

Z Machines

Simon Foster **Jim Woodcock**
University of York

20th August 2022

Overview

- 1 Modelling Z Specifications with Z Machines
- 2 Types, Stores, and State Variables
- 3 Operations and Machines
- 4 Animation and Verification
- 5 Managing Requirements



Overview

- 1 Modelling Z Specifications with Z Machines
- 2 Types, Stores, and State Variables
- 3 Operations and Machines
- 4 Animation and Verification
- 5 Managing Requirements



Motivation

- The **Z notation** is a rich specification language:

IncubatorMonitor

$temp : \mathbb{Z}$

$MIN \leq temp \leq MAX$

Increment

$\Delta IncubatorMonitor$

$temp < MAX$

$temp' = temp + 1$

- This flexibility can however make it difficult to **automate** in practice.
- **Z Machines**: restricted subset of Z in Isabelle/HOL (like **Event-B**).
- A **design pattern** for the more general Z language.
- Closer to an **implementation** and requires some **design decisions**.
- Careful handling of **types vs. set** dichotomy, which is less visible in Z.
- Includes support for (1) **proof obligation generation** and (2) **animation**.

Motivation

- The **Z notation** is a rich specification language:

IncubatorMonitor

$temp : \mathbb{Z}$

$MIN \leq temp \leq MAX$

Increment

$\Delta IncubatorMonitor$

$temp < MAX$

$temp' = temp + 1$

- This flexibility can however make it difficult to **automate** in practice.
- **Z Machines**: restricted subset of Z in Isabelle/HOL (like **Event-B**).
- A **design pattern** for the more general Z language.
- Closer to an **implementation** and requires some **design decisions**.
- Careful handling of **types vs. set** dichotomy, which is less visible in Z.
- Includes support for (1) **proof obligation generation** and (2) **animation**.

Motivation

- The **Z notation** is a rich specification language:

IncubatorMonitor

$temp : \mathbb{Z}$

$MIN \leq temp \leq MAX$

Increment

$\Delta IncubatorMonitor$

$temp < MAX$

$temp' = temp + 1$

- This flexibility can however make it difficult to **automate** in practice.
- **Z Machines**: restricted subset of Z in Isabelle/HOL (like **Event-B**).
- A **design pattern** for the more general Z language.
- Closer to an **implementation** and requires some **design decisions**.
- Careful handling of **types vs. set** dichotomy, which is less visible in Z.
- Includes support for (1) **proof obligation generation** and (2) **animation**.

Motivation

- The **Z notation** is a rich specification language:

IncubatorMonitor

$temp : \mathbb{Z}$

$MIN \leq temp \leq MAX$

Increment

$\Delta IncubatorMonitor$

$temp < MAX$

$temp' = temp + 1$

- This flexibility can however make it difficult to **automate** in practice.
- **Z Machines**: restricted subset of Z in Isabelle/HOL (like **Event-B**).
- A **design pattern** for the more general Z language.
- Closer to an **implementation** and requires some **design decisions**.
- Careful handling of **types vs. set** dichotomy, which is less visible in Z.
- Includes support for (1) **proof obligation generation** and (2) **animation**.

Motivation

- The **Z notation** is a rich specification language:

IncubatorMonitor

$temp : \mathbb{Z}$

$MIN \leq temp \leq MAX$

Increment

$\Delta IncubatorMonitor$

$temp < MAX$

$temp' = temp + 1$

- This flexibility can however make it difficult to **automate** in practice.
- **Z Machines**: restricted subset of Z in Isabelle/HOL (like **Event-B**).
- A **design pattern** for the more general Z language.
- Closer to an **implementation** and requires some **design decisions**.
- Careful handling of **types vs. set** dichotomy, which is less visible in Z.
- Includes support for (1) **proof obligation generation** and (2) **animation**.

Motivation

- The **Z notation** is a rich specification language:

IncubatorMonitor _____

$temp : \mathbb{Z}$

$MIN \leq temp \leq MAX$

Increment _____

$\Delta IncubatorMonitor$

$temp < MAX$

$temp' = temp + 1$

- This flexibility can however make it difficult to **automate** in practice.
- **Z Machines**: restricted subset of Z in Isabelle/HOL (like **Event-B**).
- A **design pattern** for the more general Z language.
- Closer to an **implementation** and requires some **design decisions**.
- Careful handling of **types vs. set** dichotomy, which is less visible in Z.
- Includes support for (1) **proof obligation generation** and (2) **animation**.

Motivation

- The **Z notation** is a rich specification language:

IncubatorMonitor

$temp : \mathbb{Z}$

$MIN \leq temp \leq MAX$

Increment

$\Delta IncubatorMonitor$

$temp < MAX$

$temp' = temp + 1$

- This flexibility can however make it difficult to **automate** in practice.
- **Z Machines**: restricted subset of Z in Isabelle/HOL (like **Event-B**).
- A **design pattern** for the more general Z language.
- Closer to an **implementation** and requires some **design decisions**.
- Careful handling of **types vs. set** dichotomy, which is less visible in Z.
- Includes support for (1) **proof obligation generation** and (2) **animation**.

Motivation

- The **Z notation** is a rich specification language:

IncubatorMonitor

$temp : \mathbb{Z}$

$MIN \leq temp \leq MAX$

Increment

$\Delta IncubatorMonitor$

$temp < MAX$

$temp' = temp + 1$

- This flexibility can however make it difficult to **automate** in practice.
- **Z Machines**: restricted subset of Z in Isabelle/HOL (like **Event-B**).
- A **design pattern** for the more general Z language.
- Closer to an **implementation** and requires some **design decisions**.
- Careful handling of **types vs. set** dichotomy, which is less visible in Z.
- Includes support for (1) **proof obligation generation** and (2) **animation**.

Motivation

- The **Z notation** is a rich specification language:

IncubatorMonitor _____

$temp : \mathbb{Z}$

$MIN \leq temp \leq MAX$

Increment _____

$\Delta IncubatorMonitor$

$temp < MAX$

$temp' = temp + 1$

- This flexibility can however make it difficult to **automate** in practice.
- **Z Machines**: restricted subset of Z in Isabelle/HOL (like **Event-B**).
- A **design pattern** for the more general Z language.
- Closer to an **implementation** and requires some **design decisions**.
- Careful handling of **types vs. set** dichotomy, which is less visible in Z.
- Includes support for (1) **proof obligation generation** and (2) **animation**.

Overview

- 1 Modelling Z Specifications with Z Machines
- 2 Types, Stores, and State Variables**
- 3 Operations and Machines
- 4 Animation and Verification
- 5 Managing Requirements



Example: Dwarf Signal Types (Z)

$LampId ::= L1 \mid L2 \mid L3$

$dark, stop, warning, drive : \mathbb{F} LampId$

$dark = \emptyset$

$stop = \{L1, L2\}$

$warning = \{L1, L3\}$

$drive = \{L2, L3\}$

$ProperState == \{dark, stop, warning, drive\}$

$Signal == \mathbb{F} LampId$

Example: Dwarf Signal Types (Z)

$LampId ::= L1 \mid L2 \mid L3$

$dark, stop, warning, drive : \mathbb{F} LampId$

$dark = \emptyset$

$stop = \{L1, L2\}$

$warning = \{L1, L3\}$

$drive = \{L2, L3\}$

$ProperState == \{dark, stop, warning, drive\}$

$Signal == \mathbb{F} LampId$

Example: Dwarf Signal Types (Z)

$LampId ::= L1 \mid L2 \mid L3$

$dark, stop, warning, drive : \mathbb{F} LampId$

$dark = \emptyset$

$stop = \{L1, L2\}$

$warning = \{L1, L3\}$

$drive = \{L2, L3\}$

$ProperState == \{dark, stop, warning, drive\}$

$Signal == \mathbb{F} LampId$

Example: Dwarf Signal Types (Z)

$LampId ::= L1 \mid L2 \mid L3$

$dark, stop, warning, drive : \mathbb{F} LampId$

$dark = \emptyset$

$stop = \{L1, L2\}$

$warning = \{L1, L3\}$

$drive = \{L2, L3\}$

$ProperState == \{dark, stop, warning, drive\}$

$Signal == \mathbb{F} LampId$

Example: Dwarf Signal Types (Z)

$LampId ::= L1 \mid L2 \mid L3$

$dark, stop, warning, drive : \mathbb{F} LampId$

$dark = \emptyset$

$stop = \{L1, L2\}$

$warning = \{L1, L3\}$

$drive = \{L2, L3\}$

$ProperState == \{dark, stop, warning, drive\}$

$Signal == \mathbb{F} LampId$

Example: Dwarf Signal Types (Isabelle)

Example: Dwarf Signal Types (Isabelle)

```
enumtype LampId = L1 | L2 | L3
```

```
type_synonym Signal = "LampId set"
```

```
enumtype ProperState = dark | stop | warning | drive
```

```
definition "ProperState = {dark, stop, warning, drive}"
```

```
fun signalLamps :: "ProperState  $\Rightarrow$  LampId set" where  
  "signalLamps dark = {}" |  
  "signalLamps stop = {L1, L2}" |  
  "signalLamps warning = {L1, L3}" |  
  "signalLamps drive = {L2, L3}"
```

Example: Dwarf Signal Types (Isabelle)

```
enumtype LampId = L1 | L2 | L3
```

```
type_synonym Signal = "LampId set"
```

```
enumtype ProperState = dark | stop | warning | drive
```

```
definition "ProperState = {dark, stop, warning, drive}"
```

```
fun signalLamps :: "ProperState  $\Rightarrow$  LampId set" where  
  "signalLamps dark = {}" |  
  "signalLamps stop = {L1, L2}" |  
  "signalLamps warning = {L1, L3}" |  
  "signalLamps drive = {L2, L3}"
```

Example: Dwarf Signal Types (Isabelle)

```
enumtype LampId = L1 | L2 | L3
```

```
type_synonym Signal = "LampId set"
```

```
enumtype ProperState = dark | stop | warning | drive
```

```
definition "ProperState = {dark, stop, warning, drive}"
```

```
fun signalLamps :: "ProperState  $\Rightarrow$  LampId set" where  
  "signalLamps dark = {}" |  
  "signalLamps stop = {L1, L2}" |  
  "signalLamps warning = {L1, L3}" |  
  "signalLamps drive = {L2, L3}"
```

Example: Dwarf Signal Types (Isabelle)

```
enumtype LampId = L1 | L2 | L3
```

```
type_synonym Signal = "LampId set"
```

```
enumtype ProperState = dark | stop | warning | drive
```

```
definition "ProperState = {dark, stop, warning, drive}"
```

```
fun signalLamps :: "ProperState  $\Rightarrow$  LampId set" where  
  "signalLamps dark = {}" |  
  "signalLamps stop = {L1, L2}" |  
  "signalLamps warning = {L1, L3}" |  
  "signalLamps drive = {L2, L3}"
```

Example: Dwarf Signal Types (Isabelle)

```
enumtype LampId = L1 | L2 | L3
```

```
type_synonym Signal = "LampId set"
```

```
enumtype ProperState = dark | stop | warning | drive
```

```
definition "ProperState = {dark, stop, warning, drive}"
```

```
fun signalLamps :: "ProperState  $\Rightarrow$  LampId set" where  
  "signalLamps dark = {}" |  
  "signalLamps stop = {L1, L2}" |  
  "signalLamps warning = {L1, L3}" |  
  "signalLamps drive = {L2, L3}"
```

Stores

- A **store** is a set of variable declarations to be present in the state.
- Corresponds to a **state schema** in Z (no dashed variables).
- Introduces several state components (**x**, **y**, **z**) and invariants.
- Invariants can be optionally **named** and are collected in **state_inv**.

Stores

- A **store** is a set of variable declarations to be present in the state.
- Corresponds to a **state schema** in Z (no dashed variables).
- Introduces several state components (**x**, **y**, **z**) and invariants.
- Invariants can be optionally **named** and are collected in **state_inv**.

Stores

- A **store** is a set of variable declarations to be present in the state.

```
zstore state =  
  x :: T1  
  y :: T2  
  z :: T3  
  where inv1: "assert"
```

- Corresponds to a **state schema** in Z (no dashed variables).
- Introduces several state components (**x**, **y**, **z**) and invariants.
- Invariants can be optionally **named** and are collected in **state_inv**.

Stores

- A **store** is a set of variable declarations to be present in the state.

```
zstore state =  
  x :: T1  
  y :: T2  
  z :: T3  
  where inv1: "assert"
```

- Corresponds to a **state schema** in Z (no dashed variables).
- Introduces several state components (**x**, **y**, **z**) and invariants.
- Invariants can be optionally **named** and are collected in **state_inv**.

Stores

- A **store** is a set of variable declarations to be present in the state.

```
zstore state =  
  x :: T1  
  y :: T2  
  z :: T3  
  where inv1: "assert"
```

- Corresponds to a **state schema** in Z (no dashed variables).
- Introduces several state components (**x**, **y**, **z**) and invariants.
- Invariants can be optionally **named** and are collected in **state_inv**.

Stores

- A **store** is a set of variable declarations to be present in the state.

```
zstore state =  
  x :: T1  
  y :: T2  
  z :: T3  
  where inv1: "assert"
```

- Corresponds to a **state schema** in Z (no dashed variables).
- Introduces several state components (**x**, **y**, **z**) and invariants.
- Invariants can be optionally **named** and are collected in **state_inv**.

State Schema and ZStore

State Schema and ZStore

Dwarf

last_proper_state : ProperState

turn_off, turn_on : \mathbb{F} LampId

last_state, current_state : Signal

desired_proper_state : ProperState

$(current_state \setminus turn_off) \cup turn_on = desired_proper_state$

$turn_off \cap turn_on = \emptyset$

State Schema and ZStore

Dwarf

last_proper_state : ProperState
turn_off, turn_on : \mathbb{F} LampId
last_state, current_state : Signal
desired_proper_state : ProperState

$(current_state \setminus turn_off) \cup turn_on = desired_proper_state$
 $turn_off \cap turn_on = \emptyset$

zstore Dwarf =

last_proper_state :: "ProperState"
turn_off :: "LampId set"
turn_on :: "LampId set"
last_state :: "Signal"
current_state :: "Signal"
desired_proper_state :: "ProperState"

where

"(current_state - turn_off) \cup turn_on = signalLamps desired_proper_state"
"turn_on \cap turn_off = {}"

Dwarf Signal Requirements Schema

Dwarf Signal Requirements Schema

NeverShowAll

Dwarf

$current_state \neq \{L1, L2, L3\}$

Dwarf Signal Requirements Schema

NeverShowAll

Dwarf

current_state $\neq \{L1, L2, L3\}$

DwarfSignal

NeverShowAll

MaxOneLampChange

ForbidStopToDrive

DarkOnlyToStop

DarkOnlyFromStop

Dwarf Signal Requirements Schema

NeverShowAll

Dwarf

current_state $\neq \{L1, L2, L3\}$

DwarfSignal

NeverShowAll

MaxOneLampChange

ForbidStopToDrive

DarkOnlyToStop

DarkOnlyFromStop

We deal with these separately in Z Machines.

Overview

- 1 Modelling Z Specifications with Z Machines
- 2 Types, Stores, and State Variables
- 3 Operations and Machines**
- 4 Animation and Verification
- 5 Managing Requirements



Operations

- **zoperation** corresponds to a Z **operation schema**, but restricted.
- Inputs ($a?$) and outputs ($b!$) are modelled by **parameters**.
- Parameters come from **sets**, not types. Can depend on the state.
- Preconditions can depend only on the undashed before state.
- Assignments update variables. Unmentioned variables unchanged.
- State invariants are **not imposed**; we need to prove they are preserved.
- Much closer to an **implementation** of the model.

Operations

- **zoperation** corresponds to a **Z operation schema**, but restricted.
- Inputs ($a?$) and outputs ($b!$) are modelled by **parameters**.
- Parameters come from **sets**, not types. Can depend on the state.
- Preconditions can depend only on the undashed before state.
- Assignments update variables. Unmentioned variables unchanged.
- State invariants are **not imposed**; we need to prove they are preserved.
- Much closer to an **implementation** of the model.

Operations

- **zoperation** corresponds to a **Z operation schema**, but restricted.

```
zoperation Op1 =  
  over state  
  params p1∈"set1" ... pm∈"setm"  
  pre "assert"  
  update "[x' = exp, ..., z' = exp]"
```

- Inputs (*a?*) and outputs (*b!*) are modelled by **parameters**.
- Parameters come from **sets**, not types. Can depend on the state.
- Preconditions can depend only on the undashed before state.
- Assignments update variables. Unmentioned variables unchanged.
- State invariants are **not imposed**; we need to prove they are preserved.
- Much closer to an **implementation** of the model.

Operations

- **zoperation** corresponds to a **Z operation schema**, but restricted.

```
zoperation Op1 =  
  over state  
  params p1∈"set1" ... pm∈"setm"  
  pre "assert"  
  update "[x' = exp, ..., z' = exp]"
```

- Inputs (*a?*) and outputs (*b!*) are modelled by **parameters**.
- Parameters come from **sets**, not types. Can depend on the state.
- Preconditions can depend only on the undashed before state.
- Assignments update variables. Unmentioned variables unchanged.
- State invariants are **not imposed**; we need to prove they are preserved.
- Much closer to an **implementation** of the model.

Operations

- **zoperation** corresponds to a **Z operation schema**, but restricted.

```
zoperation Op1 =  
  over state  
  params p1∈"set1" ... pm∈"setm"  
  pre "assert"  
  update "[x' = exp, ..., z' = exp]"
```

- Inputs (*a?*) and outputs (*b!*) are modelled by **parameters**.
- Parameters come from **sets**, not types. Can depend on the state.
- Preconditions can depend only on the undashed before state.
- Assignments update variables. Unmentioned variables unchanged.
- State invariants are **not imposed**; we need to prove they are preserved.
- Much closer to an **implementation** of the model.

Operations

- **zoperation** corresponds to a **Z operation schema**, but restricted.

```
zoperation Op1 =  
  over state  
  params p1∈"set1" ... pm∈"setm"  
  pre "assert"  
  update "[x' = exp, ..., z' = exp]"
```

- Inputs (*a?*) and outputs (*b!*) are modelled by **parameters**.
- Parameters come from **sets**, not types. Can depend on the state.
- Preconditions can depend only on the undashed before state.
- Assignments update variables. Unmentioned variables unchanged.
- State invariants are **not imposed**; we need to prove they are preserved.
- Much closer to an **implementation** of the model.

Operations

- **zoperation** corresponds to a **Z operation schema**, but restricted.

```
zoperation Op1 =  
  over state  
  params p1∈"set1" ... pm∈"setm"  
  pre "assert"  
  update "[x' = exp, ..., z' = exp]"
```

- Inputs (*a?*) and outputs (*b!*) are modelled by **parameters**.
- Parameters come from **sets**, not types. Can depend on the state.
- Preconditions can depend only on the undashed before state.
- Assignments update variables. Unmentioned variables unchanged.
- State invariants are **not imposed**; we need to prove they are preserved.
- Much closer to an **implementation** of the model.

Operations

- **zoperation** corresponds to a **Z operation schema**, but restricted.

```
zoperation Op1 =  
  over state  
  params p1∈"set1" ... pm∈"setm"  
  pre "assert"  
  update "[x' = exp, ..., z' = exp]"
```

- Inputs (*a?*) and outputs (*b!*) are modelled by **parameters**.
- Parameters come from **sets**, not types. Can depend on the state.
- Preconditions can depend only on the undashed before state.
- Assignments update variables. Unmentioned variables unchanged.
- State invariants are **not imposed**; we need to prove they are preserved.
- Much closer to an **implementation** of the model.

Operations

- **zoperation** corresponds to a **Z operation schema**, but restricted.

```
zoperation Op1 =  
  over state  
  params p1∈"set1" ... pm∈"setm"  
  pre "assert"  
  update "[x' = exp, ..., z' = exp]"
```

- Inputs (*a?*) and outputs (*b!*) are modelled by **parameters**.
- Parameters come from **sets**, not types. Can depend on the state.
- Preconditions can depend only on the undashed before state.
- Assignments update variables. Unmentioned variables unchanged.
- State invariants are **not imposed**; we need to prove they are preserved.
- Much closer to an **implementation** of the model.

Operation to Set New Proper State

Operation to Set New Proper State

SetNewProperState

$\Delta DwarfSignal$

$st? : ProperState$

$current_state = desired_proper_state$

$st? \neq current_state$

$last_proper_state' = current_state$

$turn_off' = current_state \setminus st?$

$turn_on' = st? \setminus current_state$

$last_state' = current_state$

$current_state' = current_state$

$desired_proper_state' = st?$

Operation to Set New Proper State

SetNewProperState

Δ DwarfSignal

st? : ProperState

current_state = desired_proper_state

st? \neq current_state

last_proper_state' = current_state

turn_off' = current_state \ st?

turn_on' = st? \ current_state

last_state' = current_state

current_state' = current_state

desired_proper_state' = st?

```
zoperation SetNewProperState =  
  over Dwarf  
  params st ∈ "ProperState -  
    {desired_proper_state}"  
  pre "current_state =  
    signalLamps desired_proper_state"  
  update "[  
    last_proper_state' =  
      desired_proper_state  
    , turn_off' =  
      current_state - signalLamps st  
    , turn_on' =  
      signalLamps st - current_state  
    , last_state' =  
      current_state  
    , desired_proper_state' = st]"
```

Overview

- 1 Modelling Z Specifications with Z Machines
- 2 Types, Stores, and State Variables
- 3 Operations and Machines
- 4 Animation and Verification**
- 5 Managing Requirements



Machines and Animation

- Often implicit in the Z notation; the set of all operations.
- Generates an **interaction tree** semantics, for animation.
- Animate using `animate` command. Useful for **design-space exploration**.
- Sets initial state, checks which operations + parameters are enabled.
- For animation, parameters should typically be drawn from a **finite** set.

Machines and Animation

- Often implicit in the Z notation; the set of all operations.
- Generates an **interaction tree** semantics, for animation.
- Animate using `animate` command. Useful for **design-space exploration**.
- Sets initial state, checks which operations + parameters are enabled.
- For animation, parameters should typically be drawn from a **finite** set.

Machines and Animation

- Often implicit in the Z notation; the set of all operations.

```
zmachine machine =  
  over state  
  init "[x' = exp, ..., z' = exp]"  
  operations Op1 Op2 ... Opn
```

```
animate machine
```

- Generates an **interaction tree** semantics, for animation.
- Animate using `animate` command. Useful for **design-space exploration**.
- Sets initial state, checks which operations + parameters are enabled.
- For animation, parameters should typically be drawn from a **finite** set.

Machines and Animation

- Often implicit in the Z notation; the set of all operations.

```
zmachine machine =  
  over state  
  init "[x' = exp, ..., z' = exp]"  
  operations Op1 Op2 ... Opn
```

```
animate machine
```

- Generates an **interaction tree** semantics, for animation.
- Animate using `animate` command. Useful for **design-space exploration**.
- Sets initial state, checks which operations + parameters are enabled.
- For animation, parameters should typically be drawn from a **finite** set.

Machines and Animation

- Often implicit in the Z notation; the set of all operations.

```
zmachine machine =  
  over state  
  init "[x' = exp, ..., z' = exp]"  
  operations Op1 Op2 ... Opn
```

```
animate machine
```

- Generates an **interaction tree** semantics, for animation.
- Animate using `animate` command. Useful for **design-space exploration**.
- Sets initial state, checks which operations + parameters are enabled.
- For animation, parameters should typically be drawn from a **finite** set.

Machines and Animation

- Often implicit in the Z notation; the set of all operations.

```
zmachine machine =  
  over state  
  init "[x' = exp, ..., z' = exp]"  
  operations Op1 Op2 ... Opn
```

```
animate machine
```

- Generates an **interaction tree** semantics, for animation.
- Animate using `animate` command. Useful for **design-space exploration**.
- Sets initial state, checks which operations + parameters are enabled.
- For animation, parameters should typically be drawn from a **finite** set.

Machines and Animation

- Often implicit in the Z notation; the set of all operations.

```
zmachine machine =  
  over state  
  init "[x' = exp, ..., z' = exp]"  
  operations Op1 Op2 ... Opn
```

```
animate machine
```

- Generates an **interaction tree** semantics, for animation.
- Animate using `animate` command. Useful for **design-space exploration**.
- Sets initial state, checks which operations + parameters are enabled.
- For animation, parameters should typically be drawn from a **finite** set.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).
- Machine verification requires we demonstrate:
 - The initial state assignment **establishes** the invariants.
 - Each operation **preserves** the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method **zpog_full**.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and **explore**.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).
- Machine verification requires we demonstrate:
 - The initial state assignment establishes the invariants.
 - Each operation preserves the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method **zpog_full**.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and **explore**.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).

lemma Init_inv: "Init establishes state_inv"

lemma Op1_inv: "Op1 (x, y, z) preserves state_inv"

- Machine verification requires we demonstrate:
 - ① The initial state assignment establishes the invariants.
 - ② Each operation preserves the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method **zpog_full**.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and **explore**.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).

lemma Init_inv: "Init establishes state_inv"

lemma Op1_inv: "Op1 (x, y, z) preserves state_inv"

- Machine verification requires we demonstrate:
 - 1 The initial state assignment **establishes** the invariants.
 - 2 Each operation **preserves** the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method **zpog_full**.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and **explore**.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).

lemma Init_inv: "Init establishes state_inv"

lemma Op1_inv: "Op1 (x, y, z) preserves state_inv"

- Machine verification requires we demonstrate:
 - 1 The initial state assignment **establishes** the invariants.
 - 2 Each operation **preserves** the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method **zpog_full**.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and **explore**.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).

lemma Init_inv: "Init establishes state_inv"

lemma Op1_inv: "Op1 (x, y, z) preserves state_inv"

- Machine verification requires we demonstrate:
 - 1 The initial state assignment **establishes** the invariants.
 - 2 Each operation **preserves** the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method **zpog_full**.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and **explore**.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).

lemma Init_inv: "Init establishes state_inv"

lemma Op1_inv: "Op1 (x, y, z) preserves state_inv"

- Machine verification requires we demonstrate:
 - 1 The initial state assignment **establishes** the invariants.
 - 2 Each operation **preserves** the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method *zpog_full*.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and *explore*.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).

lemma Init_inv: "Init establishes state_inv"

lemma Op1_inv: "Op1 (x, y, z) preserves state_inv"

- Machine verification requires we demonstrate:
 - 1 The initial state assignment **establishes** the invariants.
 - 2 Each operation **preserves** the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method **zpog_full**.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and **explore**.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).

lemma Init_inv: "Init establishes state_inv"

lemma Op1_inv: "Op1 (x, y, z) preserves state_inv"

- Machine verification requires we demonstrate:
 - 1 The initial state assignment **establishes** the invariants.
 - 2 Each operation **preserves** the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method **zpog_full**.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and **explore**.

Verification and Proof Obligations

- Requirements specified as **machine invariants** (cf. *DwarfSignal*).

lemma Init_inv: "Init establishes state_inv"

lemma Op1_inv: "Op1 (x, y, z) preserves state_inv"

- Machine verification requires we demonstrate:
 - 1 The initial state assignment **establishes** the invariants.
 - 2 Each operation **preserves** the invariants.
- Specified as **Hoare conjectures**: $\{state_inv\} Op1(x, y, z) \{state_inv\}$.
- Proof obligations can be generated using method **zpog_full**.
- **Weakest preconditions**; every operation is a constrained assignment.
- For large models, manage proof obligations using Isar and **explore**.

Example: Dwarf Operations

```
lemma "Init establishes Dwarf_inv"  
  by zpog_full
```

```
lemma "(SetNewProperState p) preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOn l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOff l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

Example: Dwarf Operations

```
lemma "Init establishes Dwarf_inv"  
  by zpog_full
```

```
lemma "(SetNewProperState p) preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOn l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOff l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

Example: Dwarf Operations

```
lemma "Init establishes Dwarf_inv"  
  by zpog_full
```

```
lemma "(SetNewProperState p) preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOn l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOff l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

Example: Dwarf Operations

```
lemma "Init establishes Dwarf_inv"  
  by zpog_full
```

```
lemma "(SetNewProperState p) preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOn l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOff l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

Example: Dwarf Operations

```
lemma "Init establishes Dwarf_inv"  
  by zpog_full
```

```
lemma "(SetNewProperState p) preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOn l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

```
lemma "TurnOff l preserves Dwarf_inv"  
  by (zpog_full; auto)
```

Overview

- 1 Modelling Z Specifications with Z Machines
- 2 Types, Stores, and State Variables
- 3 Operations and Machines
- 4 Animation and Verification
- 5 **Managing Requirements**



Managing Requirements

- Requirements characterised by named assertions using `zexpr`.
- System documentation and **safety argumentation**.
- Show that each operation preserves safety invariants (or fails to).
- `Req1Failed` is a **litmus test** for `req1`; enabled if `req1` fails.
- Test using animator. Verify using proof obligation generator.

Managing Requirements

- Requirements characterised by named assertions using `zexpr`.
- System documentation and **safety argumentation**.
- Show that each operation preserves safety invariants (or fails to).
- `Req1Failed` is a **litmus test** for `req1`; enabled if `req1` fails.
- Test using animator. Verify using proof obligation generator.

Managing Requirements

- Requirements characterised by named assertions using `zexpr`.

```
zexpr req1 is "assert"
```

```
zexpr req2 is "assert"
```

```
zoperation Req1Failed =
```

```
  over state
```

```
  pre " $\neg$  req1"
```

- System documentation and safety argumentation.
- Show that each operation preserves safety invariants (or fails to).
- `Req1Failed` is a litmus test for `req1`; enabled if `req1` fails.
- Test using animator. Verify using proof obligation generator.

Managing Requirements

- Requirements characterised by named assertions using `zexpr`.

```
zexpr req1 is "assert"
```

```
zexpr req2 is "assert"
```

```
zoperation Req1Failed =
```

```
  over state
```

```
  pre " $\neg$  req1"
```

- System documentation and **safety argumentation**.
- Show that each operation preserves safety invariants (or fails to).
- `Req1Failed` is a **litmus test** for `req1`; enabled if `req1` fails.
- Test using animator. Verify using proof obligation generator.

Managing Requirements

- Requirements characterised by named assertions using `zexpr`.

```
zexpr req1 is "assert"
```

```
zexpr req2 is "assert"
```

```
zoperation Req1Failed =
```

```
  over state
```

```
  pre " $\neg$  req1"
```

- System documentation and **safety argumentation**.
- Show that each operation preserves safety invariants (or fails to).
- `Req1Failed` is a **litmus test** for `req1`; enabled if `req1` fails.
- Test using animator. Verify using proof obligation generator.

Managing Requirements

- Requirements characterised by named assertions using `zexpr`.

```
zexpr req1 is "assert"
```

```
zexpr req2 is "assert"
```

```
zoperation Req1Failed =
```

```
  over state
```

```
  pre " $\neg$  req1"
```

- System documentation and **safety argumentation**.
- Show that each operation preserves safety invariants (or fails to).
- **Req1Failed** is a **litmus test** for **req1**; enabled if **req1** fails.
- Test using animator. Verify using proof obligation generator.

Managing Requirements

- Requirements characterised by named assertions using `zexpr`.

```
zexpr req1 is "assert"
```

```
zexpr req2 is "assert"
```

```
zoperation Req1Failed =
```

```
  over state
```

```
  pre " $\neg$  req1"
```

- System documentation and **safety argumentation**.
- Show that each operation preserves safety invariants (or fails to).
- **Req1Failed** is a **litmus test** for **req1**; enabled if **req1** fails.
- Test using animator. Verify using proof obligation generator.

Example: Dwarf Signal Requirements

Example: Dwarf Signal Requirements

```
zexpr NeverShowAll  
  is "current_state  $\neq$  {L1, L2, L3}"
```

Example: Dwarf Signal Requirements

```
zexpr NeverShowAll
```

```
  is "current_state  $\neq$  {L1, L2, L3}"
```

```
zexpr ForbidStopToDrive
```

```
  is "(last_proper_state = stop  $\longrightarrow$  desired_proper_state  
     $\neq$  drive)"
```

Example: Dwarf Signal Requirements

```
zexpr NeverShowAll
  is "current_state  $\neq$  {L1, L2, L3}"

zexpr ForbidStopToDrive
  is "(last_proper_state = stop  $\longrightarrow$  desired_proper_state
       $\neq$  drive)"

lemma "(SetNewProperState p) preserves NeverShowAll"
  by zpog_full
```

Example: Dwarf Signal Requirements

```
zexpr NeverShowAll
  is "current_state  $\neq$  {L1, L2, L3}"

zexpr ForbidStopToDrive
  is "(last_proper_state = stop  $\longrightarrow$  desired_proper_state
     $\neq$  drive)"

lemma "(SetNewProperState p) preserves NeverShowAll"
  by zpog_full

lemma "TurnOn l preserves NeverShowAll"
  apply zpog_full
  quickcheck
```

Example: Dwarf Signal Requirements

```
zexpr NeverShowAll
  is "current_state  $\neq$  {L1, L2, L3}"

zexpr ForbidStopToDrive
  is "(last_proper_state = stop  $\longrightarrow$  desired_proper_state
     $\neq$  drive)"

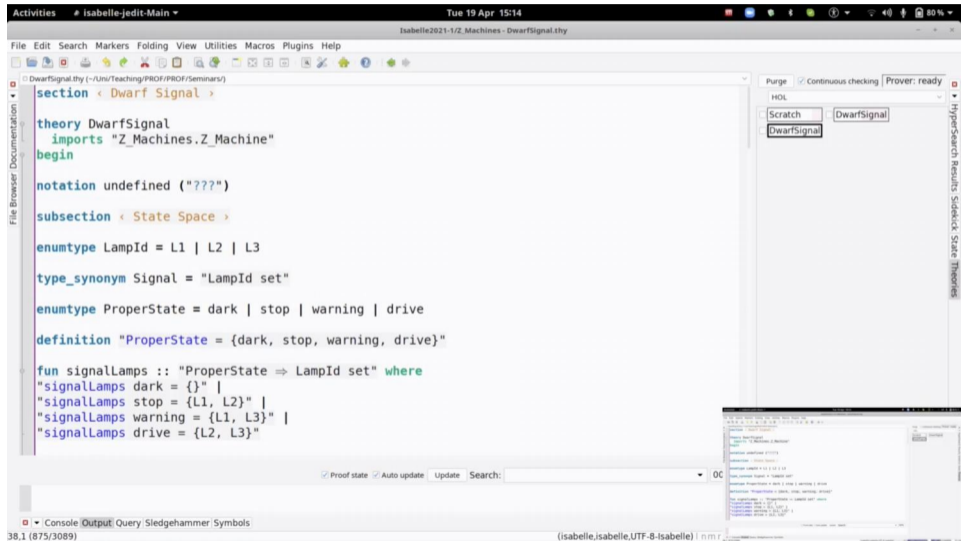
lemma "(SetNewProperState p) preserves NeverShowAll"
  by zpog_full

lemma "TurnOn l preserves NeverShowAll"
  apply zpog_full
  quickcheck

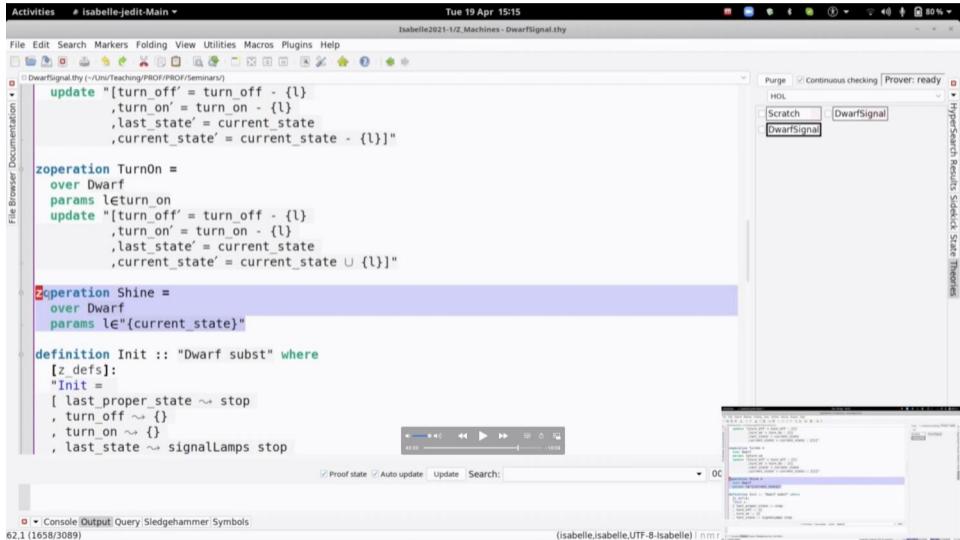
lemma "(SetNewProperState p) preserves ForbidStopToDrive"
  apply zpog_full
  quickcheck
```

Checking the Specification

Checking the Specification



Checking the Specification



Checking the Specification

The screenshot displays the Isabelle2021-1 Z/Machines - DwarfSignal.thy file in a text editor. The main window shows the following code:

```
zoperation Shine =  
  over Dwarf  
  params le "{current_state}"  
  
definition Init :: "Dwarf subst" where  
  [z defs]:  
  "Init =  
  [ last_proper_state ~ stop  
  , turn_off ~ {}  
  , turn_on ~ {}  
  , last_state ~ signalLamps stop  
  , current_state ~ signalLamps stop  
  , desired_proper_state ~ stop ]"  
  
zmachine DwarfSignal =  
  init Init  
  operations SetNewProperState TurnOn TurnOff Shine  
  
zexpr NeverShowAll  
  is "current_state ≠ {L1, L2, L3}"  
  
zexpr MaxOneLampChange  
  is "???"  
  
consts  
  Init :: "Dwarf ⇒ Dwarf"
```

The right sidebar shows the "HyperSearch Results Sidekick State Theories" panel with a search for "DwarfSignal" and a "Prover: ready" status. The bottom status bar indicates the current position is 68,11 (1781/3089) and the current theory is (isabelle.isabelle.UTF-8-isabelle) in m.r.

Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"

zmachine DwarfSignal =
  init Init
  operations SetNewProperState TurnOn TurnOff Shine

animate

zexpr NeverShowAll
  is "current_state ≠ {L1, L2, L3}"
```

The interface includes a file browser on the left, a theorem prover status bar at the top right showing "Prover: ready", and a console output area at the bottom. The status bar also shows "80.9 (2050/3099)" and "(isabelle,isabelle,UTF-8-Isabelle) | nmr".

Checking the Specification

The screenshot shows the Isabelle/ML editor interface. The main window displays the file `DwarfSignal.thy` with the following code:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"

zmachine DwarfSignal =
  init Init
  operations SetNewProperState TurnOn TurnOff Shine

animate DwarfSignal

zexpr NeverShowAll
  is "current_state ≠ {L1, L2, L3}"
```

The right sidebar shows the `HyperSearch Results` and `Sidekick State Theories` panels. The `Sidekick State Theories` panel lists `HOL`, `Scratch`, and `DwarfSignal`. The `HyperSearch Results` panel shows a search for `current_state` with results for `current_state` and `current_state`.

The bottom status bar shows the console output: `80,20 (2061/3110) Input/output complete (isabelle,isabelle,UTF-8-Isabelle) in m r o U G`.

Checking the Specification

Activities Isabelle-jedit-Main Tue 19 Apr 15:16
Isabelle2021-1/Z_Machines - DwarfSignal.thy (modified)

File Edit Search Markers Folding View Utilities Macros Plugins Help

DwarfSignal.thy (~/Uni/Teaching/PROF/PROF/Seminars/)

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"

zmachine DwarfSignal =
  init Init
  operations SetNewProperState TurnOn TurnOff Shine
```

Purge ☒ Continuous checking Prover: ready
HOL
☐ Scratch ☒ DwarfSignal
☐ DwarfSignal

File Browser Documentation HyperSearch Results Sidekick State Theories

☒ Proof state ☒ Auto update Update Search: 00%

See theory exports
Compiling animation...
See theory exports
Start animation

80,20 (2061/3110) Input/output complete (isabelle,isabelle,UTF-8-Isabelle) in m r o U G

Checking the Specification

The screenshot shows the Isabelle/ML editor interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"

zmachine DwarfSignal =
  init Init
  operations SetNewProperState TurnOn TurnOff Shine
```

The interface includes a file browser on the left, a command area at the bottom with the text "See theory exports", "Compiling animation...", "See theory exports", and "Start animation" (highlighted in green). A progress bar at the bottom indicates 00% completion. The console area at the bottom shows the output of the compilation process.

Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The top bar indicates the date and time as 'Tue 19 Apr 15:16'. The main window displays the file 'DwarfSignal.thy' with the following content:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"
```

On the right side, there is a 'HyperSearch Results Sidekick' panel showing a search for 'DwarfSignal' with results for 'HOL', 'Scratch', and 'DwarfSignal'.

The bottom panel shows the 'System' window with the following text:

```
Press %<TAB> to list built-in commands.
Run built-in with --help argument to get a brief usage message.
Run %help to view Console plugin online help.

Errors generated by compilers and some other programs are listed
for easy one-click access in the 'Plugins->Error List->Error List'
window.
~/Uni/Teaching/PROF/PROF/Seminars/> No process is currently running
~/Uni/Teaching/PROF/PROF/Seminars/>

$ISABELLE_TMP_PREFIX/process13243906329326067755/itree te> ./Simulation
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree te>
Starting ITree Simulation...
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4)
SetNewProperState Drive;
```

The bottom status bar shows the console output: '80,20 (2061/3110) Input/output complete (isabelle,isabelle,UTF-8-Isabelle) in m r e U G 1 t a'.

Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The main window displays the file `DwarfSignal.thy` with the following definition:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"
```

The right sidebar shows the "HyperSearch Results Sidekick State Theories" panel with the following selection:

- ☒ HOL
- ☐ Scratch
- ☐ DwarfSignal

The bottom panel shows the console output:

```
Run %help to view Console plugin online help.

Errors generated by compilers and some other programs are listed
for easy one-click access in the 'Plugins->Error List->Error List'
window.
~/Uni/Teaching/PROF/PROF/Seminars/> No process is currently running
~/Uni/Teaching/PROF/PROF/Seminars/>

$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10124508/simulate> ./Simulation
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10124508/simulate>
Starting ITree Simulation...
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState
SetNewProperState Drive;
3
Internal Activity...
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L2; (3) TurnOn L3;
```

The bottom status bar shows the file path: `(isabelle,isabelle,UTF-8-Isabelle) in mro UG 1 ta`.

Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"
```

The right sidebar shows the `HyperSearch Results` and `Sidekick State Theories` panels. The `Sidekick State Theories` panel lists `HOL`, `Scratch`, and `DwarfSignal`.

The bottom panel shows the `Console` output, which includes the following text:

```
for easy one-click access in the 'Plugins->Error List->Error List'
window.
~/Uni/Teaching/PROF/PROF/Seminars/> No process is currently running
~/Uni/Teaching/PROF/PROF/Seminars/>

$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10124508/simulate> ./Simulation
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10124508/simulate>
Starting ITree Simulation...
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4)
SetNewProperState Drive;
3
Internal Activity...
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L2; (3) T
2
Internal Activity...
Events: (1) Shine (Set [L1]); (2) TurnOn L3;
```

The bottom status bar shows the file path `69.29 (1810/3110)` and the encoding `(isabelle,isabelle,UTF-8-Isabelle)`.

Checking the Specification

The screenshot shows the Isabelle/THY IDE interface. The main window displays the source file `DwarfSignal.thy` with the following definition:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"
```

The right sidebar shows the `HyperSearch Results` and `State Theories` panels. The `State Theories` panel lists `HOL`, `Scratch`, and `DwarfSignal`.

The bottom panel shows the `Console Output` with the following text:

```
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10124508/simulate> ./Simulation
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10124508/simulate>
Starting ITree Simulation...
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4)
SetNewProperState Drive;
3
Internal Activity...
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L2; (3) TurnOn L3;
2
Internal Activity...
Events: (1) Shine (Set [L1]); (2) TurnOn L3;
2
Internal Activity...
Events: (1) Shine (Set [L1,L3]); (2) SetNewProperState Dark; (3) SetNewProperState Stop; (4)
SetNewProperState Drive;
```

The bottom status bar shows the file path `69.29 (1810/3110)` and the encoding `(isabelle,isabelle,UTF-8-Isabelle) in mro UG 1 to`.

Checking the Specification

The screenshot shows the Isabelle2021-1/2_Machines - DwarfSignal.thy (modified) file in the Isabelle IDE. The main window displays the following definition:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"
```

The right sidebar shows the "HyperSearch Results Sidekick State Theories" panel with the following entries:

- ☐ HOL
- ☐ Scratch
- ☐ DwarfSignal

The bottom panel shows the "System" output, which is a simulation log:

```
Starting ITree Simulation...
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4)
SetNewProperState Drive;
3
Internal Activity...
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L2; (3) TurnOn L3;
2
Internal Activity...
Events: (1) Shine (Set [L1]); (2) TurnOn L3;
2
Internal Activity...
Events: (1) Shine (Set [L1,L3]); (2) SetNewProperState
SetNewProperState Drive;
4
Internal Activity...
Events: (1) Shine (Set [L1,L3]); (2) TurnOff L1; (3) TurnOn L2;
```

The bottom status bar shows the console output: 69.29 (1810/3110) (isabelle,isabelle.UTF-8-Isabelle) in mro UG 1 ta.

Checking the Specification

The screenshot shows the Isabelle/ML editor interface. The main window displays the file `DwarfSignal.thy` with the following definition:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"
```

The right sidebar shows the `HyperSearch Results Sidekick State Theories` panel with the following entries:

- ☒ Purge
- ☒ Continuous checking
- Prover: ready
- HOL
- ☐ Scratch
- ☒ DwarfSignal

The bottom panel shows the `System` trace for the definition:

```
3
Internal Activity...
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L2; (3) TurnOn L3;
2
Internal Activity...
Events: (1) Shine (Set [L1]); (2) TurnOn L3;
2
Internal Activity...
Events: (1) Shine (Set [L1,L3]); (2) SetNewProperState Dark; (3) SetNewProperState Stop; (4)
SetNewProperState Drive;
4
Internal Activity...
Events: (1) Shine (Set [L1,L3]); (2) TurnOff L1; (3) T
2
Internal Activity...
Events: (1) Shine (Set [L3]); (2) TurnOn L2;
2
```

The bottom status bar shows the console output: `69.29 (1810/3110)` and the file path: `(isabelle,isabelle.UTF-8-Isabelle) in mro UGC 1 to`.

Checking the Specification

The screenshot displays the Isabelle2021-1/2_Machines - DwarfSignal.thy (modified) file in a text editor. The main window shows the following definition:

```
definition Init :: "Dwarf subst" where
  [z_defs]:
  "Init =
  [ last_proper_state ~ stop
  , turn_off ~ {}
  , turn_on ~ {}
  , last_state ~ signalLamps stop
  , current_state ~ signalLamps stop
  , desired_proper_state ~ stop ]"
```

The right sidebar shows the 'HyperSearch Results Sidekick State Theories' panel with the following entries:

- ☒ Purge
- ☒ Continuous checking
- ☒ Prover: ready
- ☒ HOL
- ☒ Scratch
- ☒ DwarfSignal

The bottom panel shows the 'System' output, which is a sequence of events:

```
Internal Activity...
Events: (1) Shine (Set [L1]); (2) TurnOn L3;
2
Internal Activity...
Events: (1) Shine (Set [L1,L3]); (2) SetNewProperState Dark; (3) SetNewProperState Stop; (4)
SetNewProperState Drive;
4
Internal Activity...
Events: (1) Shine (Set [L1,L3]); (2) TurnOff L1; (3) TurnOn L2;
2
Internal Activity...
Events: (1) Shine (Set [L3]); (2) TurnOn L2;
2
Internal Activity...
Events: (1) Shine (Set [L3,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Stop; (4)
SetNewProperState Warning;
```

The bottom status bar shows the console output: 69.29 (1810/3110) (isabelle.isabelle.UTF-8-Isabelle) in mro UG 1 to

Checking the Specification

The screenshot shows the Isabelle/THY editor interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
init Init
operations SetNewProperState TurnOn TurnOff Shine

animate DwarfSignal

zexpr NeverShowAll
is "current_state  $\neq$  {L1, L2, L3}"

zexpr MaxOneLampChange
is "???"

zexpr ForbidStopToDrive
is "(last_proper_state = stop  $\longrightarrow$  desired_proper_state  $\neq$  drive)"

zexpr DarkOnlyToStop
is "???"

zexpr DarkOnlyFromStop
is "???"

zexpr DwarfReq
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"
```

The right sidebar shows the `HyperSearch Results` for `State Theories`, listing `HOL`, `Scratch`, and `DwarfSignal`. The bottom status bar displays the `System` output:

```
Internal Activity...
Events: (1) Shine (Set [L1]); (2) TurnOn L3;
2
95,11 (2310/3110)
```

The bottom right corner shows the `Console Output` and `Query` tabs, with the `Query` tab selected, displaying the `HyperSearch Results` for `State Theories`.

Checking the Specification

The screenshot displays the Isabelle2021-1/2_Machines - DwarfSignal.thy (modified) window. The main editor shows the following code:

```
init Init
operations SetNewProperState TurnOn TurnOff Shine

animate DwarfSignal

zexpr NeverShowAll
is "current_state  $\neq$  {L1, L2, L3}"

zexpr MaxOneLampChange
is "???"

zexpr ForbidStopToDrive
is "(last_proper_state = stop  $\longrightarrow$  desired_proper_state  $\neq$  drive)"

zexpr DarkOnlyToStop
is "???"

zexpr DarkOnlyFromStop
is "???"

zexpr DwarfReq
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"
```

The right sidebar shows the HyperSearch Results Sidekick State Theories, with a search for "DwarfSignal" and a "Prover: ready" status.

The bottom console output shows the following text:

```
System
Internal Activity...
Events: (1) Shine (Set [L1]); (2) TurnOn L3;
2
```

The bottom status bar indicates the file path: 89,63 (2240/3110) (isabelle,isabelle,UTF-8-Isabelle) in mro UG 1 to.

Checking the Specification

The screenshot displays the Isabelle2021-1/2_Machines - DwarfSignal.thy (modified) file in a text editor. The main window shows the following code:

```
is "???"

zexpr ForbidStopToDrive
is "(last_proper_state = stop  $\rightarrow$  desired_proper_state  $\neq$  drive)"

zexpr DarkOnlyToStop
is "???"

zexpr DarkOnlyFromStop
is "???"

zexpr DwarfReq
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"

zoperation ViolNeverShowAll =
pre " $\neg$  NeverShowAll"

zmachine DwarfSignalTest =
init Init
operations SetNewProperState TurnOn TurnOff Shine ViolNeverShowAll

animate DwarfSignalTest
```

The right sidebar shows the HyperSearch Results Sidekick State Theories, with a list of theories including HOL, Scratch, and DwarfSignal. The bottom status bar indicates the system is running, showing internal activity and events: (1) Shine (Set [L1]); (2) TurnOn L3; 2.

101,23 (2475/3110) (isabelle,isabelle,UTF-8-Isabelle) in mro UG 1 to

Checking the Specification

The screenshot displays the Isabelle2021-1/2_Machines - DwarfSignal.thy (modified) file in a text editor. The file contains several definitions and lemmas for a DwarfSignal system.

```
zexpr DarkOnlyToStop
is "???"

zexpr DarkOnlyFromStop
is "???"

zexpr DwarfReq
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"

zoperation ViolNeverShowAll =
pre " $\neg$  NeverShowAll"

zmachine DwarfSignalTest =
init Init
operations SetNewProperState TurnOn TurnOff Shine ViolNeverShowAll

animate DwarfSignalTest

lemma "Init establishes Dwarf_inv"
by zpog_full

lemma "TurnOn l preserves Dwarf_inv"
by (zpog_full, auto)
```

The right sidebar shows the HyperSearch Results Sidekick State Theories, with DwarfSignal selected. The bottom status bar indicates the system is in the 'System' state, showing internal activity and events: (1) Shine (Set [L1]); (2) TurnOn L3; 2.

105,24 (2589/3089) (isabelle,isabelle,UTF-8-Isabelle) in mro UG 1 to

Checking the Specification

The screenshot shows the Isabelle/THY editor interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
zexpr DwarfReq
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"

zoperation ViolNeverShowAll =
pre " $\neg$  NeverShowAll"

zmachine DwarfSignalTest =
init Init
operations SetNewProperState TurnOn TurnOff Shine ViolNeverShowAll

animate DwarfSignalTest

lemma "Init establishes Dwarf_inv"
by zpg_full

lemma "TurnOn l preserves Dwarf_inv"
by (zpg_full, auto)
```

The right sidebar shows the "HyperSearch Results Sidekick State Theories" panel with a search for "DwarfSignal" and a "Purge" button.

The bottom console shows the execution of the simulation:

```
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10168136/simulate> ./Simulation
Process
/tmp/isabelle-simonfoster/process13243906329326067755/itree-simulate10124508/simulate/Simulation
exited with code 834
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10168136/simulate>
Starting ITree Simulation...
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4)
SetNewProperState Drive;
```

The status bar at the bottom indicates "105,24 (2589/3089) Input/output complete (isabelle.isabelle.UTF-8-Isabelle) in m r o U G 1 t n".

Checking the Specification

The screenshot displays the Isabelle/Isabelle IDE interface. The main window shows a formal specification in the `DwarfSignal.thy` file, which includes a `zexpr` definition, a `zoperation` for `ViolNeverShowAll`, a `zmachine` for `DwarfSignalTest`, and two lemmas. The `animate` command is used to execute the simulation. The console output shows the execution results, including the initial state, events, and internal activities.

File Browser Documentation

```
zexpr DwarfReq
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"

zoperation ViolNeverShowAll =
pre " $\neg$  NeverShowAll"

zmachine DwarfSignalTest =
init Init
operations SetNewProperState TurnOn TurnOff Shine ViolNeverShowAll

animate DwarfSignalTest

lemma "Init establishes Dwarf_inv"
by zpg_full

lemma "TurnOn l preserves Dwarf_inv"
by (zpg_full, auto)
```

System

```
exited with code 834
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10168136/simulate>
Starting ITree Simulation...
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4)
SetNewProperState Drive;
4
Internal Activity...
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L1; (3) TurnOn L3;
```

HyperSearch Results Sidebar: State Theories

Purge ☒ Continuous checking Prover: ready

HOL

☐ Scratch ☐ DwarfSignal

☐ DwarfSignal

Console Output Query Sledgehammer Symbols

105,24 (2589/3089) (isabelle.isabelle.UTF-8-Isabelle) in mro UG > 1 to

Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The main window displays a formal specification for a DwarfSignal system. The specification includes a `zexpr` for `DwarfReq`, a `zoperation` for `ViolNeverShowAll`, a `zmachine` for `DwarfSignalTest`, and two lemmas. The `animate` command is used to execute the model, and the console shows the resulting execution trace.

```
zexpr DwarfReq
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"

zoperation ViolNeverShowAll =
pre " $\neg$  NeverShowAll"

zmachine DwarfSignalTest =
init Init
operations SetNewProperState TurnOn TurnOff Shine ViolNeverShowAll

animate DwarfSignalTest

lemma "Init establishes Dwarf_inv"
by zpg_full

lemma "TurnOn l preserves Dwarf_inv"
by (zpg_full, auto)
```

System

Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4) SetNewProperState Drive;

4

Internal Activity...

Events: (1) Shine (Set [L1,L2]); (2) TurnOff L1; (3) TurnOn L3;

3

Internal Activity...

Events: (1) ViolNeverShowAll (); (2) Shine (Set [L2,L1,L3]); (3) TurnOff L1;

105,24 (2589/3089) (isabelle,isabelle.UTF-8-Isabelle) in mro UG 1 to

Checking the Specification

The screenshot shows the Isabelle/ML editor interface. The main window displays the file `DwarfSignal.thy` with the following lemmas:

```
lemma "(SetNewProperState p) preserves Dwarf_inv"
  by (zpgog_full; auto)

lemma "(SetNewProperState p) preserves NeverShowAll"
  by zpgog_full

lemma "(SetNewProperState p) preserves ForbidStopToDrive"
  apply zpgog_full
  quickcheck
  oops

lemma "TurnOn l preserves NeverShowAll"
  apply zpgog_full
  quickcheck
  oops

end
```

The right sidebar shows the `HyperSearch Results Sidekick State Theories` panel with the following entries:

- HOL
- Scratch
- DwarfSignal

The bottom panel shows the `System` output:

```
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4) SetNewProperState Drive;
4
Internal Activity...
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L1; (3) TurnOn L3;
3
Internal Activity...
Events: (1) ViolNeverShowAll (); (2) Shine (Set [L2,L1,L3]); (3) TurnOff L1;
```

The status bar at the bottom indicates the file is `(isabelle,isabelle,UTF-8-Isabelle)` in `improUG` 1 to 127.40 (3046/3089).

Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
params leturn_off
update "[turn_off' = turn_off - {l}
      ,turn_on' = turn_on - {l}
      ,last_state' = current_state
      ,current_state' = current_state - {l}]"

zoperation TurnOn =
over Dwarf
params leturn_on
update "[turn_off' = turn_off - {l}
      ,turn_on' = turn_on - {l}
      ,last_state' = current_state
      ,current_state' = current_state  $\cup$  {l}]"

zoperation Shine =
over Dwarf
params leturn_on
update "[turn_off' = turn_off - {l}
      ,turn_on' = turn_on - {l}
      ,last_state' = current_state
      ,current_state' = current_state  $\cup$  {l}]"
```

The right sidebar shows the `HyperSearch Results` and `State Theories` panels. The `State Theories` panel lists `HOL`, `Scratch`, and `DwarfSignal`.

The bottom panel shows the `Console Output` with the following text:

```
System
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4)
SetNewProperState Drive;
4
Internal Activity...
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L1; (3) TurnOn L3;
3
Internal Activity...
Events: (1) ViolNeverShowAll (); (2) Shine (Set [L2,L1,L3]); (3) TurnOff L1;
```

The status bar at the bottom indicates the file is `(isabelle,isabelle,UTF-8-Isabelle)` in `nmroUGs` at line `1` and column `10`.

Checking the Specification

The screenshot shows the Isabelle/THY editor interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
by (zpog_full; auto)

lemma "(SetNewProperState p) preserves NeverShowAll"
  by zpog_full

lemma "(SetNewProperState p) preserves ForbidStopToDrive"
  apply zpog_full
  quickcheck
  oops

lemma "TurnOn l preserves NeverShowAll"
  apply zpog_full
  quickcheck
  oops

end

proof (prove)
goal (1 subgoal):
1. {NeverShowAll} TurnOn l {NeverShowAll}
```

The right sidebar shows the "HyperSearch Results Sidekick State Theories" panel with a search bar and a list of theories including `HOL`, `Scratch`, and `DwarfSignal`. The bottom status bar shows the console output window with the text "127.40 (3046/3089)" and "(isabelle,isabelle,UTF-8-Isabelle) in mro UG 1 to".

Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
by (zpog_full; auto)

lemma "(SetNewProperState p) preserves NeverShowAll"
  by zpog_full

lemma "(SetNewProperState p) preserves ForbidStopToDrive"
  apply zpog_full
  quickcheck
  oops

lemma "TurnOn l preserves NeverShowAll"
  apply zpog_full
  quickcheck
  oops
end

proof (prove)
goal (1 subgoal):
1.  $\bigwedge \text{turn\_on current\_state. current\_state} \neq \{L1, L2, L3\} \Rightarrow l \in \text{turn\_on} \Rightarrow \text{insert } l \text{ current\_state} \neq \{L1, L2, L3\}$ 
end
```

The right sidebar shows the "HyperSearch Results Sidekick State Theories" panel with a search for "DwarfSignal" and a "Prover: ready" status. The bottom status bar shows the console output: "128,18 (3064/3089) (isabelle,isabelle,UTF-8-Isabelle) in mro UG 1 to".

Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The main window displays a theorem prover session for a file named `DwarfSignal.thy`. The theorem being checked is:

```
lemma "(SetNewProperState p) preserves NeverShowAll"
  by zpog_full
```

The console output shows the result of the quickcheck tactic:

```
Testing conjecture with Quickcheck-exhaustive...
Quickcheck found a counterexample:
  current_state__ = {L2, L1}
  l = L3
  turn_on__ = {L3}
```

The status bar at the bottom indicates the file is `isabelle.isabelle.UTF-8-Isabelle` and the line number is `129,7 (3071/3089)`. A small window in the bottom right corner shows a video player with a progress bar.

Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The main window displays the file `DwarfSignal.thy` with the following content:

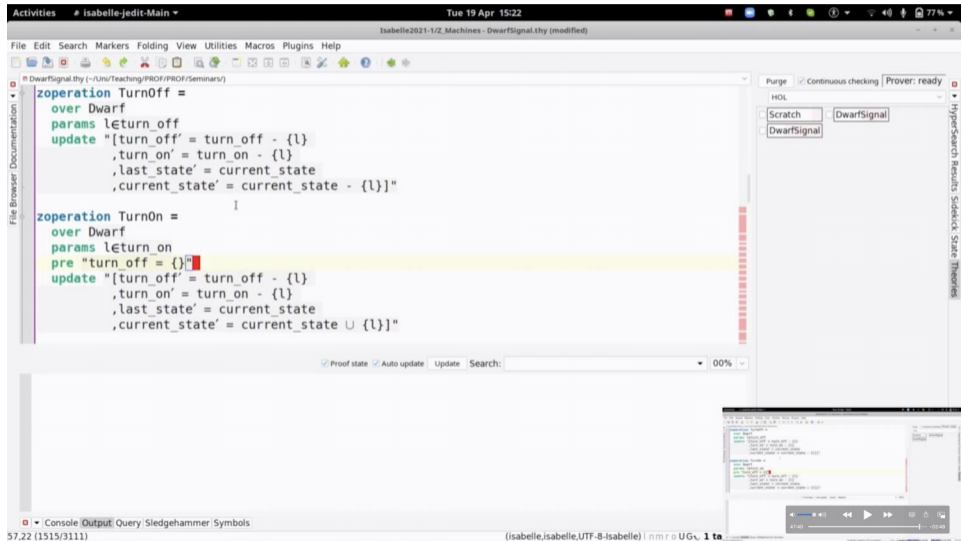
```
zoperation TurnOff =  
  over Dwarf  
  params leturn_off  
  update "{turn_off' = turn_off - {l}  
    ,turn_on' = turn_on - {l}  
    ,last_state' = current_state  
    ,current_state' = current_state - {l}}"
```

```
zoperation TurnOn =  
  over Dwarf  
  params leturn_on  
  update "{turn_off' = turn_off - {l}  
    ,turn_on' = turn_on - {l}  
    ,last_state' = current_state  
    ,current_state' = current_state  $\cup$  {l}}"
```

```
zoperation Shine =
```

The `TurnOn` operation is highlighted in yellow. The right sidebar shows the `HyperSearch Results` and `State Theories` panels. The `State Theories` panel lists `HOL`, `Scratch`, and `DwarfSignal`. The `HyperSearch Results` panel shows a search for `turn_on` with results for `turn_on` and `turn_off`. A video player is visible in the bottom right corner, showing a video titled `Isabelle/Isabelle` with a duration of 47:09.

Checking the Specification



Checking the Specification

The screenshot shows the Isabelle/Isabelle IDE interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
is "???"

zexpr DwarfReq
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"

zoperation ViolNeverShowAll =
pre " $\neg$  NeverShowAll"

zmachine DwarfSignalTest =
init Init
operations SetNewProperState TurnOn TurnOff Shine ViolNeverShowAll

animate DwarfSignalTest

lemma "Init establishes Dwarf_inv"
by zpog_full
```

The right sidebar shows the `HyperSearch Results Sidekick State Theories` panel with the following entries:

- ☐ Purge
- ☒ Continuous checking
- Prover: ready
- HOL
- ☐ Scratch
- ☐ DwarfSignal
- ☒ DwarfSignal

The bottom status bar shows the console output: `57.22 (1515/3111) (isabelle,isabelle.UTF-8-Isabelle) in mro UGv. 1 to`.

Checking the Specification

The screenshot displays the Isabelle2021-1/2_Machines - DwarfSignal.thy (modified) file in a text editor. The file contains a formal specification for a DwarfSignal system. The specification includes a `zexpr` for `DwarfReq`, a `zoperation` for `ViolNeverShowAll`, a `zmachine` for `DwarfSignalTest`, and a `lemma` for `"Init establishes Dwarf_inv"`.

```
is "???"  
  
zexpr DwarfReq  
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"  
  
zoperation ViolNeverShowAll =  
pre " $\neg$  NeverShowAll"  
  
zmachine DwarfSignalTest =  
init Init  
operations SetNewProperState TurnOn TurnOff Shine ViolNeverShowAll  
  
animate DwarfSignalTest  
  
lemma "Init establishes Dwarf_inv"  
by zpog_full
```

The console output shows the execution of the simulation:

```
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10225630/simulate> ./Simulation  
Process  
/tmp/isabelle-simonfoster/process13243906329326067755/itree-simulate10168136/simulate/Simulation  
exited with code 834  
$ISABELLE_TMP_PREFIX/process13243906329326067755/itree-simulate10225630/simulate>  
Starting ITree Simulation...  
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4)  
SetNewProperState Drive;
```

The status bar at the bottom indicates the file is in the `System` context, and the console output is complete.

Checking the Specification

The screenshot displays the Isabelle2021-1/2_Machines - DwarfSignal.thy (modified) window. The main editor shows the following code:

```
is "???"  
  
zexpr DwarfReq  
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"  
  
zoperation ViolNeverShowAll =  
pre " $\neg$  NeverShowAll"  
  
zmachine DwarfSignalTest =  
init Init  
operations SetNewProperState TurnOn TurnOff Shine ViolNeverShowAll  
  
animate DwarfSignalTest  
  
lemma "Init establishes Dwarf_inv"  
by zpog_full
```

The right sidebar shows the HyperSearch Results Sidekick State Theories, with a list of theories including HOL, Scratch, and DwarfSignal. The console output at the bottom shows the execution of the DwarfSignalTest machine:

```
System  
/tmp/isabelle-simonfoster/process13243906329326067755/itree-simulate10168136/simulate/Simulation  
exited with code 834  
$ISABELLE TMP_PREFIX/process13243906329326067755/itree-simulate10225630/simulate>  
Starting ITree Simulation...  
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4)  
SetNewProperState Drive;  
3  
Internal Activity...  
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L2;
```

The bottom status bar shows the console output, query, sledgehammer, and symbols. The bottom right corner displays the Isabelle2021-1/2_Machines - DwarfSignal.thy (modified) window.

Checking the Specification

The screenshot shows the Isabelle/ML editor interface. The main window displays the file `DwarfSignal.thy` with the following content:

```
is "???"

zexpr DwarfReq
is "NeverShowAll  $\wedge$  MaxOneLampChange  $\wedge$  ForbidStopToDrive  $\wedge$  DarkOnlyToStop  $\wedge$  DarkOnlyFromStop"

zoperation ViolNeverShowAll =
pre " $\neg$  NeverShowAll"

zmachine DwarfSignalTest =
init Init
operations SetNewProperState TurnOn TurnOff Shine ViolNeverShowAll

animate DwarfSignalTest

lemma "Init establishes Dwarf_inv"
by zpog_full
```

The right sidebar shows the "HyperSearch Results Sidekick State Theories" panel with a search for "DwarfSignal" and a "Prover: ready" status.

The bottom console window displays the output of the simulation:

```
System
Starting ITree Simulation...
Events: (1) Shine (Set [L1,L2]); (2) SetNewProperState Dark; (3) SetNewProperState Warning; (4) SetNewProperState Drive;
3
Internal Activity...
Events: (1) Shine (Set [L1,L2]); (2) TurnOff L2;
2
Internal Activity...
Events: (1) Shine (Set [L1]); (2) TurnOn L3;
```

The status bar at the bottom indicates the file path `(isabelle,isabelle,UTF-8-Isabelle)` and the command `in m r o U G C 1 t a`.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.

This lecture

- Modelling Z specifications with Z machines.
- Types, stores, and state variables.
- Operations and machines.
- Animation and verification.
- Managing requirements.