SMV continued (Lecture 5)

Analysis of Software Artifacts

Agenda

- discuss another example
- a small vending machine
- homeworks 2 and 3 will be based on this example
- discuss additional capabilities of SMV

Vending machine

- one user
- one vending machine
- takes two coins to buy a beverage

User (Variables)

- state
  - initial, one-coin, two-coins
  - making-choice, waiting
- choice
  - none, coke, diet-coke
  - sprite, mountain-dew

Vending machine (Variables)

- state
  - initial, one-coin, two-coins
  - get-choice, dispense

User (Initial values)

- initial value for the variable state

  ```
  state := initial;
  ```

- initial value for the variable choice

  ```
  choice := none;
  ```

User (Transitions)

- if `state` is `initial` and state of the
- vending machine is `initial` stay
- in `initial` or transition to `one-coin`
- using *non-determinism*

```
state = initial & (vending-machine-state = initial)
   { initial, one-coin };
```

User (Transitions)

- if state is `one-coin` move to `two-coins`
- indicates that the user has deposited the second coin
- when state is `making-choice` the user makes choice
- see the transitions for the variable choice

User (Transitions)

- see the state variable `choice`
- make a non-deterministic choice between beverages
- when `state = making-choice`
- go back to `none` when
- beverage has been dispensed

Vending machine (Initial)

- initial value of `state` is
- `initial`
- `user-state` and `choice` passed
- as parameters

Vending machine (Transitions)

- only few transitions
- can you locate them in the code?
- change state from `initial` to `one-coin`
- if the `user-state` is `one-coin`
- go to state `dispense` after `get-choice`

Macros

- in SMV you can define *macros* using the keyword `DEFINE`
- in module `vending-machine` define a macro `DISPENSED`

```
DEFINE
   DISPENSED := state = dispense;
```

Instantiating

- instantiate the module `vending-machine`
- instantiate the module `user`
- create “real” state machines
- module is like a *type definition*

Instantiating

```
MODULE main
VAR
  machine: vending-machine(msee-user.state,
                           msee-user.choice);
  msee-user: user(machine.DISPENSED,
                  machine.state);
```

Specification

- **if** the user state is `one-coin` and
- vending machine state is `initial`, **then**
- always eventually vending machine state is `dispensed`

Specification

```
SPEC
  AG(((msee-user.state = one-coin) &
      (machine.state = initial))
      -> AF(machine.state = dispense))
```

Enumerating behaviors

- sometimes you want to demonstrate a certain
- behavior or trace from a spec
- for example,

    show me a trace when transaction $T1$ is finished

Enumerating behaviors

- enumerate a trace where the vending machine dispenses
- negation of the property is `!machine.DISPENSED`
- assert that `!machine` is never true

Specification

```
SPEC

  AG(!machine.DISPENSED)
```

Negation of the property

- is **EF**(machine.dispensed)
- so the counterexample to the previous spec is a
- *trace* where the vending machine eventually
- dispenses