Formal specification in Event-B

Alexander Serebrenik,
Ulyana Tikhonova

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Outline

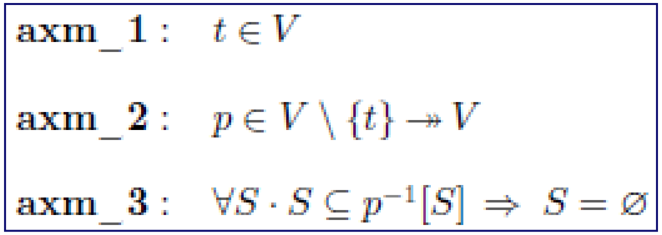
- Introduction into formal specification
- Mathematical notation of Event-B
- Event-B
- UML-B

Resources

- Summary of the Event-B notation: http://wiki.event-b.org/images/EventB-Summary.pdf
- Tutorials
  - http://handbook.event-b.org/current/html/index.html
  - http://handbook.event-b.org/current/html/mathematical_notation.html
- "Modeling in Event-B: System and Software Engineering" by Jean-Raymond Abrial
- Repository of Event-B examples: http://www.stups.uni-duesseldorf.de/bmotionstudio/index.php/User_Guide/Examples

Formal specification and formal analysis

- Formal = mathematical
  - Mathematical notation
  - Theory behind the notation



axm_1 : $t \in V$

axm_2 : $p \in V \setminus \{t\} \twoheadrightarrow V$

axm_3 : $\forall S \cdot S \subseteq p^{-1}[S] \;\Rightarrow\; S = \varnothing$

- Validation
  - Are we cooking the right product?

- Verification
  - Are we cooking it right?

Formal methods as a specification technique?

Unambiguous?

Realistic?

Verifiable?

Evolvable?

Event-B

- Z, B method, Event-B
- Event-B formalism
  - uses set theory and logic
  - has an extensive tool support
  - relatively simple, but not very expressive

Outline

- Introduction into formal specification
- Mathematical notation of Event-B
- Event-B
- UML-B

Predicates

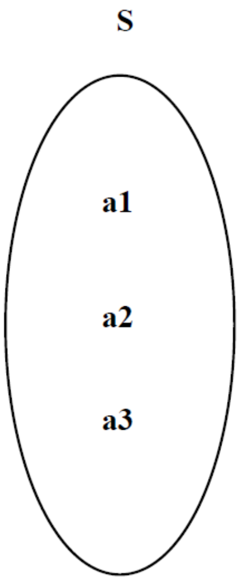
- Logical primitives: ⊤, ⊥
- Logical operators: ∧, ∨, ¬, …
- Quantifiers:
  - $\forall\ x_1, \ldots\ x_n \cdot P(x_1, \ldots\ x_n)$
  - $\exists\ x_1, \ldots\ x_n \cdot P(x_1, \ldots\ x_n)$
- Equality and inequality: =, ≠

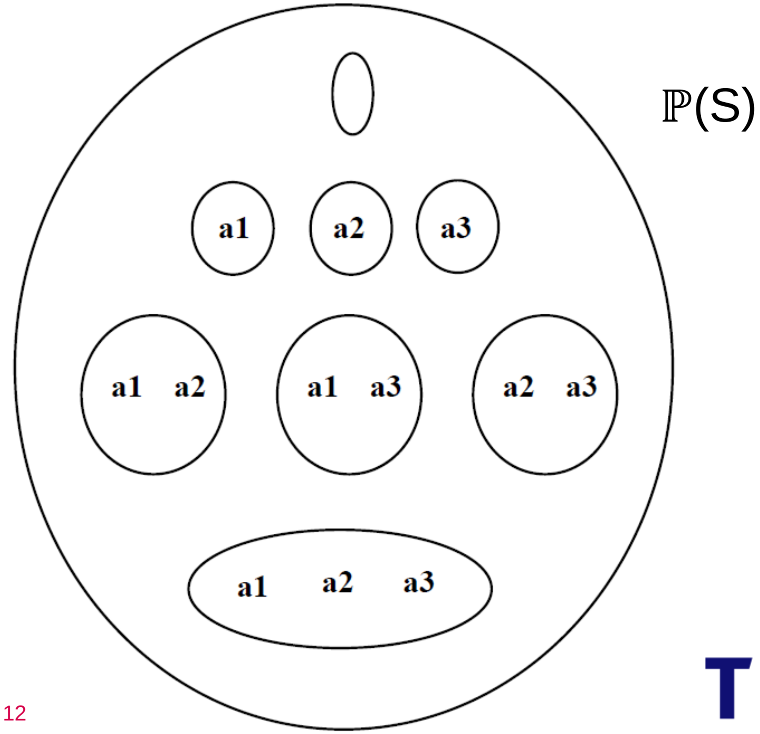
Sets

- Predefined sets: ℕ, ℤ, BOOL
- Interval: m..n
- Sets:
  - ∅
  - {expr_1, …, expr_n}
  - **{ x · P | E }**
    - − { x · x ∈ 1..10 | x ^ 2 }
  - **partition(S, {expr_1}, …, {expr_n})**
    - − partition(Course_2IW80, {Tuesday}, {Thursday})

Sets

- Membership: $a1 \in S$, $a4 \notin S$
- Subsets: $\{a1, a2\} \subseteq S$, $\{b1, b2\} \not\subseteq S$
- Operations on sets: $S \cup T$, $S \cap T$, $S \setminus T$
- Power sets: $\mathbb{P}(S)$





Is it true or false?

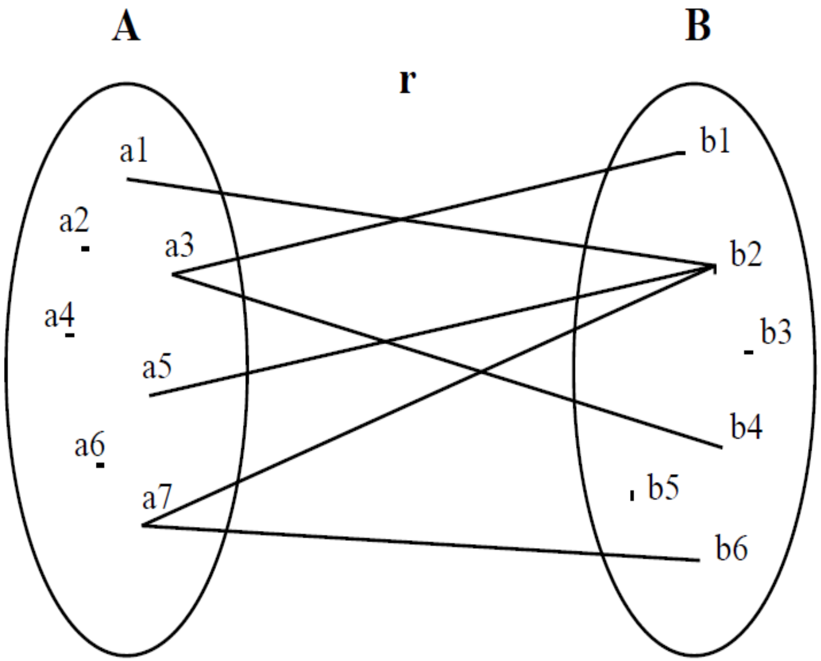
$$\begin{aligned}&\{-2..2\} \setminus \{0\} \subseteq \\ &\qquad \{\, x, y \cdot x \in \mathbb{Z} \land y \in \mathbb{Z} \land \\ &\qquad\qquad x * y = 12 \mid x \,\}\end{aligned}$$

a. $\top$

b. $\bot$

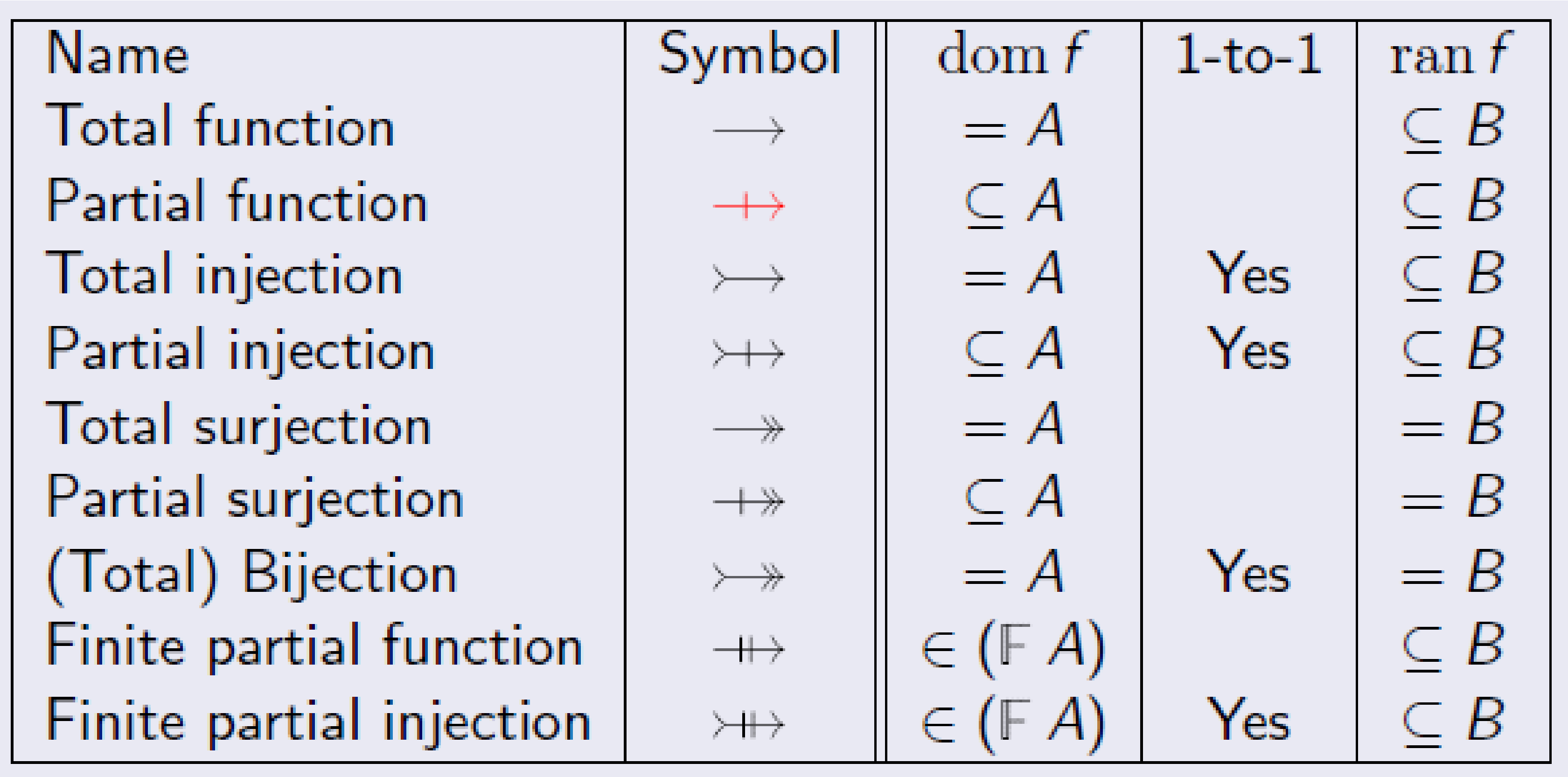
Relations

- Cartesian product
  - A × B
  - **Pair: a1 ↦ b2**
- Relations:
  - **A ↔ B**, …
- Domain and range:
  - dom(r), ran(r)
- Relational image
  - **r[{a1, a3}] is {b1,b2,b4}**



Functions

- Functions: A → B, A ⇸ B, …



Name	Symbol	dom f	1-to-1	ran f
Total function	$\rightarrow$	$= A$		$\subseteq B$
Partial function	⇸	$\subseteq A$		$\subseteq B$
Total injection	$\rightarrowtail$	$= A$	Yes	$\subseteq B$
Partial injection	⤔	$\subseteq A$	Yes	$\subseteq B$
Total surjection	$\twoheadrightarrow$	$= A$		$= B$
Partial surjection	⤀	$\subseteq A$		$= B$
(Total) Bijection	⤖	$= A$	Yes	$= B$
Finite partial function	⇻	$\in (\mathbb{F}\ A)$		$\subseteq B$
Finite partial injection	⤕	$\in (\mathbb{F}\ A)$	Yes	$\subseteq B$

Outline

- Introduction into formal specification
- Mathematical notation of Event-B
- Event-B
- UML-B

File system example

In the file system, users can create new files, execute, display (on different output devices) and delete existing files. There is a special type of delete, which removes the file permanently from the file system. The file system makes use of an access right system which specifies who the owner of each file is and what operations are allowed by which users. The owner of each file may change the access rights to the file and give or take other people's permissions to access the file. In addition to the person who creates the file, the administrator is considered the owner of all files.

Identify sets and relations

What are the sets?

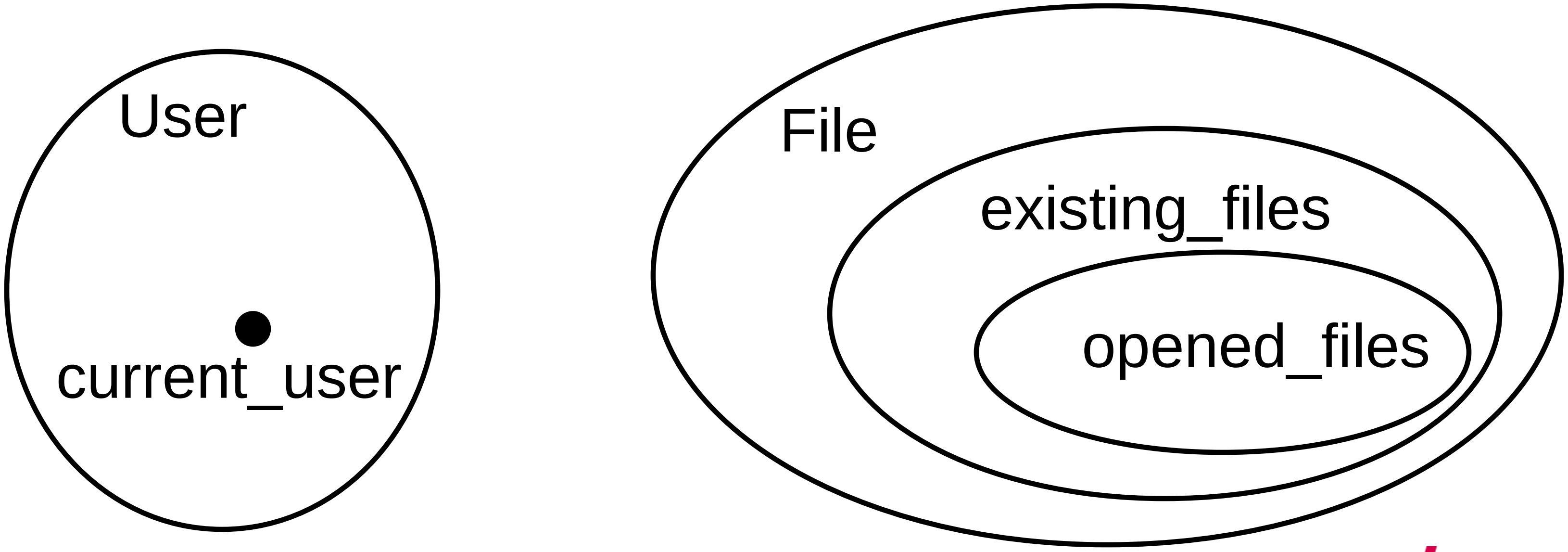
In the file system, users can create new files, execute, display (on different output devices) and delete existing files. There is a special type of delete, which removes the file permanently from the file system. The file system makes use of an access right system which specifies who the owner of each file is and what operations are allowed by which users. The owner of each file may change the access rights to the file and give or take other people's permissions to access the file. In addition to the person who creates the file, the administrator is considered the owner of all files.

User, File, Operation

Sets and subsets

- User, File, Operation
- existing_files ⊆ File
- opened_files ⊆ existing_files
- current_user ∈ User

- partition(Operation, {Execute}, {Display}, {Delete}, {Delete_permanently})



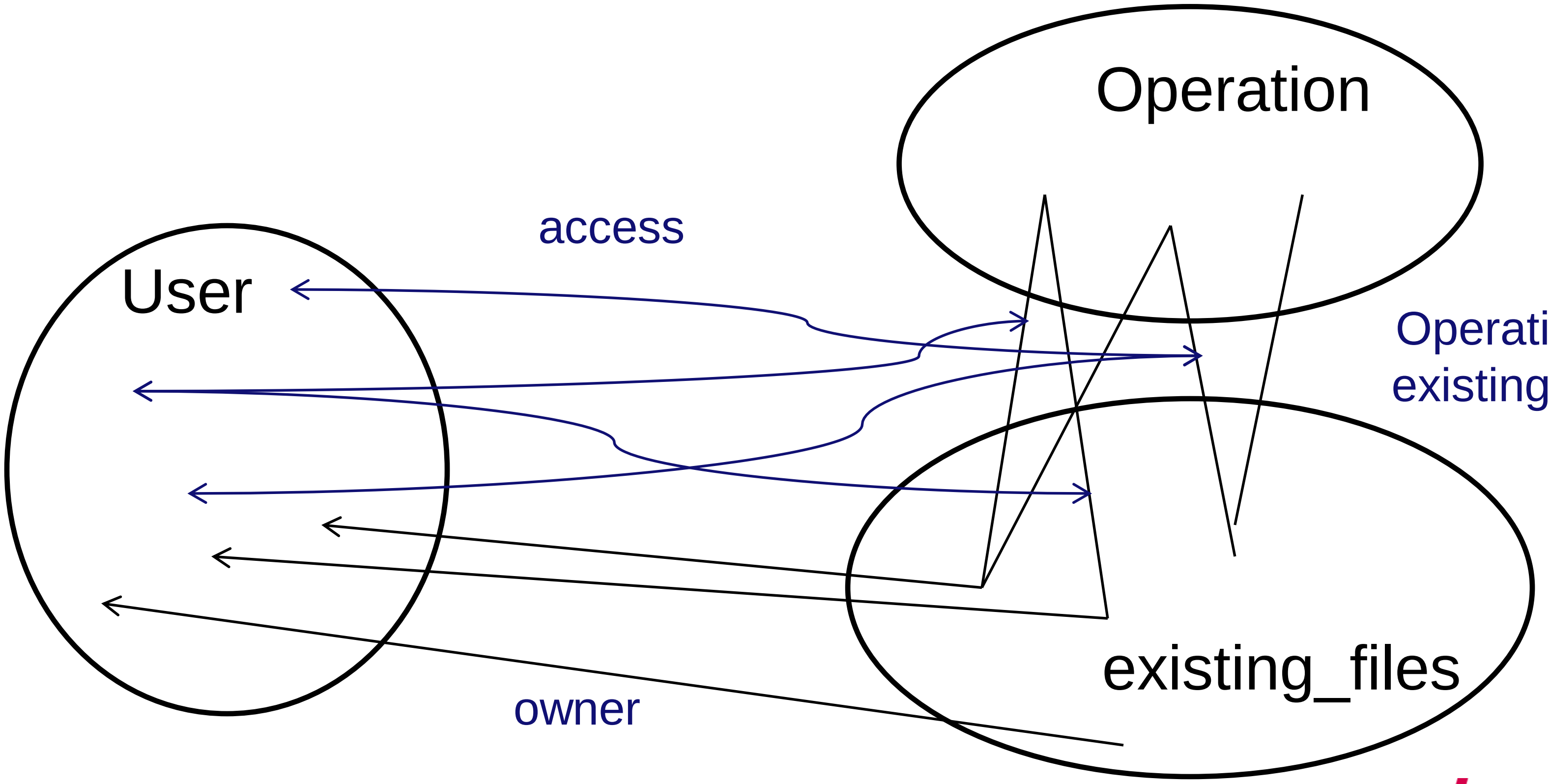
What are the relations?

In the file system, users can create new files, execute, display (on different output devices) and delete existing files. There is a special type of delete, which removes the file permanently from the file system. The file system makes use of an access right system which specifies who the owner of each file is and what operations are allowed by which users. The owner of each file may change the access rights to the file and give or take other people's permissions to access the file. In addition to the person who creates the file, the administrator is considered the owner of all files.

- **owner $\in$ existing_files → User**
- **access $\in$ User ↔ (Operation × existing_files)**

Relations

- owner ∈ existing_files → User
- access ∈ User ↔ (Operation × existing_files)



Other possibilities

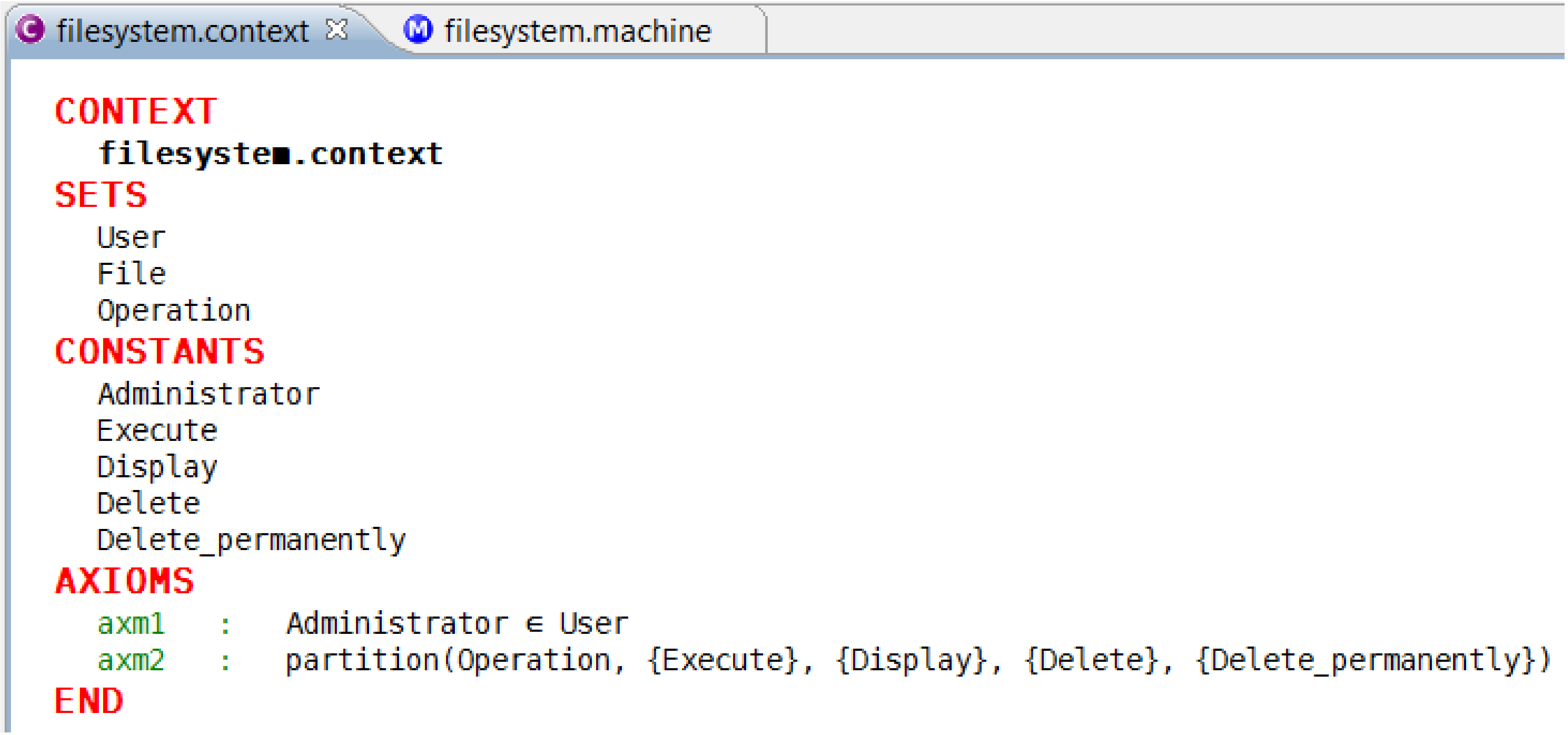
- sets User, File
- existing_files $\subseteq$ File
- owner $\in$ existing_files $\longrightarrow \mathbb{P}$(User)
- access_display $\in$ existing_files $\longrightarrow \mathbb{P}$(User)
- access_execute $\in$ existing_files $\longrightarrow \mathbb{P}$(User)
- access_delete $\in$ existing_files $\longrightarrow \mathbb{P}$(User)
- access_delete_permanently $\in$
  existing_files $\longrightarrow \mathbb{P}$(User)

Access control invariant

- $existing\_files \subseteq File$
- $opened\_files \subseteq existing\_files$
- $current\_user \in User$
- $access \in User \leftrightarrow (Operation \times existing\_files)$

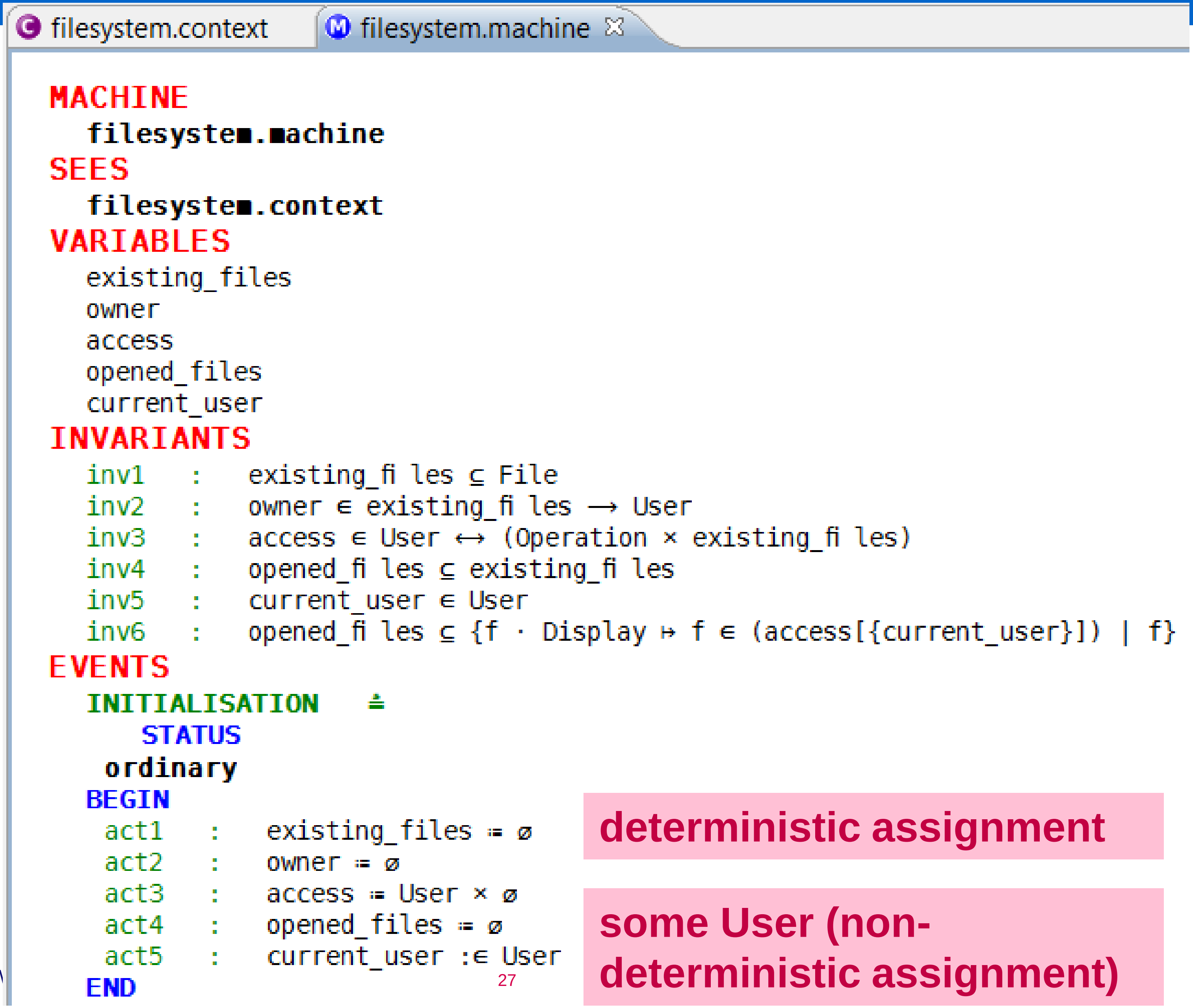
- $\mathbf{opened \subseteq \{ file \cdot}$
  $\mathbf{Display \mapsto file \in access[\{current\_user\}]}$
  $\mathbf{\mid file \}}$

File system specified in Event-B



```
CONTEXT
   filesystem.context
SETS
   User
   File
   Operation
CONSTANTS
   Administrator
   Execute
   Display
   Delete
   Delete_permanently
AXIOMS
   axm1   :   Administrator ∈ User
   axm2   :   partition(Operation, {Execute}, {Display}, {Delete}, {Delete_permanently})
END
```

File system specified in Event-B

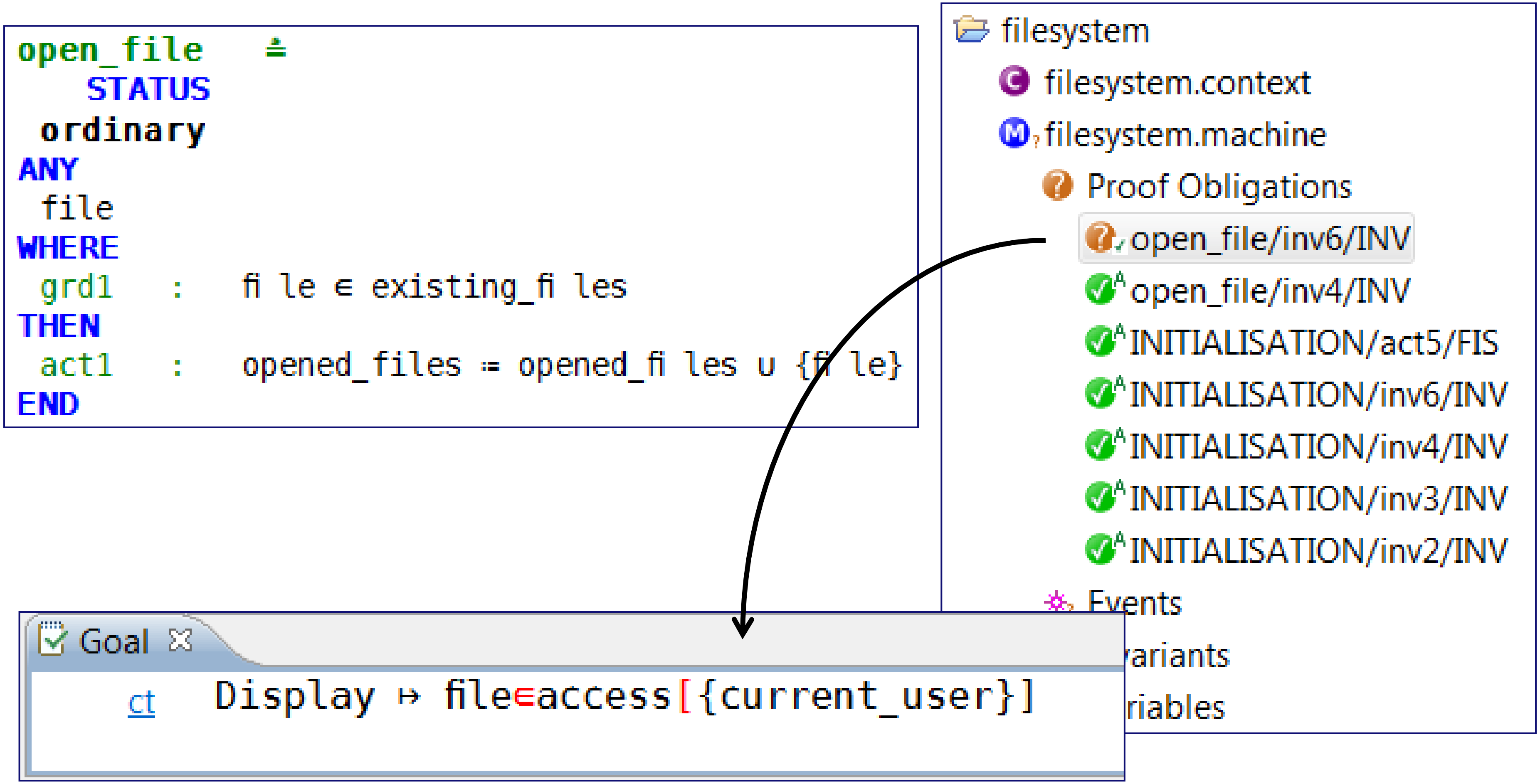


```
MACHINE
   filesystem.machine
SEES
   filesystem.context
VARIABLES
   existing_files
   owner
   access
   opened_files
   current_user
INVARIANTS
   inv1    :    existing_files ⊆ File
   inv2    :    owner ∈ existing_files → User
   inv3    :    access ∈ User ↔ (Operation × existing_files)
   inv4    :    opened_files ⊆ existing_files
   inv5    :    current_user ∈ User
   inv6    :    opened_files ⊆ {f · Display ↦ f ∈ (access[{current_user}]) | f}
EVENTS
   INITIALISATION    ≙
       STATUS
    ordinary
   BEGIN
    act1    :    existing_files ≔ ∅
    act2    :    owner ≔ ∅
    act3    :    access ≔ User × ∅
    act4    :    opened_files ≔ ∅
    act5    :    current_user :∈ User
   END
```

deterministic assignment

some User (non-deterministic assignment)

File system specified in Event-B: verification



```
open_file ≙
   STATUS
 ordinary
ANY
 file
WHERE
 grd1   :   file ∈ existing_files
THEN
 act1   :   opened_files ≔ opened_files ∪ {file}
END
```

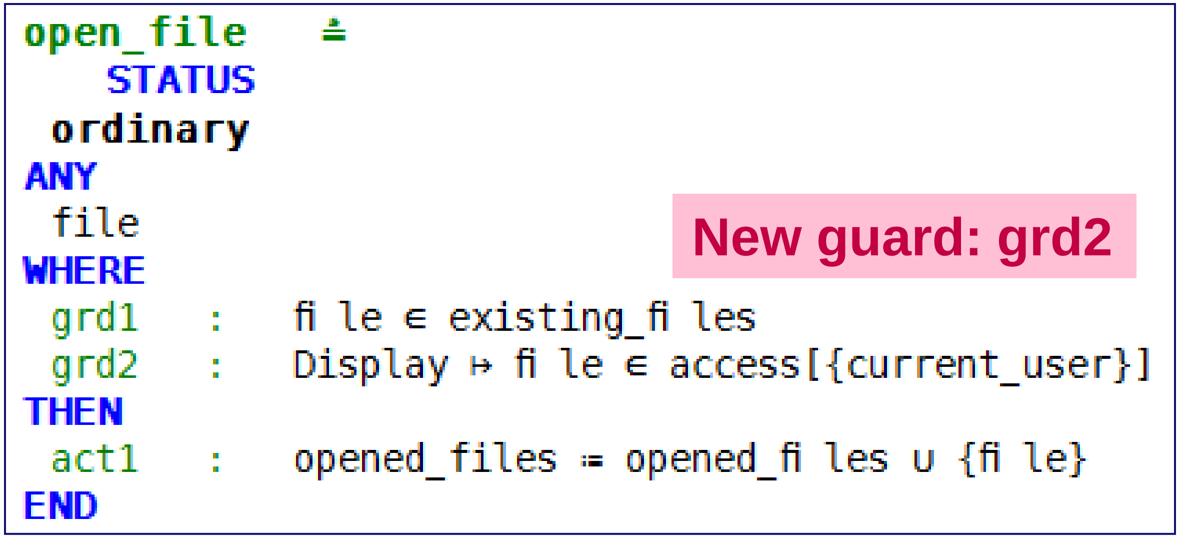
filesystem
- filesystem.context
- filesystem.machine
  - Proof Obligations
    - open_file/inv6/INV
    - ✓ open_file/inv4/INV
    - ✓ INITIALISATION/act5/FIS
    - ✓ INITIALISATION/inv6/INV
    - ✓ INITIALISATION/inv4/INV
    - ✓ INITIALISATION/inv3/INV
    - ✓ INITIALISATION/inv2/INV
  - Events
  - ariants
  - riables

Goal

```
ct   Display ↦ file∈access[{current_user}]
```

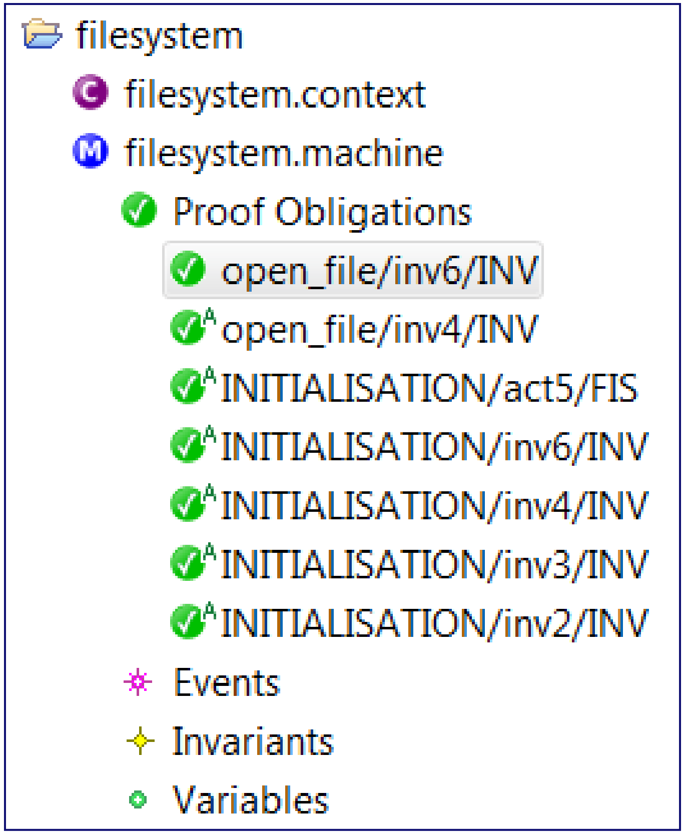
inv6: opened ⊆ { file · Display ↦ file ∈ access[{current_user}] | file }

File system specified in Event-B: verification



```
open_file  ≙
    STATUS
  ordinary
ANY
  file
WHERE
  grd1   :   file ∈ existing_files
  grd2   :   Display ↦ file ∈ access[{current_user}]
THEN
  act1   :   opened_files ≔ opened_files ∪ {file}
END
```

New guard: grd2



- filesystem
  - filesystem.context
  - filesystem.machine
    - Proof Obligations
      - open_file/inv6/INV
      - open_file/inv4/INV
      - INITIALISATION/act5/FIS
      - INITIALISATION/inv6/INV
      - INITIALISATION/inv4/INV
      - INITIALISATION/inv3/INV
      - INITIALISATION/inv2/INV
    - Events
    - Invariants
    - Variables

inv6: opened ⊆ { file · Display ↦ file ∈ access[{current_user}] | file }

Specification in Event-B: context component

Context

- Sets
  - data types
- Constants
- Axioms
  - MANDATORY: types of constants
  - properties that are assumed to be true

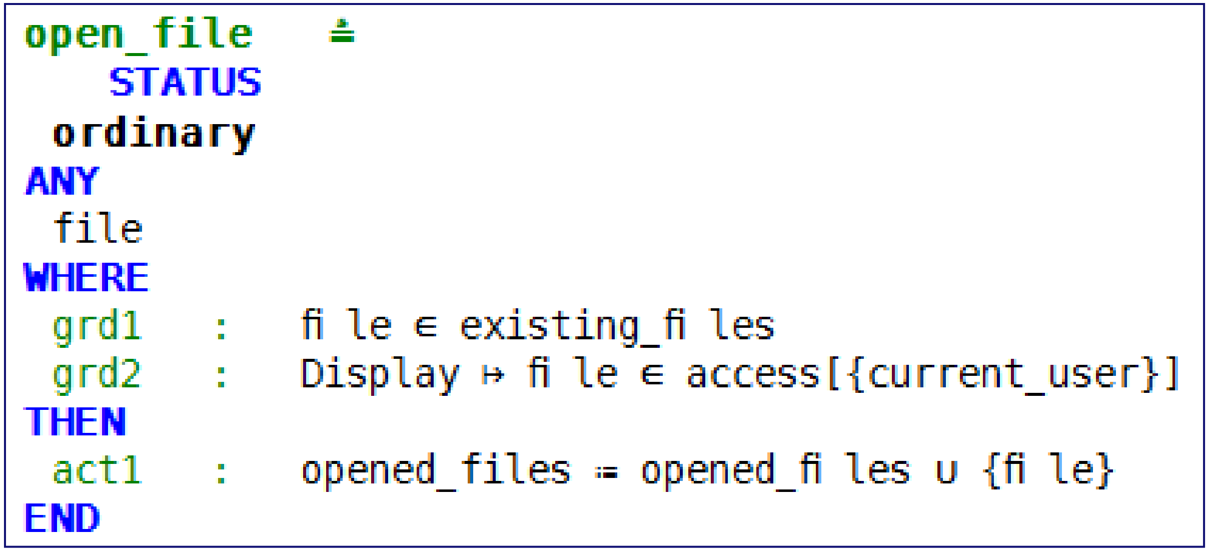
Specification in Event-B: machine component

Machine sees a context

- Variables
  - whose values are changed by events
- Invariants
  - MANDATORY: types of variables
  - properties that need to be checked
- Events
  - parameters (ANY)
  - guards (WHERE)
  - actions (THEN), executed in parallel

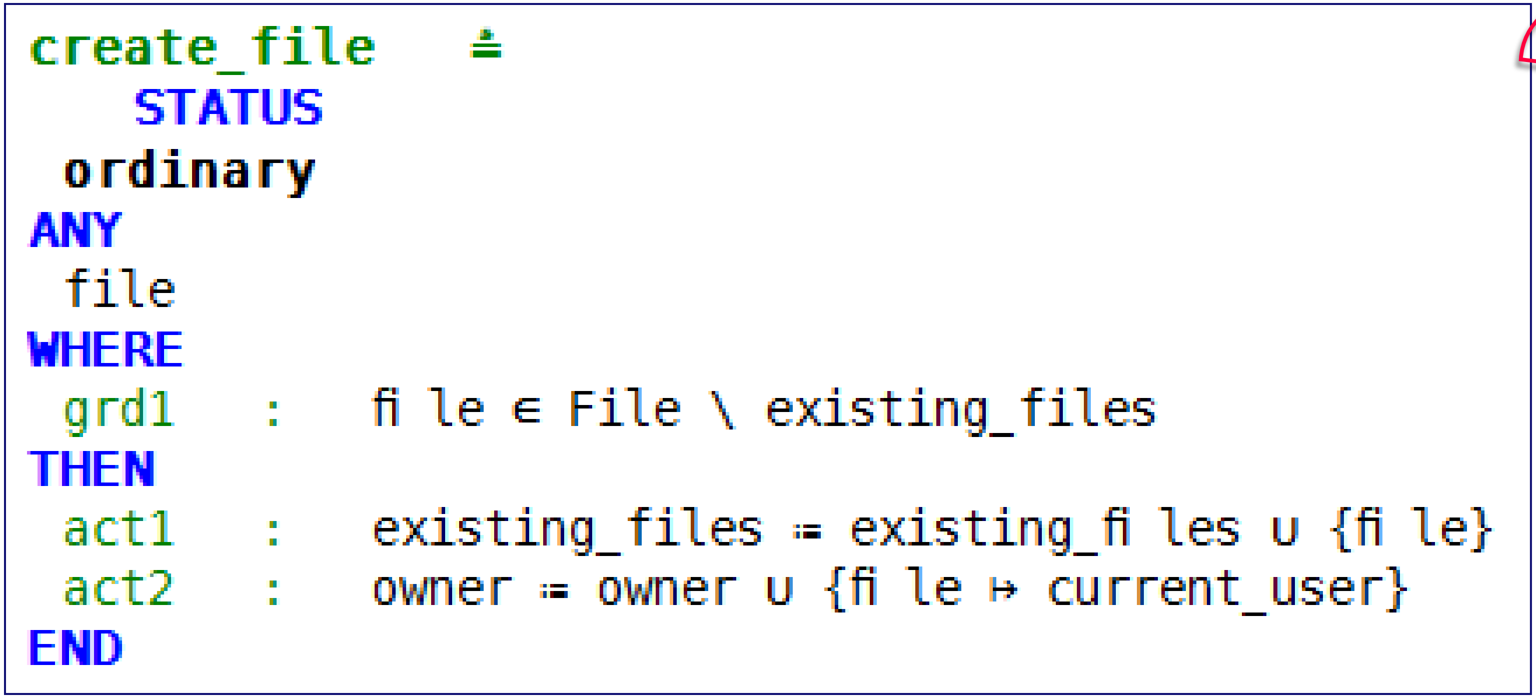
Exercise for you: create file and delete file

- sets File, User, Operation
- existing_files ⊆ File
- opened_files ⊆ existing_files
- current_user ∈ User
- owner ∈ existing_files → User
- access ∈ User ↔ (Operation × existing_files)



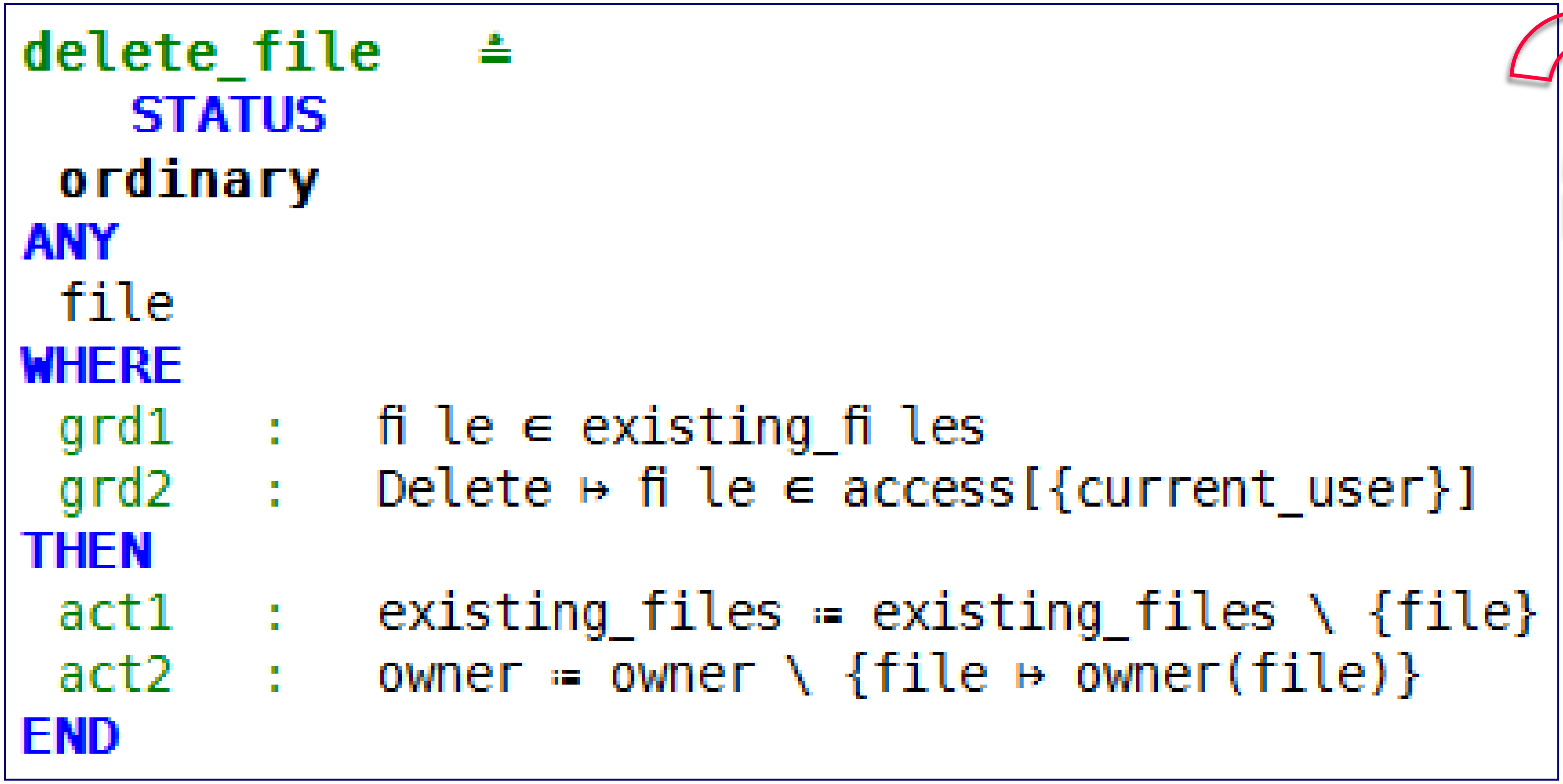
```
open_file  ≙
    STATUS
  ordinary
ANY
  file
WHERE
  grd1  :  file ∈ existing_files
  grd2  :  Display ↦ file ∈ access[{current_user}]
THEN
  act1  :  opened_files ≔ opened_files ∪ {file}
END
```

Create file



```
create_file  ≙
   STATUS
 ordinary
ANY
 file
WHERE
 grd1   :   file ∈ File ∖ existing_files
THEN
 act1   :   existing_files ≔ existing_files ∪ {file}
 act2   :   owner ≔ owner ∪ {file ↦ current_user}
END
```

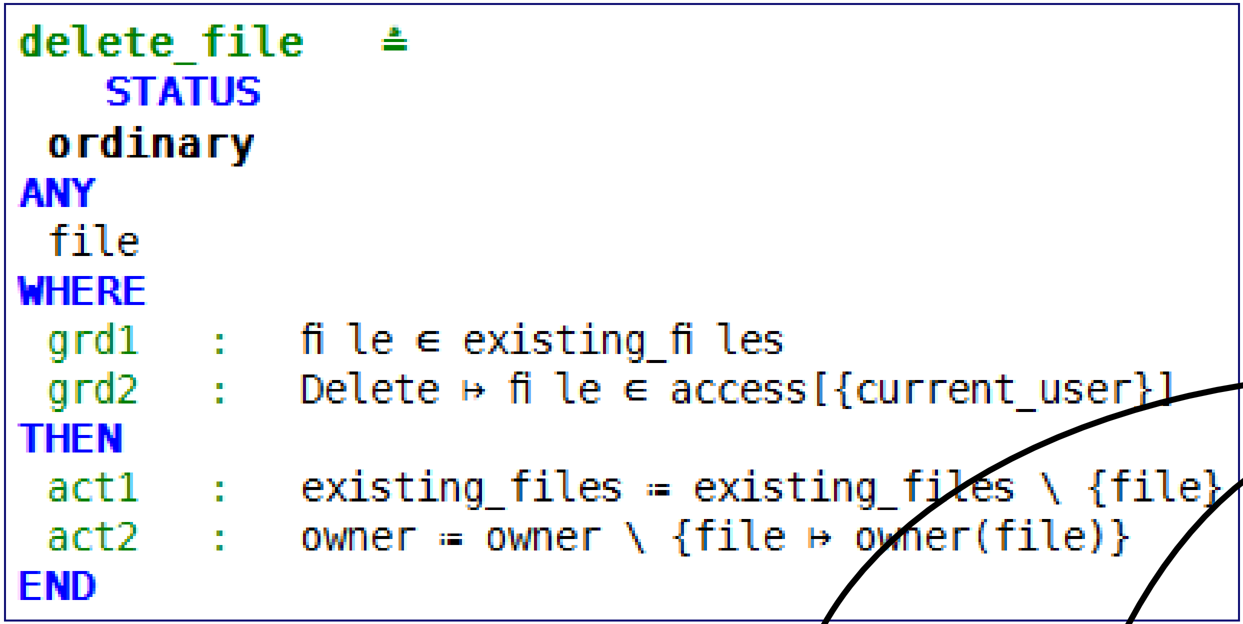
Delete file



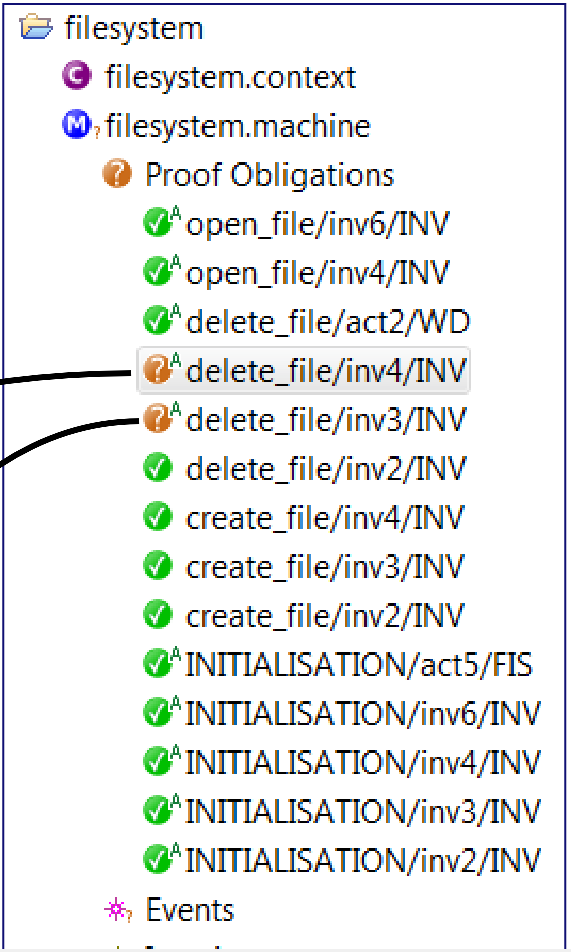
```
delete_file   ≙
    STATUS
 ordinary
ANY
 file
WHERE
 grd1   :   file ∈ existing_files
 grd2   :   Delete ↦ file ∈ access[{current_user}]
THEN
 act1   :   existing_files ≔ existing_files ∖ {file}
 act2   :   owner ≔ owner ∖ {file ↦ owner(file)}
END
```

Is it correct?

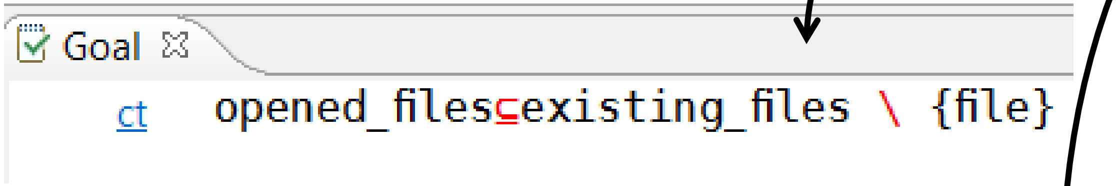
Delete file: verification problems



```
delete_file   ≙
    STATUS
 ordinary
ANY
 file
WHERE
 grd1   :   file ∈ existing_files
 grd2   :   Delete ↦ file ∈ access[{current_user}]
THEN
 act1   :   existing_files ≔ existing_files \ {file}
 act2   :   owner ≔ owner \ {file ↦ owner(file)}
END
```

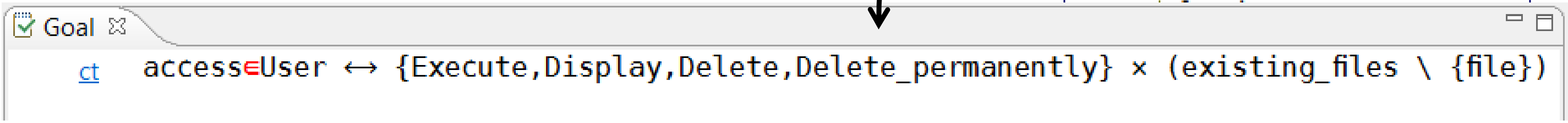


```
filesystem
  filesystem.context
  filesystem.machine
    Proof Obligations
      open_file/inv6/INV
      open_file/inv4/INV
      delete_file/act2/WD
      delete_file/inv4/INV
      delete_file/inv3/INV
      delete_file/inv2/INV
      create_file/inv4/INV
      create_file/inv3/INV
      create_file/inv2/INV
      INITIALISATION/act5/FIS
      INITIALISATION/inv6/INV
      INITIALISATION/inv4/INV
      INITIALISATION/inv3/INV
      INITIALISATION/inv2/INV
    Events
```



Goal

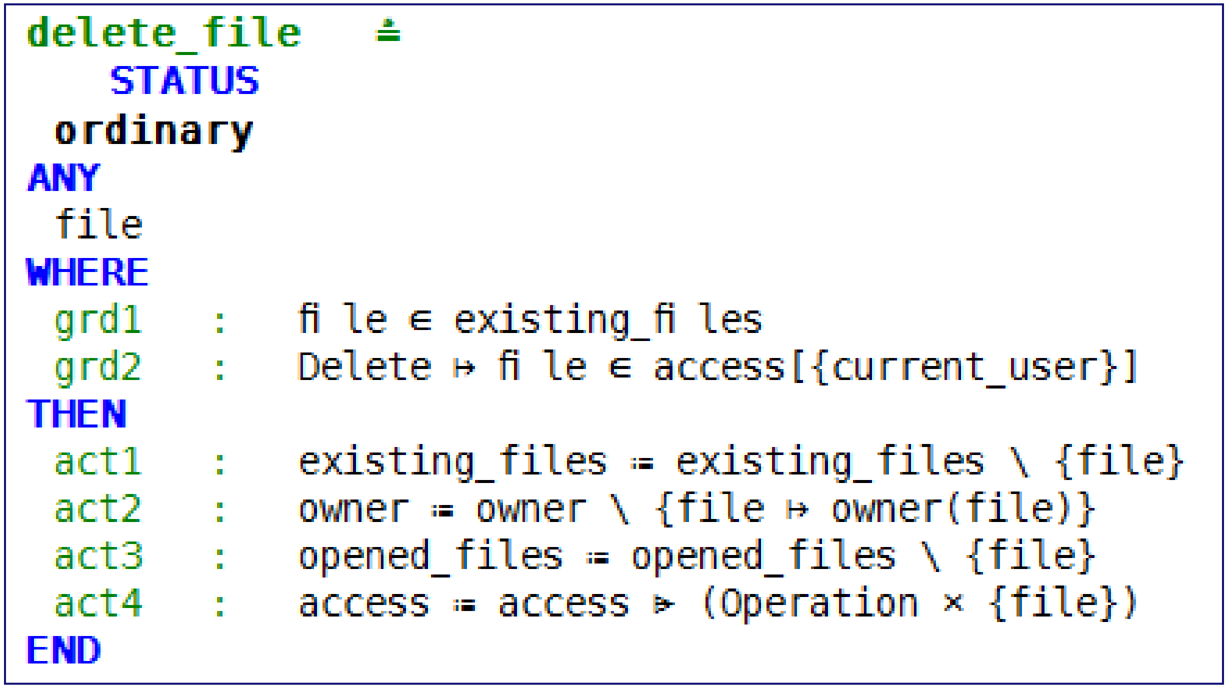
```
ct   opened_files⊆existing_files \ {file}
```



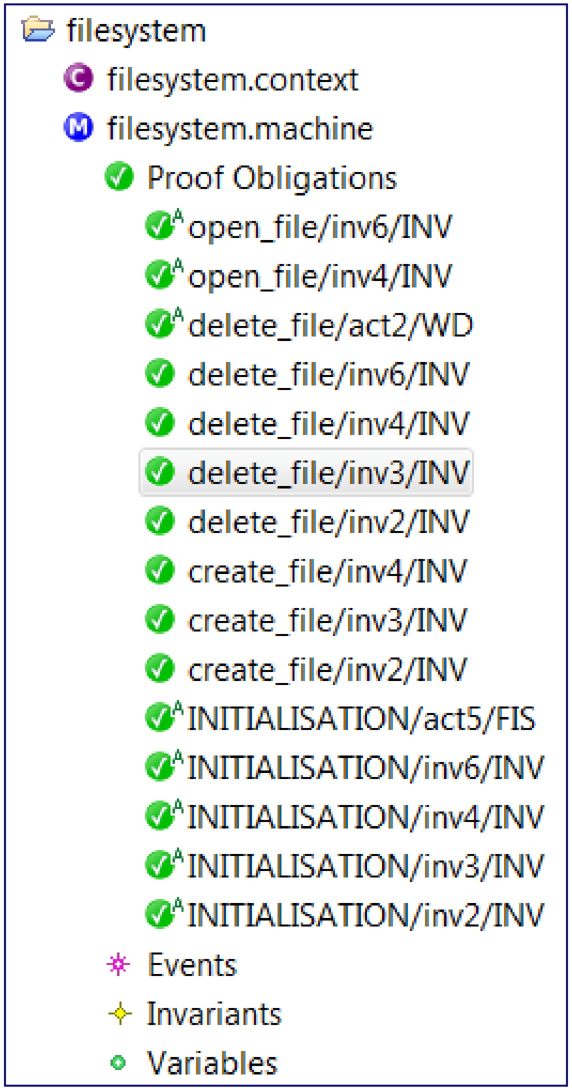
Goal

```
ct   access∈User ↔ {Execute,Display,Delete,Delete_permanently} × (existing_files \ {file})
```

Delete file: improved version



```
delete_file   ≙
    STATUS
 ordinary
ANY
 file
WHERE
 grd1   :    file ∈ existing_files
 grd2   :    Delete ↦ file ∈ access[{current_user}]
THEN
 act1   :    existing_files ≔ existing_files ∖ {file}
 act2   :    owner ≔ owner ∖ {file ↦ owner(file)}
 act3   :    opened_files ≔ opened_files ∖ {file}
 act4   :    access ≔ access ⩥ (Operation × {file})
END
```



- filesystem
  - filesystem.context
  - filesystem.machine
    - Proof Obligations
      - open_file/inv6/INV
      - open_file/inv4/INV
      - delete_file/act2/WD
      - delete_file/inv6/INV
      - delete_file/inv4/INV
      - delete_file/inv3/INV
      - delete_file/inv2/INV
      - create_file/inv4/INV
      - create_file/inv3/INV
      - create_file/inv2/INV
      - INITIALISATION/act5/FIS
      - INITIALISATION/inv6/INV
      - INITIALISATION/inv4/INV
      - INITIALISATION/inv3/INV
      - INITIALISATION/inv2/INV
    - Events
    - Invariants
    - Variables

- Domain and range restrictions

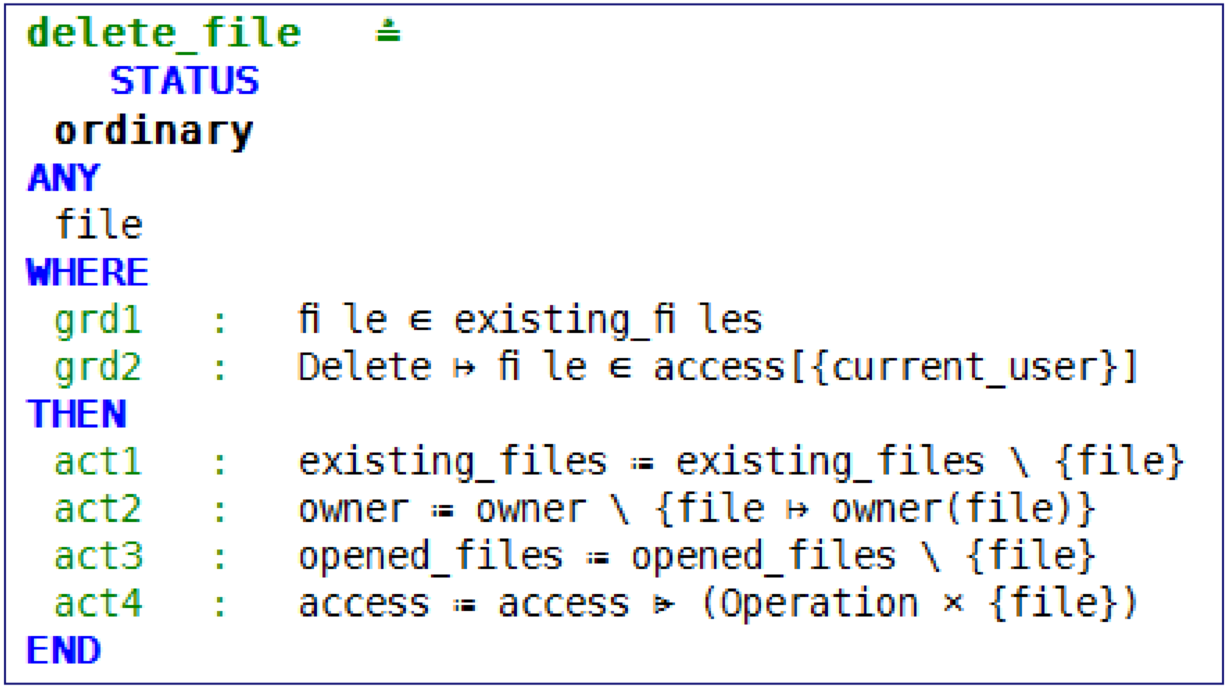
$$\mathsf{S} \lhd \mathsf{r} \mathrel{\widehat{=}} \{\, x \mapsto y \mid x \mapsto y \in \mathsf{r} \wedge x \in \mathsf{S} \}$$

$$\mathsf{S} \mathbin{⩤} \mathsf{r} \mathrel{\widehat{=}} \{\, x \mapsto y \mid x \mapsto y \in \mathsf{r} \wedge x \notin \mathsf{S} \}$$

$$\mathsf{r} \rhd \mathsf{S} \mathrel{\widehat{=}} \{\, x \mapsto y \mid x \mapsto y \in \mathsf{r} \wedge y \in \mathsf{S} \}$$

$$\mathsf{r} \mathbin{⩥} \mathsf{S} \mathrel{\widehat{=}} \{\, x \mapsto y \mid x \mapsto y \in \mathsf{r} \wedge y \notin \mathsf{S} \}$$

Exercise for you: delete file permanently



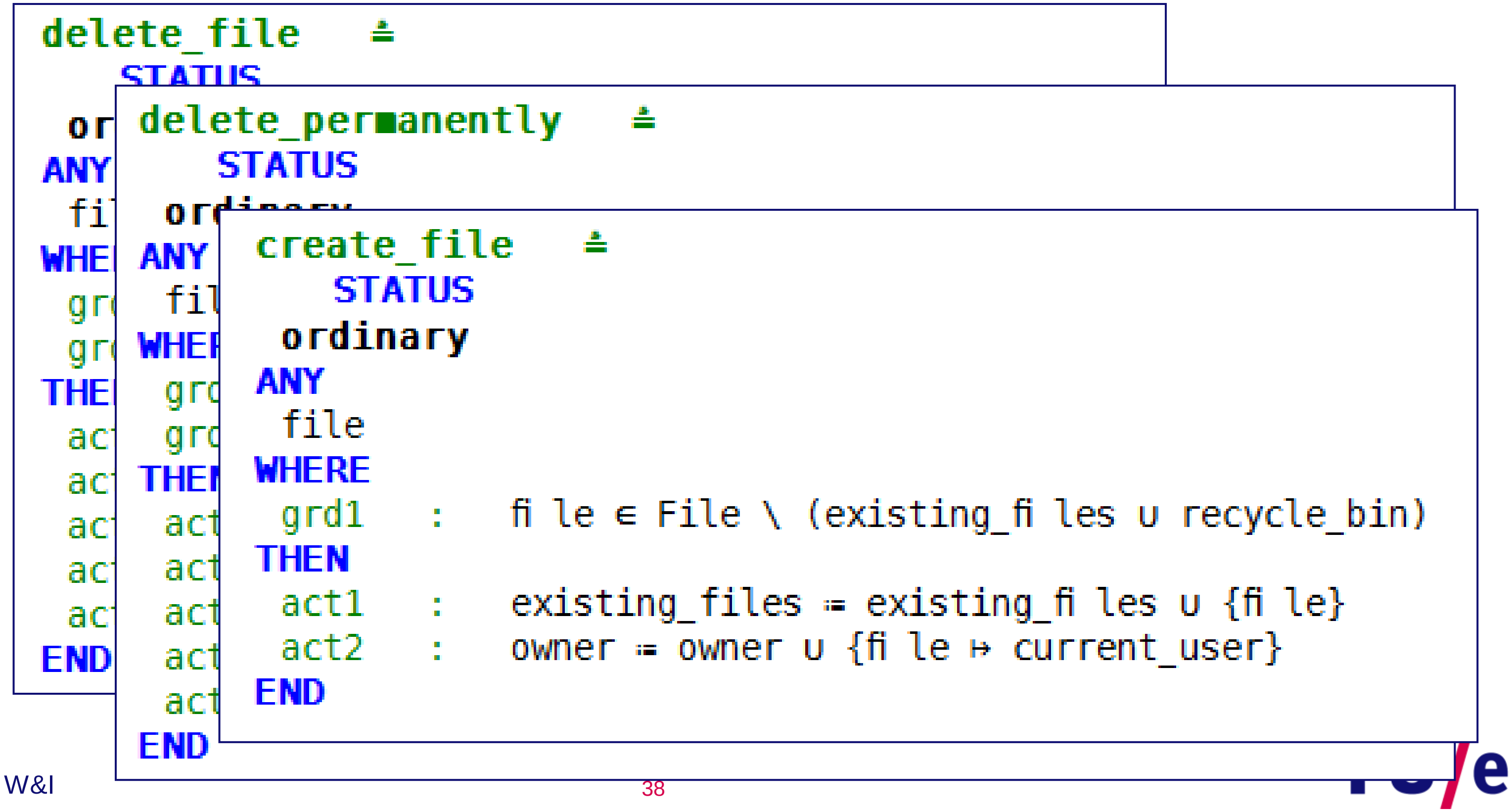
```
delete_file  ≙
    STATUS
  ordinary
ANY
  file
WHERE
  grd1   :   file ∈ existing_files
  grd2   :   Delete ↦ file ∈ access[{current_user}]
THEN
  act1   :   existing_files ≔ existing_files ∖ {file}
  act2   :   owner ≔ owner ∖ {file ↦ owner(file)}
  act3   :   opened_files ≔ opened_files ∖ {file}
  act4   :   access ≔ access ⩥ (Operation × {file})
END
```

recycle_bin …

Exercise for you: delete file permanently

- recycle_bin ⊆ File ∧
  recycle_bin ∩ existing_files = ∅

What events need to be changed?

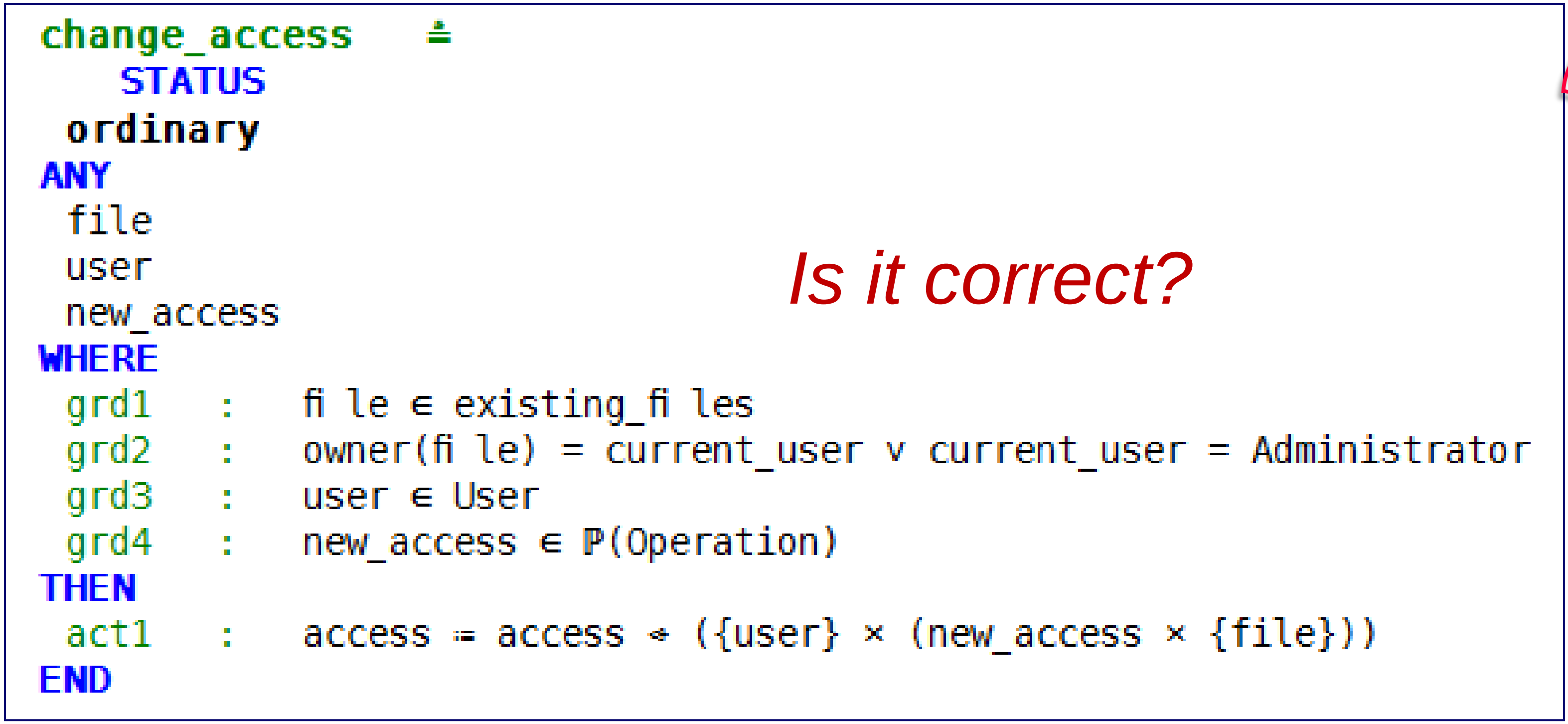


```
delete_file  ≙
    STATUS
 or
ANY
 fi
WHE
 gr
 gr
THE
 ac
 ac
 ac
 ac
 ac
END
```

```
delete_permanently  ≙
    STATUS
 ordinary
ANY
 fil
WHEF
 grd
 grd
THEN
 act
 act
 act
 act
 act
END
```

```
create_file  ≙
    STATUS
 ordinary
ANY
 file
WHERE
 grd1  :  file ∈ File \ (existing_files ∪ recycle_bin)
THEN
 act1  :  existing_files ≔ existing_files ∪ {file}
 act2  :  owner ≔ owner ∪ {file ↦ current_user}
END
```

Exercise for you: change access rights

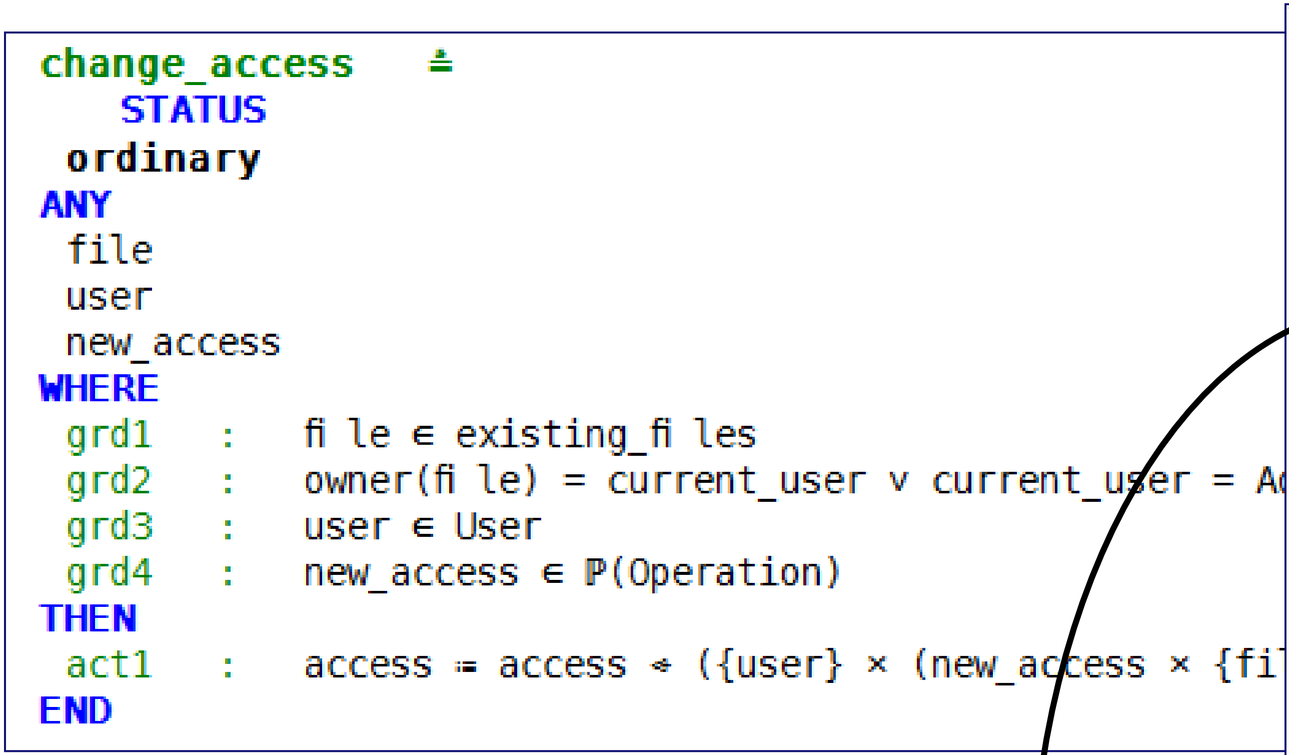


```
change_access  ≙
    STATUS
 ordinary
ANY
 file
 user
 new_access
WHERE
 grd1   :   file ∈ existing_files
 grd2   :   owner(file) = current_user ∨ current_user = Administrator
 grd3   :   user ∈ User
 grd4   :   new_access ∈ ℙ(Operation)
THEN
 act1   :   access ≔ access  ({user} × (new_access × {file}))
END
```

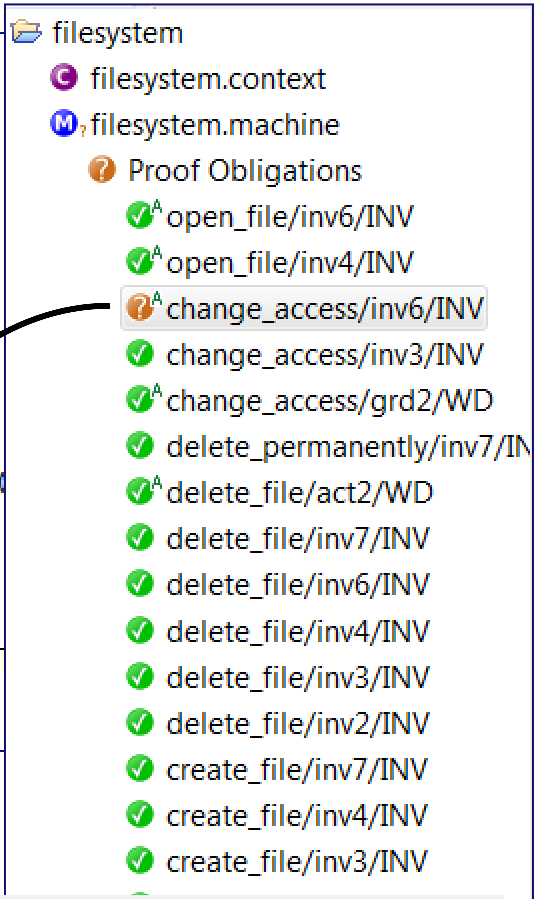
Is it correct?

- Relational override:
  $r1 \mathbin{<\!\!+} r2 = \{x \mapsto y \mid x \mapsto y \in r1 \land x \notin dom(r2)\} \cup r2$

Change access right: verification problems

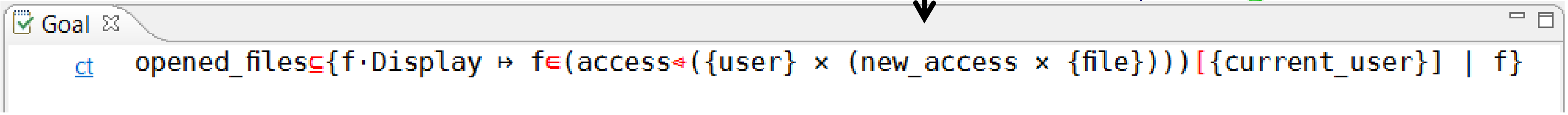


```
change_access    ≜
    STATUS
  ordinary
ANY
  file
  user
  new_access
WHERE
  grd1   :   file ∈ existing_files
  grd2   :   owner(file) = current_user ∨ current_user = A
  grd3   :   user ∈ User
  grd4   :   new_access ∈ ℙ(Operation)
THEN
  act1   :   access ≔ access  ({user} × (new_access × {fi
END
```



```
filesystem
  filesystem.context
  filesystem.machine
    Proof Obligations
      open_file/inv6/INV
      open_file/inv4/INV
      change_access/inv6/INV
      change_access/inv3/INV
      change_access/grd2/WD
      delete_permanently/inv7/IN
      delete_file/act2/WD
      delete_file/inv7/INV
      delete_file/inv6/INV
      delete_file/inv4/INV
      delete_file/inv3/INV
      delete_file/inv2/INV
      create_file/inv7/INV
      create_file/inv4/INV
      create_file/inv3/INV
      /INV
      /INV
      INITIALISATION/inv3/INV
      INITIALISATION/inv2/INV
```

How would you repair this?



```
ct   opened_files⊆{f·Display ↦ f∈(access({user} × (new_access × {file})))[{current_user}] | f}
```

inv6: opened ⊆ { file · Display ↦ file ∈ access[{current_user}] | file }

Change access right: how to repair?

- inv6: $\mathit{opened} \subseteq \{\, \mathit{file} \cdot \mathit{Display} \mapsto \mathit{file} \in \mathit{access}[\{\mathit{current\_user}\}] \mid \mathit{file} \,\}$
- Can the access rights to the file be changed when this file is opened?
  - Or: to change access right to a file, this file should be closed
- Does an owner have the access to display files that he/she owns?
  - Shall we add this to our invariants and guards?

Rodin platform

- Event-B editor
  - Write your specification
- Proof obligations
  - Generated automatically – for each pair from
  - Invariants × Events
  - Discharged by automatic provers
- Model checking
- Animation of machines
  - Execute your specification
- Graphical visualization

Outline

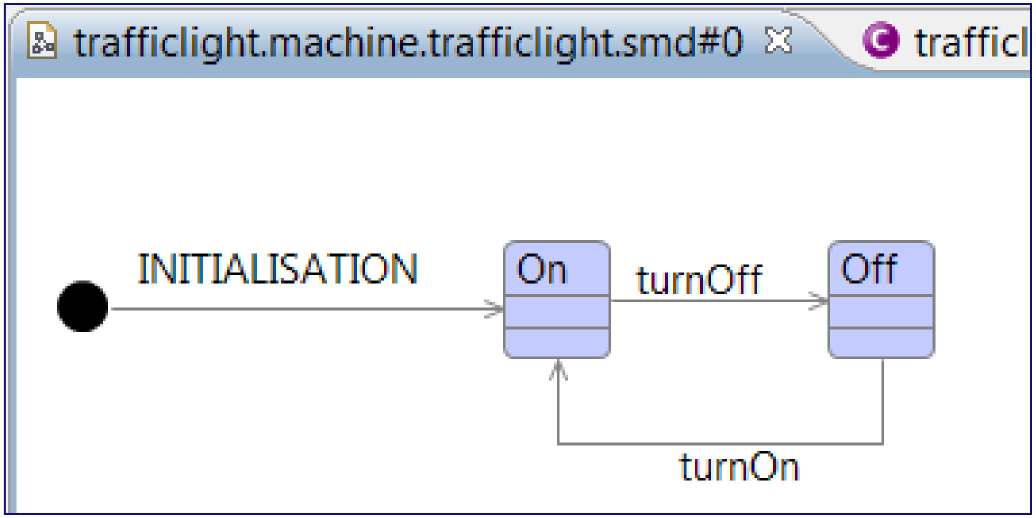
- Introduction into formal specification
- Mathematical notation of Event-B
- Event-B
- UML-B

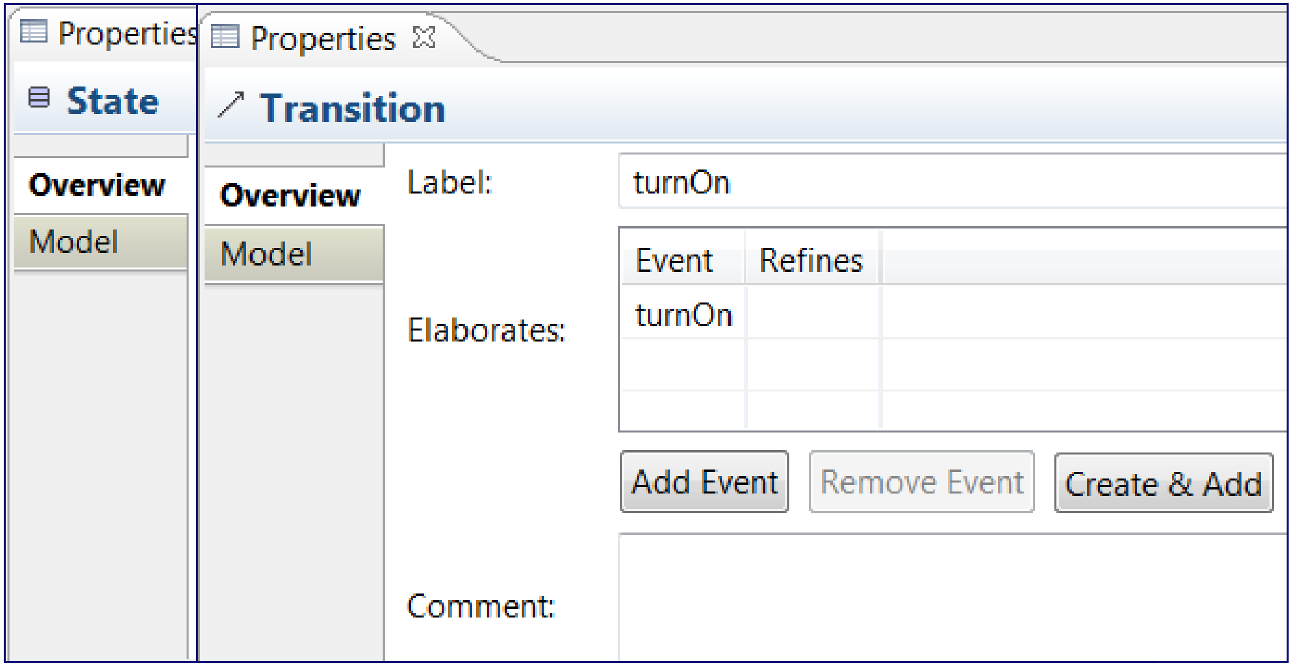
UML-B

- Draw UML diagrams
  - Class diagrams
  - State machine diagrams
- Generate Event-B specification

- iUML-B plug-in
- http://wiki.event-b.org/index.php/IUML-B
- https://www.youtube.com/watch?v=nz7ZpL2JtAM

State machine diagrams in iUML-B



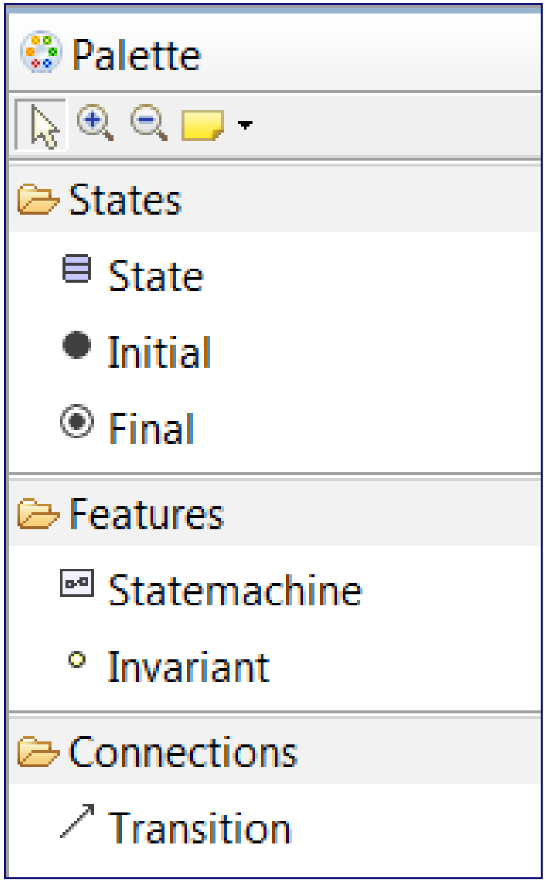


Label: turnOn

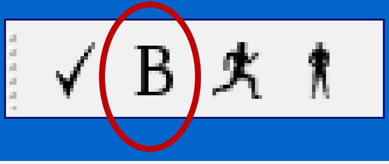
Elaborates:

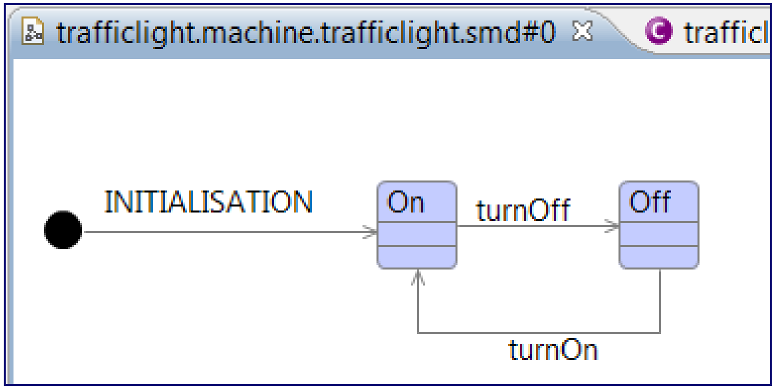
Event	Refines
turnOn	

Comment:

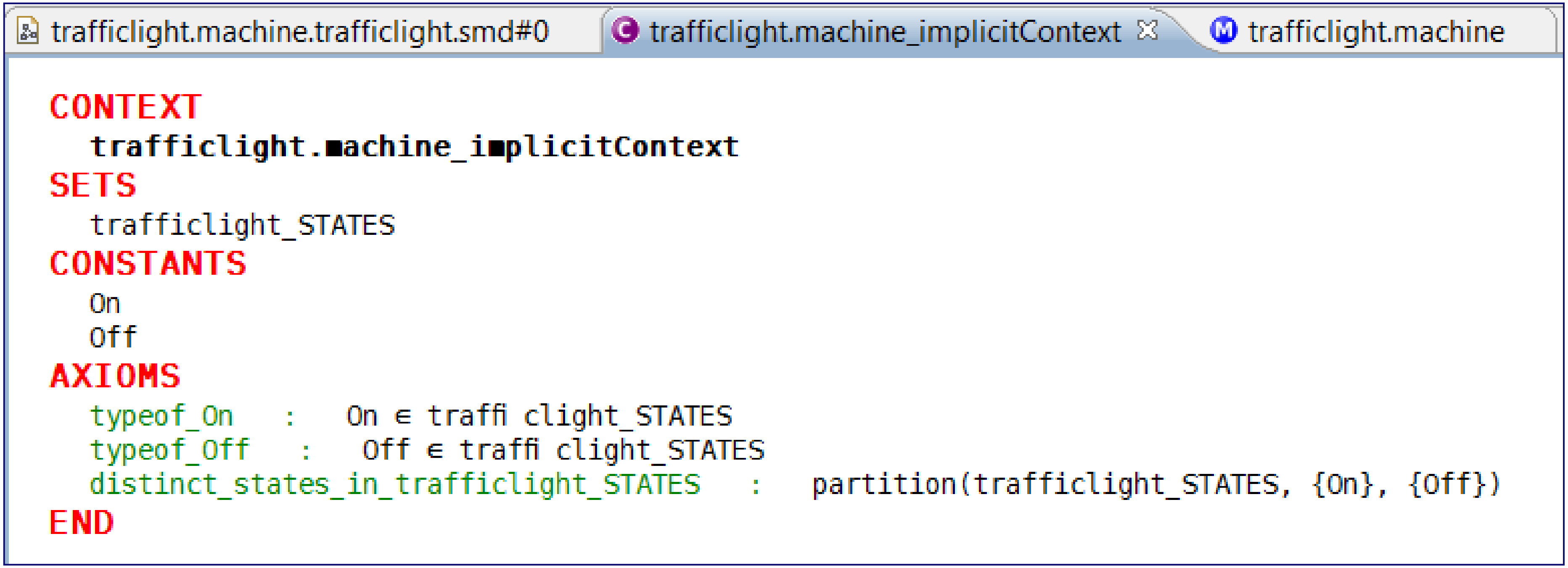


Generated Event-B specification



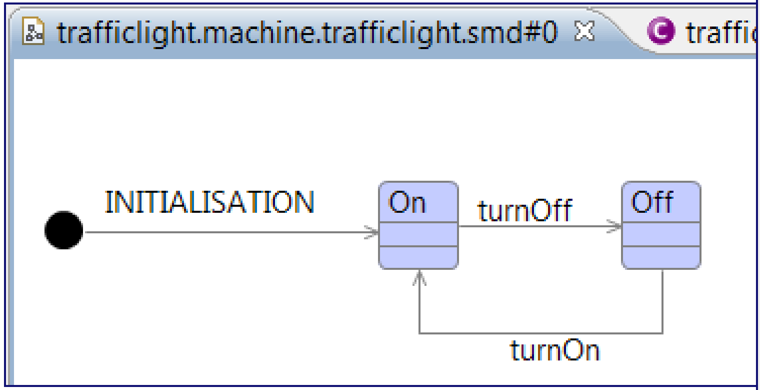


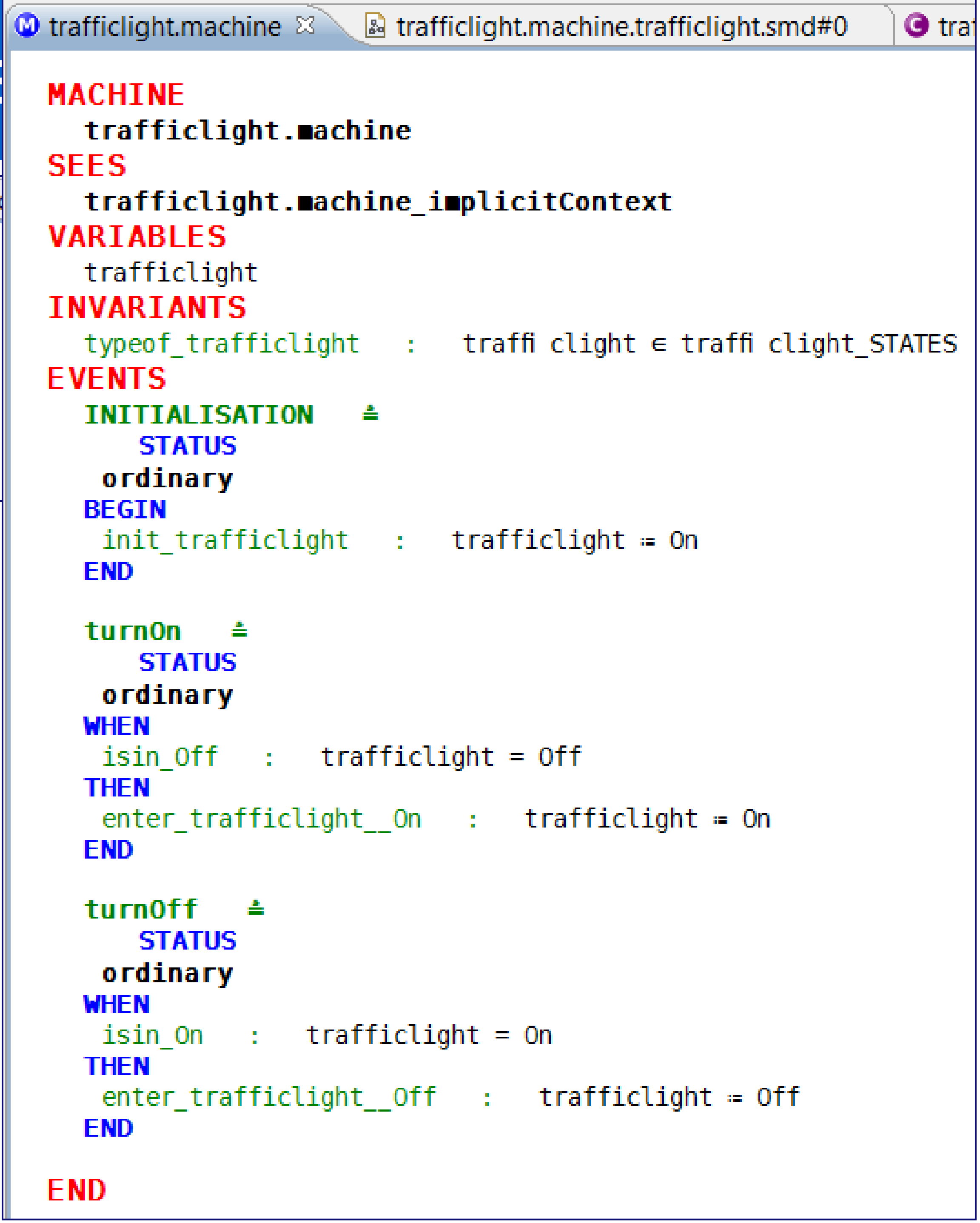
Event-B is generated from the state machine!


trafficlight.machine.trafficlight.smd#0
trafficlight.machine_implicitContext
trafficlight.machine

```
CONTEXT
   trafficlight.machine_implicitContext
SETS
   trafficlight_STATES
CONSTANTS
   On
   Off
AXIOMS
   typeof_On    :    On ∈ trafficlight_STATES
   typeof_Off    :    Off ∈ trafficlight_STATES
   distinct_states_in_trafficlight_STATES    :    partition(trafficlight_STATES, {On}, {Off})
END
```

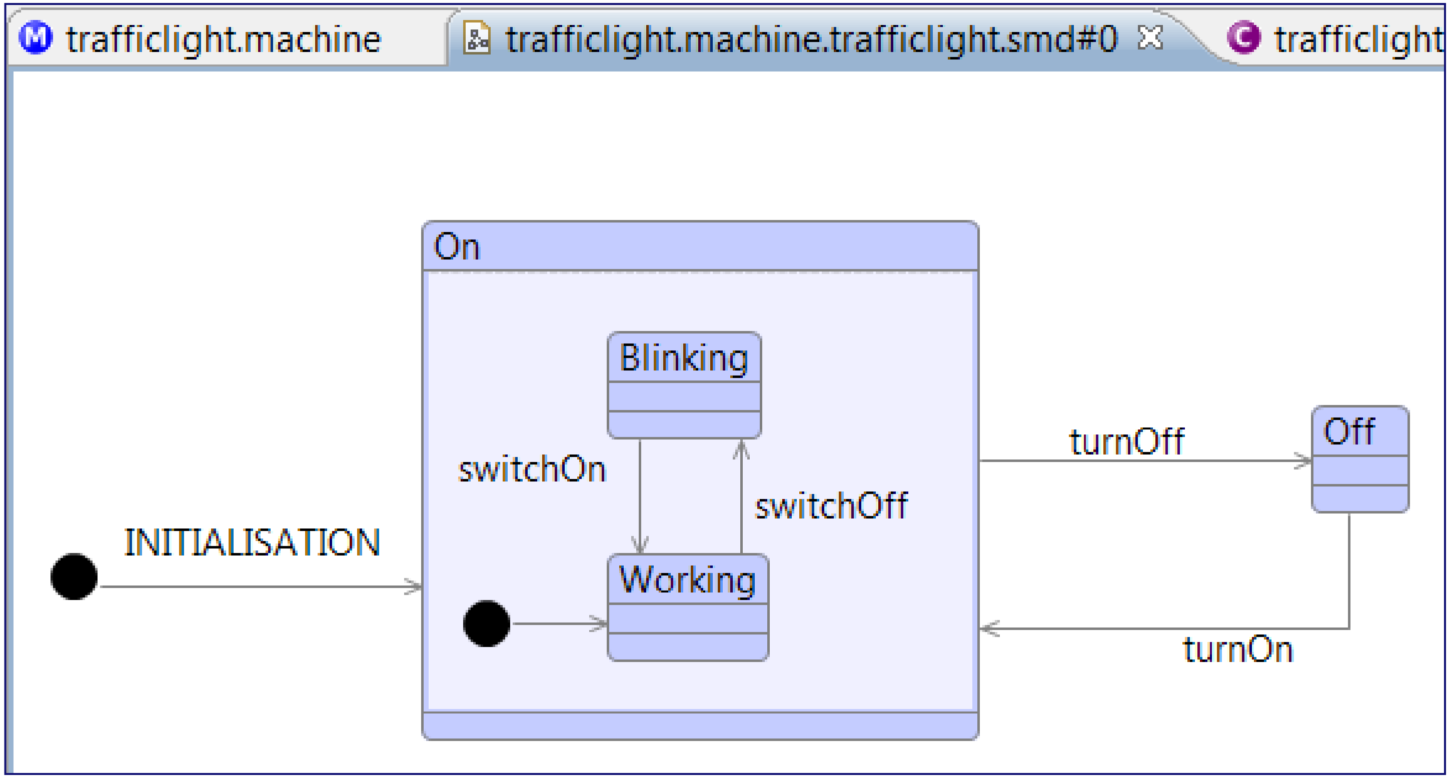
Generated Event-



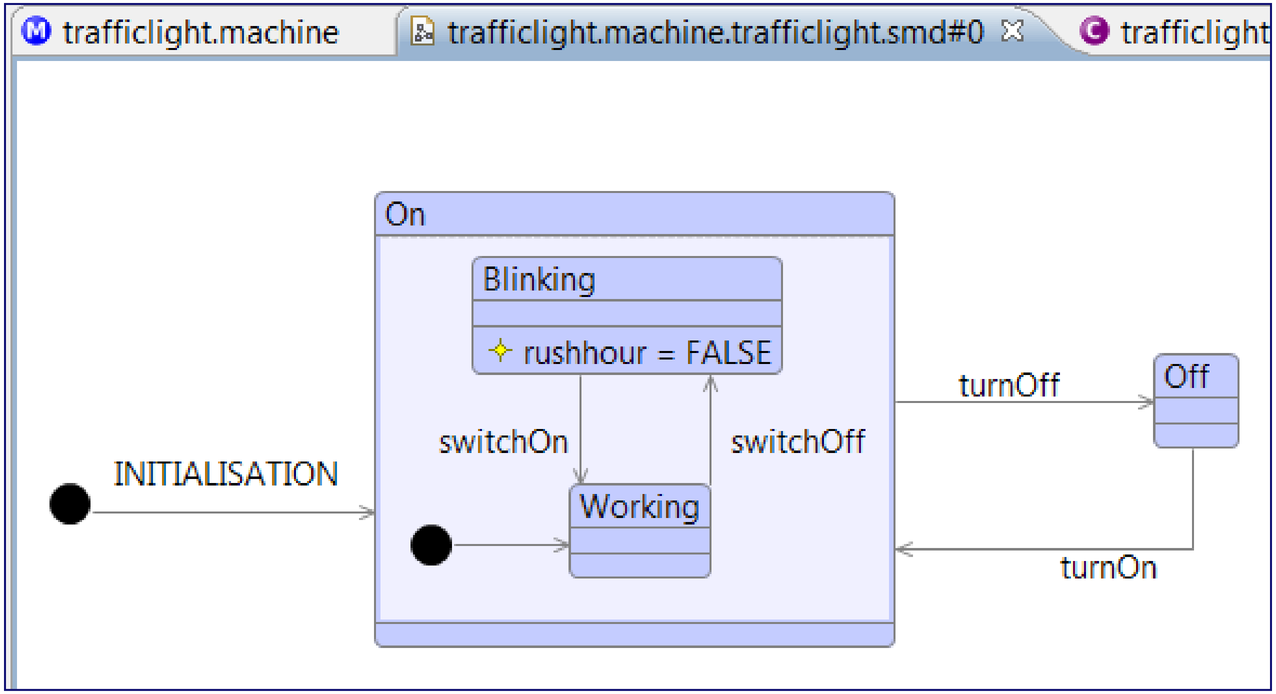


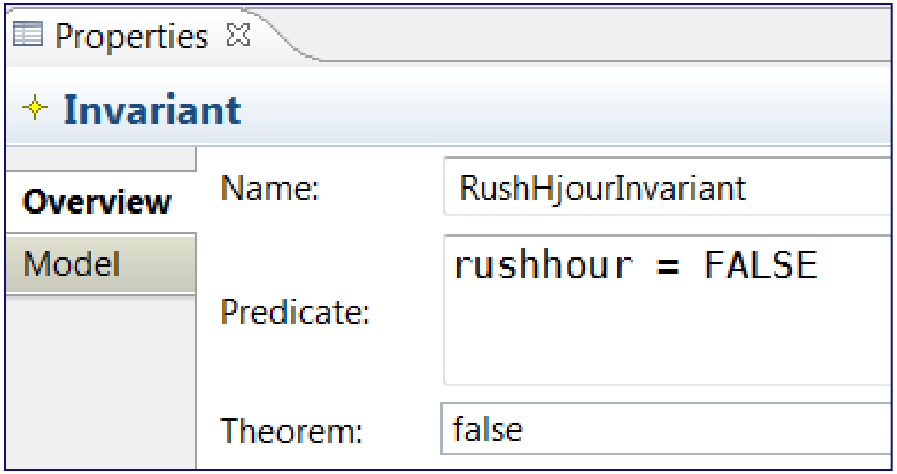
```
MACHINE
   trafficlight.■achine
SEES
   trafficlight.■achine_i■plicitContext
VARIABLES
   trafficlight
INVARIANTS
   typeof_trafficlight   :   trafficlight ∈ trafficlight_STATES
EVENTS
   INITIALISATION   ≙
       STATUS
    ordinary
   BEGIN
    init_trafficlight   :   trafficlight ≔ On
   END

   turnOn   ≙
       STATUS
    ordinary
   WHEN
    isin_Off   :   trafficlight = Off
   THEN
    enter_trafficlight__On   :   trafficlight ≔ On
   END

   turnOff   ≙
       STATUS
    ordinary
   WHEN
    isin_On   :   trafficlight = On
   THEN
    enter_trafficlight__Off   :   trafficlight ≔ Off
   END

END
```

Nested state machine



State invariant

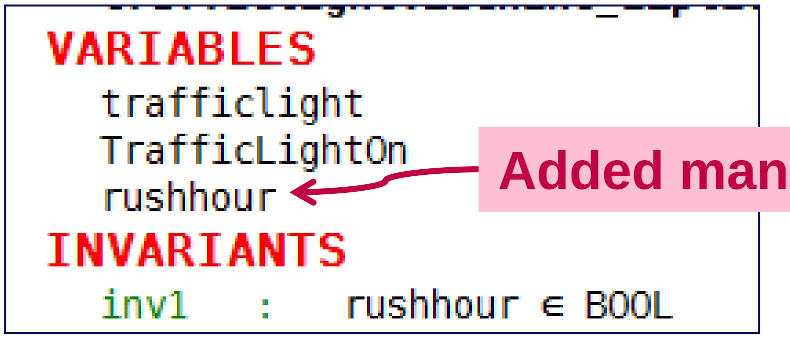




Name: RushHjourInvariant

Predicate: rushhour = FALSE

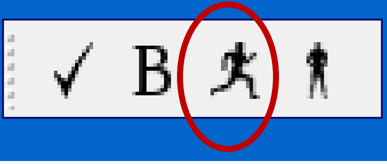
Theorem: false

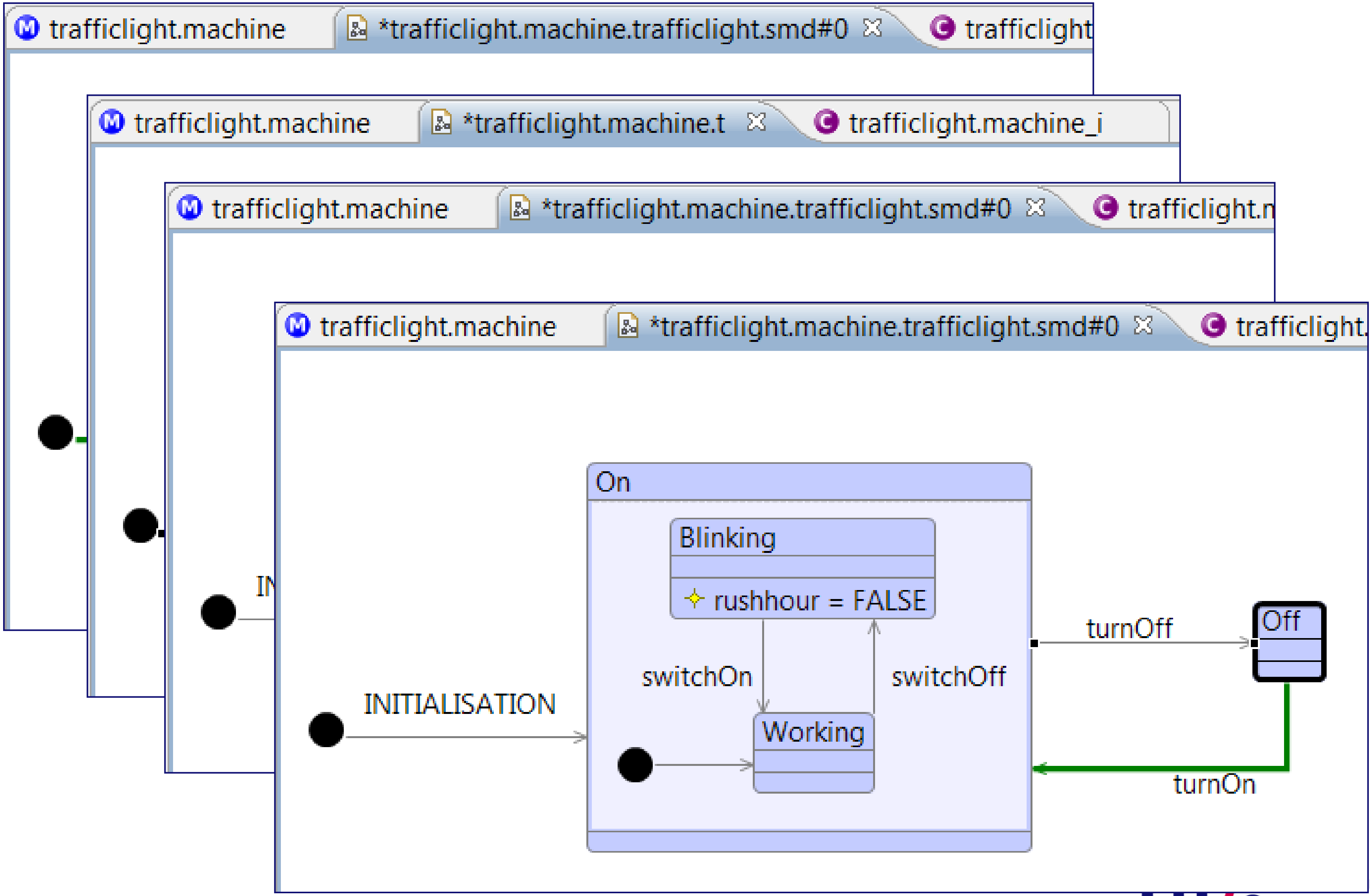


```
VARIABLES
    trafficlight
    TrafficLightOn
    rushhour
INVARIANTS
    inv1   :   rushhour ∈ BOOL
```

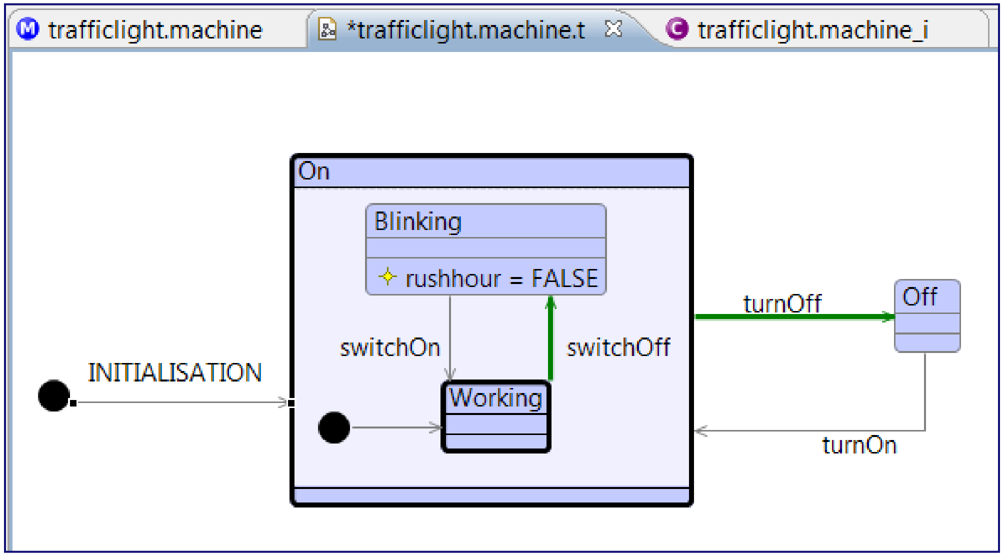
Added manually

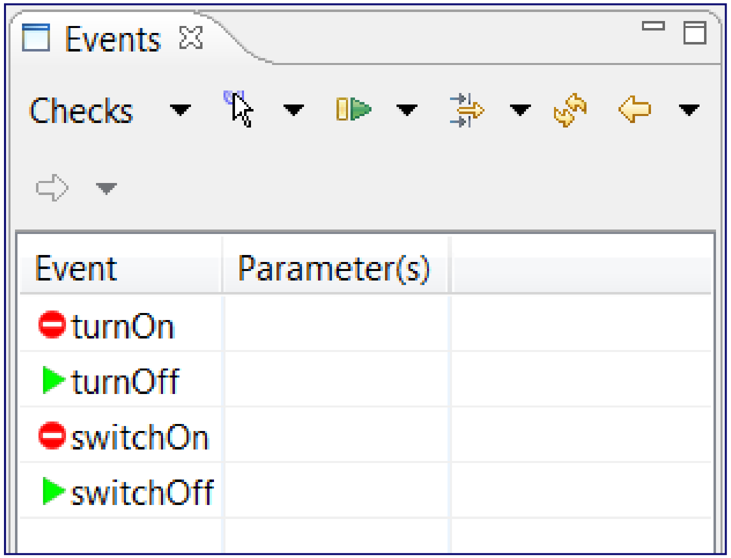
Animate state machine diagram



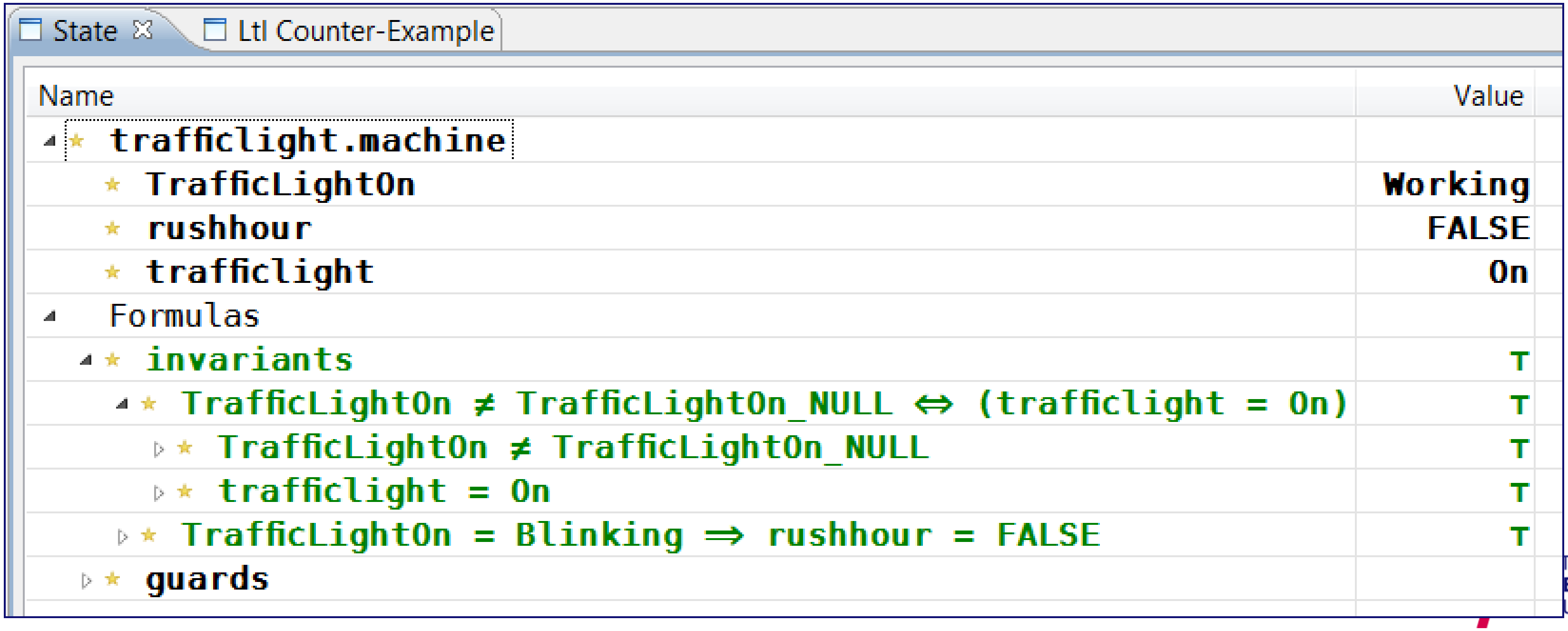


What is behind your state machine animation?





Event	Parameter(s)
turnOn	
turnOff	
switchOn	
switchOff	



Name	Value
trafficlight.machine	
TrafficLightOn	Working
rushhour	FALSE
trafficlight	On
Formulas	
invariants	T
TrafficLightOn ≠ TrafficLightOn_NULL ⇔ (trafficlight = On)	T
TrafficLightOn ≠ TrafficLightOn_NULL	T
trafficlight = On	T
TrafficLightOn = Blinking ⇒ rushhour = FALSE	T
guards	

UML vs. formal specification

- Using UML we can:
  a. Draw pictures
  b. Analyze the system
  c. Communicate with developers and customers
  d. Design the system
  e. Generate source code

UML vs. formal specification

- Using formal specification we can:
  a. Draw pictures
  b. Analyze the system
  c. Communicate with developers and customers
  d. Design the system
  e. Generate source code

Formal methods as a specification technique?

Unambiguous?

Realistic?

Verifiable?

Evolvable?

Formal methods as a specification technique?

Unambiguous?

- yes, formal notation and formal theory

Realistic?

- might require a lot of effort on modeling a correct system

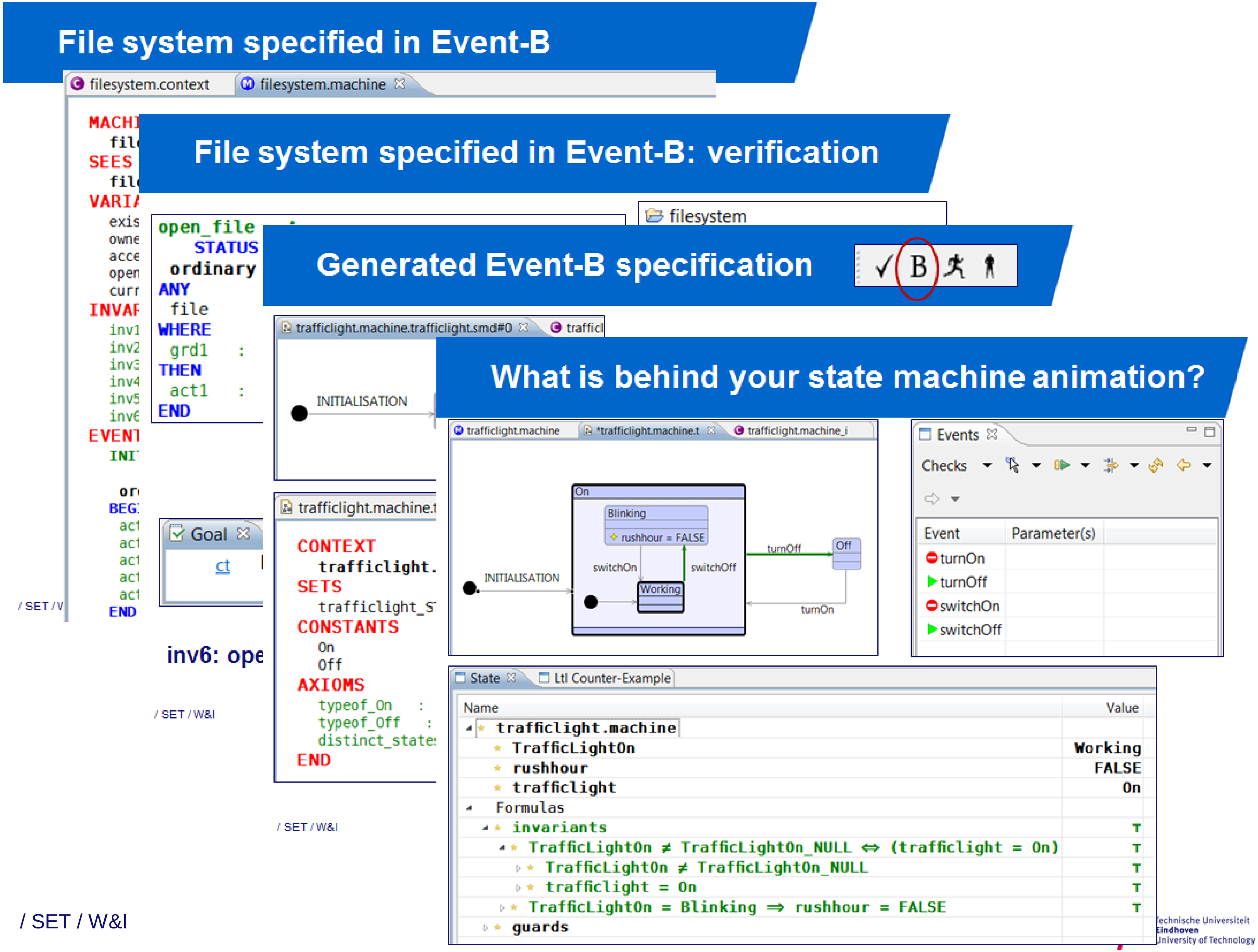
Verifiable?

- yes, formal methods

Evolvable?

- real-life systems: a lot of details ⇒
  huge formal specifications, which are hard to maintain

Brief recapitulation



File system specified in Event-B

filesystem.context | filesystem.machine

```
MACHI
  fil
SEES
  fil
VARIA
  exis
  owne
  acce
  open
  curr
INVAR
  inv1
  inv2
  inv3
  inv4
  inv5
  inv6
EVENT
  INI

   or
  BEG
   act
   act
   act
   act
   act
  END
```

/ SET / W

File system specified in Event-B: verification

```
open_file  ≙
   STATUS
 ordinary
ANY
 file
WHERE
 grd1    :
THEN
 act1    :
END
```

filesystem

Goal

ct

inv6: ope

/ SET / W&I

Generated Event-B specification

✓ B

trafficlight.machine.trafficlight.smd#0 | trafficl

trafficlight.machine.t

```
CONTEXT
  trafficlight.
SETS
  trafficlight_S
CONSTANTS
  On
  Off
AXIOMS
  typeof_On    :
  typeof_Off    :
  distinct_state
END
```

/ SET / W&I

What is behind your state machine animation?

trafficlight.machine | *trafficlight.machine.t | trafficlight.machine_i

Events

Checks

Event	Parameter(s)
⛔ turnOn	
▶ turnOff	
⛔ switchOn	
▶ switchOff	

State | Ltl Counter-Example

Name	Value
trafficlight.machine	
TrafficLightOn	Working
rushhour	FALSE
trafficlight	On
Formulas	
invariants	T
TrafficLightOn ≠ TrafficLightOn_NULL ⇔ (trafficlight = On)	T
TrafficLightOn ≠ TrafficLightOn_NULL	T
trafficlight = On	T
TrafficLightOn = Blinking ⇒ rushhour = FALSE	T
guards	