On the Verification of Diagnosis Models

Franz Wotawa and Oliver Tazl

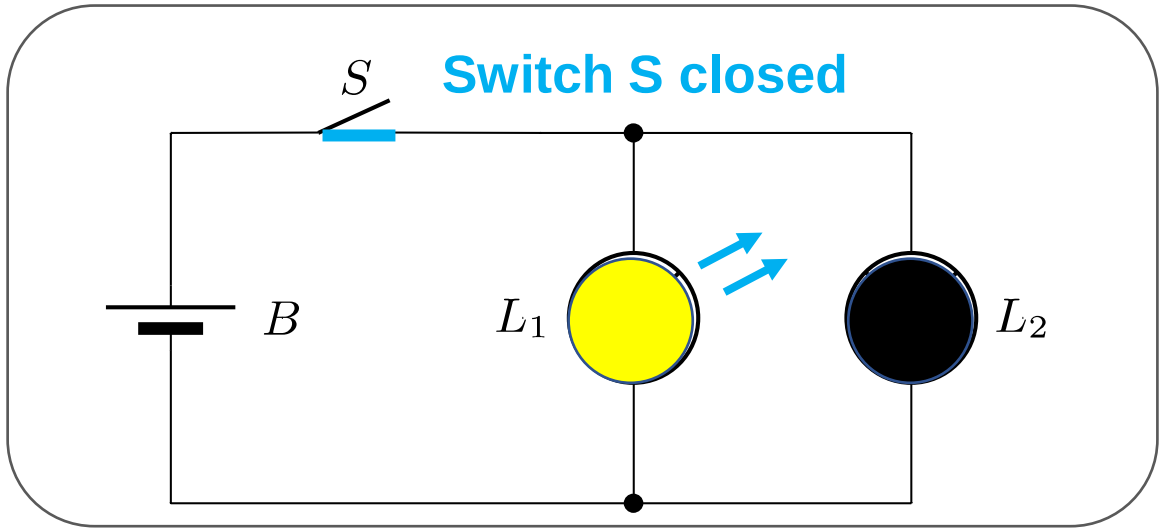
Institute for Software Technology, TU Graz

wotawa@ist.tugraz.at

EAI4IA Workshop, 25-26 July 2022, Vienna, Austria

Motivation

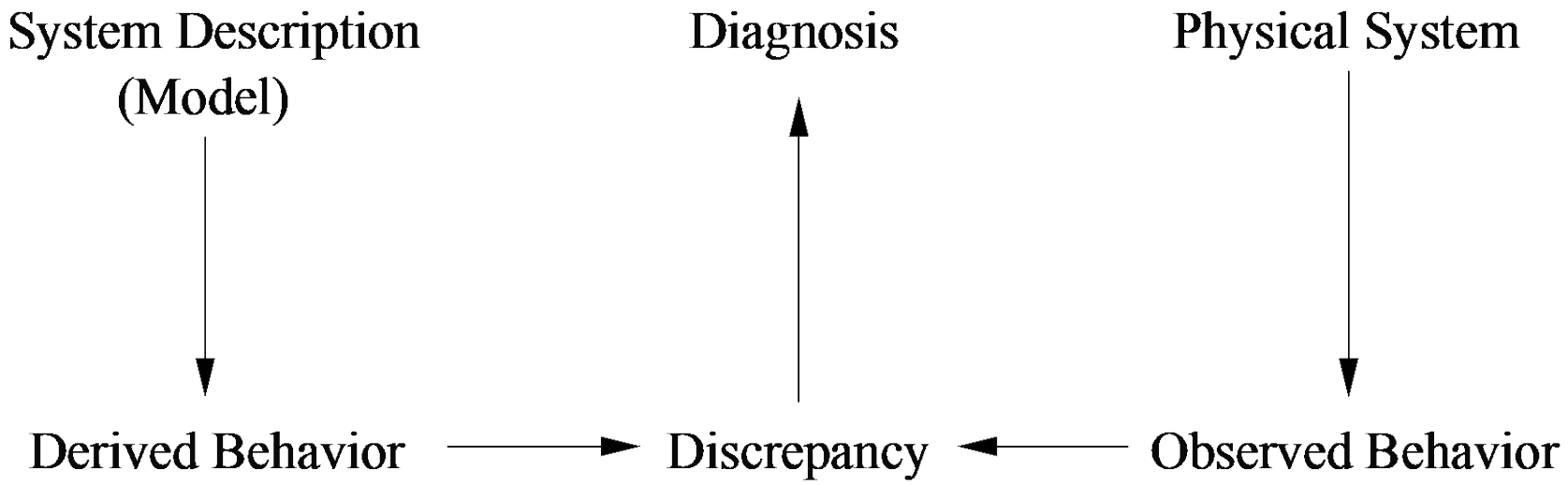
- Given a **system** together with **observations**
- Search for **root causes** explaining the observations



WHY IS THIS CIRCUIT DELIVERING A WRONG OUTPUT?

Motivation

- Want to use the model of a system directly!
- Model-based reasoning / diagnosis (MBD)



See e.g.: Davis, R.: Diagnostic reasoning based on structure and behavior. Artificial Intelligence 24, 347–410 (1984)

Motivation

- **Model-based diagnosis:**
  - Automated diagnosis based on models
  - Models: Component-Connection models
  - Easier modeling
  - Supports model re-use
  - High computational complexity! (NP-complete)
    - However, polynomial if considering diagnosis of size 1, 2, or 3
  - Algorithms usually are based on conflicts and hitting sets (e.g. HSDAG)

- **Question:** *Given that the implementation of the diagnosis procedure is correct, can we assure the correctness of diagnosis?*
  → **NO! Need to verify the underlying model as well!**

Answer Set Programming (ASP)

- An answer set is a satisfiable set of propositions that can be derived from the answer set program in an acyclic way

- **Examples**:

```
a :- b.
b :- a.
```

No answer set!

```
a :- not b.
b :- not a.
```

Two answer sets! {a} and {b}

ASP Tool `clingo`: `https://potassco.org/`

From ASP to Diagnosis

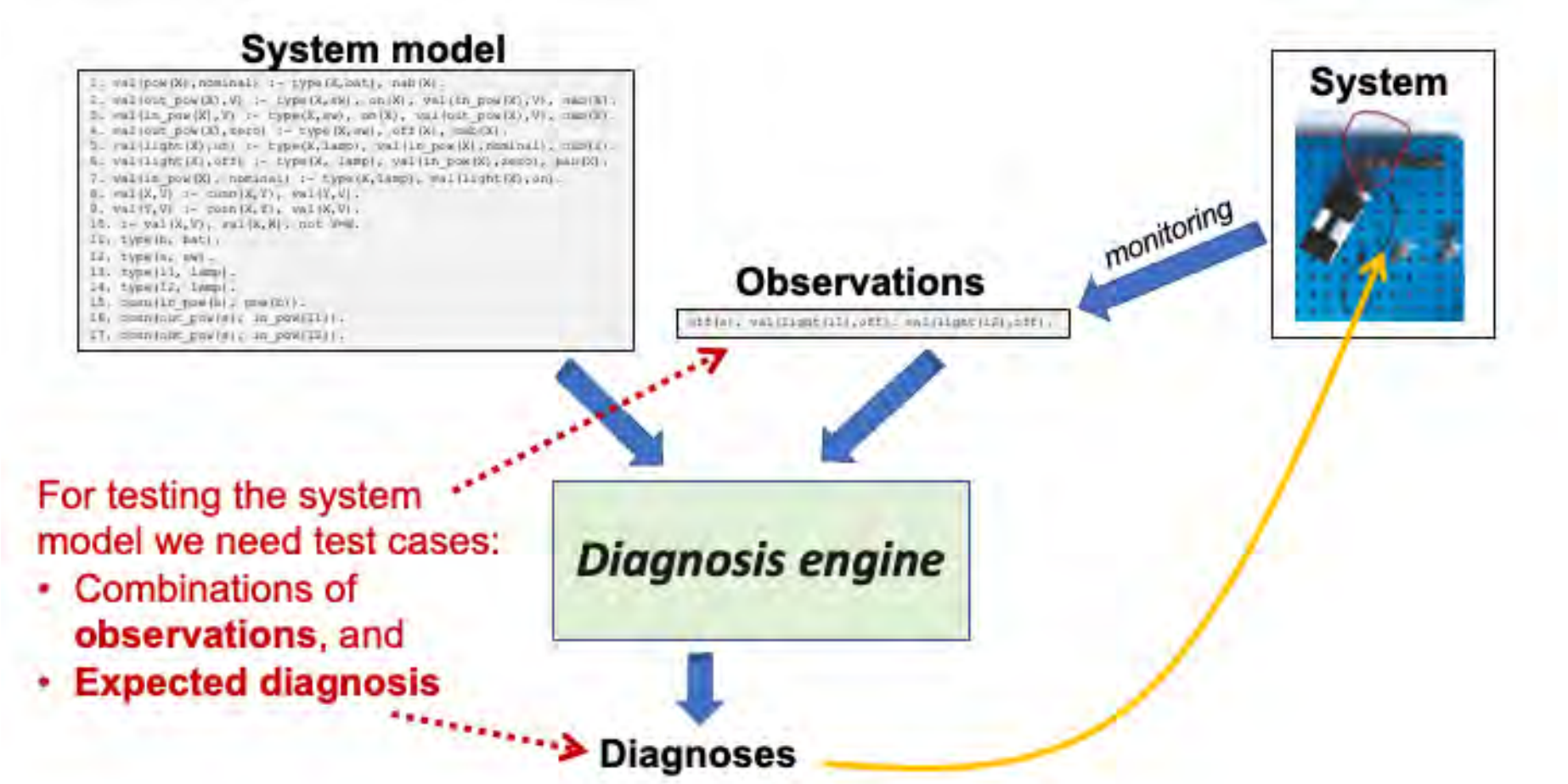
- In Model-based Diagnosis we make the health status of a component explicit!
- E.g. for components *C* we introduce a predicate $\neg ab(C)$.

- Diagnosis using ASP is searching for assignments to $\neg ab(C)$.

Model of the two-bulb system

```
1.  val(pow(X),nominal) :- type(X,bat), nab(X).
2.  val(out_pow(X),V) :-    type(X,sw), on(X),
                          val(in_pow(X),V), nab(X).
3.  val(in_pow(X),V) :-     type(X,sw), on(X),
                          val(out_pow(X),V), nab(X).
4.  val(out_pow(X),zero) :- type(X,sw), off(X), nab(X).
5.  val(light(X),on) :- type(X,lamp),
                          val(in_pow(X),nominal), nab(X).
6.  val(light(X),off) :-    type(X, lamp),
                          val(in_pow(X),zero), nab(X).
7.  val(in_pow(X), nominal) :-  type(X,lamp),
                              val(light(X),on).
8.  val(X,V) :- conn(X,Y), val(Y,V).
9.  val(Y,V) :- conn(X,Y), val(X,V).
10. :- val(X,V), val(X,W), not V=W.
11. type(b, bat).
12. type(s, sw).
13. type(l1, lamp).
14. type(l2, lamp).
15. conn(in_pow(s), pow(b)).
16. conn(out_pow(s), in_pow(l1)).
17. conn(out_pow(s), in_pow(l2)).
```

Model verification



Model verification

- Given a model M(S) for components of given types and their connections for a system $S \in \Sigma$.
- For testing we want to have the following:
  - A set of systems Σ and for each system $S \in \Sigma$ a model M(S) representing the structure, i.e., its components and connections.
  - For each system S, we want to have a set of inputs, i.e., possible observations, and a set of expected diagnoses. Note that observations include inputs and outputs of a system, and control commands (like opening or closing a switch).

Model verification (testing principle)

- For testing the two-bulb example, we need the model and a (sub-) set of observations.
- For all tests we make use of the diagnosis implementation and check whether the expected diagnoses are also computed!

- All test cases:

	Observations	Expected diagnoses	P/F
1	`on(s). val(light(l1),on). val(light(l2,on)).`	$\{\{\}\}$	$\surd$
2	`off(s). val(light(l1),off). val(light(l2,off)).`	$\{\{\}\}$	$\surd$
3	`off(s). val(light(l1),on). val(light(l2,off)).`	$\{\{s, l2\}\}$	$\surd$
4	`off(s). val(light(l1),off). val(light(l2,on)).`	$\{\{s, l1\}\}$	$\surd$
5	`off(s). val(light(l1),on). val(light(l2,on)).`	$\{\{s\}\}$	$\surd$
6	`on(s). val(light(l1),on). val(light(l2,off)).`	$\{\{l2\}\}$	$\surd$
7	`on(s). val(light(l1),off). val(light(l2,on)).`	$\{\{l1\}\}$	$\surd$
8	`on(s). val(light(l1),off). val(light(l2,off)).`	$\{\{b\}, \{s\}, \{l1, l2\}\}$	$\surd$

Discussion / Quality of testing

- Have we tested the model enough?
  - How to check this?
  - We may use ideas from software and system testing, e.g., mutation testing / mutation score
- Mutations: Slight variations of artifacts (e.g., removing one line of the model)

- Results of mutation testing for our two-bulb example:

	M1	M2	M3	M4	M5	M6	M7
1	√	√	√	√	√	√	√
2	√	√	√	√	√	√	√
3	√	√	√	×	×	√	×
4	√	√	√	×	×	√	×
5	√	√	√	×	√	√	×
6	√	√	√	√	×	√	×
7	√	√	√	√	×	√	×
8	×	×	√	√	×	√	√

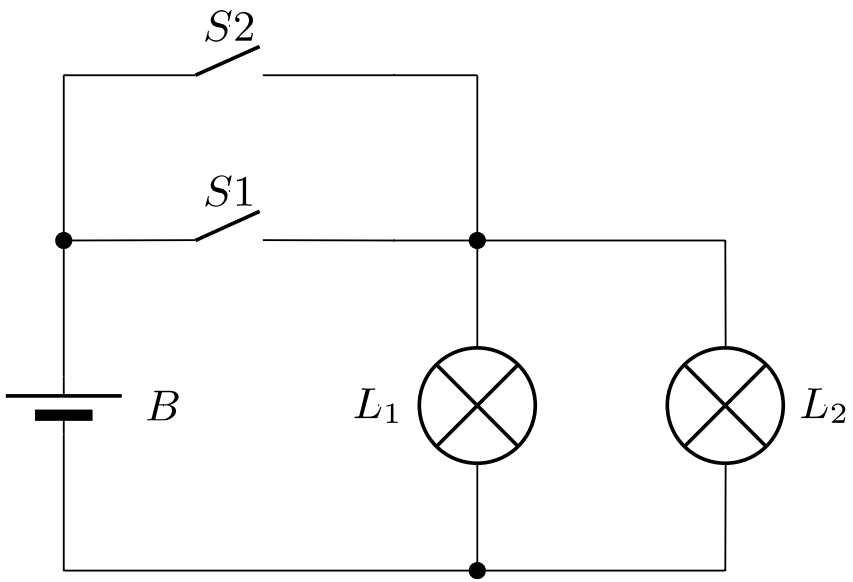
Discussion / Quality of testing

- Hence, there are some mutations that cannot be killed!
- Need more tests!
- But, have already considered all combinations of inputs!

- **More tests means different systems relying on the same behavioral model.**

Extending testing using different systems

- New system (behavior model remains the same, only structure varies)



	Observations	Expected diagnoses	P/F
1	on(s1). on(s2). val(light(l1,on)). val(light(l2),on).	{{}}	√
2	off(s1). on(s2). val(light(l1,on)). val(light(l2),on).	{{}}	×
3	on(s1). off(s2). val(light(l1,on)). val(light(l2),on).	{{}}	×
4	off(s1). off(s2). val(light(l1,off)). val(light(l2),off).	{{}}	√
5	on(s1). on(s2). val(light(l1,off)). val(light(l2),on).	{{$l1$}}	√
6	on(s1). on(s2). val(light(l1,on)). val(light(l2),off).	{{$l2$}}	√
7	on(s1). on(s2). val(light(l1,off)). val(light(l2),off).	{{b},{$s1,s2$}{$l1,l2$}}	√
8	on(s1). off(s2). val(light(l1,off)). val(light(l2),on).	{{$l1$}}	×
9	on(s1). off(s2). val(light(l1,on)). val(light(l2),off).	{{$l2$}}	×
10	on(s1). off(s2). val(light(l1,off)). val(light(l2),off).	{{b},{$s1$}{$l1,l2$}}	×
11	off(s1). on(s2). val(light(l1,off)). val(light(l2),on).	{{$l1$}}	×
12	off(s1). on(s2). val(light(l1,on)). val(light(l2),off).	{{$l2$}}	×
13	off(s1). on(s2). val(light(l1,off)). val(light(l2),off).	{{b},{$s2$},{$l1,l2$}}	×
14	off(s1). off(s2). val(light(l1,on)). val(light(l2),off).	{{$s1,s2,l2$}}	√
15	off(s1). off(s2). val(light(l1,off)). val(light(l2),on).	{{$s1,s2,l1$}}	√
16	off(s1). off(s2). val(light(l1,on)). val(light(l2),on).	{{$s1,s2$}}	√

Extending testing using different systems

- Using different system models may reveal faults in the component models!
- Require testing considering different systems as well as different observations
- May require a concept of coverage regarding potential system models

- Note that:
  - Test execution can be automated!
  - We assume that the diagnosis implementation is correct
  - For more details about challenges have a look at our paper!

Conclusions

- Considering observations and structural models as inputs is required for testing
- Mutation testing using adapted mutation operators can be applied
- The test oracle is the set of expected diagnoses, which is difficult to obtain.
- Test automation:
  - Test execution can be easily automated
  - Test case generation is challenging and requires further research

Thank you for your attention!

QUESTIONS?