (Past(temp_collector=2, 1) -> temp_collector=2) &
      (Past(temp_collector=3, 1) -> temp_collector=3) &
      (Past(temp_collector=4, 1) -> temp_collector=4) &
      (Past(temp_collector=5, 1) -> temp_collector=5)
    );
```

```
    start_update:
      AlwP(update=0) -> update=1;

// properties to check:
//  pump_start | pump_stop -> update

  controller_acts_on_update:
    (pump_start=1 | pump_stop=1) & update=0;
```
