https://refinery.tools

Efficient Graph-Based Reachability Analysis

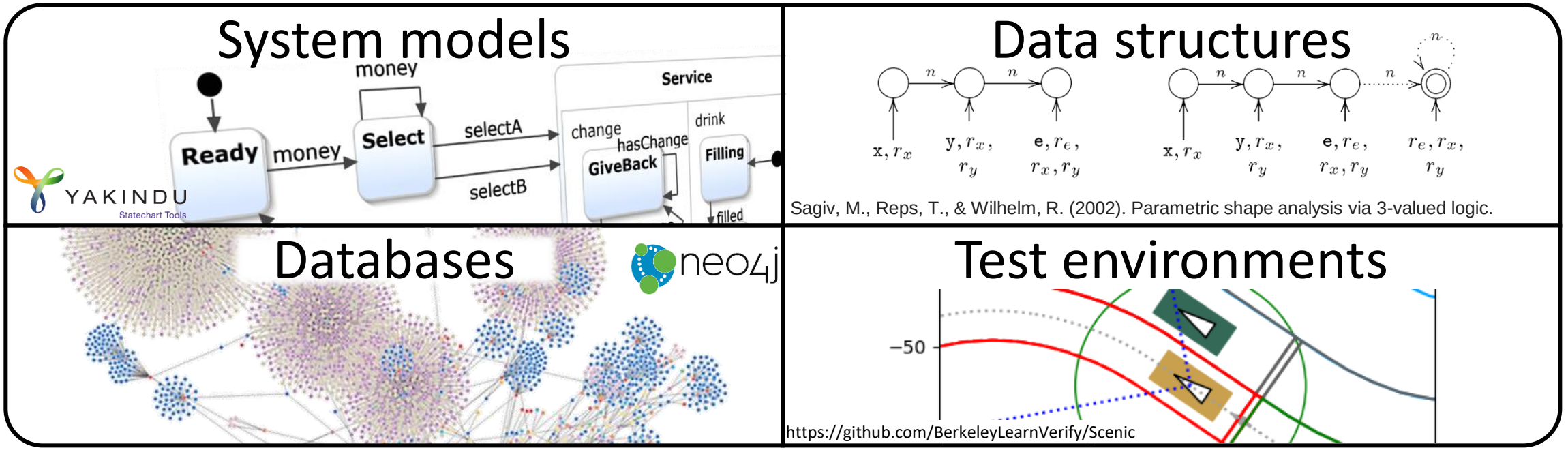
Refinery:
Refinement-Based Generation and Analysis of Consistent Models

Attila Ficsor

MŰEGYETEM 1782
ftsrg
Critical Systems Research Group
McGill UNIVERSITY
LINKÖPING UNIVERSITY
WASP WALLENBERG AI, AUTONOMOUS SYSTEMS AND SOFTWARE PROGRAM
amazon research awards
esa
Office of Naval Research GLOBAL Science & Technology

Modeling with Graphs

- Graph based models are widely used in software engineering

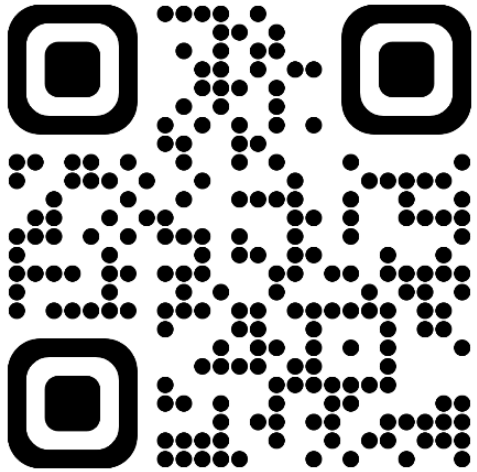


- Testing, benchmarking or design space exploration scenarios

Generating **(consistent | realistic | diverse | scalable)** models

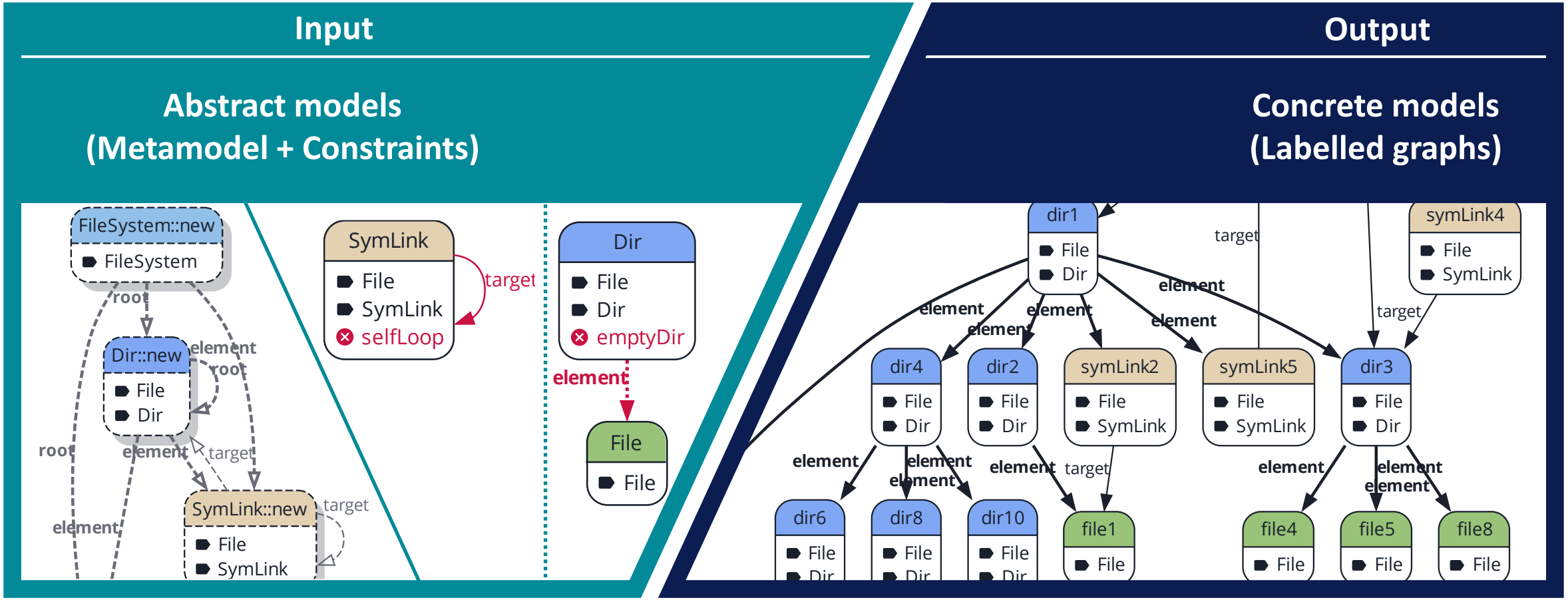
Refinery: Graph Solver as a Service

- **Logical reasoning** and **model generation** over graphs
- **Web-based editor:**
  - **Live editing** and **feedback**
  - Support for partial models, graph constraints and propagation rules
- **Efficient storage** of model versions
- Incremental query engine
- **Calculating difference** between model versions
- Framework for further graph processing tasks
  - *Ideas?*



https://refinery.services/

Models and Partial Models



AWS

Consistent + Diverse +
Scalable: up to **tens of thousands** of nodes

Refinery

<> CODE | GRAPH | TABLE | GENERATE

```
% Type system
class FileSystem {
    contains File[1] root
}
class File.
class Dir extends File {
    contains File[0..10] element
}
class SymLink extends File {
    File[1] target
}

% Facts about objects
Dir(resources).
element(resources, img).
!Dir(img).
element(resources, link).
target(link,img).

% Simple constraints:
error pred selfLoop(s) ⟷ target(s, s).
error pred emptyDir(d) ⟷ Dir(d), !element(d, _).
```

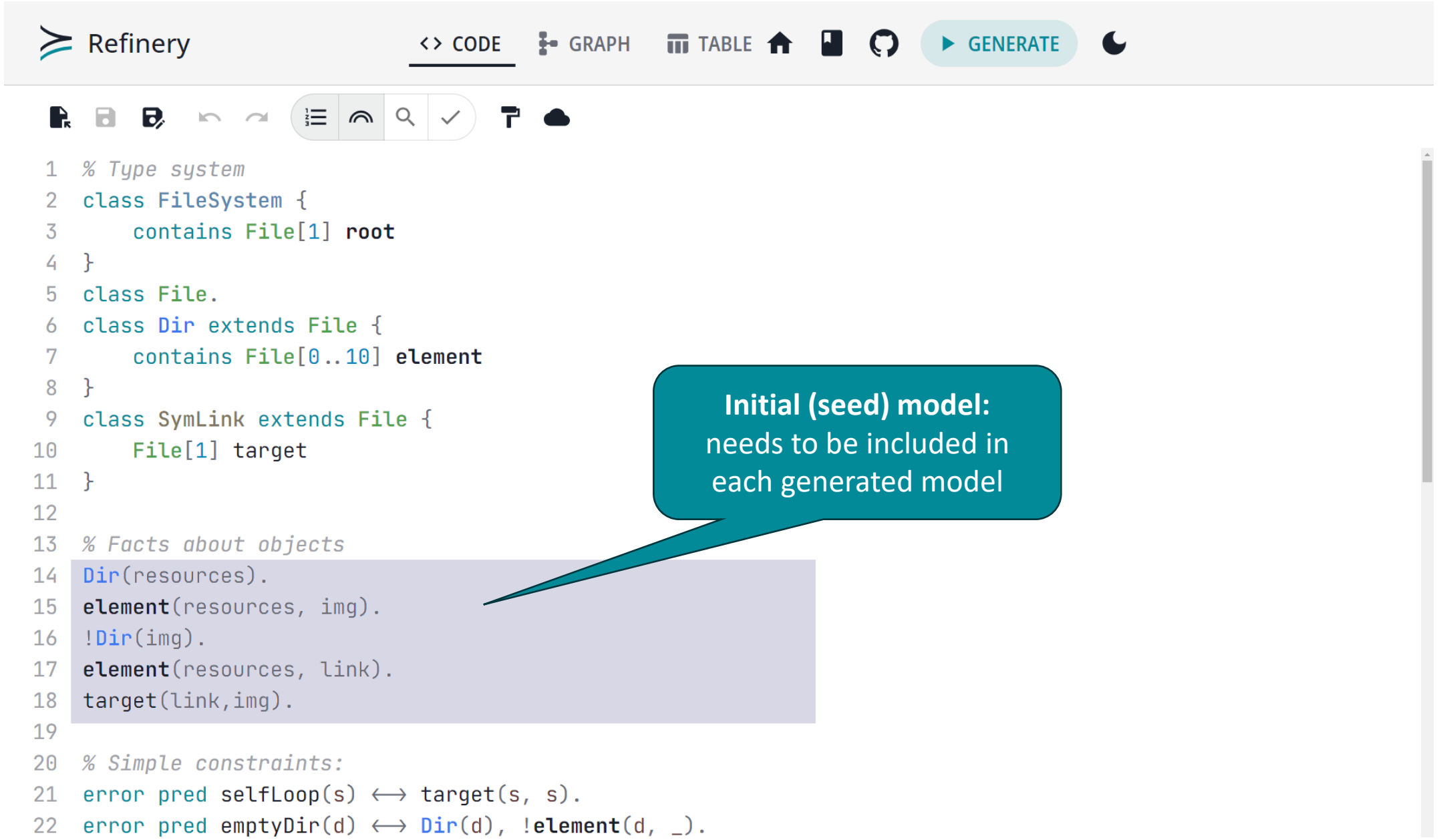
https://refinery.services/

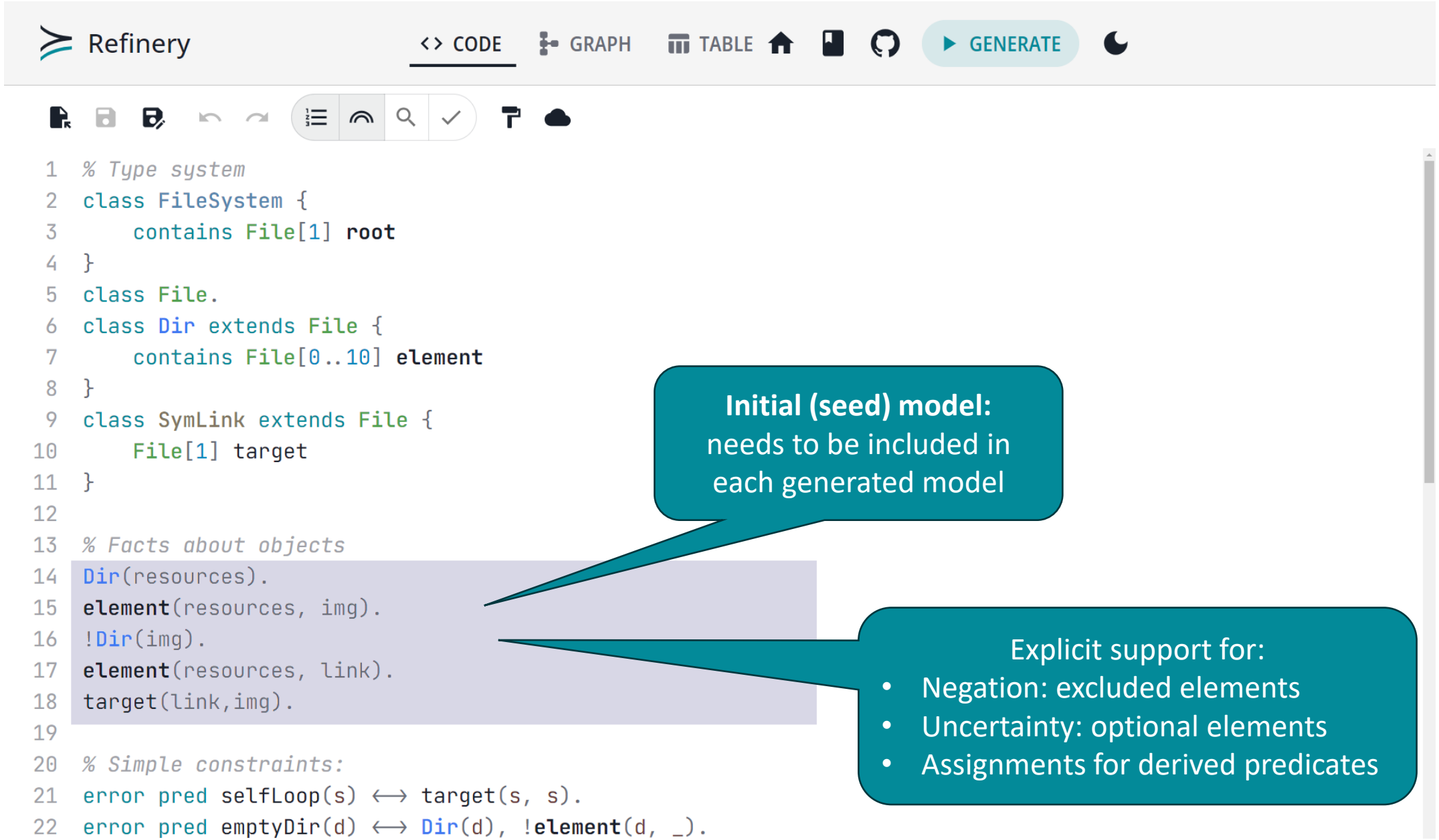
Refinery <> CODE GRAPH TABLE GENERATE

```
% Type system
class FileSystem {
    contains File[1] root
}
class File.
class Dir extends File {
    contains File[0..10] element
}
class SymLink extends File {
    File[1] target
}

% Facts about objects
Dir(resources).
element(resources, img).
!Dir(img).
element(resources, link).
target(link,img).

% Simple constraints:
error pred selfLoop(s) ⟷ target(s, s).
error pred emptyDir(d) ⟷ Dir(d), !element(d, _).
```

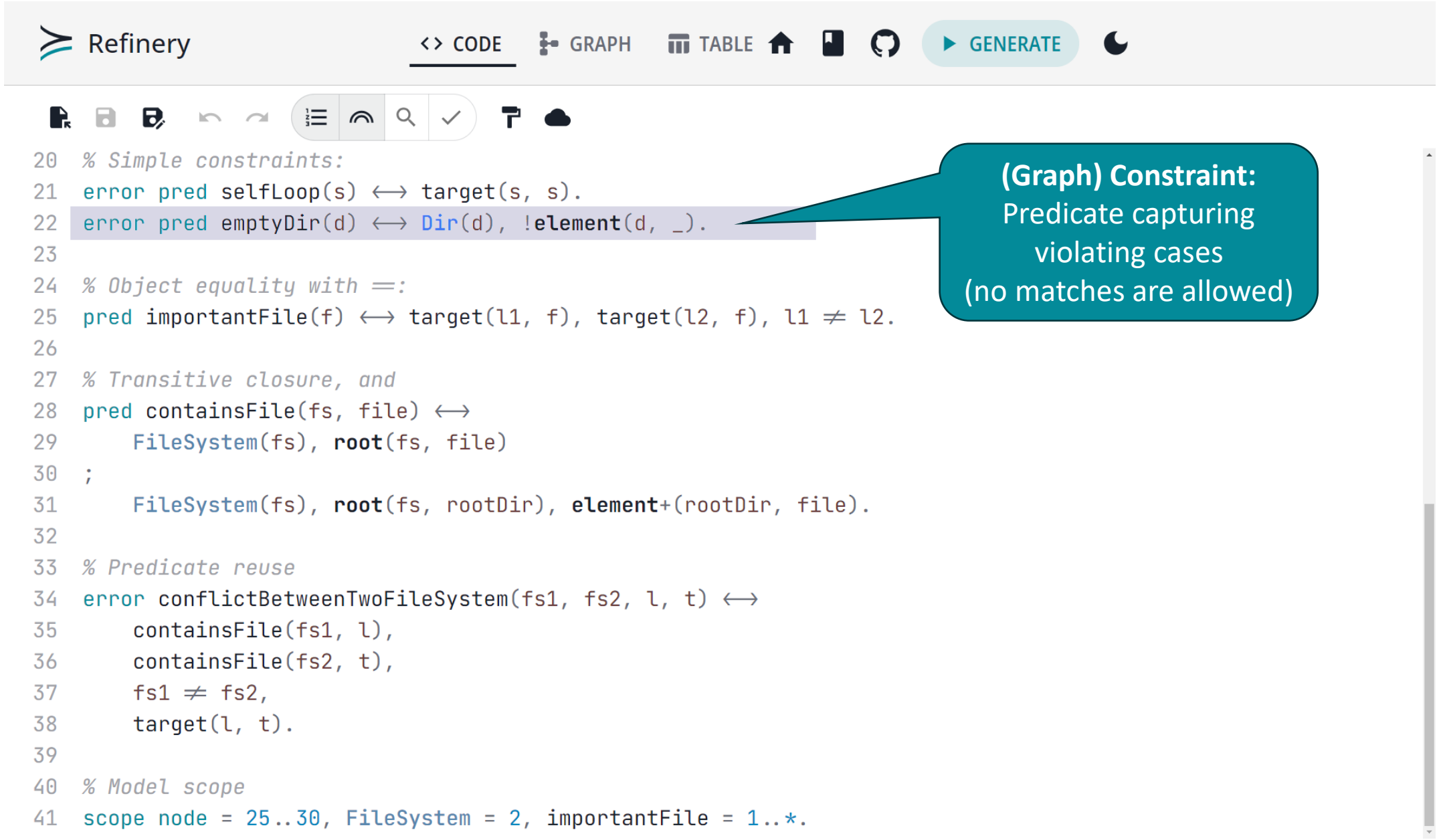
Metamodel:
Captures classes and relations
(in xcore)

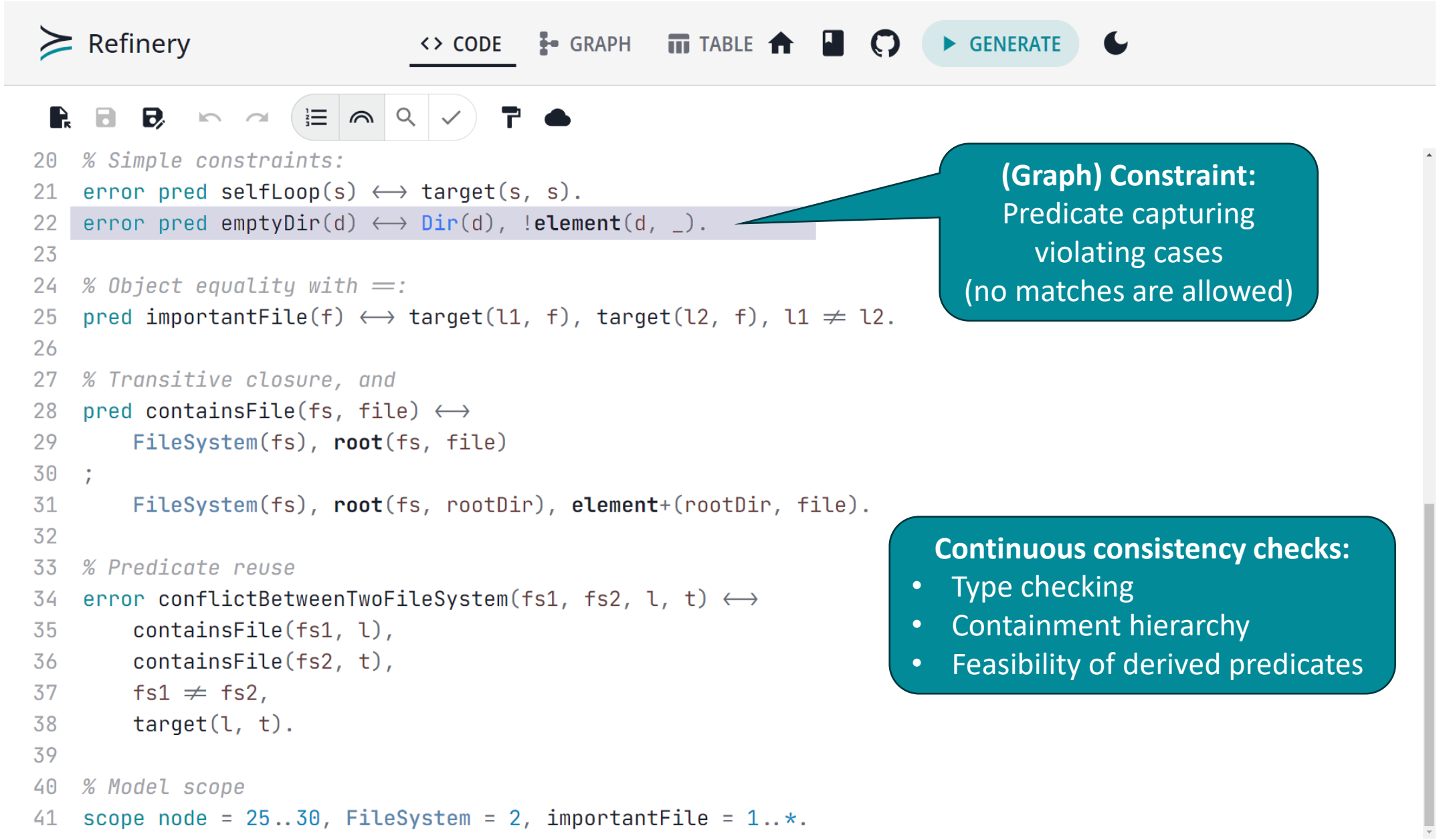
https://refinery.services/


Refinery
CODE
GRAPH
TABLE
GENERATE
1 % Type system
2 class FileSystem {
3 contains File[1] root
4 }
5 class File.
6 class Dir extends File {
7 contains File[0..10] element
8 }
9 class SymLink extends File {
10 File[1] target
11 }
12
13 % Facts about objects
14 Dir(resources).
15 element(resources, img).
16 !Dir(img).
17 element(resources, link).
18 target(link,img).
19
20 % Simple constraints:
21 error pred selfLoop(s) ⟷ target(s, s).
22 error pred emptyDir(d) ⟷ Dir(d), !element(d, _).
Initial (seed) model:
needs to be included in
each generated model


Refinery
CODE
GRAPH
TABLE
GENERATE
1 % Type system
2 class FileSystem {
3 contains File[1] root
4 }
5 class File.
6 class Dir extends File {
7 contains File[0..10] element
8 }
9 class SymLink extends File {
10 File[1] target
11 }
12
13 % Facts about objects
14 Dir(resources).
15 element(resources, img).
16 !Dir(img).
17 element(resources, link).
18 target(link,img).
19
20 % Simple constraints:
21 error pred selfLoop(s) ⟷ target(s, s).
22 error pred emptyDir(d) ⟷ Dir(d), !element(d, _).
Initial (seed) model:
needs to be included in each generated model
Explicit support for:
• Negation: excluded elements
• Uncertainty: optional elements
• Assignments for derived predicates


Refinery
CODE
GRAPH
TABLE
GENERATE
20 % Simple constraints:
21 error pred selfLoop(s) ⟷ target(s, s).
22 error pred emptyDir(d) ⟷ Dir(d), !element(d, _).
23
24 % Object equality with ==:
25 pred importantFile(f) ⟷ target(l1, f), target(l2, f), l1 ≠ l2.
26
27 % Transitive closure, and
28 pred containsFile(fs, file) ⟷
29 FileSystem(fs), root(fs, file)
30 ;
31 FileSystem(fs), root(fs, rootDir), element+(rootDir, file).
32
33 % Predicate reuse
34 error conflictBetweenTwoFileSystem(fs1, fs2, l, t) ⟷
35 containsFile(fs1, l),
36 containsFile(fs2, t),
37 fs1 ≠ fs2,
38 target(l, t).
39
40 % Model scope
41 scope node = 25..30, FileSystem = 2, importantFile = 1..*.
(Graph) Predicate:
Domain specific language
equivalent to
First Order Relational Logic
With transitive closure
Without full recursion


Refinery
CODE
GRAPH
TABLE
GENERATE
20 % Simple constraints:
21 error pred selfLoop(s) ⟷ target(s, s).
22 error pred emptyDir(d) ⟷ Dir(d), !element(d, _).
23
24 % Object equality with ==:
25 pred importantFile(f) ⟷ target(l1, f), target(l2, f), l1 ≠ l2.
26
27 % Transitive closure, and
28 pred containsFile(fs, file) ⟷
29 FileSystem(fs), root(fs, file)
30 ;
31 FileSystem(fs), root(fs, rootDir), element+(rootDir, file).
32
33 % Predicate reuse
34 error conflictBetweenTwoFileSystem(fs1, fs2, l, t) ⟷
35 containsFile(fs1, l),
36 containsFile(fs2, t),
37 fs1 ≠ fs2,
38 target(l, t).
39
40 % Model scope
41 scope node = 25..30, FileSystem = 2, importantFile = 1..*.
(Graph) Constraint:
Predicate capturing
violating cases
(no matches are allowed)

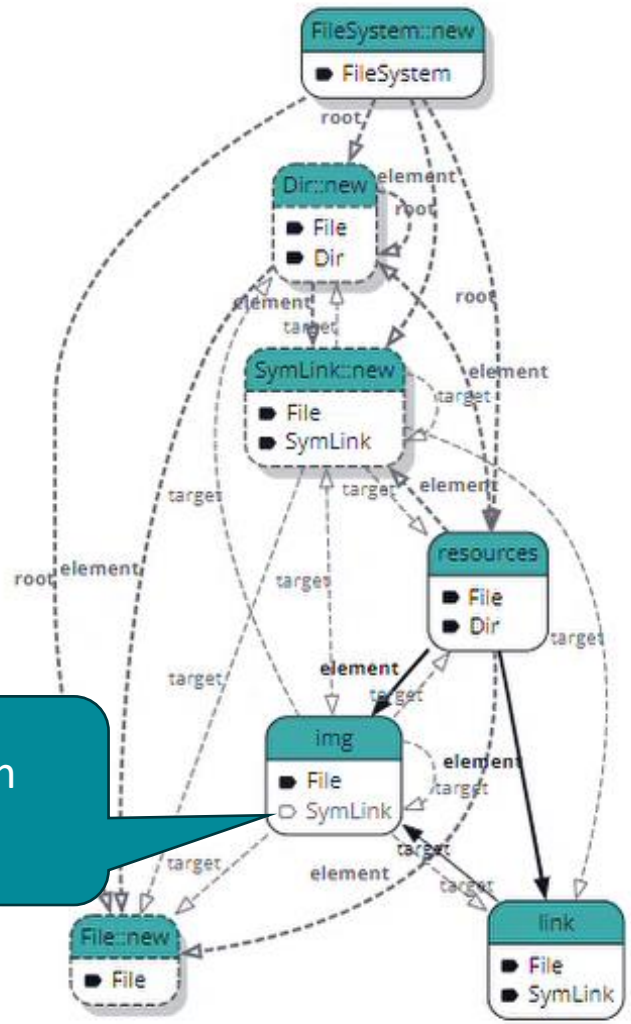

Refinery
CODE
GRAPH
TABLE
GENERATE
20 % Simple constraints:
21 error pred selfLoop(s) ⟷ target(s, s).
22 error pred emptyDir(d) ⟷ Dir(d), !element(d, _).
23
24 % Object equality with ==:
25 pred importantFile(f) ⟷ target(l1, f), target(l2, f), l1 ≠ l2.
26
27 % Transitive closure, and
28 pred containsFile(fs, file) ⟷
29 FileSystem(fs), root(fs, file)
30 ;
31 FileSystem(fs), root(fs, rootDir), element+(rootDir, file).
32
33 % Predicate reuse
34 error conflictBetweenTwoFileSystem(fs1, fs2, l, t) ⟷
35 containsFile(fs1, l),
36 containsFile(fs2, t),
37 fs1 ≠ fs2,
38 target(l, t).
39
40 % Model scope
41 scope node = 25..30, FileSystem = 2, importantFile = 1..*.
(Graph) Constraint:
Predicate capturing
violating cases
(no matches are allowed)
Continuous consistency checks:
• Type checking
• Containment hierarchy
• Feasibility of derived predicates

Refinery

<> CODE | GRAPH | TABLE | ▶ GENERATE

```
1  // Type system
2  class FileSystem {
3      contains File[1] root
4  }
5  class File.
6  class Dir extends File {
7      contains File[0..10] element
8  }
9  class SymLink extends File {
10     File[1] target
11 }
12
13 // Facts about objects
14 Dir(resources).
15 element(resources,img).
16 !Dir(img).
17 element(resources,link).
18 target(link,img).
19
20 // Predicates and constraints
```

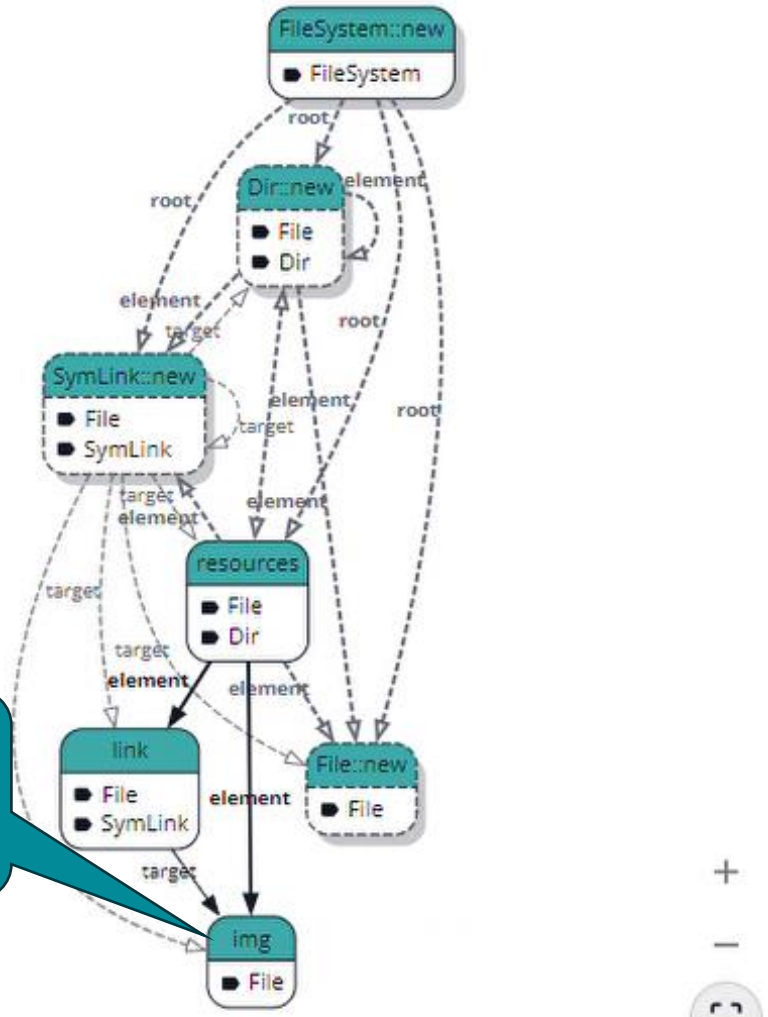
Updating the specification refines the model



Refinery

<> CODE | GRAPH | TABLE | ▶ GENERATE

```
1  // Type system
2  class FileSystem {
3      contains File[1] root
4  }
5  class File.
6  class Dir extends File {
7      contains File[0..10] element
8  }
9  class SymLink extends File {
10     File[1] target
11 }
12
13 // Facts about objects
14 Dir(resources).
15 element(resources,img).
16 !Dir(img).
17 element(resources,link).
18 target(link,img).
19 !SymLink(img).
20
```



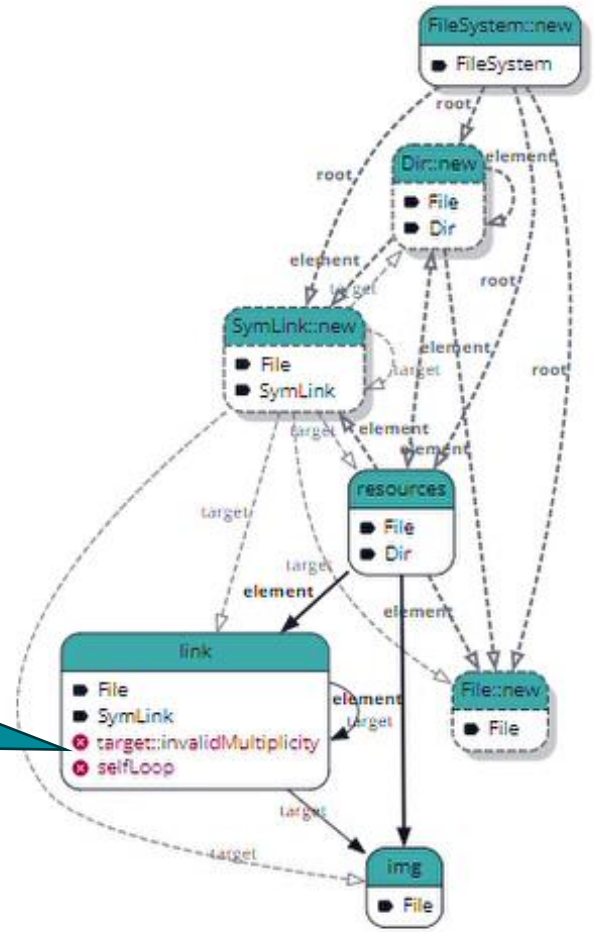
Updating the specification refines the model

Refinery

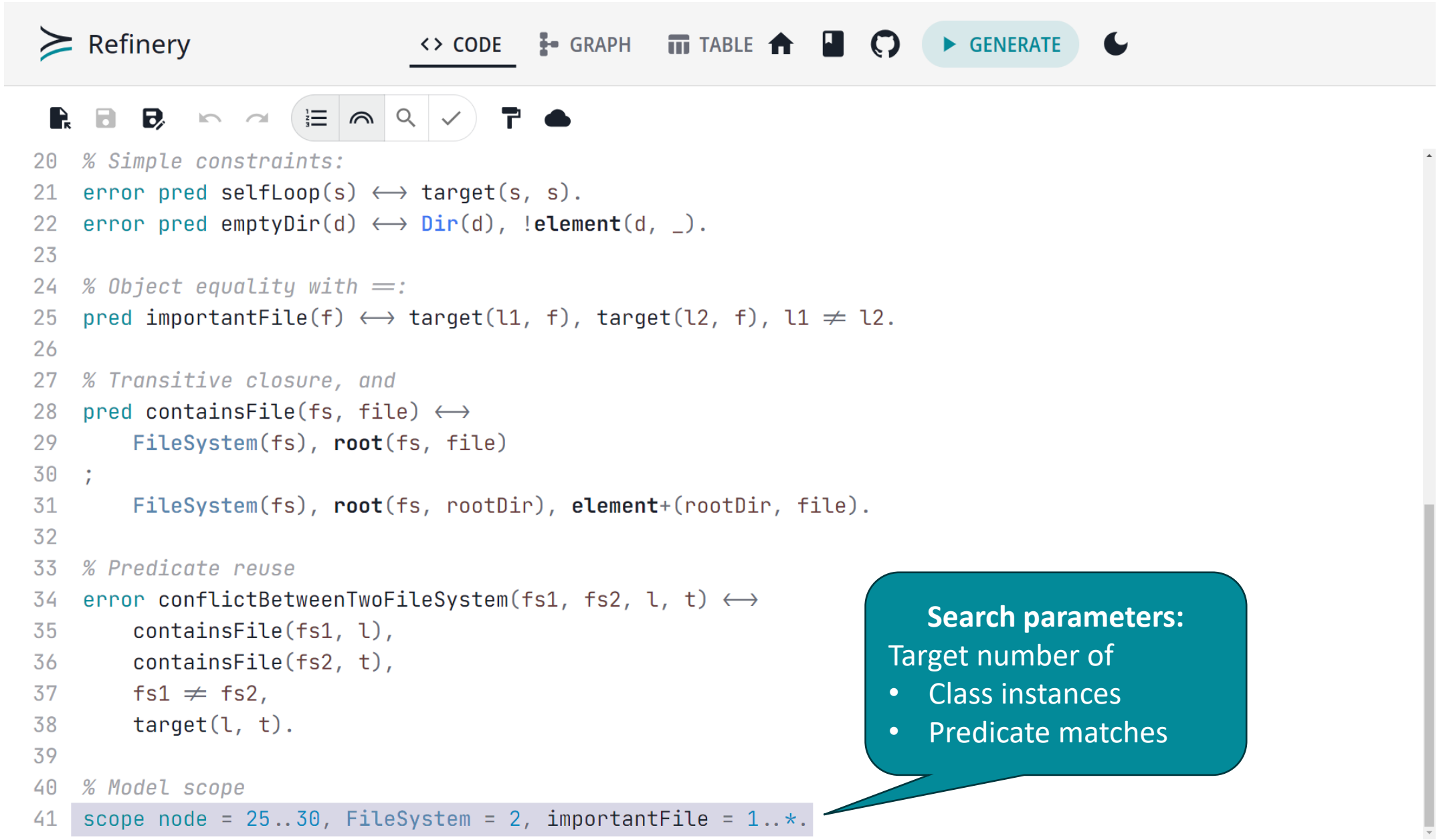
<> CODE | GRAPH | TABLE | ▶ GENERATE

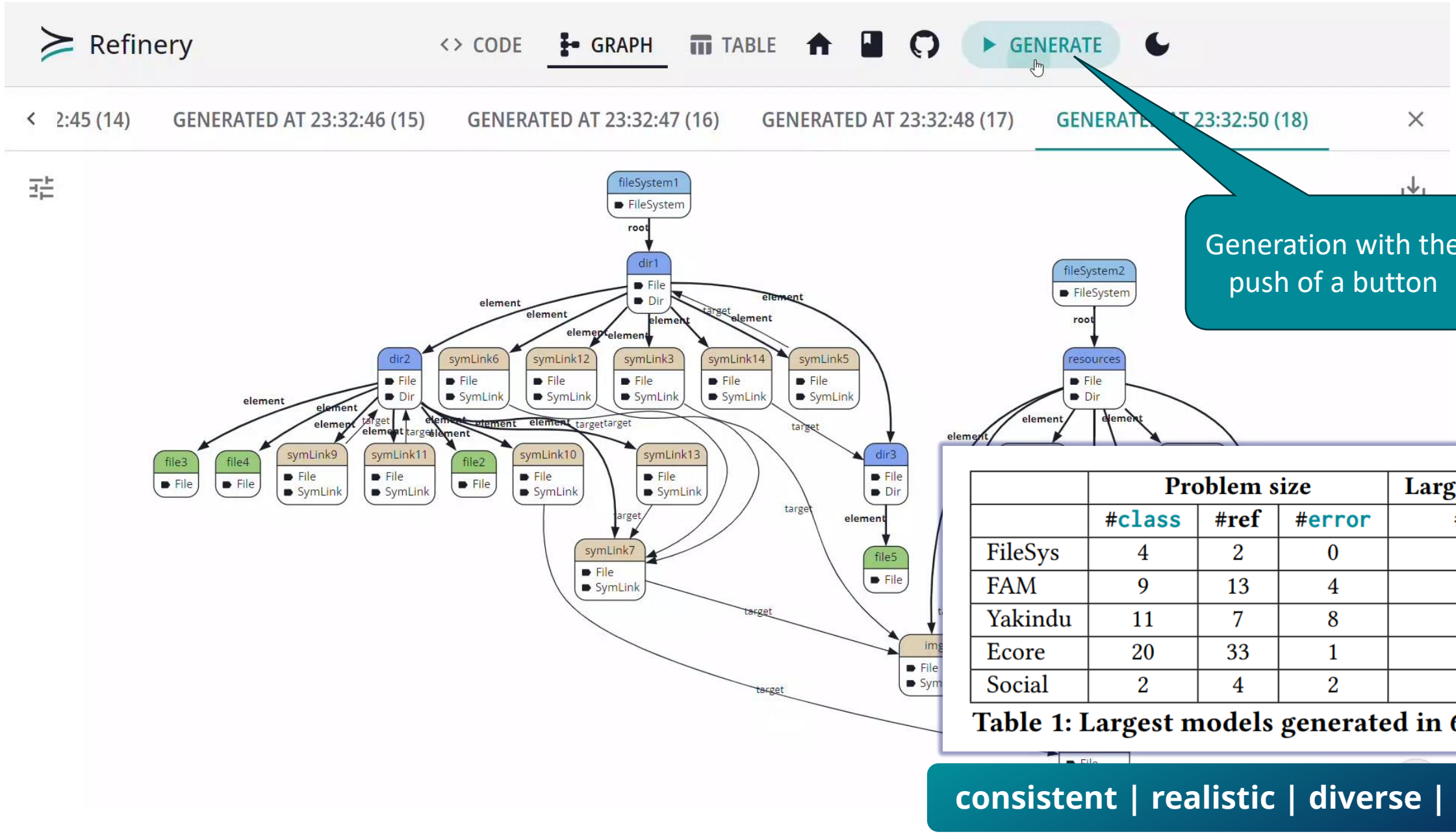
```
1  // Type system
2  class FileSystem {
3      contains File[1] root
4  }
5  class File.
6  class Dir extends File {
7      contains File[0..10] element
8  }
9  class SymLink extends File {
10     File[1] target
11 }
12
13 // Facts about objects
14 Dir(resources).
15 element(resources,img).
16 !Dir(img).
17 element(resources,link).
18 target(link,img).
19 !SymLink(img).
20 target(link,link).
```

INITIAL MODEL | GENERATED AT 18:37:56 (1)



Error: highlighting inconsistency


Refinery
CODE
GRAPH
TABLE
GENERATE
20 % Simple constraints:
21 error pred selfLoop(s) ⟷ target(s, s).
22 error pred emptyDir(d) ⟷ Dir(d), !element(d, _).
23
24 % Object equality with ==:
25 pred importantFile(f) ⟷ target(l1, f), target(l2, f), l1 ≠ l2.
26
27 % Transitive closure, and
28 pred containsFile(fs, file) ⟷
29 FileSystem(fs), root(fs, file)
30 ;
31 FileSystem(fs), root(fs, rootDir), element+(rootDir, file).
32
33 % Predicate reuse
34 error conflictBetweenTwoFileSystem(fs1, fs2, l, t) ⟷
35 containsFile(fs1, l),
36 containsFile(fs2, t),
37 fs1 ≠ fs2,
38 target(l, t).
39
40 % Model scope
41 scope node = 25..30, FileSystem = 2, importantFile = 1..*.
Search parameters:
Target number of
• Class instances
• Predicate matches



Generation with the push of a button

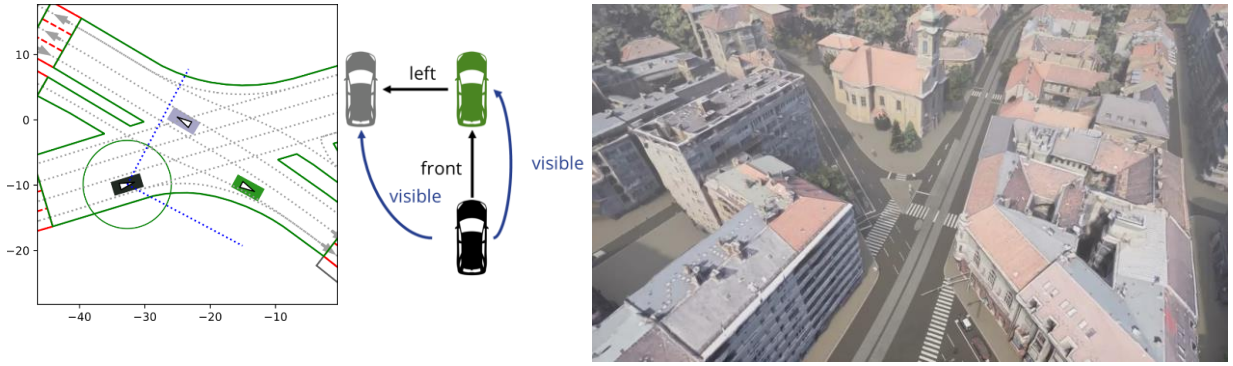
	Problem size			Largest model
	#class	#ref	#error	#node
FileSys	4	2	0	19750
FAM	9	13	4	15750
Yakindu	11	7	8	4250
Ecore	20	33	1	2000
Social	2	4	2	230

Table 1: Largest models generated in 60 seconds

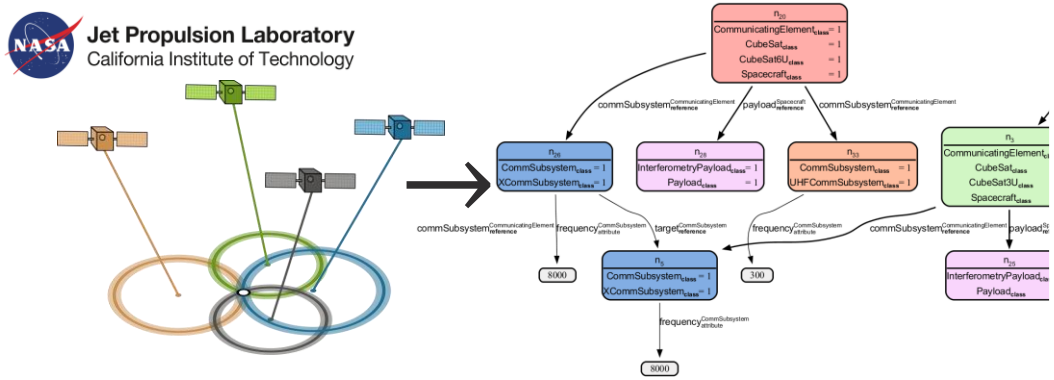
consistent | realistic | diverse | scalable

Applications

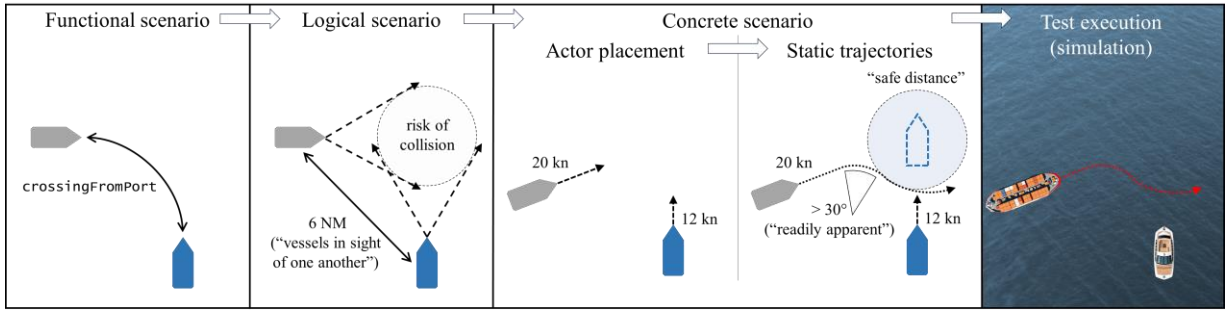
Testing vision-based AI components
by generating diverse set of traffic situations



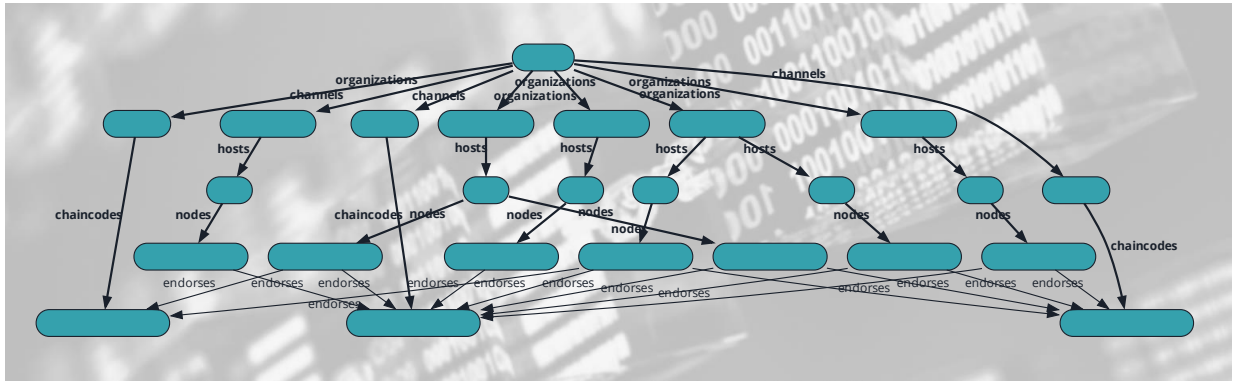
Cost optimization of satellite network
by combining numerical and structural reasoning



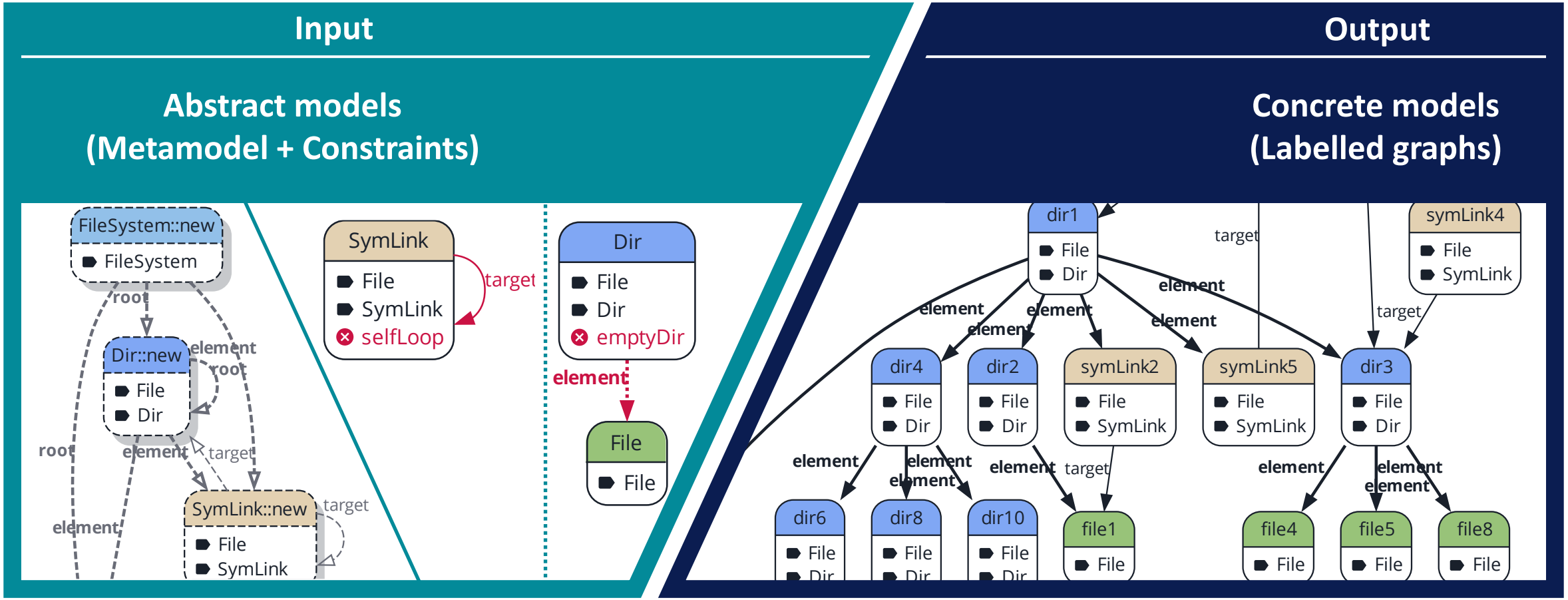
Automated test scenario synthesis for verifying collision avoidance of autonomous vessels



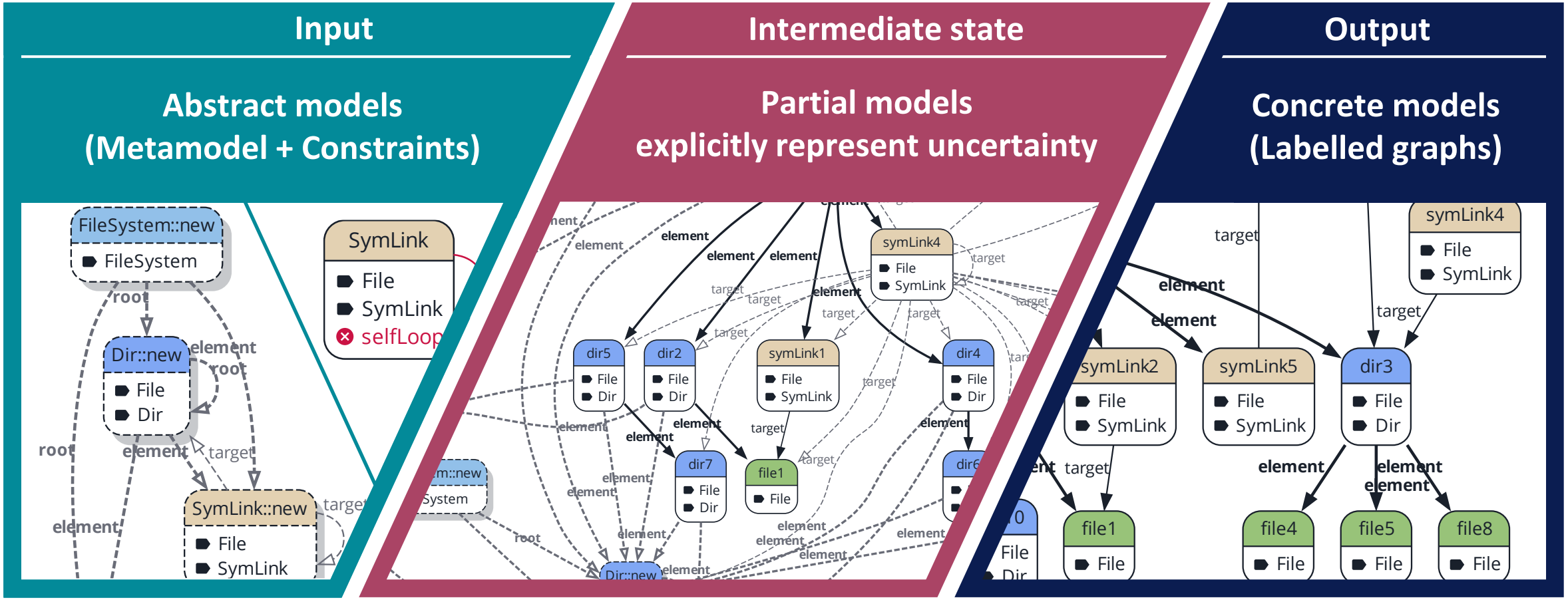
Requirement-Driven Generation of Distributed Ledger Architectures



Models and Partial Models

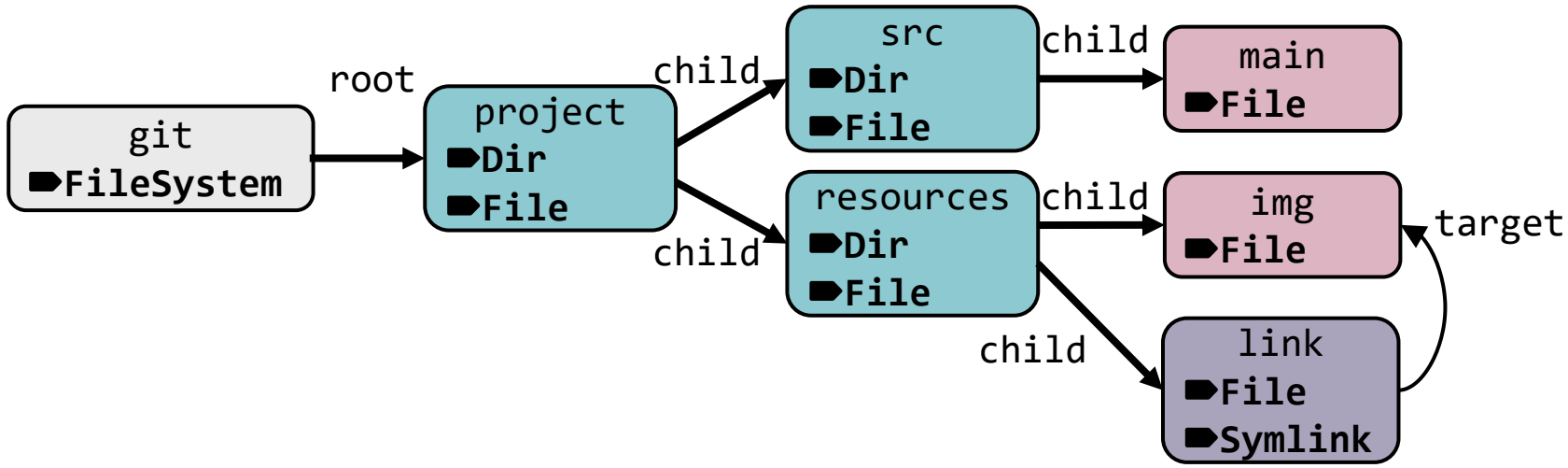


Models and Partial Models



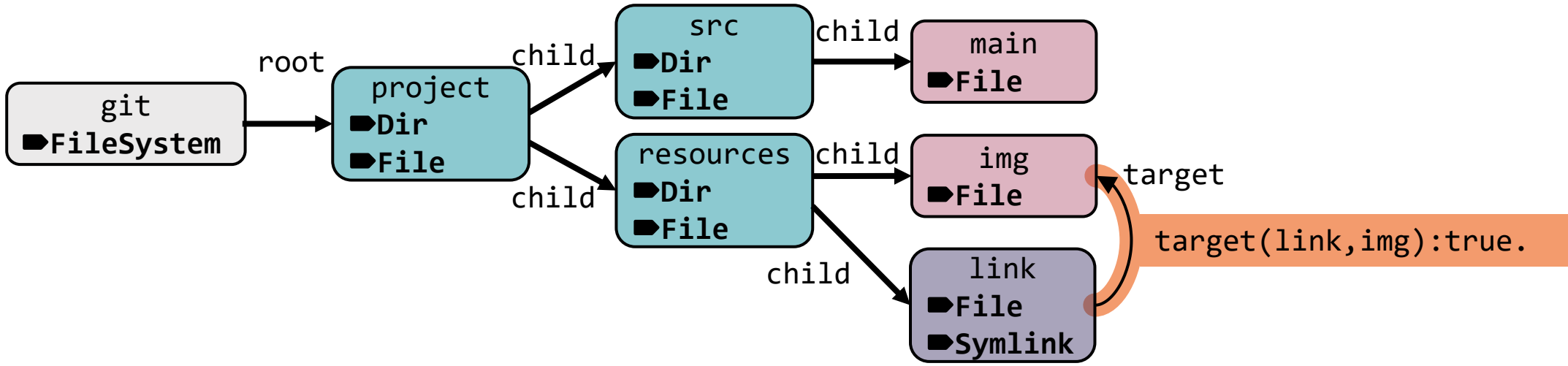
Model generation: exploration process that gradually reduces uncertainty

Partial modeling with 4-valued logic



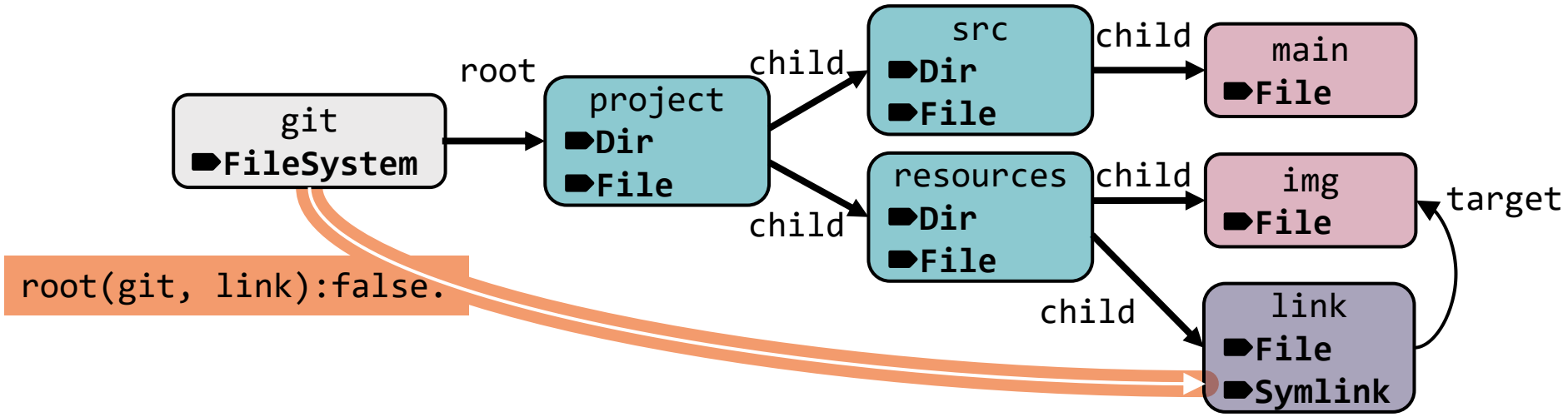
- Represent all potential extension with uncertainty
- Logic abstraction: **TRUE** | **False** | **Unknown** | **Error**

Partial modeling with 4-valued logic



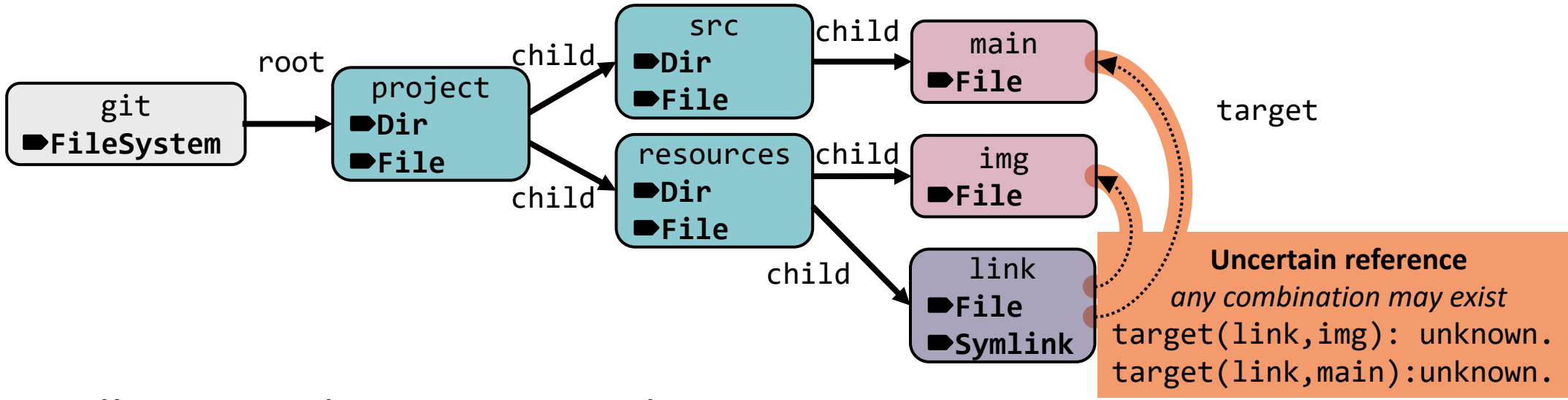
- Represent all potential extension with uncertainty
- Logic abstraction: **TRUE** | **False** | **Unknown** | **Error**

Partial modeling with 4-valued logic



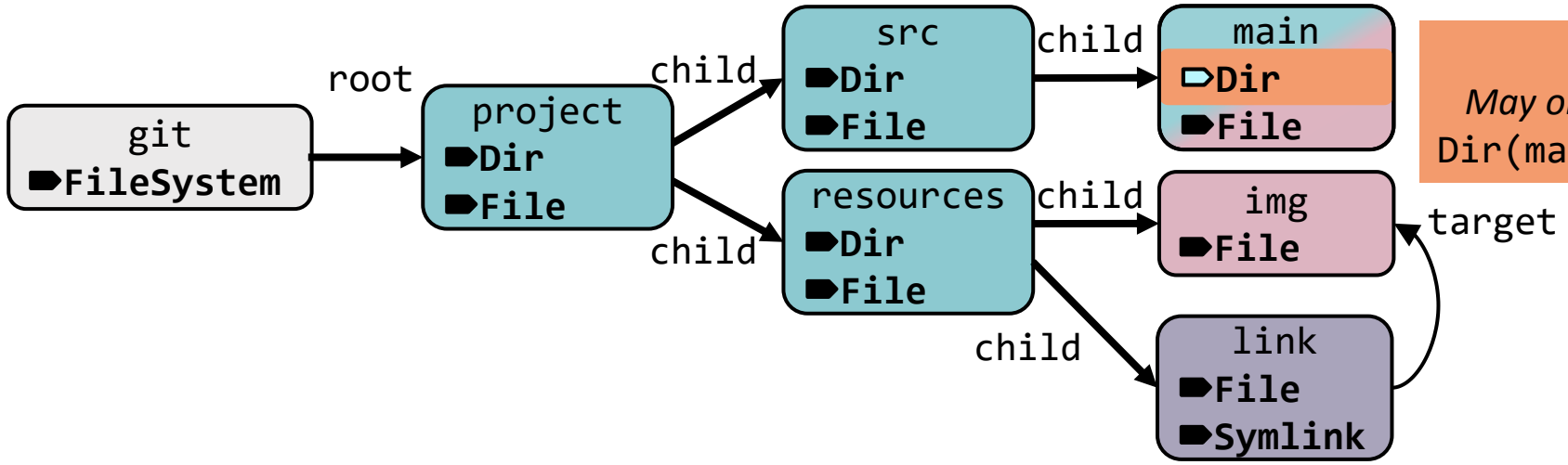
- Represent all potential extension with uncertainty
- Logic abstraction: **TRUE** | **False** | **Unknown** | **Error**

Partial modeling with 4-valued logic



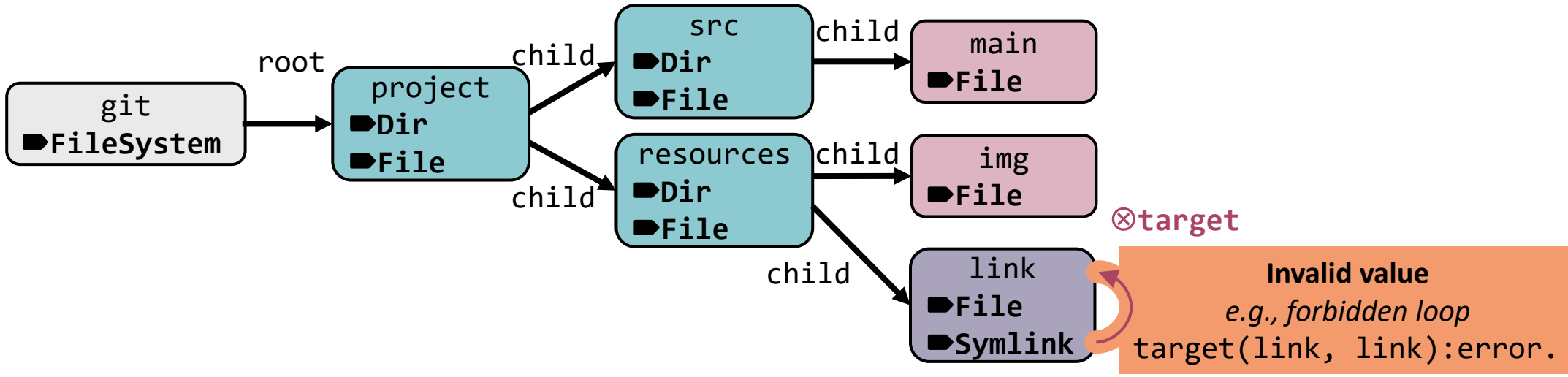
- Represent all potential extension with uncertainty
- Logic abstraction: **TRUE** | **False** | **Unknown** | **Error**

Partial modeling with 4-valued logic



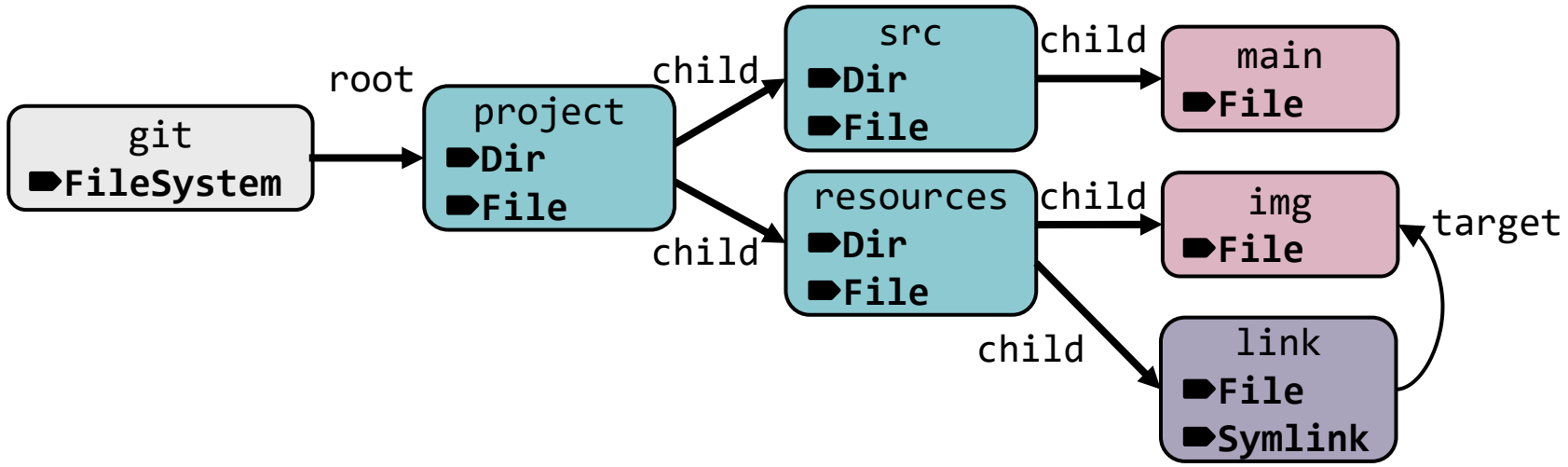
- Represent all potential extension with uncertainty
- Logic abstraction: **TRUE** | **False** | **Unknown** | **Error**

Partial modeling with 4-valued logic



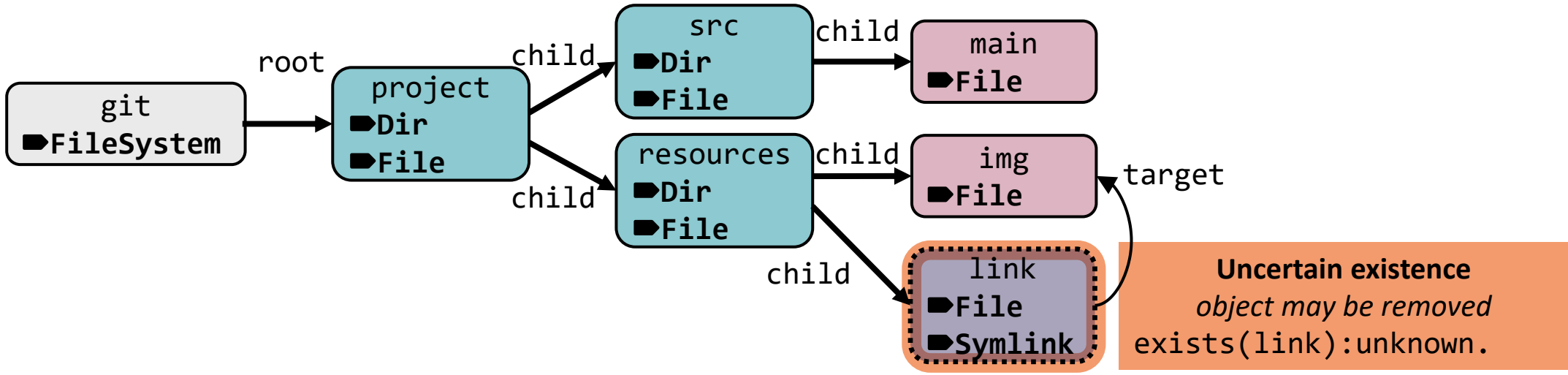
- Represent all potential extension with uncertainty
- Logic abstraction: **TRUE** | **False** | **Unknown** | **⊗Error**

Partial modeling with 4-valued logic



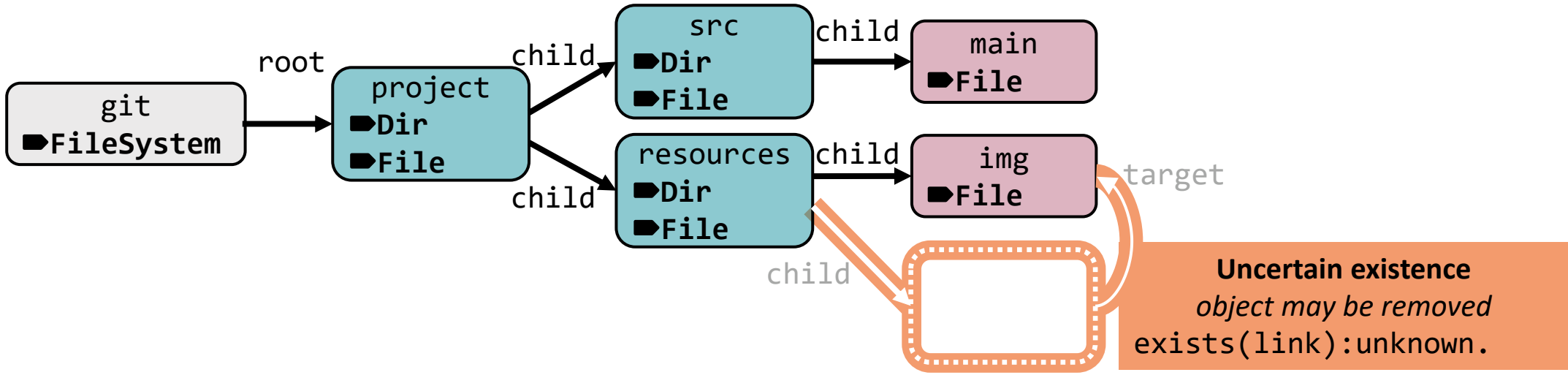
- Represent all potential extension with uncertainty
- Logic abstraction: TRUE | False | Unknown | Error
  - 4-valued **exists**: added or removed
  - 4-valued **equals**: merging or splitting

Partial modeling with 4-valued logic



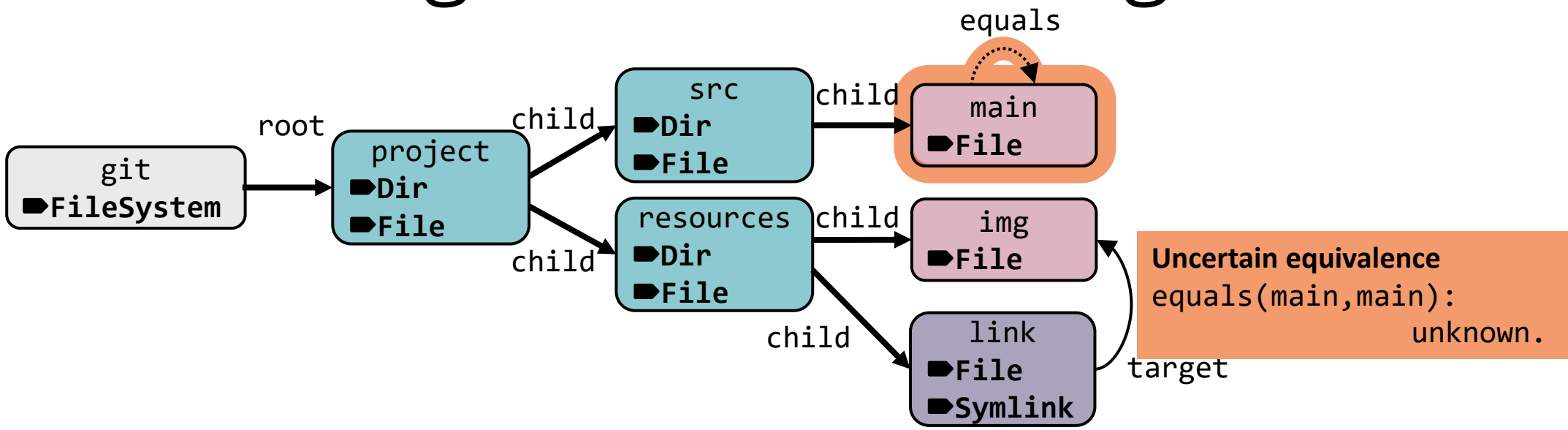
- Represent all potential extension with uncertainty
- Logic abstraction: TRUE | False | Unknown | Error
  - 4-valued **exists**: added or removed
  - 4-valued **equals**: merging or splitting

Partial modeling with 4-valued logic



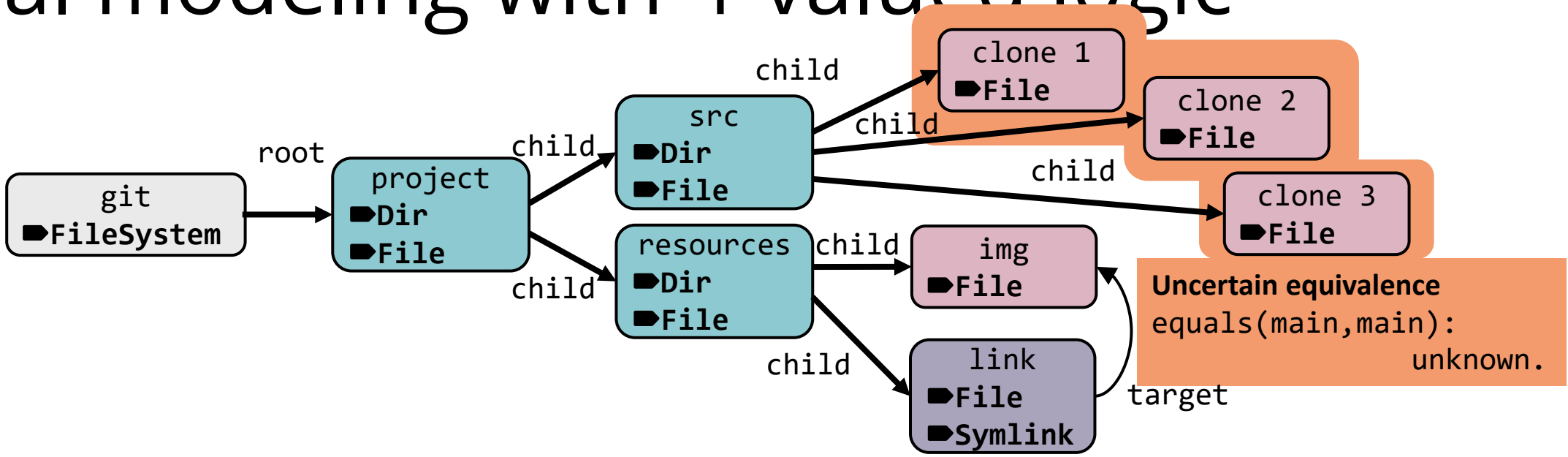
- Represent all potential extension with uncertainty
- Logic abstraction: TRUE | False | Unknown | Error
  - 4-valued **exists**: added or removed
  - 4-valued **equals**: merging or splitting

Partial modeling with 4-valued logic



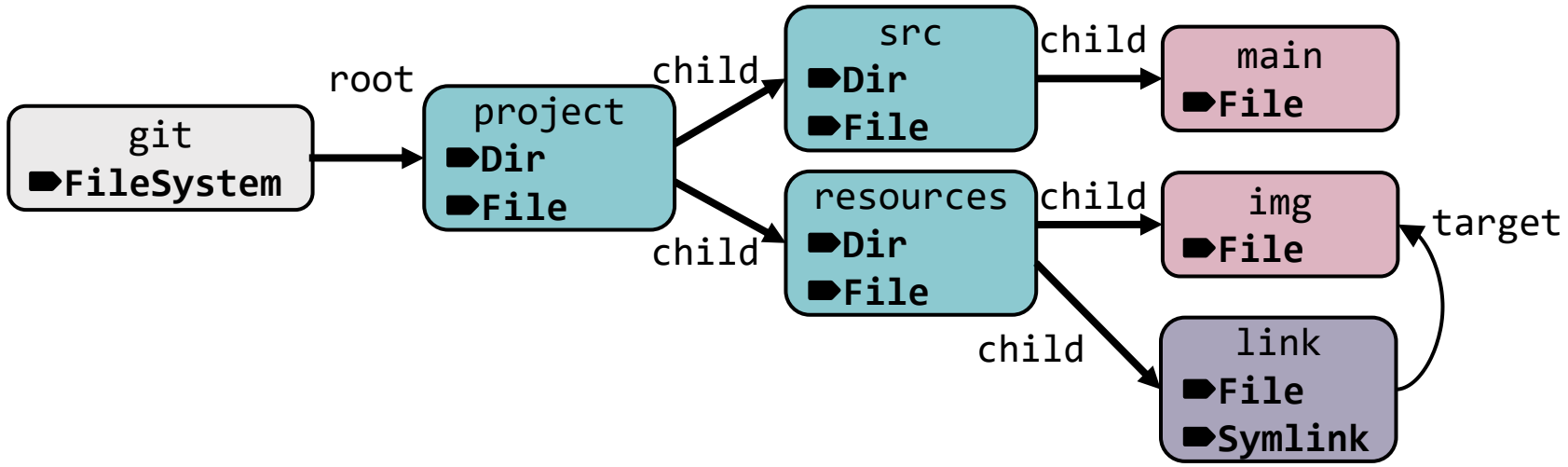
- Represent all potential extension with uncertainty
- Logic abstraction: TRUE | False | Unknown | Error
  - 4-valued **exists**: added or removed
  - 4-valued **equals**: merging or splitting

Partial modeling with 4-valued logic



- Represent all potential extension with uncertainty
- Logic abstraction: **TRUE** | **False** | **Unknown** | **Error**
  - 4-valued **`exists`**: added or removed
  - 4-valued **`equals`**: merging or splitting

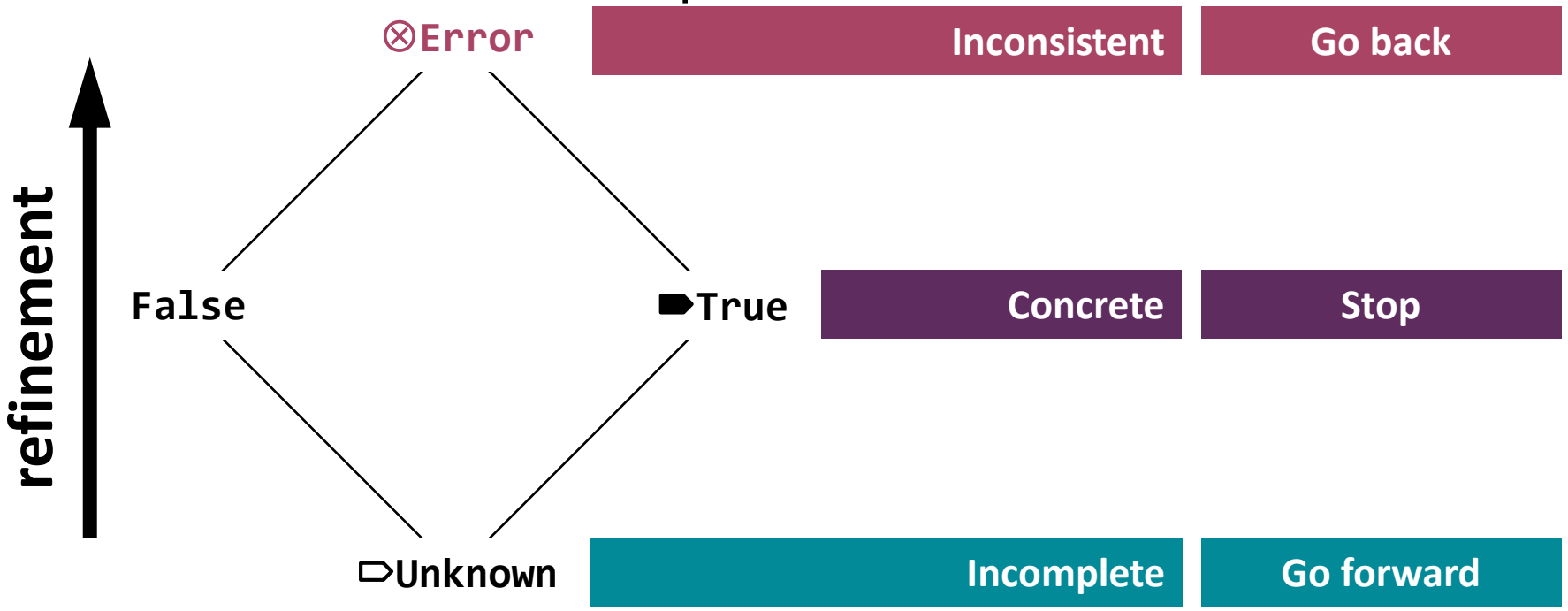
Partial modeling with 4-valued logic



- Represent all potential extension with uncertainty
- Logic abstraction: `TRUE` | `False` | `Unknown` | `Error`
  - 4-valued `exists`: added or removed
  - 4-valued `equals`: merging or splitting
- **Refinement**: reduces uncertainty → concrete models

Refinement: 4-valued logic

- Model generation is executed with respect to model refinement



E.g.: `target(_,_):unknown` —+true→ `target(_,_):true`

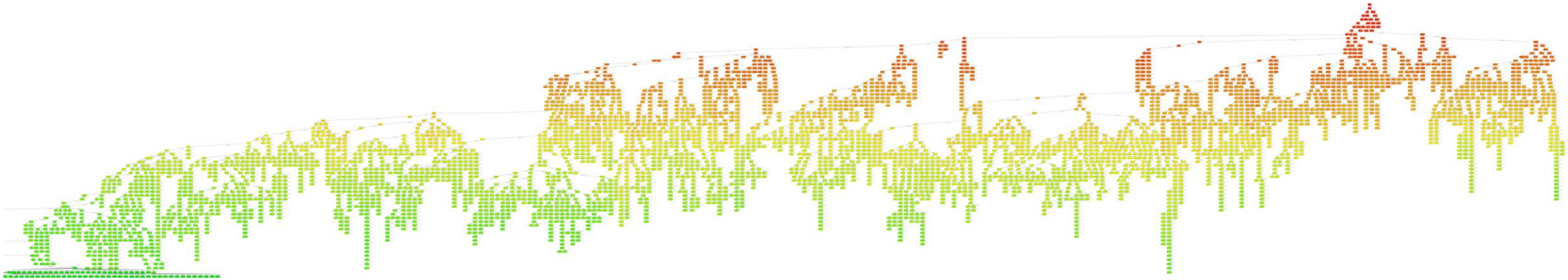
`target(_,_):true` —+false→ `target(_,_):error`

Model transformation

- Generation → Refinement rules
  - Find uncertain value
  - Refine with True or False
- Model transformation rules
  - Subgraph to match (precondition)
  - Modify the model (postcondition)
- Same logic with more complex rules

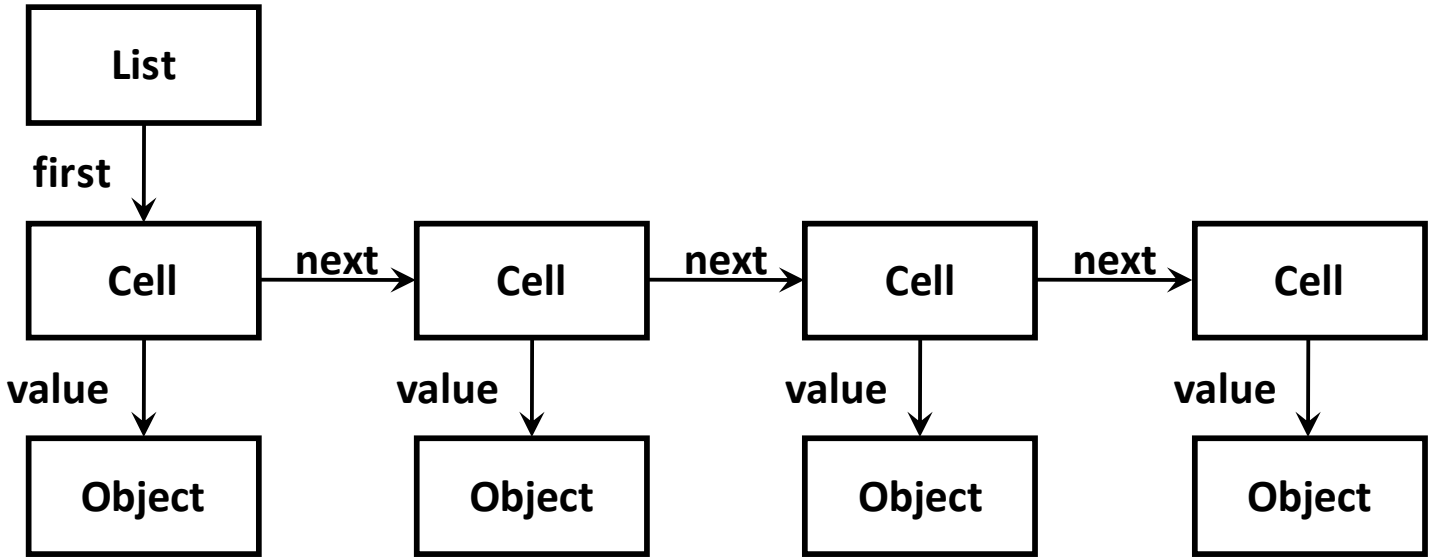
State space exploration

- Default algorithm: DFS with random jump backs
- Custom algorithms easily implemented
- Use of objective function
- Guaranteed completeness



Graph Transformation

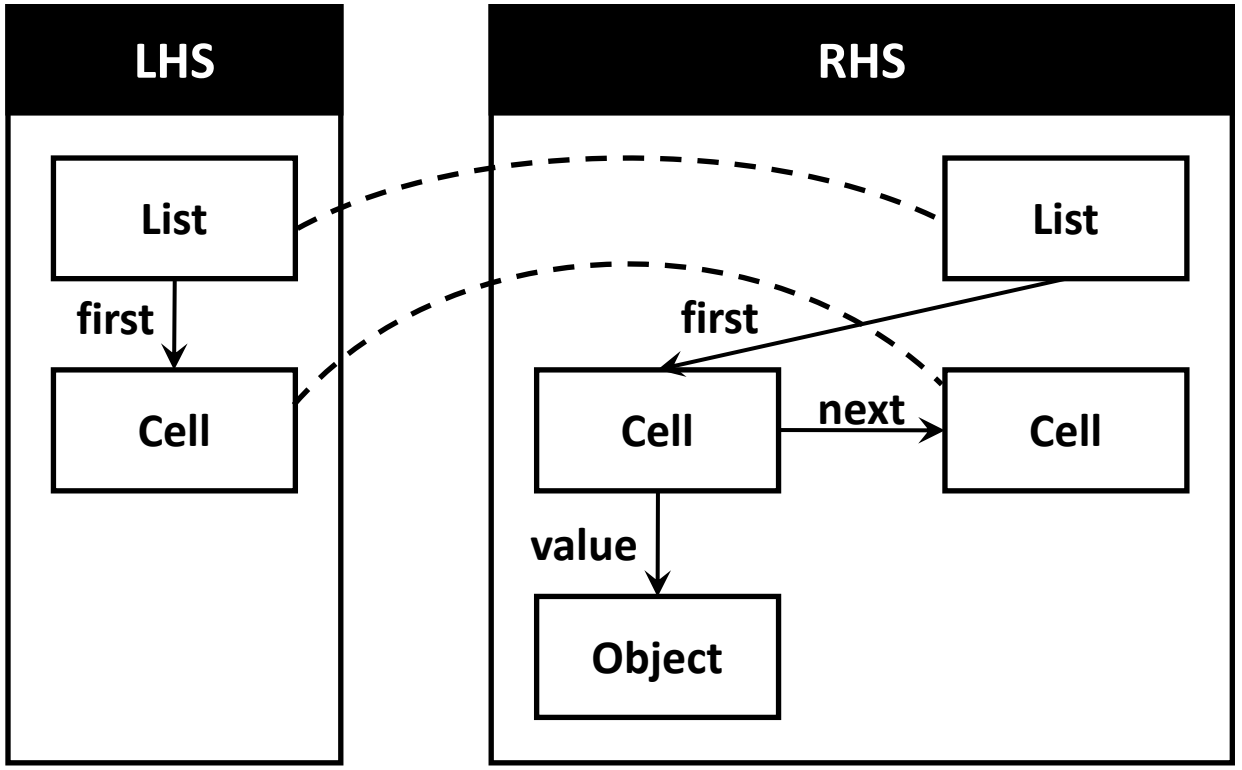
- Model = Labelled Graph



Graph Transformation rule

- Graph rewriting rule, defined with two graphs

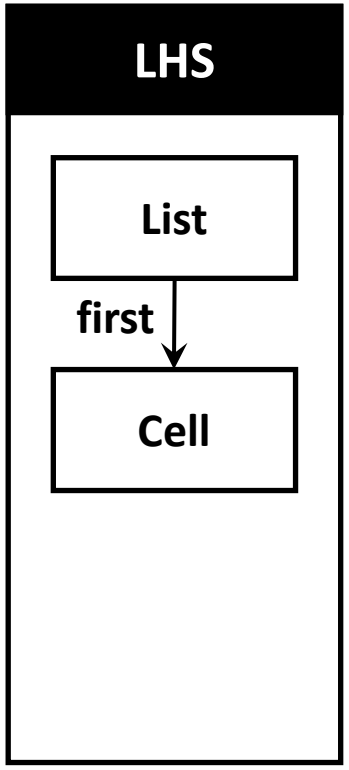
Left Hand Side Right Hand Side



Graph Transformation rule

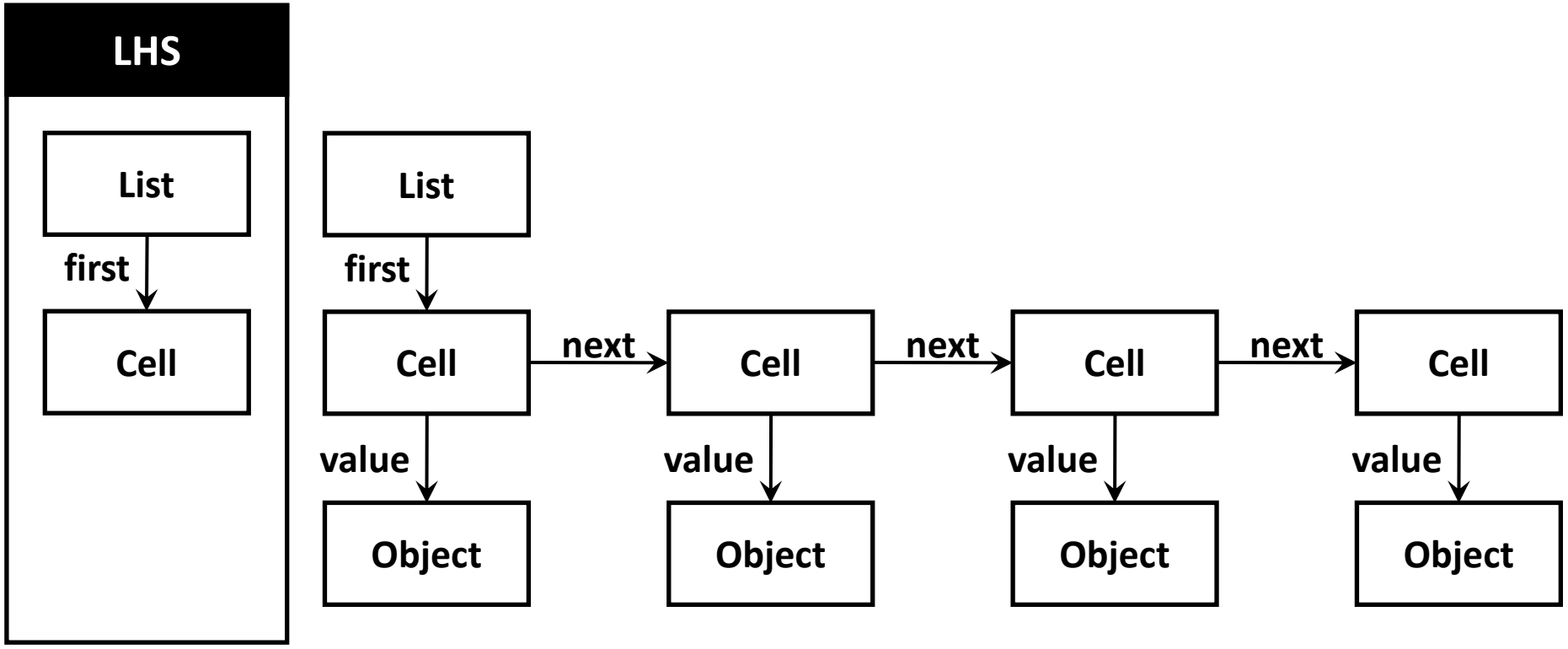
- Graph rewriting rule, defined with two graphs

Left Hand Side Right Hand Side



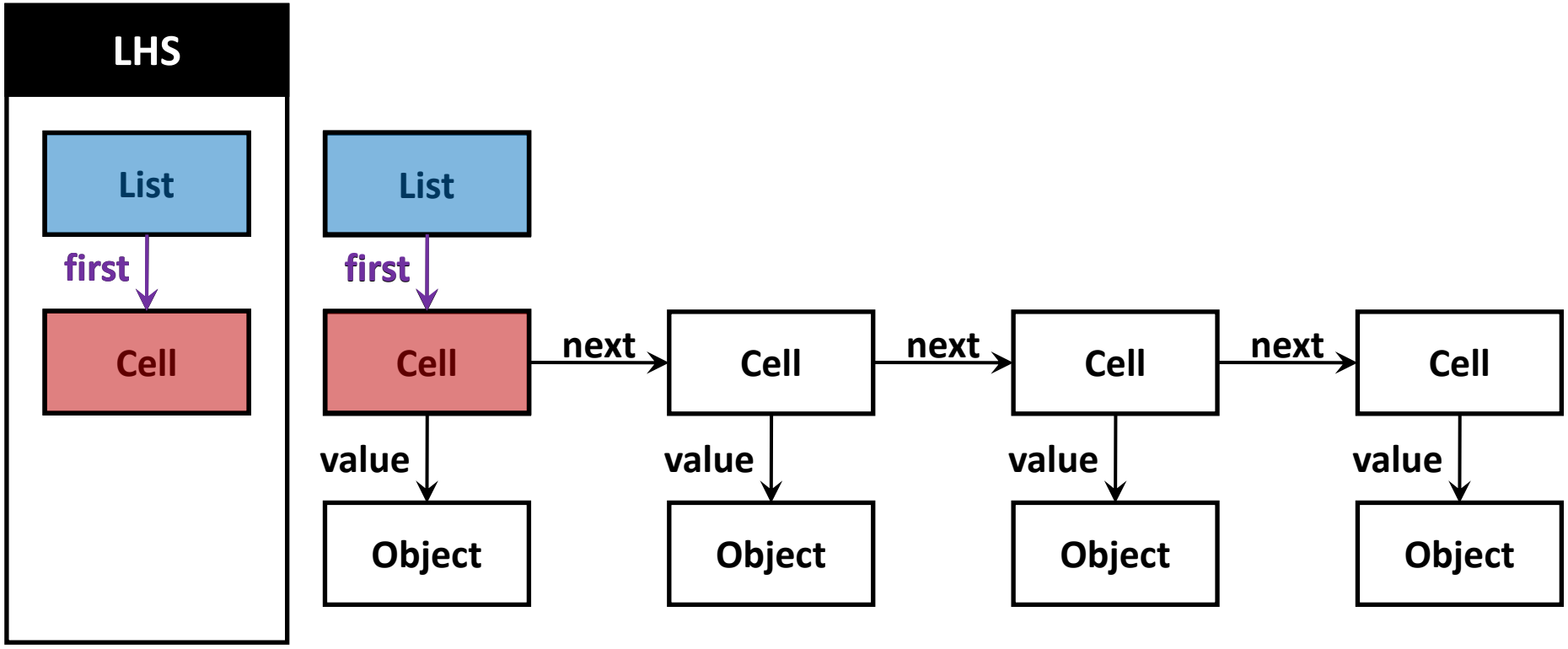
Graph Transformation: Pattern matching

- **Matching:** find the subgraphs containing LHS in the source graph



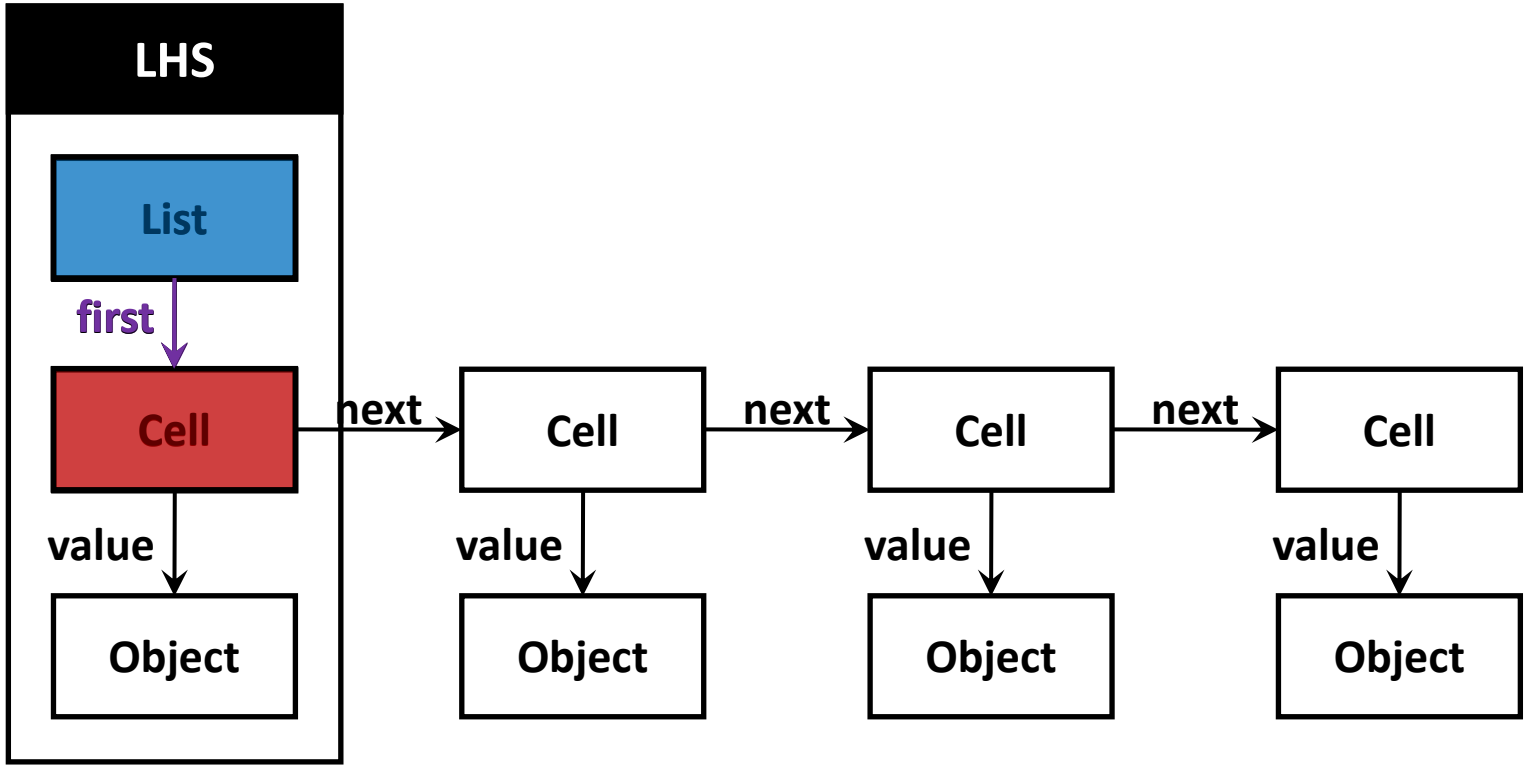
Graph Transformation: Pattern matching

- **Matching:** find the subgraphs containing LHS in the source graph



Graph Transformation: Pattern matching

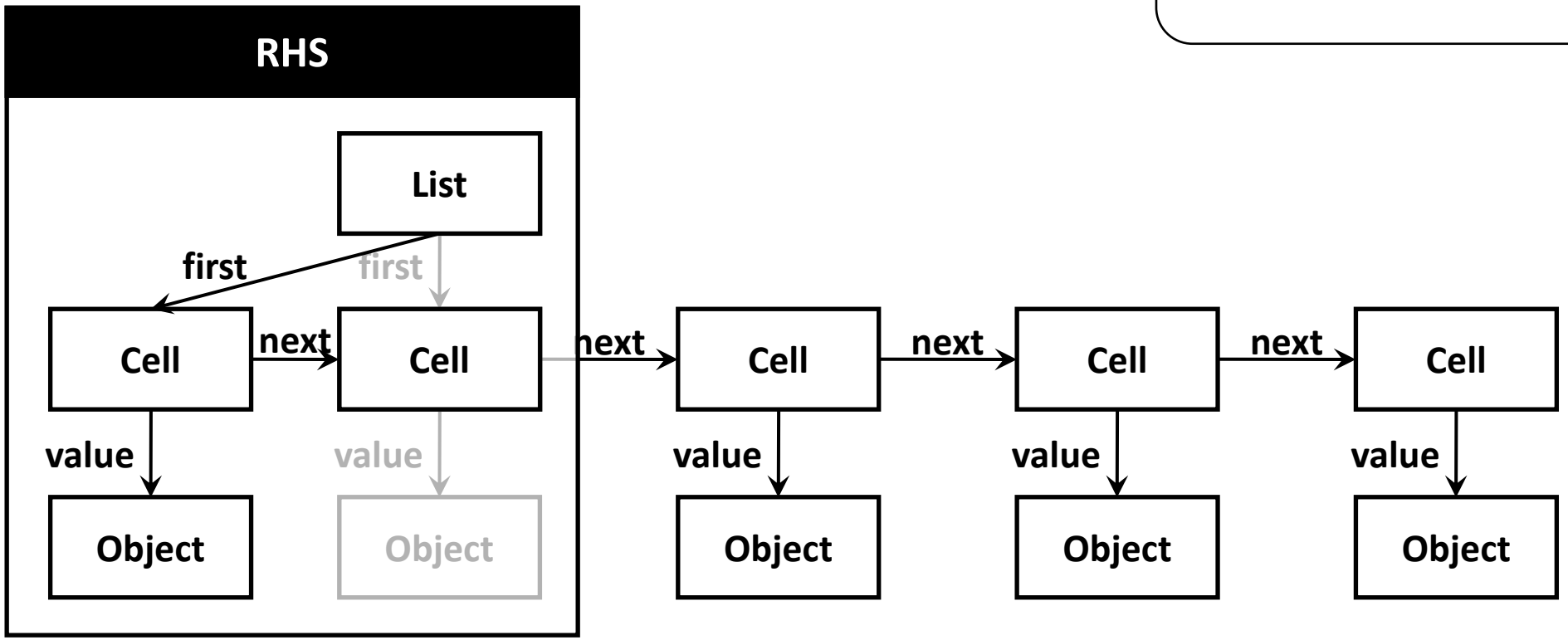
- **Matching:** find the subgraphs containing LHS in the source graph



Graph Transformation: Execution of rewriting

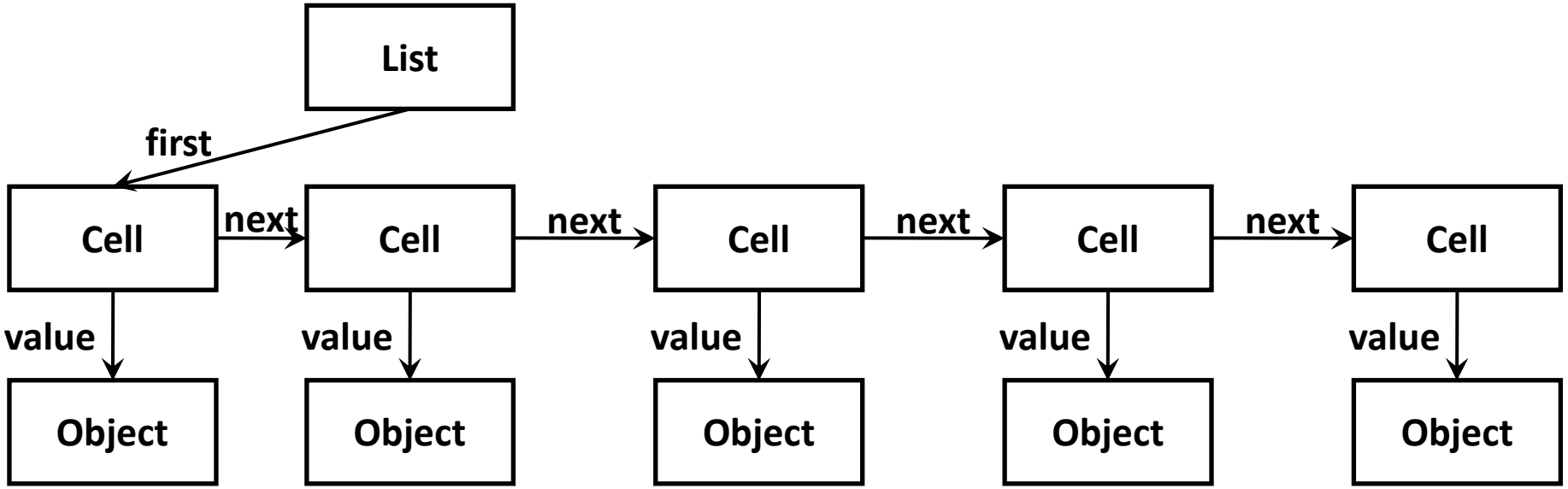
- Rewriting the graph by the match: replace LHS with RHS.

LHS\RHS → Delete
RHS\LHS → Insert
RHS∩LHS → Leave it



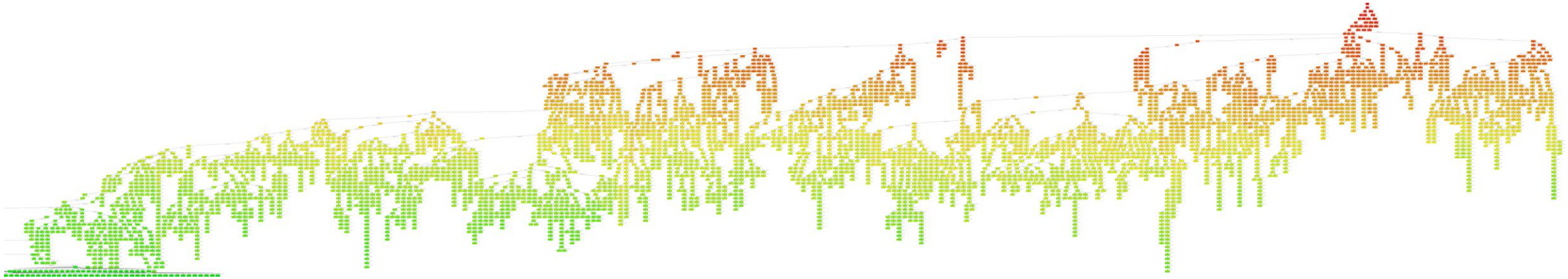
Graph Transformation: Execution of rewriting

- We get a new graph

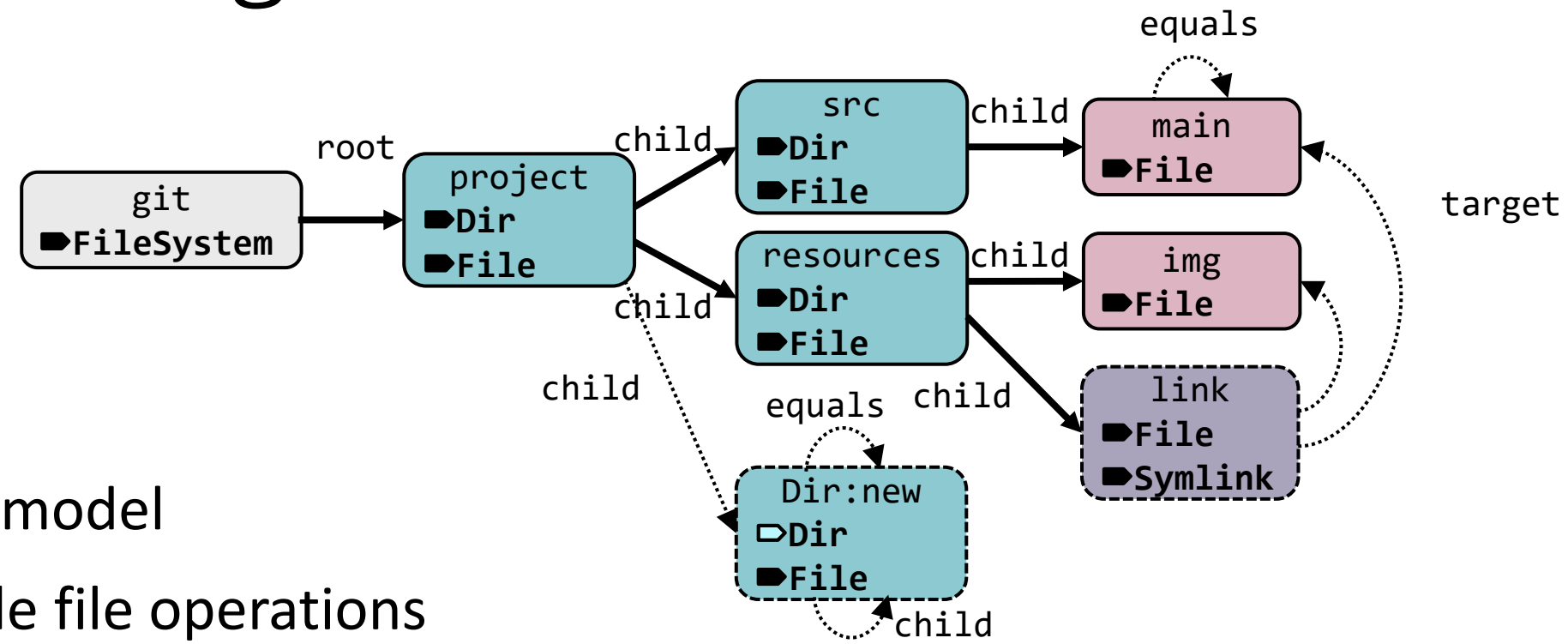


Reasoning over abstract models

- Infinite state space
- Complete state space exploration is impossible
- Abstraction must be used
- Logical reasoning required

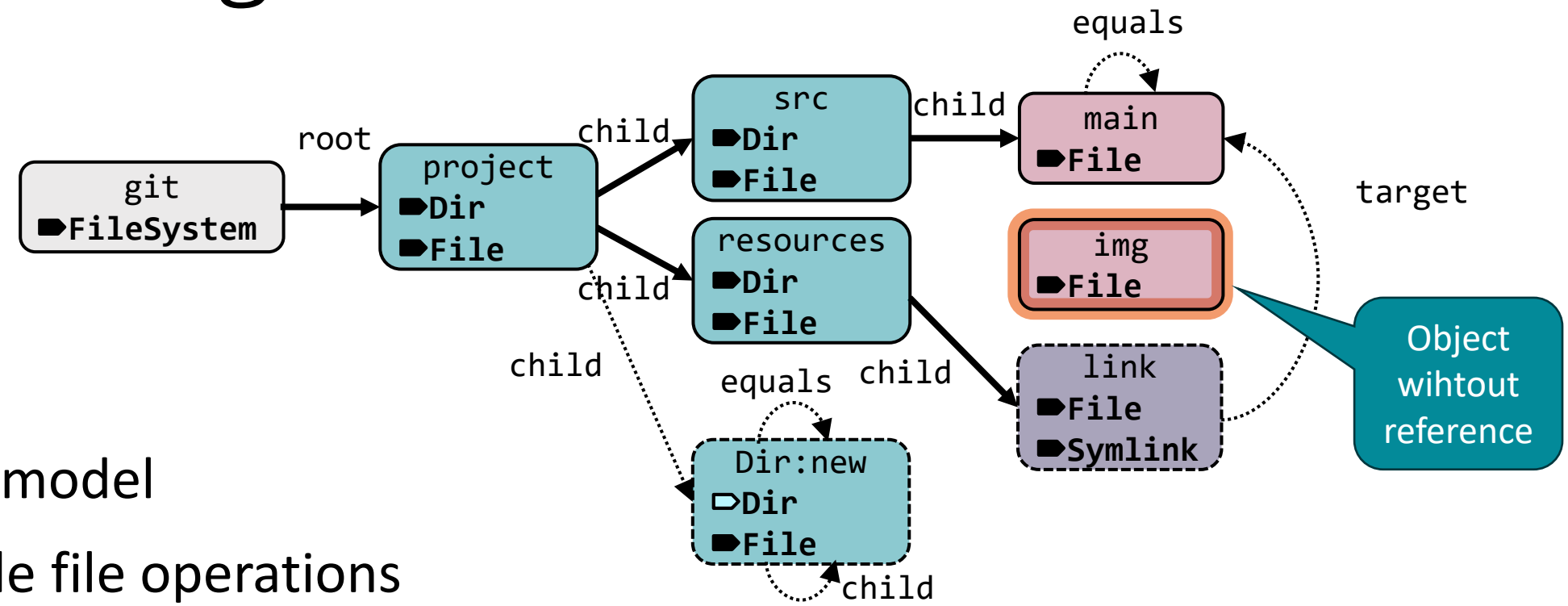


Reasoning over abstract models



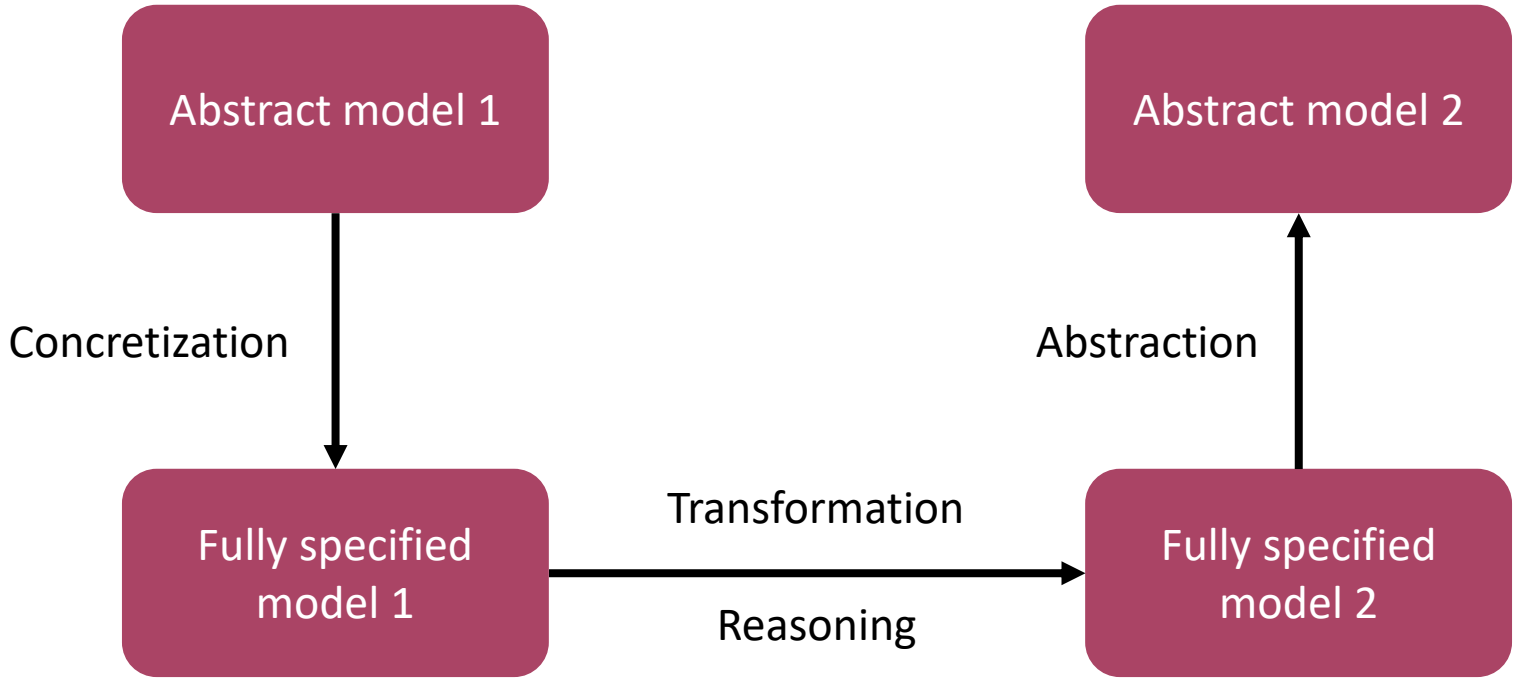
- Partial model
- Multiple file operations
  - Adding, removing and modifying files and directories
- Possibly called asynchronously
- Can we have a File without any reference pointing to it?

Reasoning over abstract models

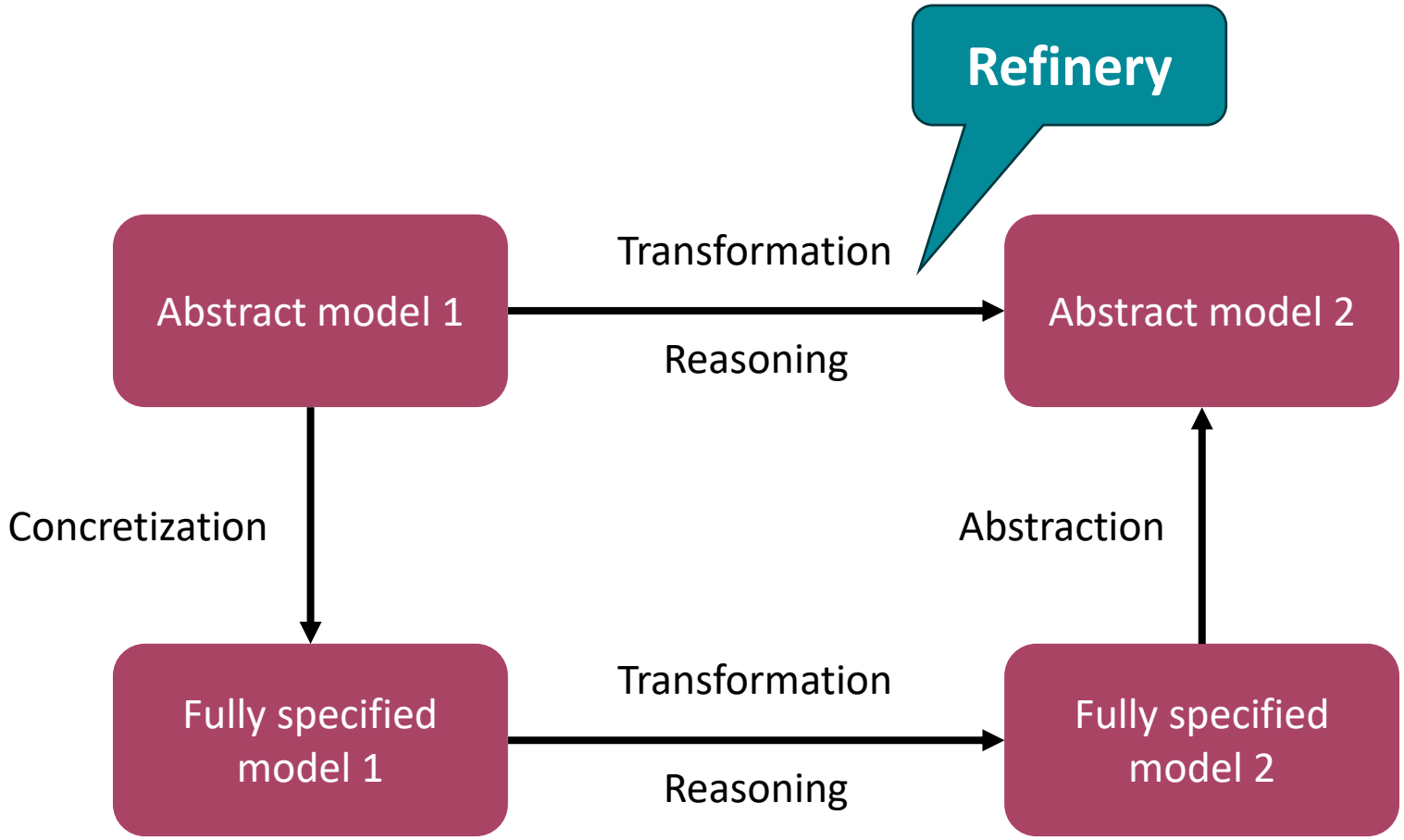


- Partial model
- Multiple file operations
  - Adding, removing and modifying files and directories
- Possibly called asynchronously
- Can we have a File without any reference pointing to it?

Reasoning over abstract models



Reasoning over abstract models



Summary of Refinery

- **Logical reasoning** and **model generation** over graphs
- **Web-based editor:**
  - **Live editing** and **feedback**
  - Support for partial models and graph constraints
- **Containerized execution:**
  - Continuously deployed at https://refinery.services/
  - Available as **Docker image**: https://refinery.tools/learn/docker/
- **Framework** for graph processing tasks