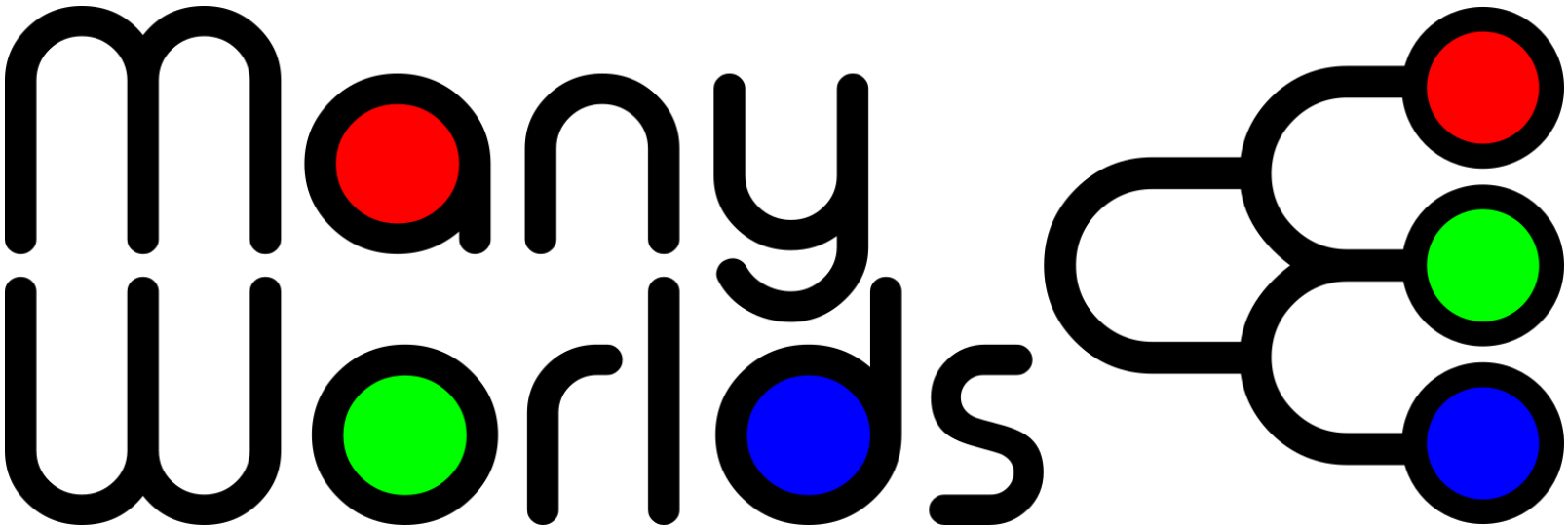


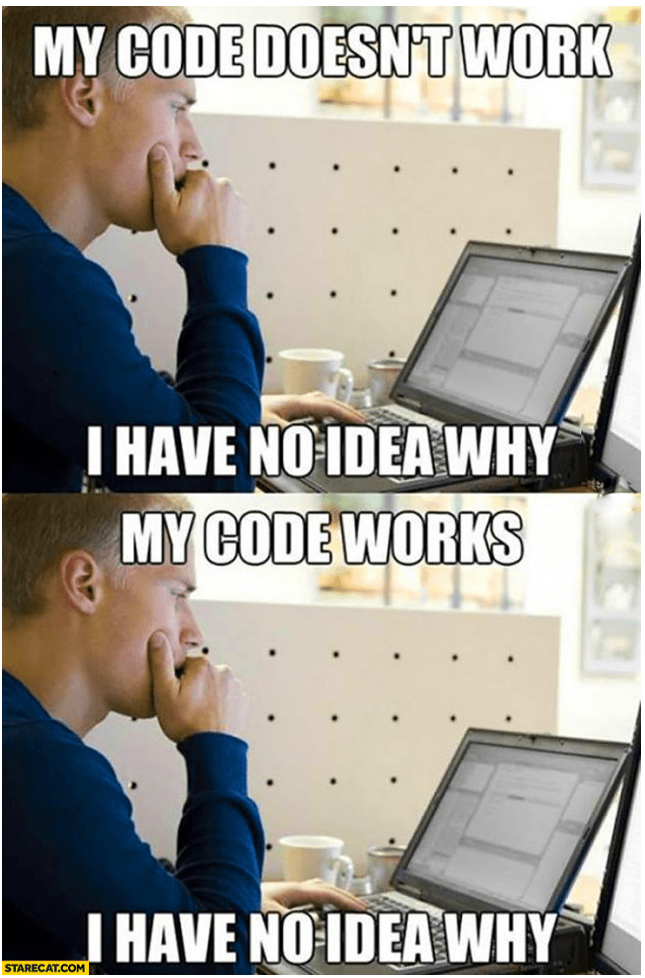
Combinatorial Programming with Functions

slides.com/jod/manyworlds-modref

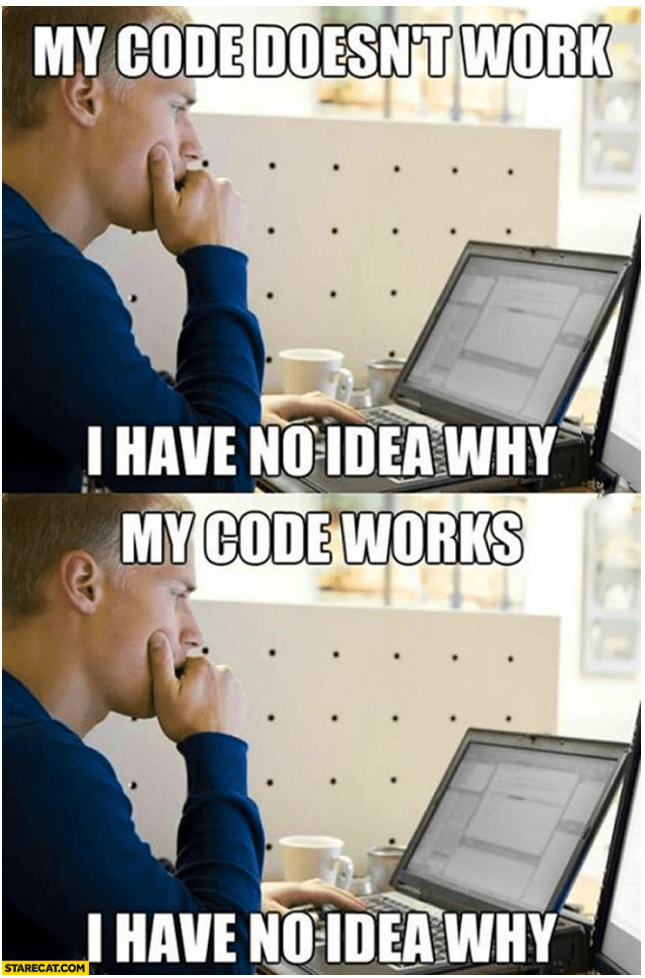
NONFICTION SOFTWARE _

Jo Devriendt

We were all n00bs once...

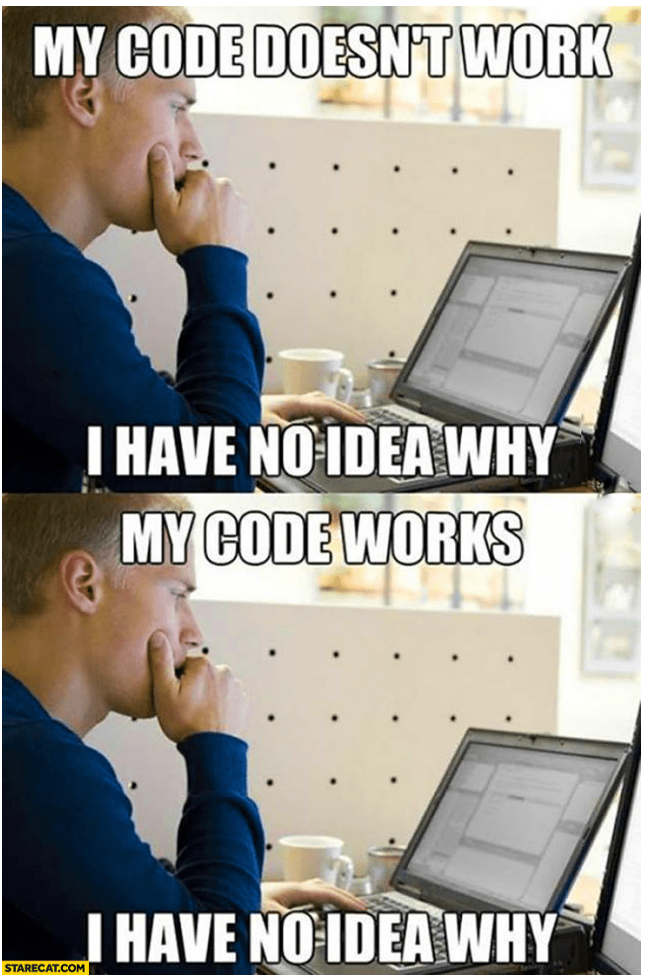


...with a problem to solve



- a tournament roster for your sports club
- a seat arrangement for a wedding
- a dice odds calculation for a game
- a class schedule for a school
- ...

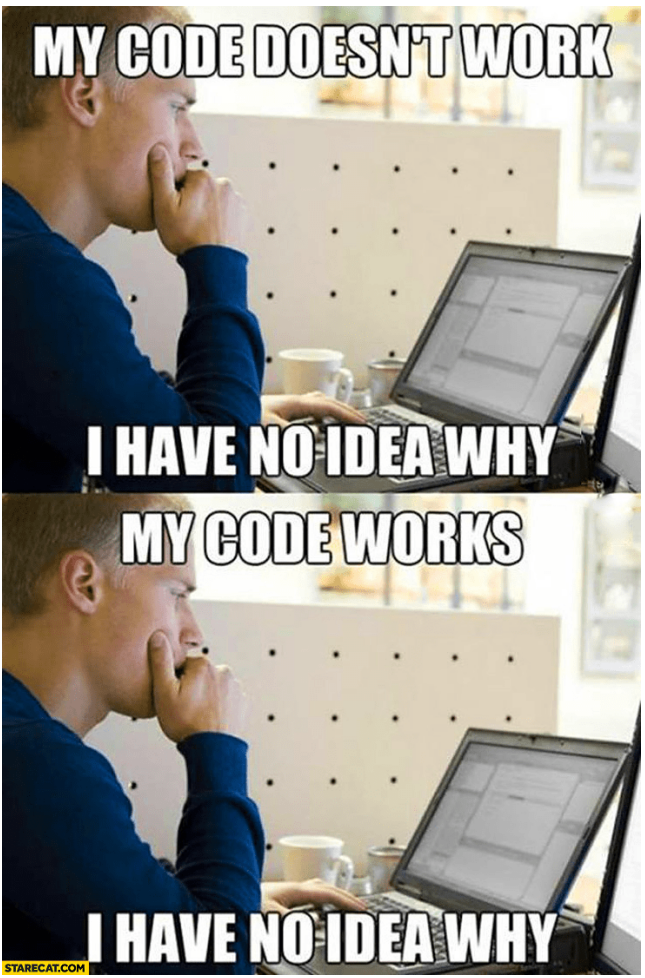
...with a problem to solve



- a tournament roster for your sports club
- a seat arrangement for a wedding
- a dice odds calculation for a game
- a class schedule for a school
- ...

You were good at high-school math, regularly use spreadsheet formulas.

...with a problem to solve

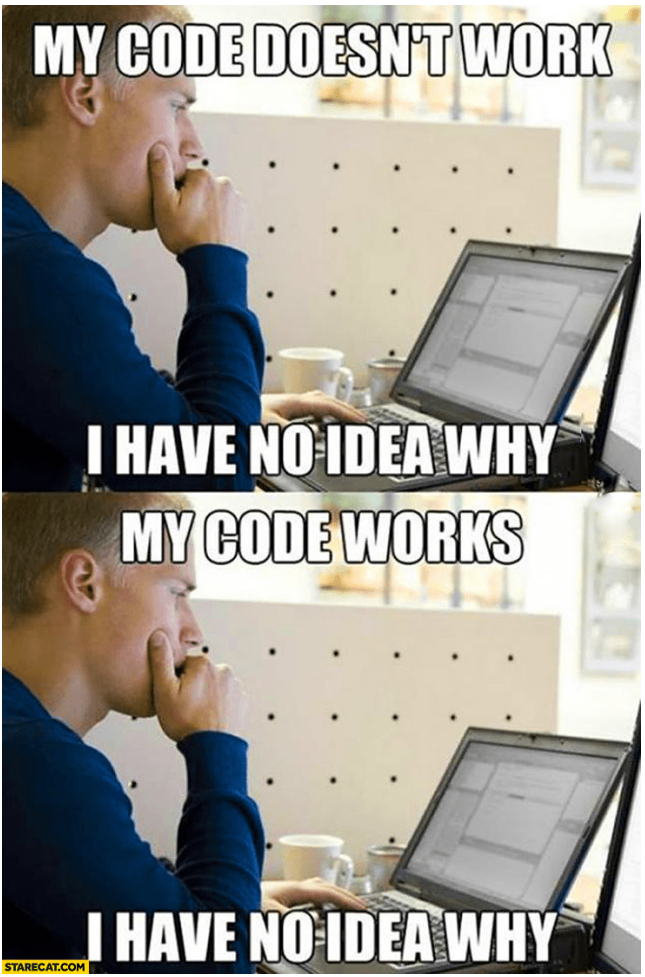


- a tournament roster for your sports club
- a seat arrangement for a wedding
- a dice odds calculation for a game
- a class schedule for a school
- ...

You were good at high-school math, regularly use spreadsheet formulas.

So you know about **arithmetic formulas** and **function application**.

...with a problem to solve



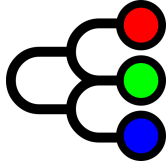
- a tournament roster for your sports club
- a seat arrangement for a wedding
- a dice odds calculation for a game
- a class schedule for a school
- ...

You were good at high-school math, regularly use spreadsheet formulas.

So you know about **arithmetic formulas** and **function application**.

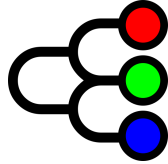
How would you go about this?

ManyWorlds



A friendly, small, simple (??) combinatorial programming language

ManyWorlds



A friendly, small, simple (??) combinatorial programming language

Basic building blocks:

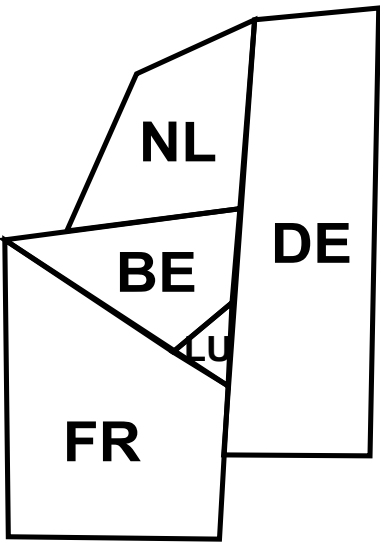
values

`bool, int, string`

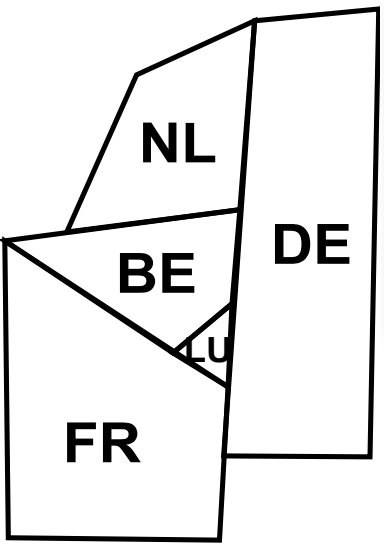
total functions

`type* -> type`

Map coloring

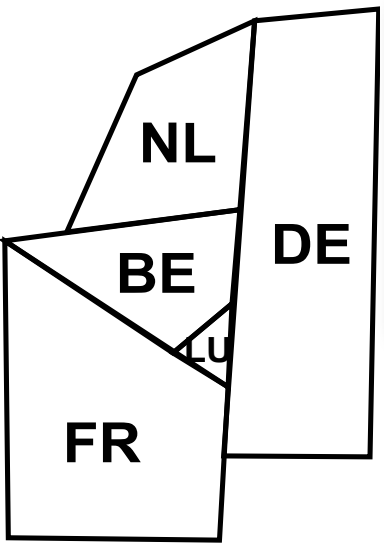


Map coloring




```
1 declare color: string -> {"r", "g", "b", "y"}.
2
3 color("NL") != color("BE"). color("NL") != color("DE").
4 color("BE") != color("LU"). color("BE") != color("DE").
5 color("BE") != color("FR"). color("FR") != color("LU").
6 color("FR") != color("DE"). color("LU") != color("DE").
```

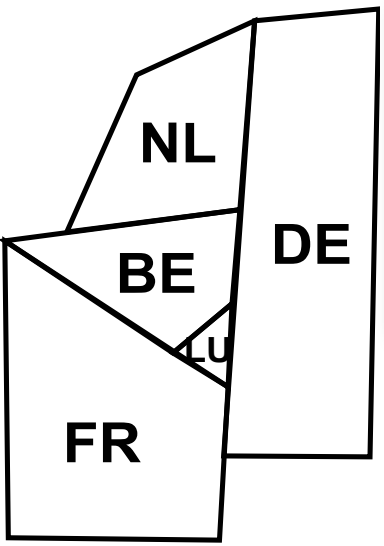
Map coloring




```
1 declare color: string -> {"r", "g", "b", "y"}.
2
3 color("NL") != color("BE"). color("NL") != color("DE").
4 color("BE") != color("LU"). color("BE") != color("DE").
5 color("BE") != color("FR"). color("FR") != color("LU").
6 color("FR") != color("DE"). color("LU") != color("DE").
```

- **User functions**: uninterpreted functions with a total co-domain [*variable(s) (arrays)*]

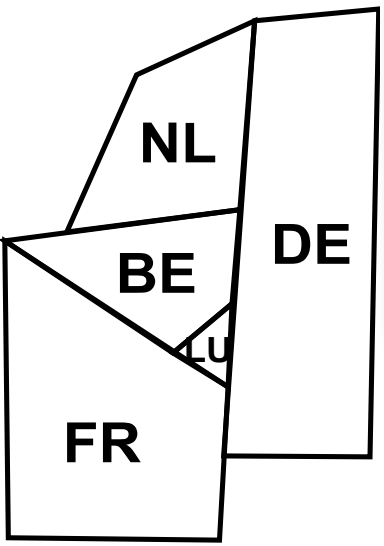
Map coloring




```
1 declare color: string -> {"r", "g", "b", "y"}.
2
3 color("NL") != color("BE"). color("NL") != color("DE").
4 color("BE") != color("LU"). color("BE") != color("DE").
5 color("BE") != color("FR"). color("FR") != color("LU").
6 color("FR") != color("DE"). color("LU") != color("DE").
```

- **User functions**: uninterpreted functions with a total co-domain [*variable(s) (arrays)*]
- **Expressions**: (nested) function applications

Map coloring

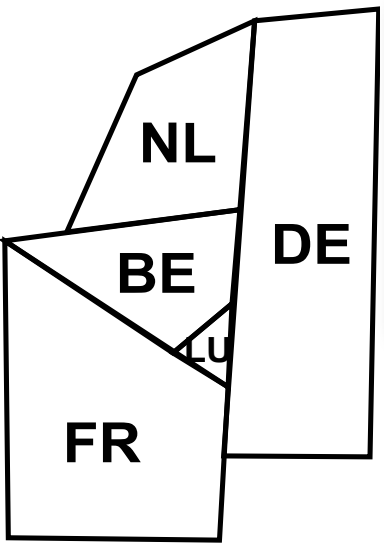



```
1 declare color: string -> {"r", "g", "b", "y"}.
2
3 color("NL") != color("BE"). color("NL") != color("DE").
4 color("BE") != color("LU"). color("BE") != color("DE").
5 color("BE") != color("FR"). color("FR") != color("LU").
6 color("FR") != color("DE"). color("LU") != color("DE").
```

- **User functions**: uninterpreted functions with a total co-domain [*variable(s) (arrays)*]
- **Expressions**: (nested) function applications
- **Constraints**: expressions that must be true

NF
S_

Map coloring

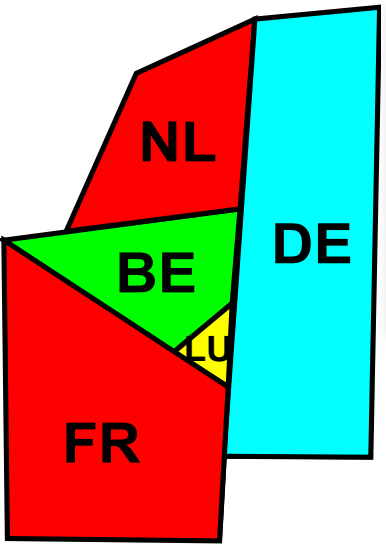



```
1 declare color: string -> {"r", "g", "b", "y"}.
2
3 color("NL") != color("BE"). color("NL") != color("DE").
4 color("BE") != color("LU"). color("BE") != color("DE").
5 color("BE") != color("FR"). color("FR") != color("LU").
6 color("FR") != color("DE"). color("LU") != color("DE").
```

- **User functions**: uninterpreted functions with a total co-domain [*variable(s) (arrays)*]
- **Expressions**: (nested) function applications
- **Constraints**: expressions that must be true
- **Worlds**: user function interpretations that make all constraints true [*solutions*]

NF
S_

Map coloring



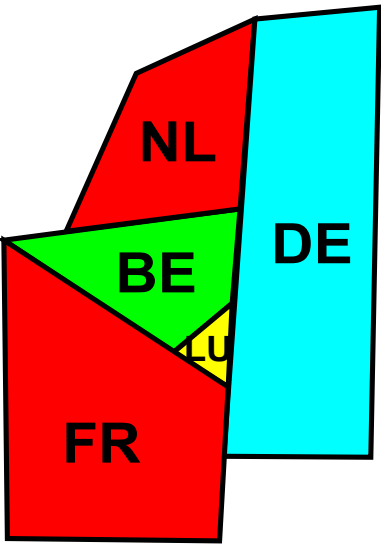

```
1 declare color: string -> {"r", "g", "b", "y"}.
2
3 color("NL") != color("BE"). color("NL") != color("DE").
4 color("BE") != color("LU"). color("BE") != color("DE").
5 color("BE") != color("FR"). color("FR") != color("LU").
6 color("FR") != color("DE"). color("LU") != color("DE").
```

```
FINDING...

FOUND WORLD
define color as {("BE","g"), ("DE","b"), ("FR","r"),
("LU","y"), ("NL","r")} default unknown.
```

- **User functions**: uninterpreted functions with a total co-domain [*variable(s) (arrays)*]
- **Expressions**: (nested) function applications
- **Constraints**: expressions that must be true
- **Worlds**: user function interpretations that make all constraints true [*solutions*]

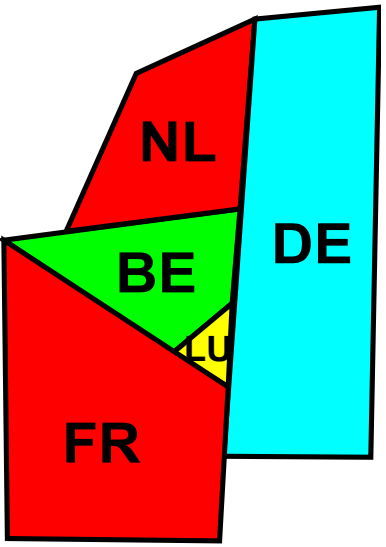
NF
S_

Map coloring bis




```
1 declare color: string -> {"r", "g", "b", "y"}.
2
3 // declare border: string, string -> bool.
4 decdef border as {("NL","BE"), ("NL","DE"),
5 ("BE","LU"), ("BE","DE"), ("BE","FR"),
6 ("FR","LU"), ("FR","DE"), ("LU","DE")}.
7
8 all [ color(x)!=color(y) for x,y where border(x,y) ].
```

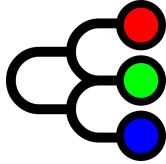
Map coloring bis




```
1 declare color: string -> {"r", "g", "b", "y"}.
2
3 // declare border: string, string -> bool.
4 decdef border as {("NL","BE"), ("NL","DE"),
5 ("BE","LU"), ("BE","DE"), ("BE","FR"),
6 ("FR","LU"), ("FR","DE"), ("LU","DE")}.
7
8 all [ color(x)!=color(y) for x,y where border(x,y) ].
```

"**Fold-Map-Filter**"
expression

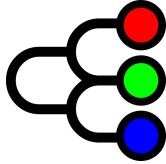
FMF expressions



```
all [ color(x)!=color(y) for x,y where border(x,y) ].
```

NFS_

FMF expressions

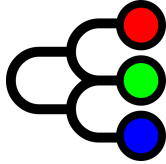


```
all [ color(x)!=color(y) for x,y where border(x,y) ].
```

Filter: select all `x,y` where `border(x,y)` holds

NFS_

FMF expressions



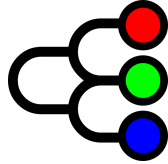
```
all [ color(x)!=color(y) for x,y where border(x,y) ].
```

Map: map those `x,y` to `color(x)!=color(y)`

Filter: select all `x,y` where `border(x,y)` holds

NFS_

FMF expressions



Fold: reduce those `color(x)!=color(y)` to `true` iff `all` are `true`

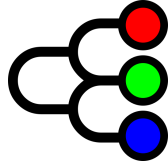
```
all [ color(x)!=color(y) for x,y where border(x,y) ].
```

Map: map those `x,y` to `color(x)!=color(y)`

Filter: select all `x,y` where `border(x,y)` holds

NFS_

FMF expressions



Fold: reduce those `color(x)!=color(y)` to `true` iff `all` are `true`

```
all [ color(x)!=color(y) for x,y where border(x,y) ].
```

Map: map those `x,y` to `color(x)!=color(y)`

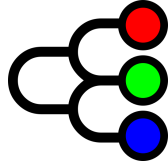
Filter: select all `x,y` where `border(x,y)` holds

```
all
any
none
count
sum
product
```

```
min
max
distinct
same
odd
even
```

NF
S_

FMF expressions



Fold: reduce those `color(x)!=color(y)` to `true` iff `all` are `true`

```
all [ color(x)!=color(y) for x,y where border(x,y) ].
```

Map: map those `x,y` to `color(x)!=color(y)`

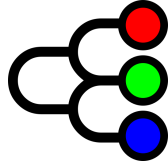
Filter: select all `x,y` where `border(x,y)` holds

`all`	`min`
`any`	`max`
`none`	`distinct`
`count`	`same`
`sum`	`odd`
`product`	`even`

- generalize quantification / aggregates

NFS_

FMF expressions



Fold: reduce those `color(x)!=color(y)` to `true` iff `all` are `true`

```
all [ color(x)!=color(y) for x,y where border(x,y) ].
```

Map: map those `x,y` to `color(x)!=color(y)`

Filter: select all `x,y` where `border(x,y)` holds

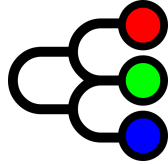
```
all
any
none
count
sum
product
```

```
min
max
distinct
same
odd
even
```

- generalize quantification / aggregates
- take over role of "joker" global constraints - e.g., alldiff_except_0

NFS_

FMF expressions



Fold: reduce those `color(x)!=color(y)` to `true` iff `all` are `true`

```
all [ color(x)!=color(y) for x,y where border(x,y) ].
```

Map: map those `x,y` to `color(x)!=color(y)`

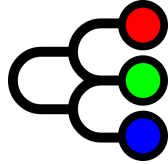
Filter: select all `x,y` where `border(x,y)` holds

`all`	`min`
`any`	`max`
`none`	`distinct`
`count`	`same`
`sum`	`odd`
`product`	`even`

- generalize quantification / aggregates
- take over role of "joker" global constraints - e.g., alldiff_except_0
- nestable

NFS_

More language extensions

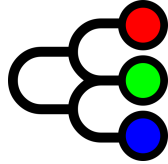


Done

- builtin functions and operators
- syntactic sugar
- arbitrary precision integers
- user types
- intensional definitions
- python definitions

NF
S_

More language extensions



Done

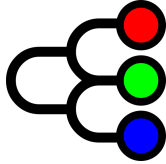
- builtin functions and operators
- syntactic sugar
- arbitrary precision integers
- user types
- intensional definitions
- python definitions

To do

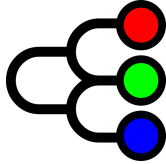
- pseudo-rational division
- tuple values
- recursive definitions
- ...

NF
S_

Inferences & objectives



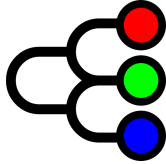
Inferences & objectives

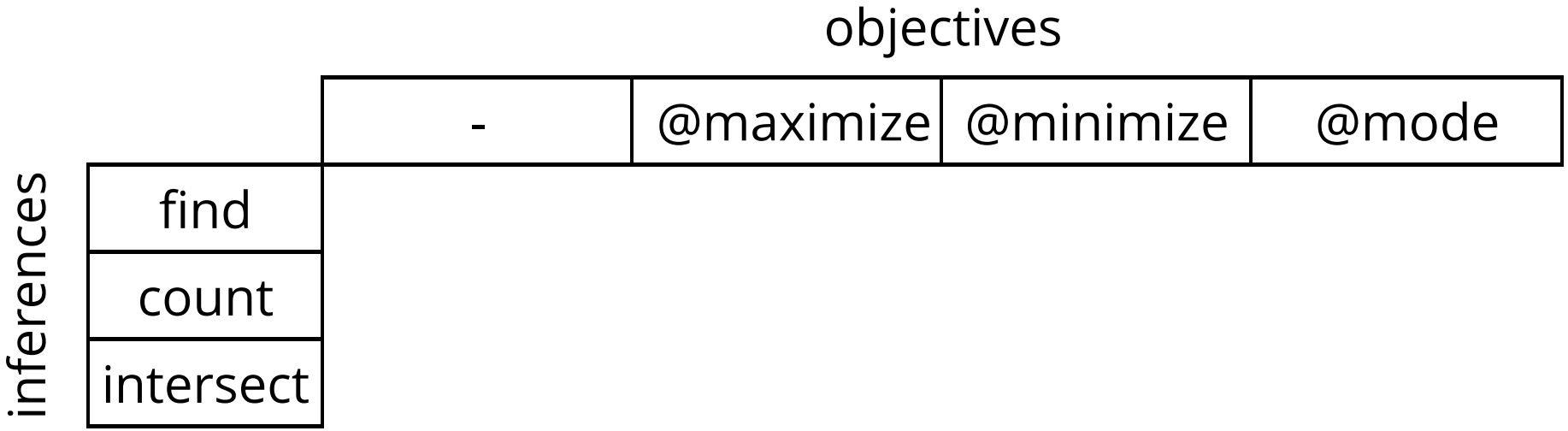


inferences

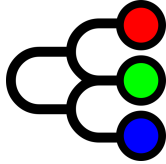
find
count
intersect

Inferences & objectives





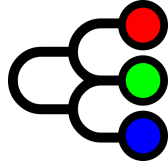
Inferences & objectives



objectives

inferences	-	@maximize	@minimize	@mode
find	✓	✓	✓	✓
count	✓	✓	✓	✓
intersect	✓	✓	✓	✓

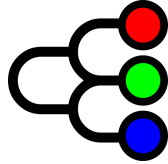
Count + @mode example



```
1  // we have 5 dice
2  decdef die as {"d" 1 .. 5}.
3
4  // with 1 to 6 dots
5  decdef dots as {1 .. 6}.
6
7  // rolling assigns a number of dots to each die
8  declare roll: die -> dots.
9
10 // the sum of the dice rolls must be 14
11 sum[ roll(x) for x where die(x) ] = 14.
12
13 // at most two dice can have rolled the same dots
14 all[
15   count[ roll(x) = y for x where die(x) ] <= 2
16 for y where dots(y) ].
17
18 // show the most common value ('mode') of the highest die
19 // this also yields statistics on the highest die
20 @mode max[ roll(x) for x where die(x) ].
21
```

NF
S_

Count + @mode example

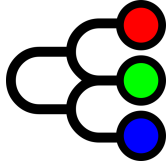


```
1  // we have 5 dice
2  decdef die as {"d" 1 .. 5}.
3
4  // with 1 to 6 dots
5  decdef dots as {1 .. 6}.
6
7  // rolling assigns a number of dots to each
8  declare roll: die -> dots.
9
10 // the sum of the dice rolls must be 14
11 sum[ roll(x) for x where die(x) ] = 14.
12
13 // at most two dice can have rolled the same
14 all[
15   count[ roll(x) = y for x where die(x) ] <= 2
16 for y where dots(y) ].
17
18 // show the most common value ('mode') of the highest die
19 // this also yields statistics on the highest die
20 @mode max[ roll(x) for x where die(x) ].
21
```

```
PARSING...
#seconds 0.000000

COMPILING...
#vars 61 #constraints 124 #seconds 0.00100
7776 candidate(s) exist

CALCULATING DISTRIBUTION OF WORLDS...
#seconds 0.060000
450 world(s) exist
-> 5.787037% of candidates

5: 210 (mode objective fixed to this)
6: 150
4: 90
mean:   5.133333 (77/15)
median: 5
```

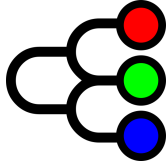
NF
S_

Development support



- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

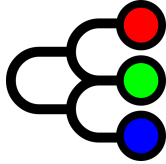
Development support



- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

(3-valued) **expression evaluation**

Development support



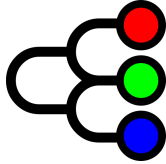
- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

(3-valued) **expression evaluation**

```
declare drinksAlcohol: -> bool.
declare age: -> {0 .. 150}.

age() >= 18 implies drinksAlcohol().

define drinksAlcohol() as true.
define age() as 0.
```

NF
S_

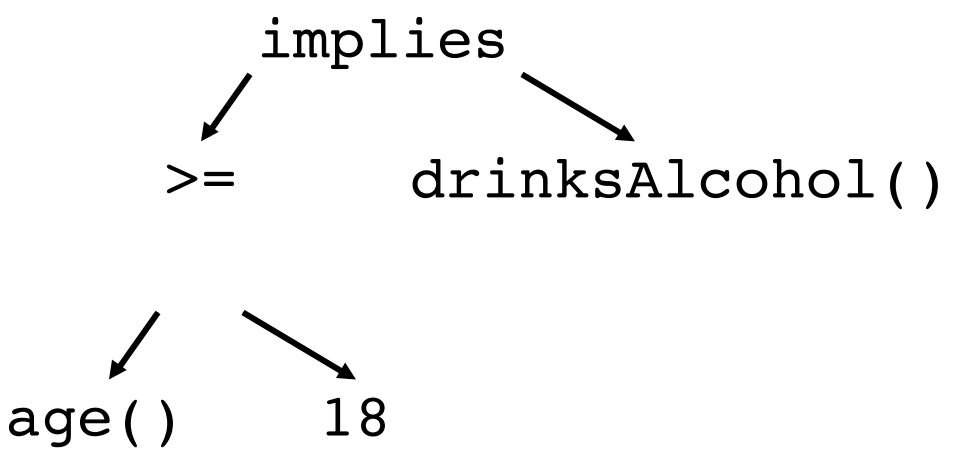
Development support



- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

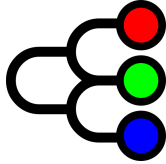
(3-valued) **expression evaluation**

```
declare drinksAlcohol: -> bool.
declare age: -> {0 .. 150}.

age() >= 18 implies drinksAlcohol().

define drinksAlcohol() as true.
define age() as 0.
```



NFS_

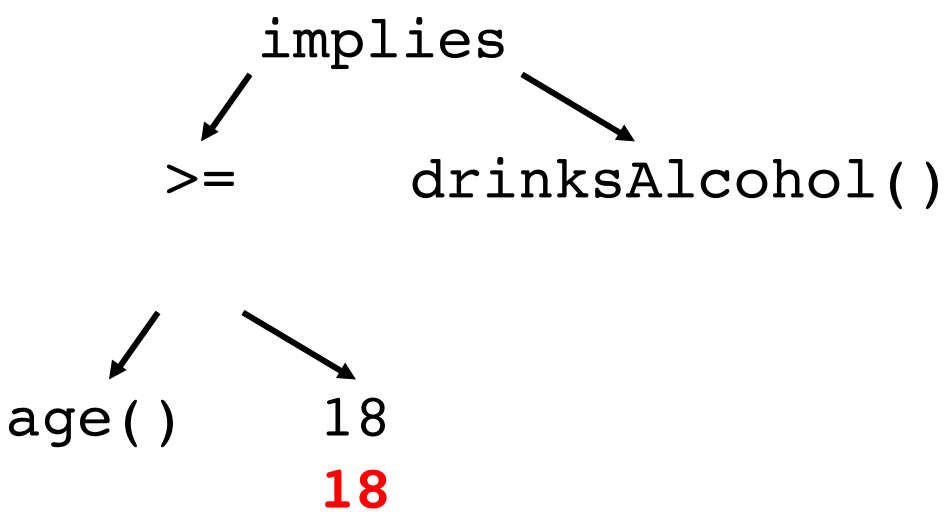
Development support



- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

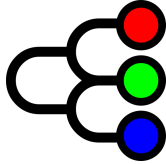
(3-valued) **expression evaluation**

```
declare drinksAlcohol: -> bool.
declare age: -> {0 .. 150}.

age() >= 18 implies drinksAlcohol().

define drinksAlcohol() as true.
define age() as 0.
```



NFS_

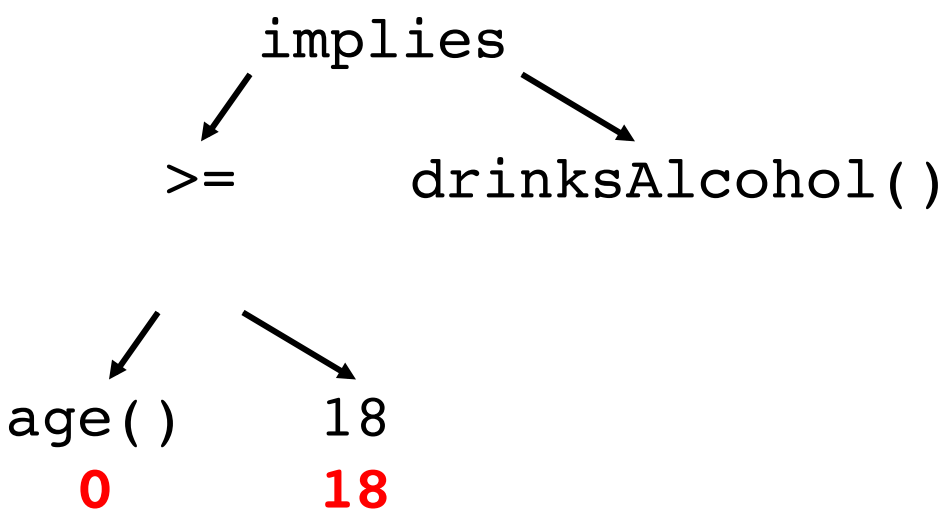
Development support



- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

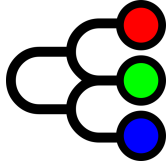
(3-valued) **expression evaluation**

```
declare drinksAlcohol: -> bool.
declare age: -> {0 .. 150}.

age() >= 18 implies drinksAlcohol().

define drinksAlcohol() as true.
define age() as 0.
```



NFS_

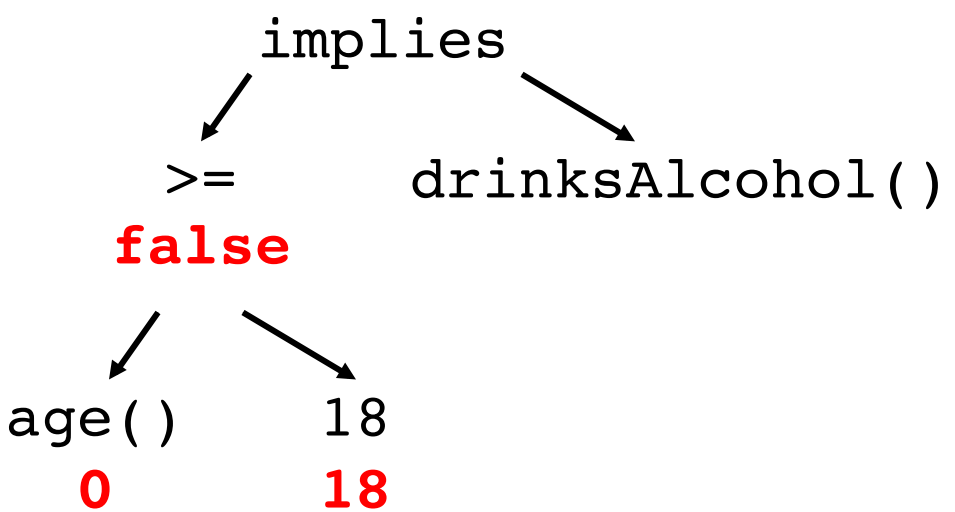
Development support



- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

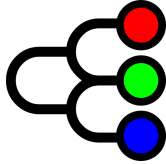
(3-valued) **expression evaluation**

```
declare drinksAlcohol: -> bool.
declare age: -> {0 .. 150}.

age() >= 18 implies drinksAlcohol().

define drinksAlcohol() as true.
define age() as 0.
```



NFS_

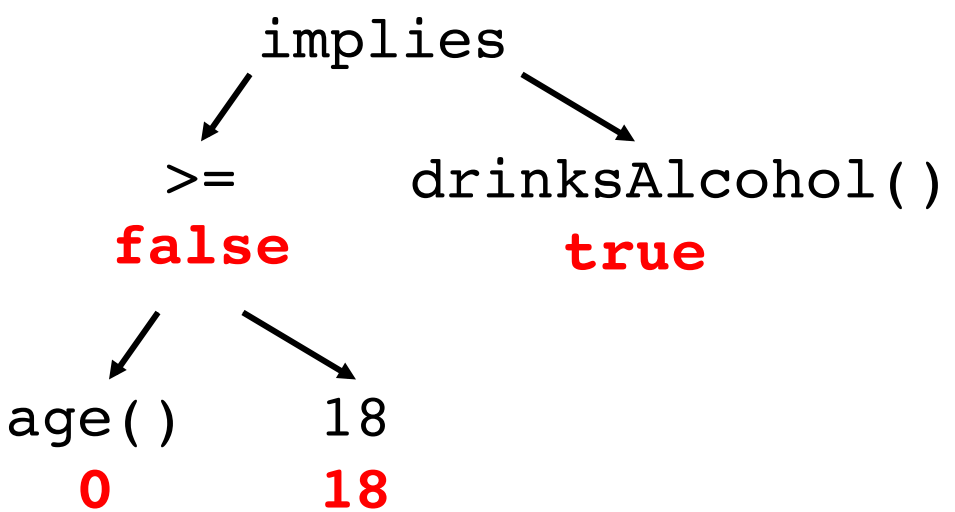
Development support



- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

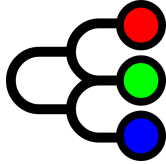
(3-valued) **expression evaluation**

```
declare drinksAlcohol: -> bool.
declare age: -> {0 .. 150}.

age() >= 18 implies drinksAlcohol().

define drinksAlcohol() as true.
define age() as 0.
```



NFS_

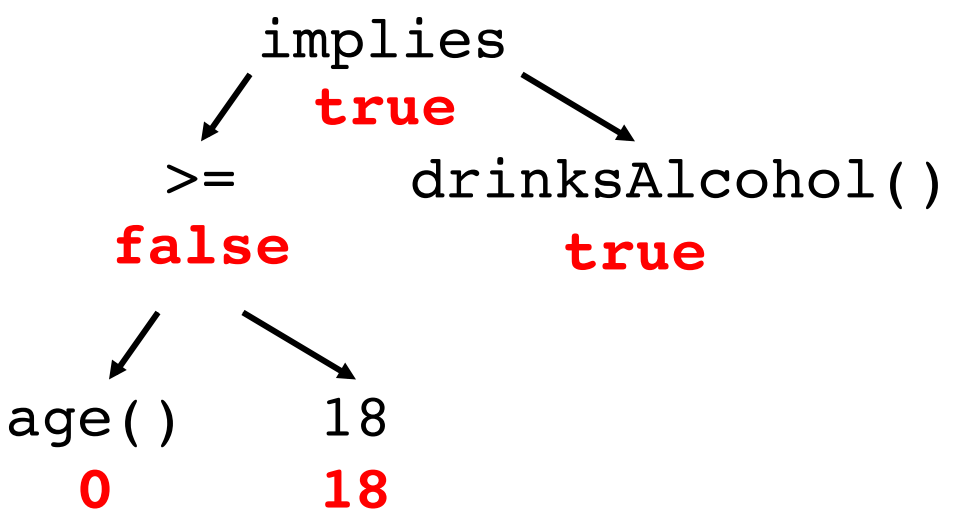
Development support



- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

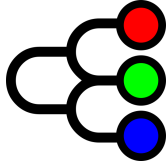
(3-valued) **expression evaluation**

```
declare drinksAlcohol: -> bool.
declare age: -> {0 .. 150}.

age() >= 18 implies drinksAlcohol().

define drinksAlcohol() as true.
define age() as 0.
```



NFS_

Development support

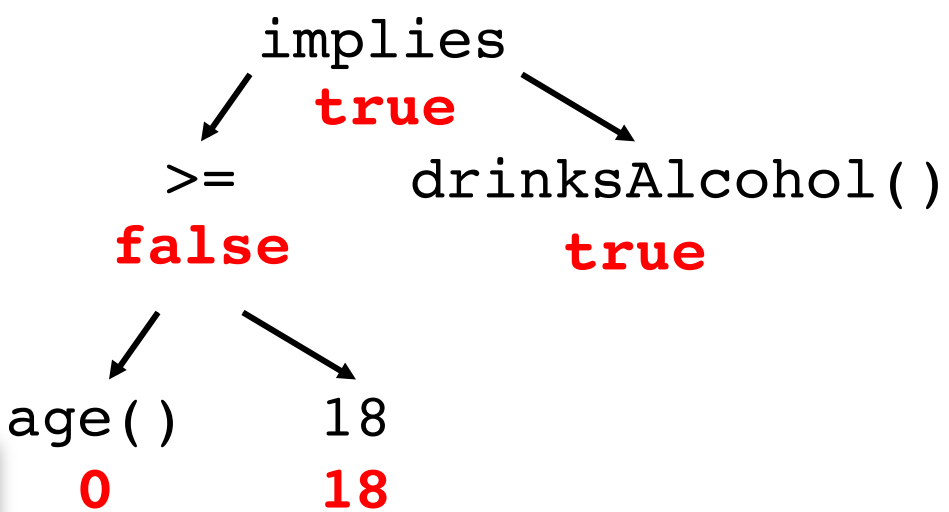


- online IDE
- simple syntax highlighting
- helpful error/warning messages
- **debugging** support

(3-valued) **expression evaluation**

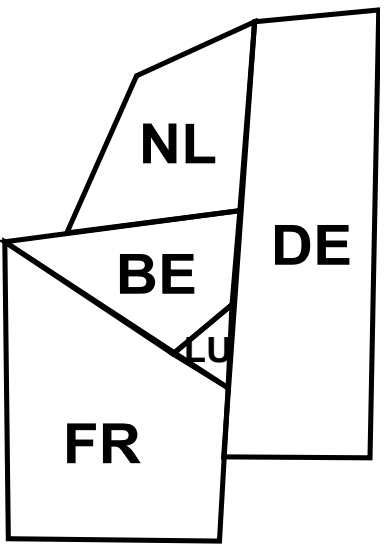
```
declare drinksAlcohol: -> bool.
declare age: -> {0 .. 150}.

age() >= 18 implies drinksAlcohol().

define drinksAlcohol() as true.
define age() as 0.
```

```
·    ·    age() [0]
·    >= [false]
·    ·    18
implies [true]
·    drinksAlcohol() [true].
```



NFS_

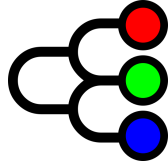
Development support



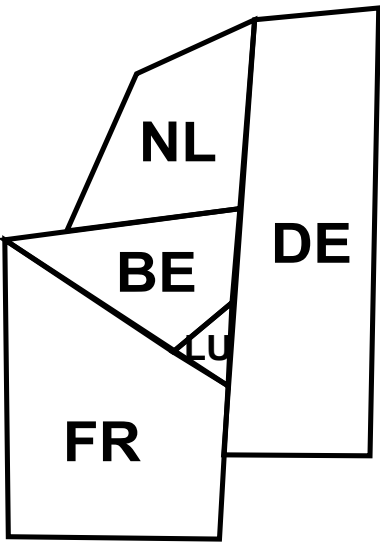
detailed **UNSAT explanation**

```
1 declare color: string -> {"r", "g", "b", "y"}.
2
3 decdef border as {("NL","BE"), ("NL","DE"),
4 ("BE","LU"), ("BE","DE"), ("BE","FR"),
5 ("FR","LU"), ("FR","DE"), ("LU","DE")}.
6
7 all [
8     color(x)!=color(y)
9 for x,y where border(x,y) ].
```

Development support



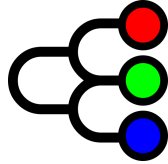
detailed **UNSAT explanation**



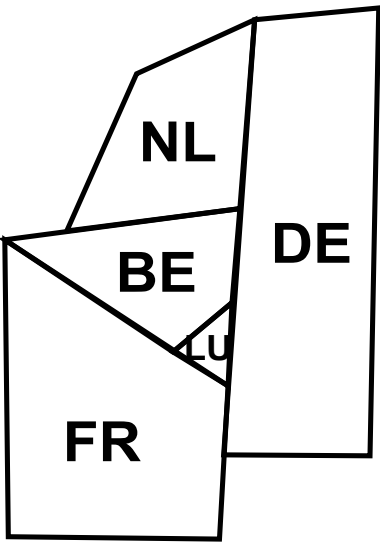

```
1 declare color: string -> {"r", "g", "b"}.
2
3 decdef border as {("NL","BE"), ("NL","DE"),
4 ("BE","LU"), ("BE","DE"), ("BE","FR"),
5 ("FR","LU"), ("FR","DE"), ("LU","DE")}.
6
7 all [
8     color(x)!=color(y)
9 for x,y where border(x,y) ].
```

NFS_

Development support



detailed **UNSAT explanation**

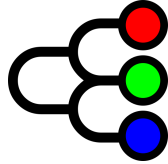



```
1 declare color: string -> {"r", "g", "b"}.
2
3 decdef border as {("NL","BE"), ("NL","DE"),
4 ("BE","LU"), ("BE","DE"), ("BE","FR"),
5 ("FR","LU"), ("FR","DE"), ("LU","DE")}.
6
7 all [
8     color(x)!=color(y)
9 for x,y where border(x,y) ].
```

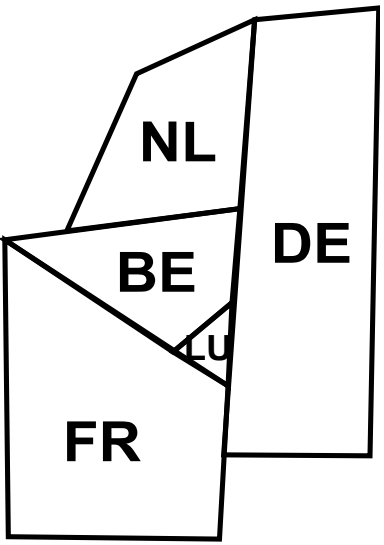
```
FOUND UNSATISFIABILITY

BLOCKERS
Line 7: all[not color(x)=color(y) for x,y where border(x,y)]
```

NFS_

Development support



detailed **UNSAT explanation**

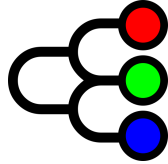



```
1 declare color: string -> {"r", "g", "b"}.
2
3 decdef border as {("NL","BE"), ("NL","DE"),
4 ("BE","LU"), ("BE","DE"), ("BE","FR"),
5 ("FR","LU"), ("FR","DE"), ("LU","DE")}.
6
7 all [
8     color(x)!=color(y)
9 for x,y where border(x,y) ].
```

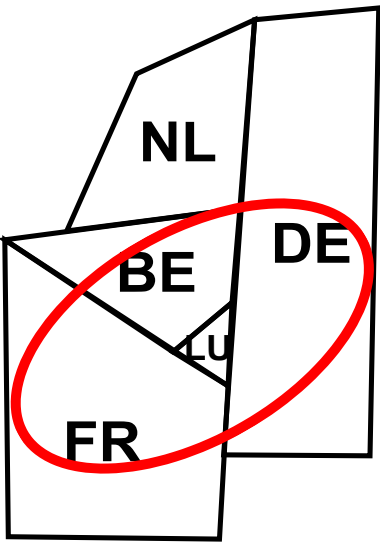
```
FOUND UNSATISFIABILITY

DETAILED BLOCKERS
Line 7: not color("BE")=color("DE")
Line 7: not color("BE")=color("FR")
Line 7: not color("BE")=color("LU")
Line 7: not color("DE")=color("FR")
Line 7: not color("DE")=color("LU")
Line 7: not color("FR")=color("LU")
```

NFS_

Development support



detailed **UNSAT explanation**

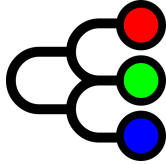



```
1 declare color: string -> {"r", "g", "b"}.
2
3 decdef border as {("NL","BE"), ("NL","DE"),
4 ("BE","LU"), ("BE","DE"), ("BE","FR"),
5 ("FR","LU"), ("FR","DE"), ("LU","DE")}.
6
7 all [
8     color(x)!=color(y)
9 for x,y where border(x,y) ].
```

```
FOUND UNSATISFIABILITY

DETAILED BLOCKERS
Line 7: not color("BE")=color("DE")
Line 7: not color("BE")=color("FR")
Line 7: not color("BE")=color("LU")
Line 7: not color("DE")=color("FR")
Line 7: not color("DE")=color("LU")
Line 7: not color("FR")=color("LU")
```

NFS_

Performance?

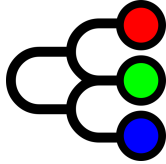


Backend solver: Exact

https://gitlab.com/nonfiction-software/exact

NF
S_

Performance?



Backend solver: Exact

https://gitlab.com/nonfiction-software/exact

On-call rostering

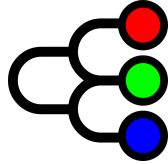
	MiniZinc (Gecode)		MiniZinc (OR-Tools)		ManyWorlds (Exact)	
Instance	Objective	Time	Objective	Time	Objective	Time
2s-200d	65	25 000+	**64**	**6204.30**	**64**	**0.04**
4s-100d	58	25 000+	61	25 000+	**2**	**0.24**
10s-100d-C	47	25 000+	**47**	**1.29**	**47**	**0.06**
20s-100d-B	59	25 000+	58	25 000+	**16**	**0.43**
30s-400d-A	293	25 000+	299	25 000+	**2**	**11.12**

Table 1 Objective values and runtimes (in seconds) for three different approaches to solve the on-call rostering problem. Entries in bold denote that optimality was proven.

https://github.com/MiniZinc/mzn-challenge/blob/develop/2018/on-call-rostering/oc-roster.mzn

NF
S_

Where to get



Running

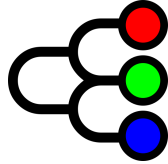
- Website with examples & online editor: **manyworlds.site**
- Source code (should compile on Linux) gitlab.com/nonfiction-software/manyworlds
- Dockerfile and Docker image

Learning

- The ModRef paper
- These slides: slides.com/jod/manyworlds-modref
- Mailing list: groups.google.com/g/manyworlds-lang
- Subreddit: reddit.com/r/manyworlds

NF
S_

Where to get



Running

- Website with examples & online editor: **manyworlds.site**
- Source code (should compile on Linux) gitlab.com/nonfiction-software/manyworlds
- Dockerfile and Docker image

Learning

- The ModRef paper
- These slides: slides.com/jod/manyworlds-modref
- Mailing list: groups.google.com/g/manyworlds-lang
- Subreddit: reddit.com/r/manyworlds

NF
S_

Thanks for your attention!