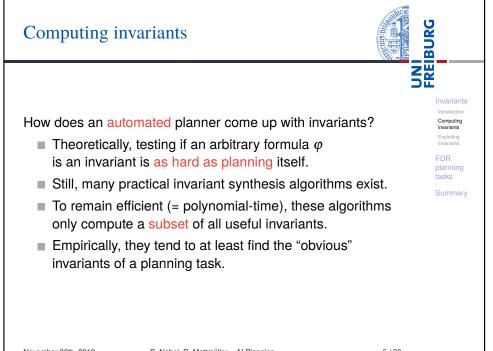


1 Invariants		BURG
 Introduction Computing in Exploiting in 	invariants	Literatoria Introduction Computing Invariants Exploiting Invariants Inva
November 30th, 2012	B. Nebel, R. Mattmüller – Al Planning	3 / 20



November 30th, 2012

Invariant synthesis algorithms



Introduction

Computing

invariants

Exploiting

FDR

planning tasks

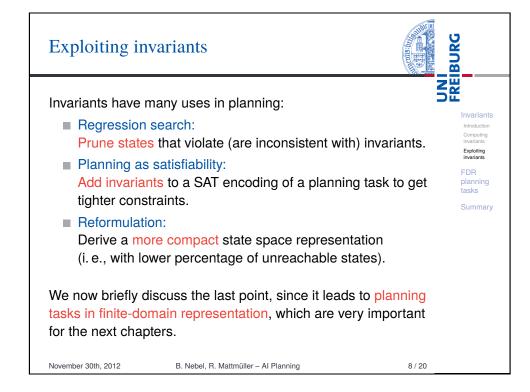
Most algorithms for generating invariants are based on a generate-test-repair paradigm:

- Generate: Suggest some invariant candidates, e.g., by enumerating all possible formulas φ of a certain size.
- **Test:** Try to prove that φ is indeed an invariant. Usually done inductively:
 - **1** Test that initial state satisfies φ .
 - **2** Test that if φ is true in the current state, it remains true after applying a single operator.
- **Repair:** If invariant test fails, replace candidate φ by a weaker formula, ideally exploiting why the proof failed.

November 30th, 2012

B. Nebel, R. Mattmüller - Al Planning

6/20



Invariant synthesis: references



Computing

invariants

Exploiting

planning

FDR

. tasks

We discussed invariant synthesis in detail in previous courses on AI planning, but this year we will focus on other aspects of planning.

Literature on invariant synthesis:

- DISCOPLAN (Gerevini & Schubert, 1998)
- TIM (Fox & Long, 1998)
- Edelkamp & Helmert's algorithm (1999)
- Rintanen's algorithm (2000)
- Bonet & Geffner's algorithm (2001)
- Helmert's algorithm (2009)

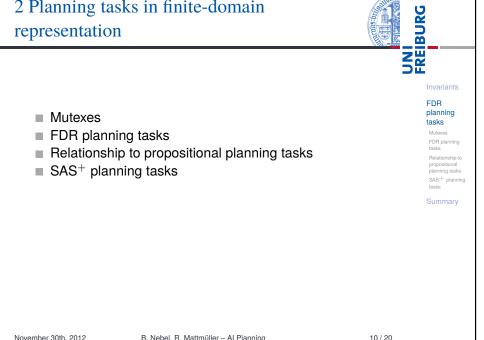
November 30th, 2012

November 30th, 2012

B. Nebel, R. Mattmüller - Al Planning

2 Planning tasks in finite-domain

7/20



B. Nebel, R. Mattmüller - Al Planning

Mutexes



Invariants that take the form of binary clauses are called mutexes because they state that certain variable assignments cannot be simultaneously true and are hence mutually exclusive.

Example (Blocksworld)

The invariant $\neg A$ -on- $B \lor \neg A$ -on-C states that A-on-B and A-on-C are mutex.

Summary

planning

tasks

tasks

Mutexes

SAS+ plannin

Often, a larger set of literals is mutually exclusive because every pair of them forms a mutex.

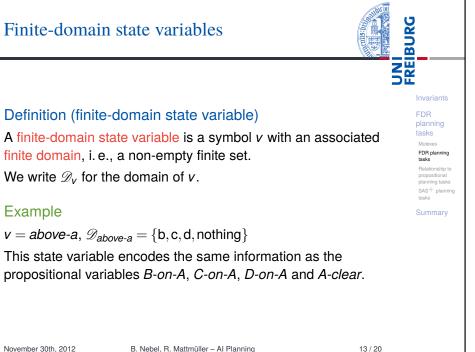
Example (Blocksworld)

Every pair in {*B-on-A*, *C-on-A*, *D-on-A*, *A-clear*} is mutex.

November 30th, 2012

B. Nebel, R. Mattmüller - Al Planning

11/20



Encoding mutex groups as finite-domain variables



FDB

tasks

Mutexes FDR planning

Relationship propositional

SAS+ plannin tasks

Let $L = \{I_1, ..., I_n\}$ be mutually exclusive literals over *n* different variables $A_{L} = \{a_{1}, ..., a_{n}\}.$

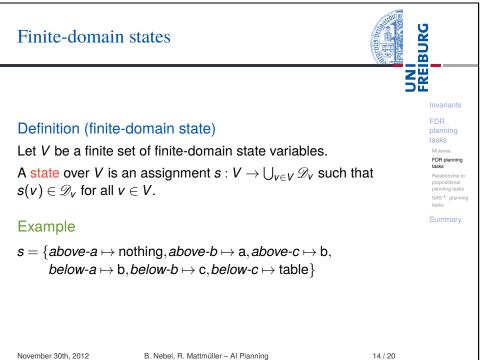
Then the planning task can be rephrased using a single finite-domain (i.e., non-binary) state variable v_l with n+1possible values in place of the *n* variables in A_l :

- n of the possible values represent situations in which exactly one of the literals in L is true.
- The remaining value represents situations in which none of the literals in L is true.
 - Note: If we can prove that one of the literals in *L* has to be true in each state, this additional value can be omitted.

In many cases, the reduction in the number of variables can dramatically improve performance of a planning algorithm.

- November 30th, 2012
- B. Nebel, R. Mattmüller Al Planning

12/20



Finite-domain formulae



planning

tasks

tasks

tasks

15/20

Mutexes

FDR planning

Relationship t

planning tasks

SAS+ planning

Definition (finite-domain formulae)

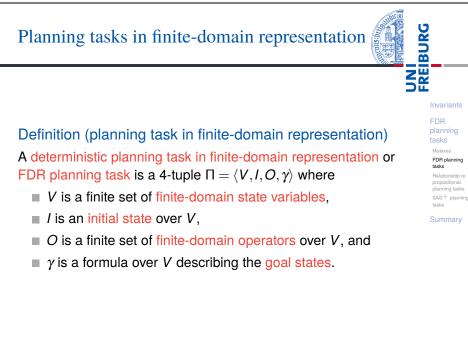
Logical formulae over finite-domain state variables *V* are defined as in the propositional case, except that instead of atomic formulae of the form $a \in A$, there are atomic formulae of the form v = d, where $v \in V$ and $d \in \mathcal{D}_{v}$.

Example

The formula $(above-a = \text{nothing}) \lor \neg (below-b = c)$ corresponds to the formula A-clear $\lor \neg B$ -on-C.

November 30th, 2012

B. Nebel, R. Mattmüller - Al Planning



Finite-domain effects



16/20

FDR

tasks

Mutexes

tasks

tasks

FDR planning

Relationship

propositional planning tasks

SAS+ plannin

Summary

Definition (finite-domain effects)

Effects over finite-domain state variables *V* are defined as in the propositional case, except that instead of atomic effects of the form *a* and $\neg a$ with $a \in A$, there are atomic effects of the form v := d, where $v \in V$ and $d \in \mathscr{D}_v$.

Example

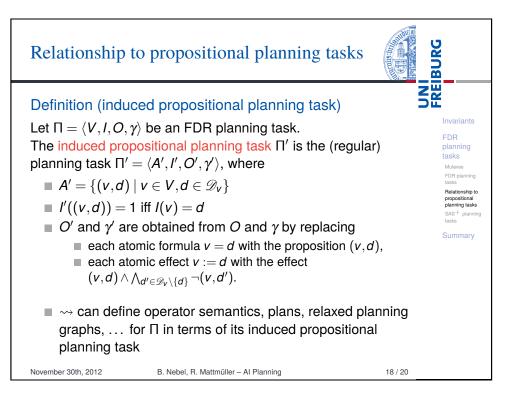
The effect

 $(below-a := table) \land ((above-b = a) \triangleright (above-b := nothing))$ corresponds to the effect $A-on-T \land \neg A-on-B \land \neg A-on-C \land \neg A-on-D \land (A-on-B \triangleright B-clear).$

~ definition of finite-domain operators follows from this

November 30th, 2012

B. Nebel, R. Mattmüller – Al Planning

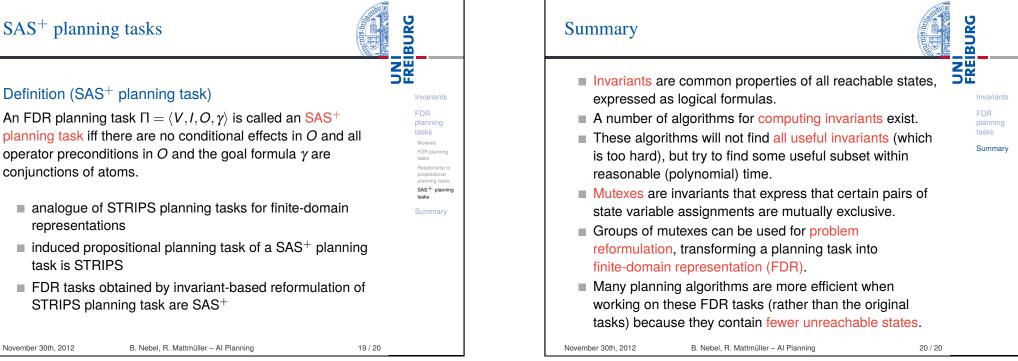


SAS⁺ planning tasks

conjunctions of atoms.

representations

task is STRIPS



November 30th, 2012

B. Nebel, R. Mattmüller - Al Planning