Alcino Cunha

SPECIFICATION AND MODELING

ELECTRUM OVERVIEW

Universidade do Minho & INESC TEC

2019/20

TRASH

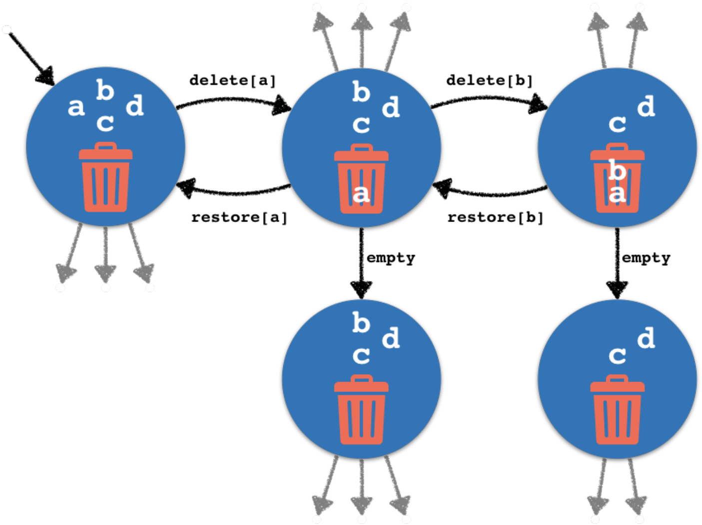
Design a trash component such that:

- A deleted file can still be restored if the trash is not emptied

TASKS

- Design of the structure and behavior (operations) of the component
- Validate this design by simulation
- Elicit and verify expected properties

TRANSITION SYSTEMS



STATE

- A *state* is an assignment of values to variables
- In abstract design, it is useful to rely on standard mathematical structures to describe states

Alloy

- Values are sets and relations
- Inhabited by (tuples of) uninterpreted *atoms*
- Sets are declared with the `sig` keyword

Electrum

- Mutable sets (state variables) are also declared with the `var` keyword

TRASH STATE

```
var sig File {}
var sig Trash in File {}
```

EXPLICIT MODELING OF TRANSITION SYSTEMS

- A transition system can be modeled explicitly:
  - Define which are the initial states
  - Define how the next state(s) can be obtained from the current one
- In formal software design, all states are usually required to always have at least one successor

SMV

The transition system is explicitly modeled with a DSL
The verification tool can detect *deadlocks*

IMPLICIT BEHAVIOR SPECIFICATION

- The behaviour of a transition system can be abstracted by its set of *infinite traces*
  - This is known as a *linear model of time*
- This set of traces can be modeled implicitly:
  - By a property that “recognises” the valid traces among all possibles sequences of states
  - This property can be specified with a *linear temporal logic*
  - Ideally combined with a first order logic to specify properties of states

Electrum

The transition system is implicitly modeled with a linear temporal logic specification enclosed in a `fact`
The (infinite) traces satisfying this specification are also known as *instances*

FIRST ORDER LOGIC

Alloy	Math
not	$\neg$
and	$\wedge$
or	$\vee$
implies	$\longrightarrow$
all x : e | p	$\forall x \cdot x \in e \longrightarrow p$
some x : e | p	$\exists x \cdot x \in e \wedge p$

SET OPERATORS

Alloy	Math
`in`	$\subseteq$
`+`	$\cup$
`&`	$\cap$
`-`	$\setminus$
`no e`	$e = \emptyset$
`some e`	$e \neq \emptyset$

LINEAR TEMPORAL LOGIC

Electrum	Meaning
`always p`	`p` is always true from now on
`after p`	`p` is true in the next state
`once p`	`p` was once true
...	...
`e'`	the value of `e` in the next state

AN ELECTRUM PATTERN FOR BEHAVIOR SPECIFICATION

```
fact init { ... }
fact transitions { always (event1 or event2 or ...) }
```

- The specification of every event typically involves:
  - *Guard* - a state formula that checks if the event can occur
  - *Effect* - a formula with primes specifying how some state variables change
  - *Frame* - a formula with primes stating what does *not* change

TRASH BEHAVIOR

```
fact init { no Trash }

fact transitions {
  always (
    // delete file
    (some f: File | f not in Trash and       -- guard
                    Trash' = Trash + f and   -- effect
                    File' = File) or         -- frame
    // restore file
    ... or
    // empty trash
    ...
  )
}
```

TRASH BEHAVIOR REFACTORED WITH PREDICATES

```
pred delete[f : File] {
  f not in Trash
  Trash' = Trash + f
  File' = File
}
pred restore[f : File] { ... }
pred empty { ... }

fact transitions {
  always (
    (some f: File | delete[f] or restore[f]) or empty
  )
}
```

SIMULATION

- Models include analysis commands
- A **run** command asks for an instance (checking the consistency of the facts)
- Further instances can be obtained by an interactive exploration mode akin to simulation
- All commands have a scope that bounds the size of the signatures
- The default is 3, but can be changed with the **for** keyword


DEMO

TRASH BEHAVIOR FIXED

```
pred delete[f : File] { ... }
pred restore[f : File] { ... }
pred empty { ... }
pred do_nothing {
  Trash' = Trash
  File' = File
}

fact transitions {
  always (
    (some f: File | delete[f] or restore[f]) or empty or do_nothing
  )
}
```

ASSERTIONS

- In Electrum, the same first order temporal logic is used for
  - modeling
  - specification of expected properties – *assertions*
- The latter can be enclosed in named `assert` paragraphs

EXAMPLE ASSERTIONS

```
assert restoreAfterDelete {
  -- Every restored file was once deleted
  always (all f : File | restore[f] implies once delete[f])
}

assert deleteAll {
  -- If the trash contains all files and is emptied
  -- then no files will ever exist afterwards
  always ((File in Trash and empty) implies always no File)
}
```

VERIFICATION

- **check** commands are used to verify assertions
- The verification is fully automatic, but limited to the specified scope
- The set of counter-examples can also be explored like instances

DEMO

FIXED ASSERTION

```
assert deleteAll {
  -- If the trash contains all files and is emptied
  -- then no files will ever exist afterwards
  always ((File in Trash and empty) implies after (always no File))
}
```