Structure-based Configuration

What is "configuration"?

- **Assembling a (technical) system from individual parameterisable objects to a configuration that fulfills a certain task (or purpose)**
- **"Configuration" means:**
  - **the process of combining and**
  - **the result of the process, i.e. a list of components**
  - **the field in AI, which deals with knowledge-based configuration**

What is "structure-based configuration"?

- **Configuration based on a declarative representation of the compositional structure of configurable systems**

Examples: Structure-based configuration of industrial machinery, cars, aircraft cabins, chemical structures, software

1

History of Structure-based Configuration Systems

- **Success of rule-based configuration with XCON (1982 - 1988)**
- **AI funding of German government offered for "Technische Expertensysteme für Konstruktion TEX-K" (1984).**
- **Development of the configuration system shell PLAKON (1986 - 1990) in a joint project with partners Batelle (Frankfurt), Philips (Hamburg), Siemens (Erlangen), URW (Hamburg) and Hamburg University.**
- **Development of the configuration system shell KONWERK with application-specific modules in a joint project led by Hamburg University (1991 - 1995).**
- **Development of the commercial configuration tool EngCon based on KONWERK (1996 - 1998).**

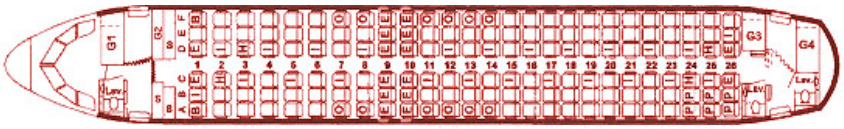
2

Main Components of Structure-based Configuration

- **Domain objects (components, aggregates)**
  Conceptual descriptions of technical components
- **Configuration model**
  Conceptual description of permissable configurations
- **Concrete configuration task**
  Description of customer wishes
- **Configuration strategies**
  Control strategies, e.g. top-down, least-commitment

3

Application Example Cabin-layout

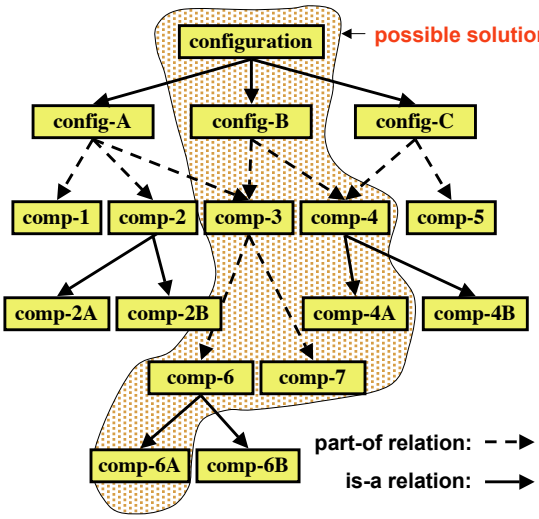


Placement of cabin equipment in view of customer wishes, technical constraints, legal constraints and optimality criteria

Domain objects:	**seats, kitchens, toilets, etc.**
Configuration model:	**possible kinds of domain objects and ways to place them into a cabin**
Configuration task:	**> 200 seats economy, as little leg room as legally possible, 50 seats business, 2 toilets, etc.**
Configuration strategy:	**focus business seats, interactive**

4

Representation of a Configuration Model



- boxes (frames) specify aggregate and component properties
- has-part relations bind components to aggregates
- is-a relations describe variants of entities
- constraints between entities (not shown) restrict choices and parameter combinations

5

Object Description Language

In a configuration model, an "object" describes possible choices for a configuration component (primitive or aggregate) by means of a conceptual expression.

In the following, we present the concept language BHIPS ("Begriffshierarchie-Beschreibungssprache") developed for the configuration system frameworks PLAKON and KONWERK.

Example:

```
(ist!     (ein Auto)
          (ein Konstruktionsobjekt
               (Geschwindigkeit [0km/h  300km/h] )
               (Farbe {rot  grün  blau  schwarz} )
               (Hersteller (eine Autofirma)
               (Has-Parts (:set   (ein Motor)
                                  (eine Karosserie)
                                  (ein Fahrgestell)))))
```

6

Concept for ´Galley´ of an Airbus A340

```
def-concept
     :name galley
     :super-concept {cabin-interior-component rectangle}
     :parameters
          ref-nr [integer 2531000 2533999]
          door {1 2 4}
          trolleys {0 2 3 4 5 6 7 8 9 10}
          half-size-trolleys {0 1 2 3 4 5}
          meals [integer 28 140]
          type {longitudinal transversal}
          height {full half} (default 'full)
     :relations
          part-of [passenger-class]
```

7

Concept Expressions in BHIBS

Concept expression for objects in a configuration model:

```
(ist!     (ein <concept name>)
          (ein <concept expression1>)
          ...
          (ein <concept expressionK>)
```

Concept expressions:

```
(ein <concept parent name>
          <object descriptor1>
          ...
          <object descriptorN>)
```

Compare to frame languages!

8

Object Descriptors (1)

Specific values:	**(colour red)** **(weight 35t)**
Value sets:	**(colour {red, green, blue})**
Ranges:	**(speed [0kmh 300kmh])**
Predicates:	**(number-of-wheels (:satisfies evenp))**
Logical operators:	**(:and [4 12] (:satisfies evenp))**

Object Descriptors (2)

Complex sets:

(:set :some [(ein <concept$_0$>) m_0 n_0] :>
:some [(ein <concept$_1$>) m_1 n_1]
. . .
:some [(ein <concept$_k$>) m_k n_k])

Between m_0 and n_0 elements of concept$_0$, consisting (among others) of m_1 to n_1 elements of concept$_1$, ... , m_k to n_k elements of concept$_k$.

Similarly: (:set :some [(ein <concept$_0$>) m_0 n_0] :=
:some [(ein <concept$_1$>) m_1 n_1]
. . .
:some [(ein <concept$_k$>) m_k n_k])

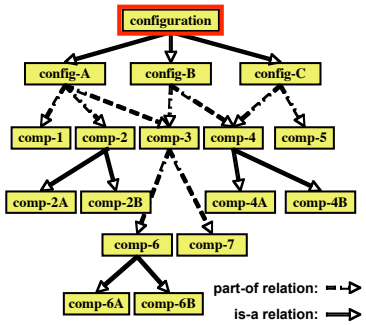
Example: (:set :some [(ein Autoteil>) 3 3] :=
:some [(ein Motor) 1 1]
:some [(ein Fahrgestell) 1 1]
:some [(eine Karosserie) 1 1])

Parts of Aggregates

Together with thespecialization hierarchy, the specification of parts is important for guiding the configuration process.

```
(ist! (ein Auto)
      (ein Konstruktionsobjekt
           (Has-Parts (:set (ein Motor)
                            (eine Karosserie)
                            (ein Fahrgestell)))))
```

Aggregates and parts constitute a compositional hierarchy with a top node denoting all possible configurations.

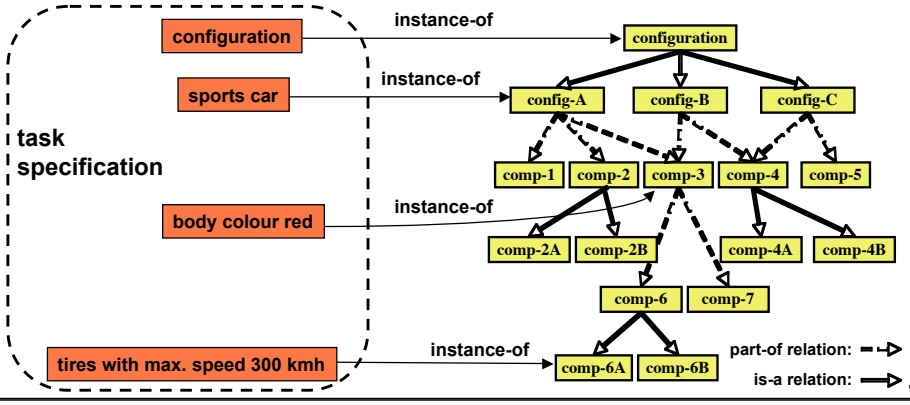


11

Task Specification

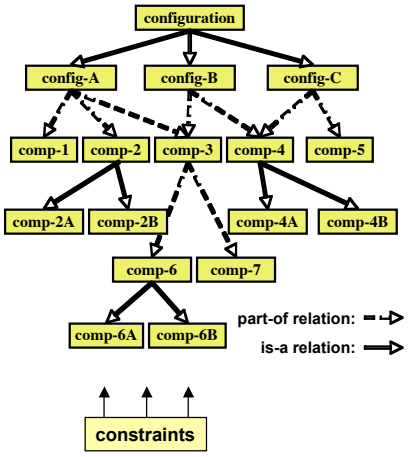
A concrete configuration task is specified in terms of instances of concepts and relations of the configuration model which are interpreted as desired parts of the solution.

An instance of the top node of the compositional hierarchy ("configuration") is always part of a task description.



12

Configuration Steps



Kinds of configuration steps during the configuration process:

- instance specialization
- aggregate expansion
- aggregate instantiation
- parameterization
- instance merging

At any time, constraint propagation may be initiated to narrow down choices or detect conflicts.

Configuration steps must be able to create all configurations permitted by the configuration model.

13

Configuration Cycle

Repeat
- Check for goal completion
- Determine current strategy
- Determine possible configuration steps
  - Select from agenda and execute one of
    - { instance specialisation,
    - aggregate expansion,
    - aggregate instantiation,
    - parametrization,
    - instance merging }
- Propagate constraints
- Check for conflict

Goal completion:

Specific values are selected for all choices for instances selected from the configuration model.

14

Constraints

Concept expressions involve only unary predicates and parts specifications for a single aggregate.

N-ary predicates between parameters of arbitrary aggregates can be expressed by constraints.

Constraints are part of the configuration model and subject to constraint propagation during the configuration process.

Constraints are defined as conceptual constraints. Constraint instances are generated during the configuration process and integrated into a constraint net.

15

Conceptual Constraints

Examples:

The displacement of a cylinder is the squared diameter times the height of stroke times π/4.

```
(constrain  ((#?Z  (ein Zylinder)
            (Square (#?Z  Durchmesser)  ?Q)
            (Multiply ?Q  π/4 ?P)
            (Multiply  ?P  (#? Hubhöhe)  (#? Hubraum))
```

The displacement of a motor is the sum of the displacement values of all cylinders which are constrained to be equal.

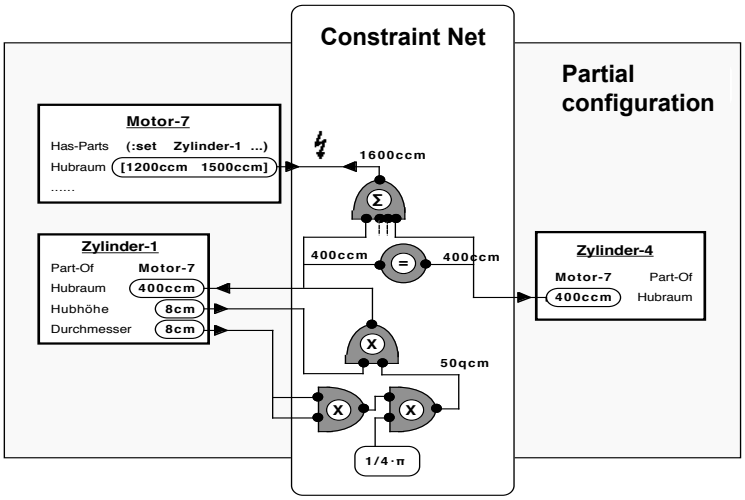
```
(constrain  ((#?M  (ein Motor) (#?Z  :all (ein Zylinder (part-of #?M))))
            (All-Equal (#?Z Hubraum))
            (Sum  (#?M Hubraum)  (#?Z Hubraum)))
```

Constraint propagation is multi-directional in general!

16

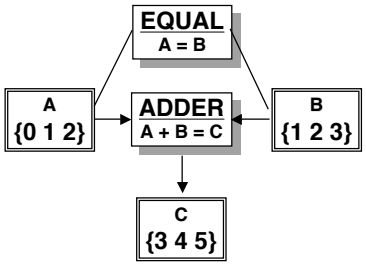
Constraint Net

A constraint net is created when concepts of the configuration model are instantiated during the configuration process.

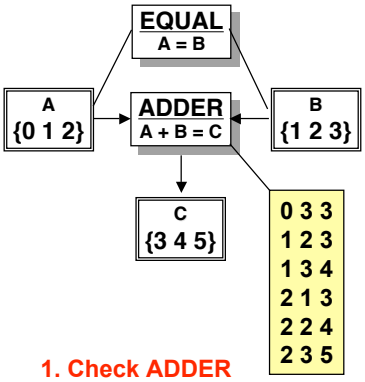


17

Local Propagation (1)



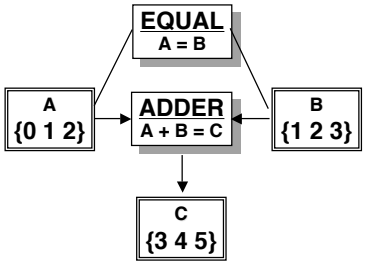
Constraints are evaluated locally to determine path consistency

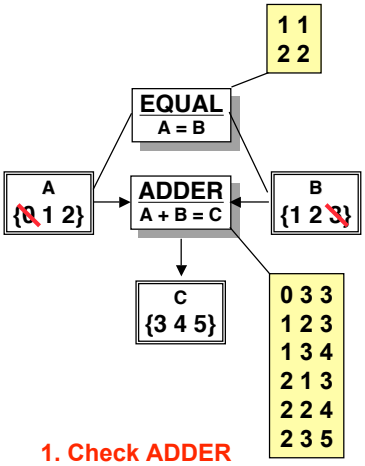


1. Check ADDER

18

Local Propagation (2)



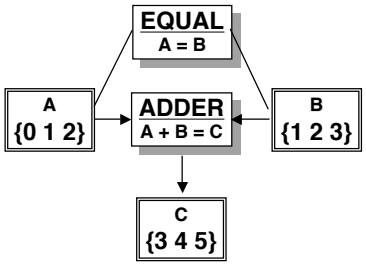


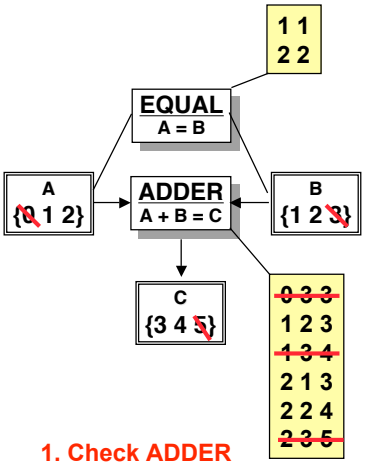
Constraints are evaluated locally to determine path consistency

1. Check ADDER
2. Check EQUAL

19

Local Propagation (3)



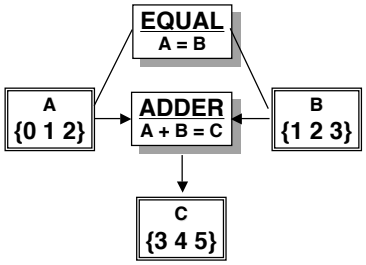


Constraints are evaluated locally to determine path consistency

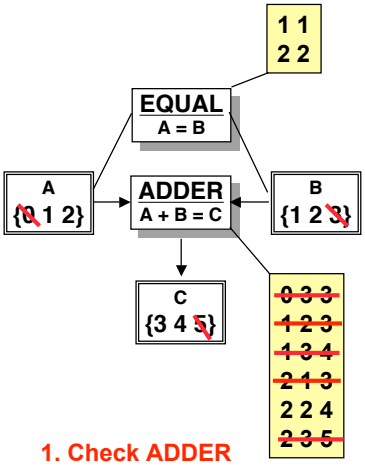
1. Check ADDER
2. Check EQUAL
3. Check ADDER

20

Propagation for Global Consistency



Constraints are evaluated
dependent on each other
to determine global consistency



1. Check ADDER
2. Check EQUAL
3. Check ADDER
Check global consistency

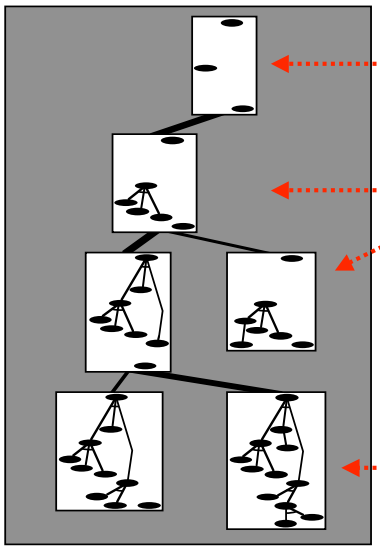
21

Control Knowledge

Each configuration cycle involves one configuration step selected from the agenda. Control knowledge determines which step to select.

22

Illustration of Stepwise Configuration



23

Interactive Configuration

- Based on the configuration model and the task specification, the user interactively constructs a configuration
- The system supports the user and checks correctness and completeness of the solutions
- Uniquely inferable decisions are automatically made by the system

24

Heuristic Configuration

- Based on the configuration model and the task specification, the configuration system automatically makes heuristic decisions.
- Heuristic knowledge is represented as
  - rules
  - default values
  - computable values

25

Building the Agenda

Example:

Instantiation of ENGINE gives rise to several new agenda entries

```
Instance ENGINE-007
is-a:      Construction-Object
is-a-inv:  Rallye-Engine, Racing Engine
power:     [10hp  250hp]
part-of:   Car
has-parts: (:set [Cylinder 2 12]
                 Ignition
                 [Turbocharger 0 1])
```

Agenda:
...

specialize	ENGINE-7	
decompose	ENGINE-7	slot HAS-PARTS
parametrize	ENGINE-7	slot POWER

26

Selecting from the Agenda

Types of agenda selection criteria:

- **Selection by Pattern**
- **Selection by Rating**
- **Selection by Sequence**

Examples for selection criteria:

Prefer Specialization Steps
Prefer Decomposition Steps
Prefer steps concerning the ENGINE
Prefer slot POWER of ENGINE

27

Value Determination Methods

A decision for a concrete value is obtained by one of several value determination methods:

- **using a static or dynamic default**
- **querying the user**
- **evaluating a function**
- **evaluating a heuristic rule**
- **propagating constraints**
- **evaluating external simulation methods**
- **using a library solution**

28

Resolution of configuration conflicts

Retracing of decisions

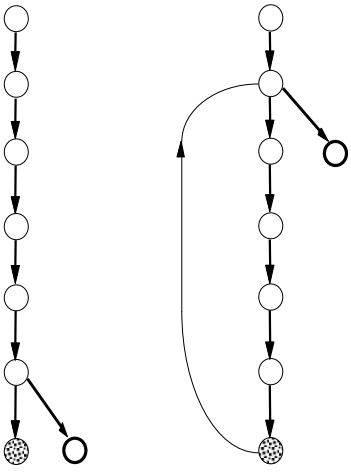
- Chronological Backtracking
- Dependency-based Backtracking
- Knowledge-based Backtracking
- Marking backtracking points
- Interactive Backtracking
- Backtracking with data migration

Repair

- Repair statements
- Interactive modification
- Constraint-Relaxation

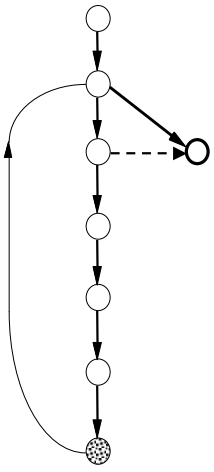
29

Backtracking



Chronological Backtracking

"Intelligent" Backtracking



"Intelligent" Backtracking with data migration

30

Applications realized with KONWERK

- Configuring aircraft passenger cabins
- Design of liquid crystals
- Configuration and dimensioning of cardan shafts
- Selection and dimensioning of slide bearings
- Generating an optimal layout of a logistic system
- Pre-design of a space transportation system
- Configuration of hydro-geological models
- Configuration of elevators
- Applications for modelling environmental problems
- Selection and configuration of measuring devices for chemical plants
- Design of digital analogue circuits
- Configuration and dimensioning of photo-voltaic systems
- Dimensioning and parameterisation of drive systems
- Configuring driver assistance systems
- Configuring software for car engines

31

Logical interpretation

Language constructs can be mapped to logical constructs of a description logic by using:

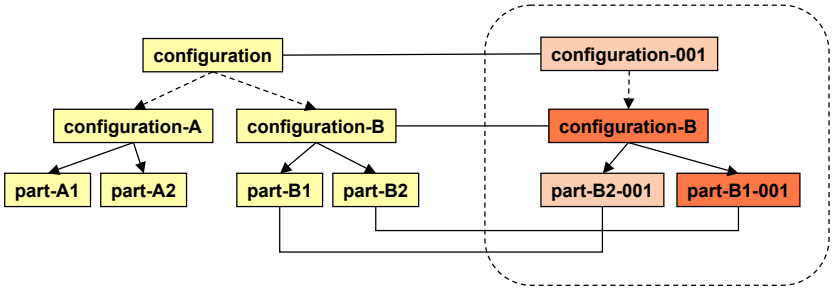
- Conjunction
- Negation and disjunction with atomic concepts
- Value restrictions
- Qualifying number restrictions
- Inverse roles
- Sets
- Concrete domains over $\mathcal{R}$

Configuration can be interpreted as logical model construction or abduction.

32

Configuration as Logical Model Construction

Instantiate the configuration model consistent with task description.



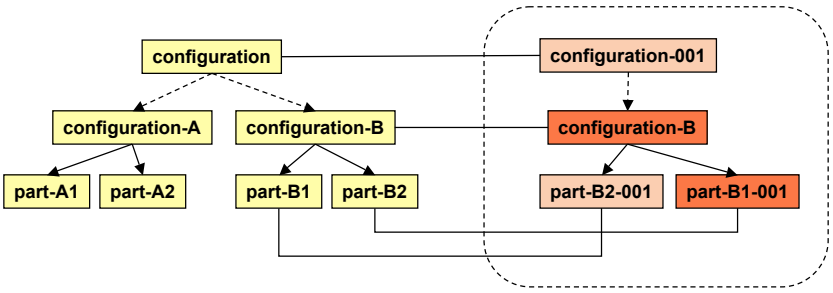
configuration model

constructed logical model involving task description

33

Configuration as Abduction

Find "proof" for task description in terms of a derivation from the configuration model.



configuration model

constructed proof by adding hypotheses about missing configuration components

34

Using the Structure-based Configuration Approach for Other Problem Types

The general idea behind structure-based configuration is

- declarative representation of problems and their solutions, not of the steps leading up to a solution
- separation of domain knowledge and control

This is possible due to a representation language which allows to define

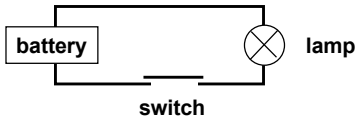
- object classes
- ranges of property values
- possible decompositions
- constraints between arbitrary properties

Can this approach be also applied to problem types other than configuration?

35

Problem-Solving Space for Diagnosis

Example:



What are possible causes for the lamp to be dark?

The objects of knowledge representation are components of the domain and situations described by symptoms and their causes.

The diagnosis task is carried out by using a task specification in terms of current symptoms for selecting fitting situations and finding out about causes.

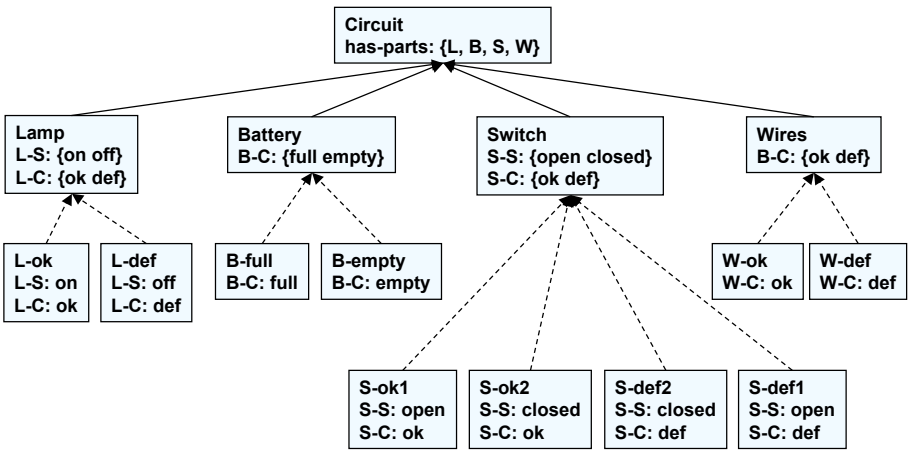
Compare to configuration:

Configuration is carried out by using a task specification in terms of desired components for selecting a configuration.

36

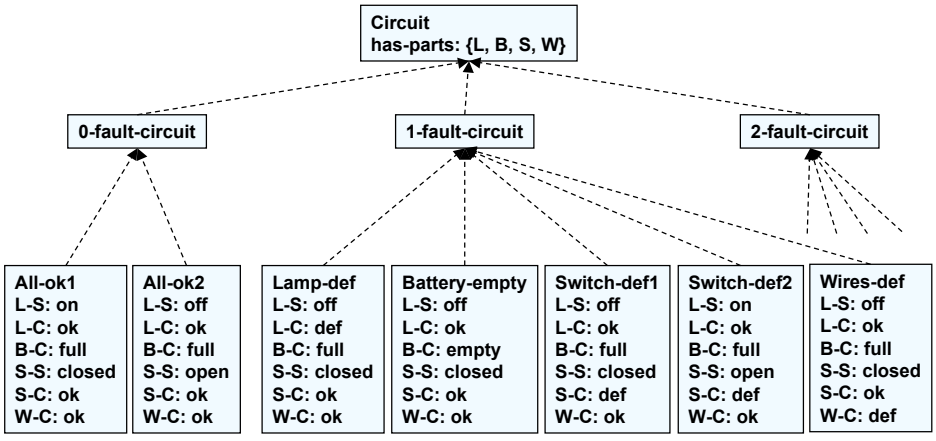
Problem-solving Space for Example (1)

Situations decribed by symptoms (S) and causes (C)
L = Lamp, B = Battery, S = Switch, W = Wires



37

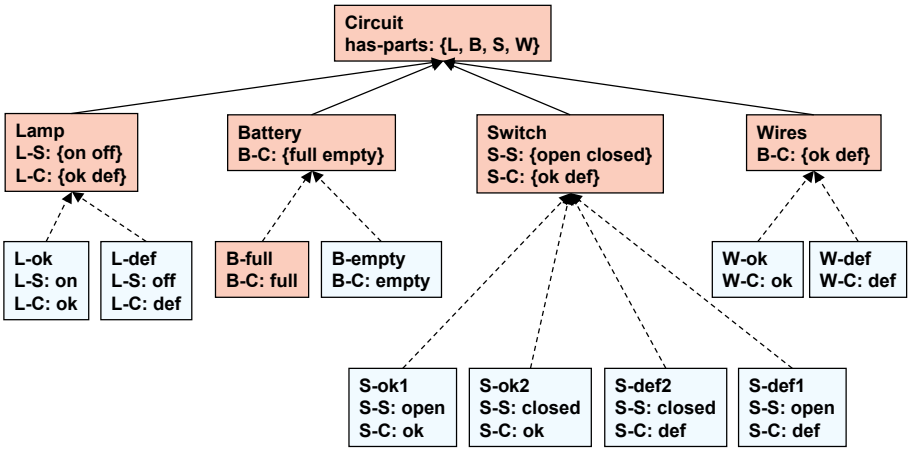
Problem-solving Space for Example (2)



38

Concrete Diagnosis Task (1)

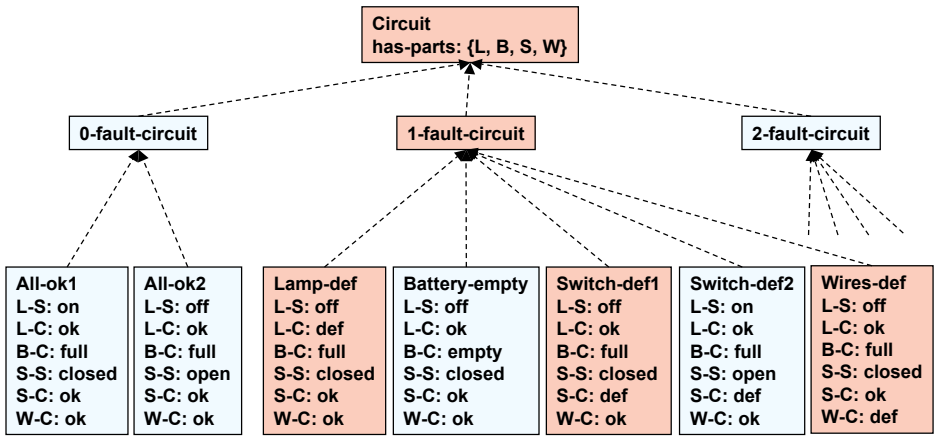
Observed:
- lamp is off (L-S: off)
- switch is closed (S-S: closed)
- battery is full (B-S: full)



39

Concrete Diagnosis Task (2)

Observed:
- lamp is off (L-S: off)
- switch is closed (S-S: closed)
- battery is full (B-S: full)



40