University of Basel

A Comparison of Invariant Synthesis Methods

Master's Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Artificial Intelligence
https://ai.dmi.unibas.ch/

Examiner: Prof. Dr. Malte Helmert
Supervisor: Dr. Silvan Sievers

Severin Wyss
severin.wyss@stud.unibas.ch
2013-053-707

October 5, 2020

Acknowledgement

I want to thank Prof. Helmert for the opportunity to write my Master's thesis in his research group, as well as for giving valuable advice on the more complex topics. An enormous thanks goes to my supervisor Silvan Sievers who supported me throughout the whole process in many different ways and for the patient sparring on my wilder ideas.

Further gratitude goes to my family for their great love and support at all times. Similar thanks are earned by my partner who always believed in me and helped me while she also wrote her thesis. Last but not least, I thank all my friends for their help, emotional support or just simply for their honest interest. Additionally, for their help proofreading this thesis, I would like to explicitly thank my mother, brother, Mélanie Müller, Marvin Buf, Alexander Korb and Philippe Götschman.

Abstract

We implemented the invariant synthesis algorithm proposed by Rintanen and experimentally compared it against Helmert's mutex group synthesis algorithm as implemented in Fast Downward.

The context for the comparison is the translation of propositional STRIPS tasks to FDR tasks, which requires the identification of mutex groups.

Because of its dominating lead in translation speed, combined with few and marginal advantages in performance during search, Helmert's algorithm is clearly better for most uses. Meanwhile Rintanen's algorithm is capable of finding invariants other than mutexes, which Helmert's algorithm per design cannot do.

Contents

1 Introduction

There are many real world tasks that benefit from being performed in an optimal order as opposed to a random order. For example a robot in a warehouse that has to bring objects from one location to another. While bringing them one by one would work, the robot can save time by analysing the task at hand and planning an optimal sequence of actions. A human is often able to find a solution to a simple task. But the robot example described above in a warehouse with a thousand locations and a thousand objects which each have separate targets would make it hard for a human to find an optimal solution before the next truck arrives and delivers new objects.

In planning we seek to find general algorithms that allow computers to find solutions, so-called plans, to problems. These plans are sequences of actions that change the world (which we describe as being in a state) to reach a goal state. The task is always to start in an initial state and to find a plan from there. We will only consider deterministic actions, where the outcome is fixed. A program that finds plans is called a planner. The way problems are commonly encoded for a general planner is by using propositional PDDL.

A very simple yet as powerful as any form of problem is STRIPS[5]. In STRIPS a state is encoded as variables describing one fact as either true or false. This binary nature hides certain concepts from the planner. In the given robot example, the planner does not immediately know that the robot can only be at one location. However, knowing this can be advantageous for estimating the distance to the goal, which in turn enables a better guided search for a plan. Such estimates are called heuristics. In a so-called finite domain representation proposed by Helmert [9], the state is encoded as variables. Each variable has a domain of values which describe facts that cannot be true at the same time. In each state the variables are mapped to exactly one of these values. An example would be a variable for the location of the robot that has as a domain all the locations the robot could be at.

When starting from a propositional PDDL[12], we can only combine propositional variables into a finite domain variable, if of all of those propositional variables, only ever one can be true in any state that we can reach from the initial state. This can be expressed as looking for a formula over propositional variables, which holds in every reachable state. A formula that holds in every reachable state is called an invariant. Here we are especially interested in mutexes, invariants that are a disjunction of exactly two negative facts. A mutex expresses that at most one of the two facts can be true. If we have a set of propositional variables and a corresponding set of mutexes over each pair of those propositional variables, then we call this set of propositional variables a mutex group. A mutex group indicates a domain for a finite domain variable.

Invariants that are no mutexes can also be used to prune states that can never reach the goal. To illustrate, let us consider a more complex problem where the robot has a time limit. This can be modeled in STRIPS, even though it is tedious. Then we might be able to determine an invariant, backwards from the goal, expressing that the robot will take at least a certain amount of time for the last two packages. Consequently, we could prune any state where there is do not enough time left.

In this thesis we implemented a ground algorithm for invariant synthesis on planning tasks in STRIPS as proposed by Rintanen [14]. We experimentally evaluated the ground algorithm against the build in mutex group synthesis algorithm from Fast Downward that was proposed by Helmert [9]. To get mutexes from a set of invariants, we implemented a greedy clique computation proposed by Rintanen [13].

2 Background

In this thesis we will work only with planning problems, that adhere to the restrictions of STRIPS [5]. First we establish schematic propositional tasks. While the definitions will start with the schematic version, we will also introduce ground versions within the same definition. Consequently, finite domain representation will be defined.

We will describe invariants, a special kind of invariant, the mutex, and two types of invariant candidates. These will be required for the definition of the algorithms by Rintanen and Helmert for computing invariant candidates.

A problem from the gripper domain will serve as a running example through the chapter. In gripper, there are balls, rooms and robots with arms. The balls and the robot are always in exactly one room. A robot can pick up and drop balls with each arm, as well as move (with the balls if carrying any) to another room.

2.1 Planning Notation

The definitions in this chapter were heavily influenced by Rintanen [15] and Helmert [9]. While we did not simply copy them, because of the nature of the topic and the thoroughness of their definitions, we often ended up using them or very similar ones, simply because those are the best ways to phrase it that we are aware of.

Definition 1 *Types[15]*
Let O be a set of objects. Let there be a finite set T of types. To each type $t \in T$ a non-empty set $D(t) \subseteq O$ of objects is associated by the domain function $D : T \rightarrow O$.

For the gripper example, let

$$\begin{aligned} O =& \{ball_1, ball_2, ball_3, ball_4, left, right, room_A, room_B, room_C\} \\ T =& \{ball, gripper, room\} \\ D(ball) =& \{ball_1, ball_2, ball_3, ball_4\} \\ D(gripper) =& \{left, right\} \\ D(room) =& \{room_A, room_B, room_C\} \end{aligned}$$

Definition 2 *Schematic Variables[15]*
Let T be a set of types. A schematic variable v has a type $\tau_{var}(v) \in T$.

There can be an arbitrary, not fixed number of schematic variables with any name. For example we could have the schematic variables x, z and *hand* with the types $\tau_{var}(x) = \mathit{ball}$, $\tau_{var}(z) = \mathit{ball}$ and $\tau_{var}(\mathit{hand}) = \mathit{gripper}$.

Definition 3 *Predicates[15]*
Let T be a set of types. A predicate p has arity $ar(p) \in \mathbb{N}$ and an associated type $\tau_{pre}(p) \in T^{ar(p)}$, the latter given by the typing function $\tau_{pre}(p)$.

The following four predicates appear in the gripper example:

$$\begin{aligned} P_{example} =& \{\mathit{at\text{-}robby}, \mathit{at}, \mathit{carry}, \mathit{free}\} & & \\ ar(\mathit{at\text{-}robby}) =& 1 & \tau_{pre}(\mathit{at\text{-}robby}) &= \langle \mathit{room} \rangle \\ ar(\mathit{at}) =& 2 & \tau_{pre}(\mathit{at}) &= \langle \mathit{ball}, \mathit{room} \rangle \\ ar(\mathit{carry}) =& 2 & \tau_{pre}(\mathit{carry}) &= \langle \mathit{ball}, \mathit{gripper} \rangle \\ ar(\mathit{free}) =& 1 & \tau_{pre}(\mathit{free}) &= \langle \mathit{room} \rangle \end{aligned}$$

Definition 4 *Schematic Atoms and Literals[15]*
Let O be a set of objects, D a domain function associating to subsets of O, V a set of variables, $p \in P$ be a predicate of arity $n = ar(p)$ and with type $\tau_{pre}(p) = (t_1, ..., t_n)$. Then schematic atoms (facts) are of the form $p(q_1, ..., q_n)$ where each q_i is either an object $o \in D(t_i)$ or a variable $v \in V$ with $\tau_{var}(v) = t_i$.

The set $gnd(P, \tau_{pre}, D)$ of ground atoms consists of all $p(o_1, ..., o_n)$ such that $p \in P$, $ar(p) = n$, and $o_i \in D(t_i)$ where $\tau_{pre} = (t_1, ..., t_n)$. Similarly, with a typing function $\tau_{var}(v)$ for $v \in V$, the set $Atoms(P, V, \tau_{var}(v), \tau_{pre}, D)$ is the set of all atoms over P, V and O.

Let x be an atom. Then x and $\neg x$ are literals to x.

Both x and *hand* are schematic variables with $\tau_{var}(x) = \mathit{ball}$ and $\tau_{var}(\mathit{hand}) = \mathit{gripper}$. Meanwhile *left* and room_A are objects. These are examples for schematic atoms using predicates from the previous example:

- $\mathit{at\text{-}robby}(\mathit{room}_A)$ (which is also a ground atom)
- $\mathit{carry}(x, \mathit{left})$
- $\mathit{free}(\mathit{hand})$

The complete set of ground atom for our gripper example is

$$\begin{aligned}
gnd(P_{example}, \tau_{pre}, D) = \{&\textit{at-robby}(room_A), \textit{at-robby}(room_B), \textit{at-robby}(room_C),\\
&at(ball_1, room_A), at(ball_1, room_B), at(ball_1, room_C),\\
&at(ball_2, room_A), at(ball_2, room_B), at(ball_2, room_C),\\
&at(ball_3, room_A), at(ball_3, room_B), at(ball_3, room_C),\\
&at(ball_4, room_A), at(ball_4, room_B), at(ball_4, room_C),\\
&carry(ball_1, left), carry(ball_1, right), carry(ball_2, left),\\
&carry(ball_2, right), carry(ball_3, left), carry(ball_3, right),\\
&carry(ball_4, left), carry(ball_4, right),\\
&free(left), free(right)\}
\end{aligned}$$

While the literal $\textit{at-robby}(room_B)$ represents that the robot is in the room called $room_B$, the literal $\neg\textit{at-robby}(room_B)$ represents that it is not there.

Definition 5 *State[15]*
Let P be a set of predicates with type $\tau_{pre}(p)$ for $p \in P$ and D a domain function. Then a state s is a mapping from $gnd(P, \tau_{pre}, D)$ to $\{0, 1\}$, indicating the truth-value of every atom.

A state can only be ground. An example for a state is s_0[1], where all balls and the robot are in $room_A$:

$$\begin{aligned}
s_0(o_{true}) = 1 \text{ for } o_{true} \in &\{\ \textit{at-robby}(room_A),\ free(left), free(right)\}\\
&\cup \{at(x, room_A) \mid x \in \{ball_1, ball_2, ball_3, ball_4\}\}\\
s_0(o_{false}) = 0 \text{ for } o_{false} \in &\{\textit{at-robby}(room_B),\ \textit{at-robby}(room_C)\}\\
&\cup \{at(x, y) \mid x \in \{ball_1, ball_2, ball_3, ball_4\}\\
&\qquad \textit{and } y \in \{room_B, room_C\}\}\\
&\cup \{carry(x, z) \mid x\ \in \{ball_1, ball_2, ball_3, ball_4\}\\
&\qquad \textit{and } z \in \{left, right\}\}
\end{aligned}$$

Listing the assignment to each atom can be quite exhaustive. Therefore it is helpful to introduce a shorthand for states, where only the positive[2] atoms are mentioned, in form of a set:

$$\begin{aligned}
s_0 = \langle &\textit{at-robby}(room_A), free(left), free(right), at(ball_1, room_A),\\
&at(ball_2, room_A), at(ball_3, room_A), at(ball_4, room_A)\rangle
\end{aligned}$$

Definition 6 *Schematic Formulas[15]*
Let O be a set of objects, V a set of variables and P a set of predicates with arities $ar(p)$ for every $p \in P$. The following are schematic formulas over O, P and V:

[1] This state will be useful later

[2] Only listing all negative atoms would also be possible.

1. schematic atoms $p(s_1, ..., s_n)$ over O and V

2. $\phi_1 \wedge \phi_2$, if ϕ_1 and ϕ_2 are schematic formulas

3. $\phi_1 \vee \phi_2$, if ϕ_1 and ϕ_2 are schematic formulas

4. $\neg\phi$, if ϕ is a schematic formula

A schematic formula that is a schematic formula using only definitions 1 and 4 is called an atomic schematic formula. A schematic formula where $V = \emptyset$ is called a ground formula.

Using the atoms $at(ball_1, room_A)$, $at(ball_2, room_C)$ and $free(hand)$, a few examples for schematic formulas are:

- $at(ball_1, room_A)$
- $at(ball_2, room_C) \vee free(hand)$
- $\neg at(ball_1, room_A) \wedge free(hand)$

Definition 7 *Schematic Effect [15]*
Let O be a set of objects, D a domain function associating to O, V a set of variables, and P a set of predicates with arity $ar(p)$ for every $p \in P$. A literal e to an atom $atom(e) \in Atoms(P, V, \tau_{var}(v), \tau_{pre}, D)$ is a schematic effect over $Atoms(P, V, \tau_{var}(v), \tau_{pre}, D)$.

If e is an effect over $Atoms(P, V, \tau_{var}(v), \tau_{pre}, D)$ with $V = \emptyset$, then e is a ground effect.

Definition 8 *Schematic Action[9]*
Let O be a set of objects, V a set of variables, and P a set of predicates with arities $ar(p)$ for every $p \in P$. Let then $Atoms(P, V, \tau_{var}(v), \tau_{pre}, D)$ be the set of all atoms over P, O and V. A schematic action is a 3-tuple $\langle \chi, e, c\rangle$ with χ a schematic formula over O and V called the precondition, e an schematic effect over $Atoms(P, V, \tau_{var}(v), \tau_{pre}, D)$ and the cost $c \in \mathbb{N}$.

If both χ and e are ground, then the action is a ground action. The set of add effects $add(a)$ of an action $a = \langle \chi, e, c\rangle$ is defined as $add(a) = \{e_i \mid e_i \in e \text{ and } e_i \text{ is a positive literal}\}$. Similarly, the set of delete effects to a is the set of positive literals $del(a) = \{\neg e_i \mid e_i \in e \text{ and } e_i \text{ is a negative literal}\}$. Further we define $pre(a) = \chi$.

Here are a few example for schematic actions for the gripper example using the objects and types from former examples and the variables b, g with $\tau_{var}(b) = ball$ and $\tau_{var}(g) = gripper$ (Note that a_1 is also a ground action):

$$\begin{aligned}
a_1 =& \langle at\text{-}robby(room_A), \{\neg at\text{-}robby(room_A), at\text{-}robby(room_B)\}, 1\rangle \\
a_2 =& \langle at\text{-}robby(room_B) \wedge at(ball_3, room_B) \wedge free(right), \\
& \{\neg at(ball_3, room_B), \neg free(right), carry(b, g)\}, 1\rangle \\
a_3 =& \langle at\text{-}robby(room_B) \wedge carry(b, left), \\
& \{\neg carry(ball_2, left), at(ball_2, room_B), free(left)\}, 1\rangle
\end{aligned}$$

$$\text{del}(a_3) = \{carry(ball_2, left)\}$$

Definition 9 *Schematic STRIPS Task [9]*
A schematic STRIPS task is a 7-tuple $\Pi = \langle T, O, D, P, s_0, \chi_*, A \rangle$ *with the following components:*

- T *is a finite set of Types*
- O *is a finite set of objects*
- $D : T \to O$ *is a domain function*
- P *is a finite set of predicates*
- s_0 *is a state over* O *and* P *called the initial state*
- χ_* *is a ground formula over* O *and* P *called the goal formula*
- A *is a finite set of schematic actions over* O *and* P

A schematic STRIPS task is a ground STRIPS task if A contains only ground actions. We will also refer to STRIPS tasks as propositional (planning) tasks.

In former examples we already defined T, O, D, P and s_0 for the gripper example. The goal formula is $\chi_* = \mathit{at}(\mathit{ball}_1, \mathit{room}_B) \wedge \mathit{at}(\mathit{ball}_2, \mathit{room}_B) \wedge \mathit{at}(\mathit{ball}_3, \mathit{room}_B) \wedge \mathit{at}(\mathit{ball}_4, \mathit{room}_B)$. The set of ground actions A is best understood as the union of three types of actions:

$$\begin{aligned}
A_{move} =& \{\langle \mathit{at\text{-}robby}(r), \{\neg \mathit{at\text{-}robby}(r), \mathit{at\text{-}robby}(r')\}, 1\rangle \\
& \mid \langle r, r'\rangle \in D(\mathit{room}) \times D(\mathit{room}) \text{ and } r \neq r'\} \\
A_{pick\text{-}up} =& \{\langle \mathit{at\text{-}robby}(r) \wedge \mathit{at}(b, r) \wedge \mathit{free}(g), \{\neg \mathit{at}(b, r), \neg \mathit{free}(g), \mathit{carry}(b, g)\}, 1\rangle \\
& \mid r \in D(\mathit{room}) \text{ and } b \in D(\mathit{ball}) \text{ and } g \in D(\mathit{gripper})\} \\
A_{drop} =& \{\langle \mathit{at\text{-}robby}(r) \wedge \mathit{carry}(b, g), \{\neg \mathit{carry}(b, g), \mathit{at}(b, r), \mathit{free}(g)\}, 1\rangle \\
& \mid r \in D(\mathit{room}) \text{ and } b \in D(\mathit{ball}) \text{ and } g \in D(\mathit{gripper})\} \\
A =& \ A_{move} \cup A_{pick\text{-}up} \cup A_{drop}
\end{aligned}$$

For brevity we introduce the following short hands:

$$\begin{aligned}
\mathit{move}(r_J, r_k) &\text{ for actions from the set } A_{move} \\
\mathit{pick\text{-}up}(r, b, g) &\text{ for actions from the set } A_{pick\text{-}up} \\
\mathit{drop}(r, b, g) &\text{ for actions from the set } A_{drop}
\end{aligned}$$

For example $\mathit{move}(\mathit{room}_A, \mathit{room}_B)$ represents the action:

$$\langle \mathit{at\text{-}robby}(\mathit{room}_A), \{\neg \mathit{at\text{-}robby}(\mathit{room}_A), \mathit{at\text{-}robby}(\mathit{room}_B)\}, 1\rangle$$

In the following we use the general term FDR[9]. However, considering that we restricted ourselves to STRIPS, these definitions represent and only hold for SAS$^+$[2].

Definition 10 *Ground FDR Task[9]*
Let T be a set of types, O a set of objects, $D : T \rightarrow O$ a domain function and P a set of predicates.

An FDR atom domain function d is a mapping from FDR variables to a set of values (its domain). An FDR state s maps variables to values in their domain. We define these domains as subsets of the union of literals to $gnd(P, \tau_{pre}, D)$ and $\{none\text{-}of\text{-}those\}$.

An FDR formula is a schematic formula where, instead of a schematic atom, an assignment atom $\mapsto$ label is an (atomic) FDR formula. The remaining recursive definitions are the same as with schematic formulas. An atomic FDR formula atom $\mapsto$ label holds in a state, if that state maps the atom to the value label.

An FDR effect is an atomic FDR state, meaning it only maps one atom one of the values in its domain. We write them short as just the assignment $e = atom_i \mapsto d(atom_i)$. FDR actions are schematic actions where instead of schematic effects there are FDR effects and instead of a schematic formula there is an FDR formula as precondition.

Then $\Pi = \langle T, O, D, P, s_i, \chi_*, d, A\rangle$ is a ground FDR planning task with d an FDR atom domain function, s_i a FDR state, χ_* an FDR formula and A a set of FDR actions. We only consider ground FDR tasks in this thesis.

Each propositional ground task can be expressed in an FDR task where $d(atom) = \{atom, \neg atom\}$ for all atoms over O and P.

Definition 11 *Successor State*
Let s be a state and $a = \langle \chi, e, c\rangle$ a ground action. The action a is applicable in s if $s \models \chi$. Then the successor state to s when applying action a is s'. The state s' is created from s by replacing all mappings of variables that appear in an effect of a to the value that the effect indicates. This can be written as $s \xrightarrow{a} s'$.

Let a successor state to the initial state in the gripper example be given by $s_0 \xrightarrow{a_0} s'$. Furthermore, let a_0 be defined as $a_0 = move(room_A, room_B)$. Then the successor state looks as follows:

$$s' = \langle at\text{-}robby(room_B), free(left), free(right), at(ball_1, room_A),\\ at(ball_2, room_A), at(ball_3, room_A), at(ball_4, room_A)\rangle$$

Definition 12 *Path, Solution and optimal Solution*
Let Π be a propositional planning task $\Pi = \langle T, O, D, P, s_0, \chi_*, A\rangle$ or an FDR planning task $\Pi = \langle T, O, D, P, s_0, \chi_*, d, A\rangle$ and S the set of all states over D and P. A path for Π is a sequence of actions $\pi = \langle a_1, ...a_m\rangle$ with $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_m} s_m$ with $a_i \in A$ and $s_i \in S$ (as well as $s \in S$). If $s = s_0$ and $s_m \models \chi_*$ then the path is a solution (or plan) to Π. The cost of a plan with $a_i = \langle \chi_i, e_i, c_i\rangle$ is:

$$c(\pi) = \sum_{i \in \{1,...,m\}} c_i$$

An optimal solution π_* for Π is a solution with $c(\pi_*) \leq c(\pi_i)$ for all possible solutions π_i to Π.

An optimal solution to our example planning task Π would be the action sequence

$$\begin{aligned}\pi_* = \langle & \textit{pick-up}(\textit{room}_A, \textit{ball}_1, \textit{left}), \textit{pick-up}(\textit{room}_A, \textit{ball}_2, \textit{right}),\\ & \textit{move}(\textit{room}_A, \textit{room}_B), \textit{drop}(\textit{room}_B, \textit{ball}_1, \textit{left}), \textit{drop}(\textit{room}_B, \textit{ball}_2, \textit{right}),\\ & \textit{move}(\textit{room}_B, \textit{room}_A),\\ & \textit{pick-up}(\textit{room}_A, \textit{ball}_3, \textit{left}), \textit{pick-up}(\textit{room}_A, \textit{ball}_4, \textit{right}),\\ & \textit{move}(\textit{room}_A, \textit{room}_B), \textit{drop}(\textit{room}_B, \textit{ball}_3, \textit{left}), \textit{drop}(\textit{room}_B, \textit{ball}_4, \textit{right})\rangle\end{aligned}$$

The cost of π_* is $c(\pi_*) = 11$

Definition 13 *Reachable State*
Let Π be a propositional planning task $\Pi = \langle T, O, D, P, s_0, \chi_*, A\rangle$ or an FDR planning task $\Pi = \langle T, O, D, P, s_0, \chi_*, d, A\rangle$ and s_* a state over D and P. If there exists a plan $\pi = \langle a_1, ..., a_m\rangle$ so that $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} ... \xrightarrow{a_m} s_*$, then the state s_* is reachable (from the initial state).

Definition 14 *Heuristic*
Let S be a set of all states to a propositional task or FDR states to an FDR task. A heuristic h is a function $h : S \to \mathbb{N}$.

Definition 15 *Regression of a propositional State [14]*
Let $a = \langle \chi, e, c\rangle$ be an action and s a state to a propositional planning task Π. The formula φ to a state is the conjunction of literals where all ground atoms appear in exactly one literal with negation determined by the mapping of the state. Then the regression of s through a, written $rg_a(s)$, is the following formula:
$rg_a(s = \chi \wedge \psi$ with $\psi = \bigwedge \psi_i$ where $\psi_i \in \varphi$ and $\psi_i \notin e$

2.2 Invariants

Definition 16 *Invariant[9]*
Let $\Pi = \langle T, O, D, P, s_i, \chi_*, A\rangle$ be a schematic propositional planning task with atoms $Atoms(P, V, \tau_{var}(v), \tau_{pre}, D)$ and states S over O and V. An invariant for Π is a schematic formula ϕ over literals over O and V such that $s \models \phi$ for all states reachable from the intial state. Note that it is also possible that both the invariant and Π are ground.

In the gripper example, the following formulas are invariants:

$$\begin{aligned}\phi_1 =& \textit{at-robby}(\textit{room}_A) \vee \textit{at-robby}(\textit{room}_B)\\ \phi_2 =& \neg \textit{free}(\textit{left}) \vee \neg \textit{carry}(\textit{ball}_1, \textit{left}) \vee \neg \textit{carry}(\textit{ball}_2, \textit{left})\\ & \vee \neg \textit{carry}(\textit{ball}_3, \textit{left}) \vee \neg \textit{carry}(\textit{ball}_4, \textit{left})\\ \phi_3 =& \neg \textit{carry}(\textit{ball}_4, \textit{left}) \vee \neg \textit{carry}(\textit{ball}_4, \textit{right})\end{aligned}$$

Definition 17 *Mutex and Mutex Group[9]*
A mutex is a special case of an invariant. An invariant is a mutex if it is a disjunction of exactly 2 negative literals.

Let Γ be the set of all invariants to a propositional task Π. The set of literals M is a mutex group if there is a set of mutexes $\Gamma_M \subseteq \Gamma$ so that $\neg\phi \wedge \neg\phi' \in \Gamma$ for all $\phi, \phi' \in M$ with $\phi \neq \phi'$.

An example for a mutex is ϕ_3. A mutex group for a propositional planning task denotes a set of atoms of which there is at most one occurring as a positive literal in any reachable state.

Mutexe groups are helpful, as they allow the transformation from a STRIPS task to a *SAS*+ task. As only ever one atom of a mutex group can be true in any reachable state, they can be represented by one SAS+ variable with the domain equal to all the atoms in the mutex, possibly combined with a default "none" value.

Definition 18 *Reachability Invariant Candidates (RIC)[15]*
Let O be a set of objects and V be a set of variables with $D(\tau_{var}(v)) \subseteq O$ for all $v \in V$. Reachability invariant candidates (RIC) are tuples $\gamma = \langle\psi, \phi\rangle$ where ϕ is a disjunction of literals and ψ is a (possibly empty) conjunction of inequalities $x \neq x'$ where x and x' are objects or variables occurring in literals in ϕ. An RIC with ψ empty and gournd literals, is a ground reachability invariant candidate[3]*. We write the RICs $\gamma_J = \langle\psi_J, \phi_J\rangle$ with $\phi_{0,2} = l_1 \vee l_2$, $\phi_{1,3} = l_1$ and $\psi_{2,3} = \emptyset$ as follows:*

$$\begin{aligned}\gamma_0 =& \psi_0 \rightarrow (l_1 \vee l_2)\\ \gamma_1 =& \psi_1 \rightarrow l_1\\ \gamma_2 =& (l_1 \vee l_2)\\ \gamma_3 =& l_1\end{aligned}$$

The set of ground RIC to a schematic RIC $\gamma_s = \langle\psi_s, \phi_s\rangle$ is the set of RIC's Γ where all $\gamma_J = \langle\psi_J, \phi_J\rangle$ fulfill $\gamma_J \in \Gamma$, $\psi_J = \emptyset$ and when applying the mapping $\alpha : V \rightarrow O$, that is implied by ϕ_s and ϕ_J, to ψ_s, no object occurs twice in the same tuple (no inequality is violated).

Definition 19 *Monotonicity Invariant Candidates (MIC)[9]*
A monotonicity invariant candidate (MIC) for a task Π is a pair $\theta = \langle H, \phi\rangle$ where H is a set of variables and ϕ is a set of literals. Variables $h_i \notin H$ in the atom $p(q_1, ..., q_m)$ from the literal $l_J \in \phi$ are called counted variables. We write an MIC $\langle\{v_0, v_1, ..., v_m\}, \{l_0, l_1, ..., l_k\}\rangle$ as follows:

$$\forall v_0 v_1 ... v_m (l_0 + l_1 + ... + l_k) \downarrow$$

The weight $w = weight(\theta, s)$ of a ground MIC $\langle H, \phi\rangle$ in a state s is the sum over ground atoms $a_i \in \phi$ that are true in s.

[3]For example $a \vee b$ with $a, b \in O$

A MIC $\theta = \langle H, \phi \rangle$ with weight(θ, s_0)=1 that is proven invariant represents a mutex group that can be constructed, in form of a set of atoms, using the following procedure:

$$\textit{for each } q_i \in \textit{counted-variables}(\phi):$$
$$\phi = \{p(q_0, ..., o, ..., q_m) \mid o \in \tau_{var}(q_i)\} \cup \phi \setminus \{p(q_0, ..., q_i, ..., q_m)\}$$

An example serves to explain the case where two counted variables occur in a MIC: Let p be a predicate and a, b, c, d variables. Let the MIC $\theta = \forall ab(p(a, b, c, d)) \downarrow$ be proven invariant for the propositional planning task Π. Then θ using the domain function D from Π represents the mutex group M:

$$D(\tau_{pre}(a)) = \{1, 2\} \qquad D(\tau_{pre}(c)) = \{5, 6\}$$
$$D(\tau_{pre}(b)) = \{3, 4\} \qquad D(\tau_{pre}(d)) = \{7, 8\}$$
$$M = \{p(a, b, 5, 7), p(a, b, 5, 8), p(a, b, 6, 7), p(a, b, 6, 8)\}$$

A way to generate invariants is with a generate-test-repair algorithm[9], invariant candidates could be RICs, MICs or others:

- 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 a invariant. Usually done inductively:
  - Test that initial state satisfies ϕ.
  - Test that if ϕ is true in the current state, it remains true after applying a sinlge action.
- Repair: If invariant test fails, replace invariant candidate ϕ by a weaker[4] formula, ideally exploiting why the proof failed.

2.3 Ground Iterative Reachability Invariant Synthesis (G-IRIS)

Rintanen [14] suggested an invariant synthesis method based on the use of the regression of actions for the check part in a guess-check-repair approach for propositional ground planning tasks. We will call this method Ground Iterative Reachability Invariant Synsthesis (G-IRIS).

Let $\Pi = \langle T, O, D, P, s_0, \chi_*, A \rangle$ be a propositional ground planning task and $G = gnd(P, \tau_{pre}, D)$ the set of all atoms in Π. The algorithm is presented in

[4]It is also possible to construct a generate-test-repair algorithm for invariants where the formulas are made stronger in the repair step.

Algorithm 11. The goal of the algorithm is to find the set of strongest ground RICs up to the size n to Π.

Algorithm 1: Algorithm for Invariant computation by Rintanen [14]. The function *lits*(c) returns the number of atoms in the formula.

```
Input: A finite set of ground atoms $G$, the initial state $s_i$ a finite set of
       grounded actions $A$ and $n \in \mathbb{N}$.
Output: $\Gamma$ (set of ground RIC proven invariant)
1 function invariants($G, s_0, A, n$):
2     $\Gamma := \{g \in G \mid s_0 \models g\} \cup \{\neg g \mid g \in G; s_0 \not\models g\}$
3     while $\Gamma \neq \Gamma'$ do
4         $\Gamma' := \Gamma$
5         foreach $a \in A$ and $\gamma \in \Gamma$ s.t. $\Gamma' \cup \{rg_a(\neg\gamma)\} \in SAT$ do
6             $\Gamma := \Gamma \backslash \{\gamma\}$
7             if $|\texttt{lits}\ (\gamma)| < n$ then
8                 $\Gamma := \Gamma \cup \{\gamma \vee g \mid g \in G\} \cup \{\gamma \vee \neg g \mid g \in G\}$
9     return $\Gamma$
```

On line 1 we create the RICs from the literals in the initial state. Then there is a search for a fix point. This fix point is reached when the set of RICs has not changed after an iteration. In the iteration, it checks which of the candidates can be falsified by testing satisfiability of the current set of invariant candidates with the regression of the negation of that candidate. If the candidate can be falsified, then it is discarded. If a candidate is discarded this way and not yet weaker than we want to consider (consisting of a disjunction of more than n atoms, usually $n = 2$ or $n = 3$), the candidate is weakened. The weakening on line 7 builds the set of all disjunctions with the discarded candidate and every atomic literal over G.

For the satisfiability test, it is possible to use an incomplete test which does not falsely indicate unsatisfiable for a set of satisfiable formulas. For STRIPS the test can be done using unit resolution.

In the following, we will show some of the interesting steps on the running example. Let us first recap the initial state:

$$s_0 = \langle \textit{at-robby}(\textit{room}_A), \textit{free}(\textit{left}), \textit{free}(\textit{right}), \textit{at}(\textit{ball}_1, \textit{room}_A),\\ \textit{at}(\textit{ball}_2, \textit{room}_A), \textit{at}(\textit{ball}_3, \textit{room}_A), \textit{at}(\textit{ball}_4, \textit{room}_A)\rangle$$

The initial set of candidates C_0 contains the following RICs:

$$\begin{aligned} C_0 = \{&\textit{at-robby}(\textit{room}_A), \neg\textit{at-robby}(\textit{room}_B), \neg\textit{at-robby}(\textit{room}_C), \textit{free}(\textit{left}),\\ &\textit{free}(\textit{right})\}\\ &\cup \{\textit{at}(\textit{ball'}, \textit{room}_A) \mid \textit{ball}' \in D(\textit{ball})\}\\ &\cup \{\neg\textit{at}(\textit{ball}', \textit{room}') \mid \textit{ball}' \in D(\textit{ball}) \text{ and } \textit{room}' \in D(\textit{room})\}\\ &\cup \{\neg\textit{carry}(\textit{ball}', \textit{gripper}') \mid \textit{ball}' \in D(\textit{ball}) \text{ and } \textit{gripper}' \in D(\textit{gripper})\}\end{aligned}$$

Let the following c be the first candidate and a the first action considered:

$$c = \mathit{at\text{-}robby}(\mathit{room}_A)$$
$$a = \langle \mathit{at\text{-}robby}(\mathit{room}_A), \{\neg \mathit{at\text{-}robby}(\mathit{room}_A), \mathit{at\text{-}robby}(\mathit{room}_B)\}, 1\rangle$$

The regression of $\neg c$ with a is then $\mathit{at\text{-}robby}(\mathit{room}_A)$. As this is again c which we know is in C', we also know $\{c\} \cup C'$ is satisfiable. Therefore, we remove c from C and attempt to weaken it. Because c has fewer than $n = 2$ literals, we may weaken it. The weakening set contains all disjunctions of the form $c \vee l$ where l is a literal to $gnd(P_{example}, \tau_{pre}, D$. The candidate $\mathit{at\text{-}robby}(\mathit{room}_A) \vee \mathit{at\text{-}robby}(\mathit{room}_A)$ could be discarded, because it simplifies to c which we just discarded. Meanwhile, the candidate $\mathit{at\text{-}robby}(\mathit{room}_A) \vee \neg \mathit{at\text{-}robby}(\mathit{room}_A)$ is actually a bit more interesting. Of course, it is a trivial candidate, but its presents contains at least a small bit of information. Namely, that neither $\mathit{at\text{-}robby}(\mathit{room}_A)$ nor $\neg \mathit{at\text{-}robby}(\mathit{room}_A)$ are invariants for the planning task.

A key insight for understanding the algorithm is, that the ground RIC $\gamma_1 = \neg \mathit{at}(\mathit{ball}_1, \mathit{room}_B)$ can only be discarded in the second iteration. Because in the first, the satisfiable test runs against the initial C and there we have $\neg \mathit{carry}(\mathit{ball}_1, \mathit{left})$ as well as $\neg \mathit{carry}(\mathit{ball}_1, \mathit{right})$. While we find out during the first iteration, that these two are indeed not invariants, we cannot discard γ_1.

After reaching the fix point set, there are two kinds of invariants for our example. First 25 of the form $x \vee \neg x$. Secondly, there are 63 mutexes which can be split in six sets:

$$\begin{aligned}
\Gamma_1 =& \{\neg \mathit{free}(g) \vee \neg \mathit{carry}(b, g) \mid b, g \in D(\mathit{ball}) \times D(\mathit{gripper})\}(\text{size } 8)\\
\Gamma_2 =& \{\neg \mathit{carry}(b, g) \vee \neg \mathit{carry}(b', g)\\
&\quad \mid b, b' \in D(\mathit{ball}) \times D(\mathit{ball}) \text{ and } b \neq b' \text{ and } g \in D(\mathit{gripper})\}(\text{size } 12)\\
\Gamma_3 =& \{\neg \mathit{carry}(b, g) \vee \neg \mathit{carry}(b, g')\\
&\quad \mid b \in D(\mathit{ball}) \text{ and } g, g' \in D(\mathit{gripper}) \times D(\mathit{gripper}) \text{ and } g \neq g'\}(\text{size } 4)\\
\Gamma_4 =& \{\neg \mathit{at\text{-}robby}(r) \vee \neg \mathit{at\text{-}robby}(r')\\
&\quad \mid r, r' \in D(\mathit{room}) \times D(\mathit{room}) \text{ and } r \neq r'\}(\text{size } 3)\\
\Gamma_5 =& \{\neg \mathit{at}(b, r) \vee \neg \mathit{at}(b, r')\\
&\quad \mid b \in D(\mathit{ball}) \text{ and } r, r' \in D(\mathit{room}) \times D(\mathit{room})\}(\text{size } 12)\\
\Gamma_6 =& \{\neg \mathit{at}(b, r) \vee \neg \mathit{carry}(b, g)\\
&\quad \mid b \in D(\mathit{ball}) \text{ and } r \in D(\mathit{room}) \text{ and } g \in D(\mathit{gripper})\}(\text{size } 24)
\end{aligned}$$

Our example run is performed with $n = 2$. For $n = 3$, the algorithm would weaken candidates further and also find for example this invariant:

$$\mathit{at\text{-}robby}(\text{room}_A) \vee \mathit{at\text{-}robby}(\text{room}_B) \vee \mathit{at\text{-}robby}(\text{room}_C)$$

2.4 Schematic Monotonicity Invariant Synthesis (S-MIS)

Helmerts porposed an algorithm[9] for proving that an MIC is invariant. This can be proven by examining the schematic actions of the planning task.

Definition 20 *Threatened Monotonicity Invariant Candidates [9]*
A monotonicity invariant canditate θ is threatened by a schematic action a iff one of the following two contitions holds:

- *The schematic action a has an add effect that can increase the weight of an instance of θ in some state, but no delete effect that is guaranteed to decrease the weight of the same instance in the same state. In this case, we say that the schematic action a is* ***unbalanced*** *with regard to θ.*

- *When ignoring delete effects, the schematic action a can increase the weight of some instance of θ in some stat****e by at least 2. In this case, we say that the schematic action*** *a* ***is*** *too heavy for θ.*

If a is unbalanced with regards to the MIC θ, meaning it has more add than delete effects corresponding to θ, then a refinement is attempted. This refinement works by finding an atom which gets deleted by a, which when added to the θ results in a MIC for which a is balanced.

Invariant candidates are discarded if there is a schematic action that is too heavy for them. This essentially means that the schematic action has more than one add effect corresponding to the candidate. This avoids non-trivial checks for cases where a schematic action could be balanced if predicates are over different objects, but is not so if they are over the same object.

The schematic monotonicity invariant synthesis S-MIS uses a guided guess-check-repair approach which is depicted in Algorithm 2. The function prove-invariant is applied to every MIC which are hold in a list. All refined MICs are added to this list. The initial list consist of all MICs $\theta = \langle H, \phi \rangle$ with ϕ a formula over exactly one atom and at most one counted variable.

Algorithm 2: Algorithm for proving that an MIC $\langle H, \phi \rangle$ is an invariant and possibly refining it if it is rejected.

```
Input: A finite set of variables $H$ and a set of predicates $\Phi$ over those
       variables.
Output: Boolean describing if $(H, \Phi)$ is an invariant.
function prove-invariant($H, \phi$):
    foreach schematic action $a$ do
        if is-action-too-heavy($a, H, \phi$) then
            return false {Reject the candidate.}
        foreach effect $e$ of add($a$) that affects a predicate in $\phi$ do
            if is-add-effect-unbalanced($a, e, H, \phi$) then
                return false {Reject the candidate and try to refine it.}
        return true {Accept the candidate.}
```

The order in Algorithm 2 is important, as we want to reject every MIC for which there exists a too heavy action[5]. Helmert explains that while this is technically stricter than necessary, it is suitable for the computation of mutex groups which will be used for the construction of a finite domain representation.

[5]No refinement could lower the heaviness of the action towards the new candidate.

We could imagine an action that has weight 2 and does not violate an MIC. However, this would mean that when we translate this action using the mutex group derived from the MIC, the action could have two different effects for the same variable. This is not allowed because a successor state cannot map the same variable to two values at once. The heaviness test is described in Algorithm 3.

Algorithm 3: Function determining if the action is to heavy with regard to the MIC $\langle H, \phi \rangle$.

```
1 function is-action-too-heavy($a, H, \phi$):
2     Let $a'$ be a copy of $a$.
3     Duplicate all (non-trivially) quantified effects of $a'$.
4     Assign unique names to all quantified variables in effects of $a'$.
5     foreach pair($e, e'$) of add effects of $a'$ that affect a predicate in $\phi$ do
6         if the parameters of action $a'$ can be renamed so that ($atom(e) \neq$
            $atom(e')$ and covers($H, \phi, atom(e)$) and covers($H, \phi, atom(e')$)
            and $pre(a') \wedge \neg atom(e) \wedge \neg atom(e')$ is satisfiable) then
7             return true {The action is to heavy.}
8     return false {The action is not to heavy.}
```

The heaviness test informally described checks if two different effects appear in the sum of the MIC and whether the action can actually cause both effects at once. The function covers is defined in Algorithm 4 and needs to be understood in the context where it is used. Note how there is a renaming of the parameters of a' before the call to covers. Inside covers, the counted variables are renamed. So at that point we renamed all variables in order to it at all possible enable $\varphi = \psi$. The function covers, again informally phrased, tells if an atom appears in the sum of the MIC.

Algorithm 4: covers

```
  Input: A finite set of variables $H$, a set of predicates $\Phi$ over those
         variables and a formula $\psi$
  Output: Boolean describing whether $\phi$ covers $\psi$.
1 function covers ($H, \phi, \psi$):
2     foreach $\varphi \in \phi$ do
3         if the counted variables in $\varphi$ (those not in $H$) can be renamed
            so that $\varphi = \psi$ then
4             return false
5     return false
```

Finally, in the unbalanced check in Algorithm 5 we look for a delete effect who's variables can be renamed so that it appears in the MIC and which can be applied in a state where it is not already deleted. The renaming of the quantified variables for covers is this time done for the effect.

Algorithm 5: Function determining if the action is unbalanced for the MIC $\langle H, \phi \rangle$ in regard of the specific add effect.

```
1 function is-add-effect-unbalanced($a, e, H, \phi$):
2     Let $a'$ be a copy of $a$ where the parameters are minimally renamed
        so that $covers(H, \phi, atom(e))$ is true.
3     foreach delete effect $e'$ of $a'$ that affects a predicate in $F$ do
4         if the quantified variables of $e'$ can be renamed so that
            ($atom(e) \neq atom(e')$ and covers($H, \phi, atom(e')$) and $pre(a') \wedge$
            $\neg atom(e) \models atom(e')$) then
5             return false {e' balances e}
6     return true {the add effect is unbalanced}
```

If a candidate is rejected because it is unbalanced, we attempt to refine it with Algorithm 6.

Algorithm 6: Refinement of a candidate $\langle H, \phi \rangle$ that is unbalanced with respect to the action a and add effect e

```
1 function refine-candidate($a, e, H, \phi$):
2     Select some schematic action $a$ and add effect $e$ such that
        is-add-effect-unbalanced($a, e, H, \phi$) returns true.
3     foreach atom $\psi'$ over variables from $H$ and at most one other
        variable for which covers $(H, \phi, \psi')$ is not true do
4         $\phi' = \phi \cup \{\psi'\}$
5         Simplify $\phi'$ by removing atoms from $\phi$ that are covered by $\psi'$.
6         Simplify $\phi'$ by removing unused parameters.
7         if if not is-add-effect-unbalanced($a, e, H, \phi'$) then
8             Add $\langle H, \phi \rangle$ to the set of MIC
```

Moving to our example, the set of initial MICs is the following:

$$\begin{aligned}
\theta_1 &= \forall(\textit{at-robby}(r)) \downarrow & \theta_2 &= \forall r(\textit{at-robby}(r)) \downarrow \\
\theta_3 &= \forall r(\textit{at}(b, r)) \downarrow & \theta_4 &= \forall b(\textit{at}(b, r)) \downarrow \\
\theta_5 &= \forall b, r(\textit{at}(b, r)) \downarrow & \theta_6 &= \forall b(\textit{carry}(b, g)) \downarrow \\
\theta_7 &= \forall g(\textit{carry}(b, g)) \downarrow & \theta_8 &= \forall b, g(\textit{carry}(b, g)) \downarrow \\
\theta_9 &= \forall(\textit{free}(g)) \downarrow & \theta_{10} &= \forall g(\textit{free}(g)) \downarrow
\end{aligned}$$

The first MIC $\theta_1 = \forall(\textit{at-robby}(r)) \downarrow$ can be proven invariant. There is only one shematic action affecting the predicate *at-robby*, namely $a_{move} = \textit{move}(r_J, r_k)$. It is clearly not to heavy, as it only has one add effect. Indeed it is balanced with regard to the add effect $\textit{at-robby}(r)$, because of the delete effect $\neg\textit{at-robby}(r')$. Therefore θ_1 is proven invariant. Now the difference of what happens in both test when examining θ_2 reveals much about how they work. The test both fail at their respective "if“ condition, because it is not possible to rename the quantitative variables in θ_2 to enable both effects to be covered by the MIC while being not the same atom. Note however, that we discuss the balance test here only in theory, because the θ_2 is already discarded after being found to heavy. To show the refinement, let us look at θ_4. It is unbalanced with regards to the add effect $at(b, r)$ in $a_{drop}(r, b, g)$. The schematic action has no delete effect with predicate *at* and can therefore not balance the add effect. The only delete effect in $adrop$ is $\neg carry(b, g)$. We can create a new MIC $\theta_{11} = \forall b(at(b, r) + carry(b, g)) \downarrow$. This new candidate is not only balanced with regards to $a_{drop}(r, b, g)$, but also with regards to $a_{pick\text{-}up}(r, b, g)$. As no other schematic action affects these predicates and neither schematic action is to heavy with regards to θ_{11}, it is proven invariant.

In the end, we will find three MICs that are proven invariant:

$$\begin{aligned}\theta_1 =& \forall(at\text{-}robby(r)) \downarrow\\ \theta_{11} =& \forall b(at(b, r) + carry(b, g)) \downarrow\\ \theta_{12} =& \forall g(free(g) + carry(b, g)) \downarrow\end{aligned}$$

3 Implementation

This chapter will go into details of the implementation. There are aspects of the algorithms discussed which where not definitively described by the author. The implementation was written in Python 3. However, this thesis will only use pseudo code.

3.1 G-IRIS implemenation

Our implementation is presented in Algorithm 7. We do compute a map from literals to clauses where the literals appear negated in each outer iteration. This can be done once for every C' and can then be used for every 2-SAT test in this iteration. We also added a douplicate detection mechanism with the set canidates-conisdered. This serves to compensate the relative naive weakening which due to its definition is bound to create many douplicates.

Algorithm 7: G-IRIS proposed by Rintanen [14] as implemented for this thesis.

```
Input: $G$: a finite set of grounded atoms; $s_0$: the initial state; $A$: a finite set of ground actions; $n \in \mathbb{N}$
Output: C: set of ground RICs
1  function invariants($G, s_0, A, n$):
2      remove-trivial-actions($A$)
3      $G_{\text{affected}}$ = affected-atoms($A$)
4      $C := \{g \in G_{\text{affected}} \mid s_0 \models g\} \cup \{\neg g \mid g \in G_{\text{affected}}$ and $s_0 \not\models g\}$
5      $C_{\text{trivial}} = \{g \in G \setminus G_{\text{affected}} \mid s_0 \models g\} \cup \{\neg g \mid g \in G \setminus G_{\text{affected}}$ and $s_0 \not\models g\}$
6      while True do
7          $C' := C.copy()$
8          literal_to_clauses_where_negated(C')
9          foreach $c \in C$ do
10             candidates-considered = candidates-considered $\cup\{c\}$
11             foreach $a \in A$ do
12                 if 2-SAT($\{$STRIPS-regress$(a, \neg c)\}, C', G$) then
13                     $C := C \setminus \{c\}$
14                     if $|$lits $(c)| < n$ then
15                         weakened-c = $\{c \vee g \mid g \in G\} \cup \{c \vee \neg g \mid g \in G\}$
16                         weakened-c = weakened-c $\setminus$ candidates-considered
17                         foreach weak-c $\in$ weakened-c do
18                             if 2-SAT ($\{$STRIPS-regress$(a, \neg$weak-c$)\}, C', G$) then
19                                 $C := C \cup \{$weak-c$\}$ (extends loop at 7, but no restart)
20         if $C$ != $C'$ then
21             break
22     return $C$
```

Another optimization we added is to check all new candidates with the same action. We already know, that this action affects at least one of the disjuncts. It would not make any sense to add an action, that does not resolve the reason of the rejection, this being the action in this context.

The regression depicted in Algorithm 8 is fairly straight forward.

Algorithm 8: Regression for a conjunction in STRIPS.

```
Input: c: a Conjunction of literals (handled as set of literals); a: a
       propositional ground action
Output: The regression of c by a in the form of a conjunction of literals
function STRIPS-regress ($o, c$):
    foreach $e \in del(a)$ do
        if $e \in c$ then return $false$
        $c := c \setminus e$
    foreach $e \in add(a)$ do
        if $\neg\ e \in c$ then return $False$
        $c := c \setminus e$
    return $c$
```

Our first implementation used a incomplete unsatisfiability test by resolution for Line 12 in Algorithm 7, which would have allowed for invariants of greater size. However, the performance of resolution on the scale we observed already on a simple gripper problem was poor. As we are primarily interested in using this implementation for generating mutex groups, we can restrict the algorithm to $n = 2$ without losing any mutexes and thereby enabling us to use an algorithm for 2-SAT. The 2-SAT test in Algorithm 9 lead to a significant performance increase.

Algorithm 9: 2-SAT

```
Input: Set of disjunctions $C$, and a set of unit clauses $C'$
Output: true if $C \cup C'$ is satisfiable and false it is not
function 2-SAT ($C, c$):
    unforced-binary-disjunctions,unsolvable = unit-resolution(C,c)
    if $unsolvable$ then
        return $false$
    if $unforced\text{-}binary\text{-}disjunctions\ is\ empty$ then
        return $true$
    edges= $\{\langle \neg a, b\rangle, \langle a, \neg b\rangle \mid \langle a, b\rangle \in$ unforced-binary-disjunctions$\}$
    sccs = tarjan-get-sccs(edges)
    foreach $scc \in sccs$ do
        foreach $l \in scc$ do
            if $\neg l \in scc$ then
                return $false$
    return $true$
```

We will first talk about the second part of the 2-SAT algorithm. There we construct a graph using all literals as nodes. We add two directed edges for each disjunction in our list. This represents the implication that if one side of the disjunction is not fulfilled, the only way to fulfill it is if the other side is fulfilled. In this graph, we can now do a test for satisfiability by testing whether any literal can reach its negation as well as its negation reaching it. A way to compute this for all literals at once is by using strongly connected component (SCC).

SCCs are defined as a set of vertices, where every vertices can reach every other vertices. When we have the SCCs of a graph, we can simply test if any literal appears in a SCC together with its negation. For computing SCCs of a graph, we implemented the algorithm proposed by tarjan[18]. Our implementation closely follows the pseudo code that can be found on Wikipedia[6].

Our implementation of the unit-resolution is depicted in Algorithm 10. The idea that optimizes the computation here is, that we do not actually need to produce a resolvent to test for unsatisfiability. If we remember which literals we have looked at, we can simply count how many of the literals in each claus are not yet in conflict with what we saw up till now. As soon as there is only one literal left, we know that this one either holds or $C \cup c$ is unsatisfiable. At this point we use the mapping from literals to clauses that we computed in Algorithm 7.

Algorithm 10: unit-resolution

```
Input: A set of disjunctions with length at most 2 C, and a set of unit
       clauses c
Output: binary disjunctions that are not forced true or false if $C \wedge c$ is
        unsatisfiable by unit-resolution
function unit-resolution(C,c):
    forced-literals = $\{l \mid l \in c\}$
    literal-marked-forced = $\{l \mid l \in c\} \cup \{\text{claus} \in C \mid \text{length(claus)} = 1\}$
    if $l \in$ literal-marked-forced and $\neg l \in$ literal-marked-forced for some $l$ then
        return false
    unmarked-literal-per-claus = claus $\rightarrow$ length(claus)
    while $l \in$ forced-iterals do
        forced-iterals = forced-iterals $\setminus l$
        foreach claus $\in C$ with $\neg l \in$ claus do
            unmarked-literal-per-claus[claus] =
              unmarked-literal-per-claus[claus]$-1$
            if unmarked-literal-per-claus[claus] $= 1$ then
                forced-literal = $\{\text{lit} \mid \text{lit} \in \text{claus and lit} \notin$
                  literal-marked-forced$\}$ (at most one)
                if not forced-literal then
                    return false
                forced-literals.add( forced-literal)
                literal-marked-forced.add( forced-literal)
    foreach claus, nmb-unmarked $\in$ unmarked-literal-per-claus do
        if nmb-unmarked $> 1$ then
            if $l \in$ literal-marked-forced for each $l \in$ claus then
                binary-clauses.add(claus)
    return binary-clauses
```

[6] wikipedia.org/wiki/Tarjan's_strongly_connected_components_algorithm last accessed 11:00 4.10.2020

The unit resolution alone can however not determine if a set of disjunctive clauses, where we can not force all but one literal, is satisfiable. Therefore, this set is handled by the aforementioned testing using SCCs.

Let Γ be a set of RIC that was proven invariant using Rintanens invariant synthesis method using $n = 2$. In order to get mutex groups from this set, we first filter out all invariants that are no mutexes. Then we use a greedy clique partitioning algorithm by Rintanen [13] to get a set of disjunct mutex groups from our set of mutexes. This partitioning into cliques starts by assigning all vertices to the same clique candidate. It then iteratively identifies literals in a clique candidate that have no edge between them and creates a new clique from one of the two and every other vertices that has an edge to it. It terminates when the clique candidate is a clique, as then all the sets of vertices we have are cliques. The use of a greedy algorithm is due to the problem of finding max cliques being NP-complete[11].

Considering that we throw away everything but the mutexes, the question arises why we did not simply limite our candidates in Algorithm 7 to canidates that are mutexes. The reason is, that the fewer candidates there are in Γ', the easier it is to find an operator s.t. $\Gamma' \cup \{\text{STRIPS-regress}(\neg\gamma)\} \in SAT$, thereby discarding the candidate, even though it might be a mutex for the task.

4 Evaluation

We evaluated the implementation on 35 STRIPS domains from past IPCs[7]. There are 1119 problems in our optimal track and 1133 in our satisficing track.

4.1 Translation Results

Let us first examine how many tasks can actually be translated. The default implementation of the translator for Fast Downward sets a time limit of 5 minutes. As can be seen in Table 1, G-IRIS can only solve about 13 % of the tasks in this time. Therefore we also report the results for higher time limits for G-IRIS. However, it was unable to translate even half of the problems in time.

Table 1: instances

		opt		sat	
algorithm	limit	done	out of time	done	out of time
S-MIS	5m	1133	0	1119	0
G-IRIS	5m	151	982	140	979
G-IRIS	3h	295	838	261	858

Next we take a look at how the algorithms compare on the task that could be translated by both. As can be seen in Figure 1, S-MIS is always faster than

[7]https://github.com/aibasel/downward-benchmarks (last accessed 22.June 2020, commit: d6febbba6012806ec92214a00b7f842fe3dacd34)

G-IRIS. For all problems that are not solved within a tenth of a second by S-MIS, G-IRIS is about 2 or more magnitudes slower.

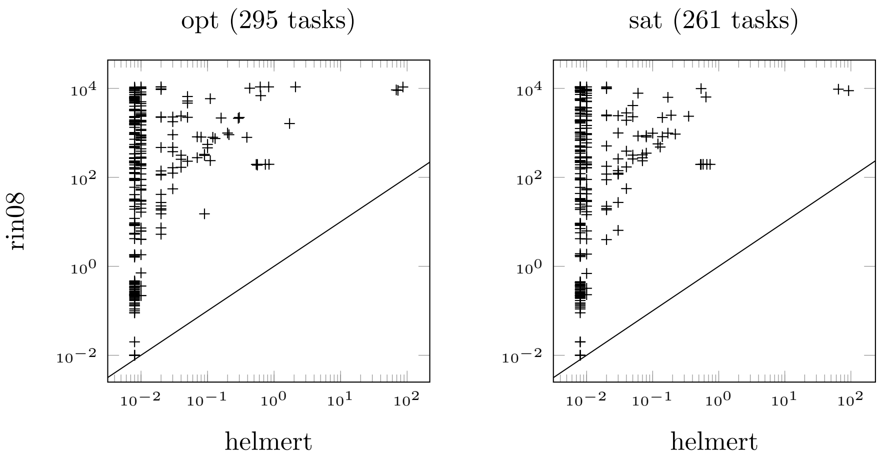

Figure 1: Time each algorithm required to find invariants to problems for optimal domains. Measurement resolution of 0.01 seconds. Defaulted 0.0 to 0.008 for logarithmic visualisation.

To find out why G-IRIS is slow, we only have to look at the number of tests that are performed. While there is some variation due to the fact, that not every test is equally difficult. However, there is a linear trend that can be seen in Figure 2. The behaviour below 0.01 can be explained with the measurement resolution which rounded everything between 0.000 and 0.010.

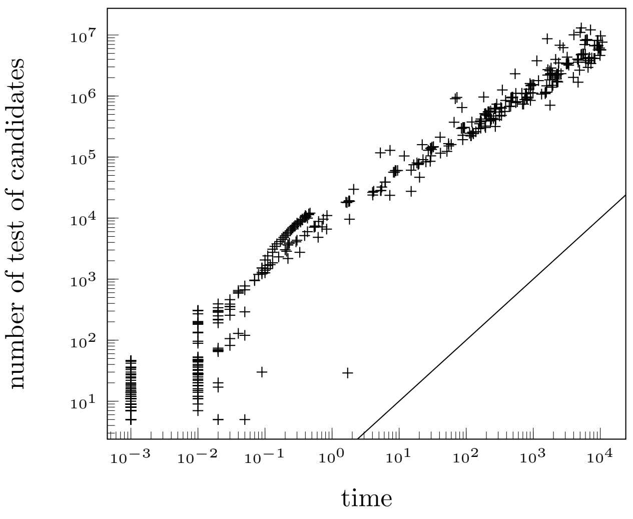

Figure 2: Time plotted against the total number of 2-SAT tests for each translated problem.

4.2 Search Results

We used A*[7] with the following heuristics: blind, hmax[6], ipdb[8], lm-cut[10] and merge-and-shrink[16] using the merge strategy SCC-DFP [17]. We run A* with those heuristics on all problems that both algorithms could translate. The memory was limited to 3.5 GB and the time to 30 min.

In Table 2 we see the number of problems from the optimal track that could be solved when translated using G-IRIS and S-MIS. We observe, that the translation from S-MIS can be solved more often. Indeed independent of the heuristic, the translation by S-MIS can be solved at least as often as the translation by G-IRIS. There are a few problems, where A* on either translation runs into time or memory limits.

solved by	blind	hmax	ipdb	lmcut	m&s
only S-MIS	0	2	23	25	4
both	267	269	264	259	285
only G-IRIS	0	0	0	0	0

Table 2: Problems that where translated by both algorithms and how often the translation could be solved for the optimal track.

For the satisficing track we report the results in Table 3. Here similar to the optimal track, the translation from S-MIS can be solved at least as often as the one by G-IRIS.

Besides the search time, another interesting measure is the number of states that were evaluated before reaching the final f-layer. We will refer to evaluations

solved by	blind	hmax	ipdb	lmcut	m&s
only S-MIS	0	0	16	19	1
both	237	237	235	234	253
only G-IRIS	0	0	0	0	0

Table 3: Problems that where translated by both algorithms and how often the translation could be solved for the satisficing track.

until the final f-layer with the shorthand f-evaluations. For the heuristics blind, hmax and lmcut we can observe mostly similar search times and number of f-evaluations. We report all those in Appendix A. Meanwhile, the same attributes for A* with ipdb or merge and shrink are less regular as can be seen in the example on f-evaluations in Figure 3. All four plots can be found in Appendix A. On all of them we see the following things:

- Most pairs of translations of problems are solved with similar number of f-evaluations and in similar time.
- There are a few problems where one of the two translations is solved more or less instantly, while the other takes significantly more time or f-evaluations.
- There are a few problems, where both take some time or f-evaluations. However, one translation performs significantly better than the other.

We used the ratio of the absolute difference between the time for both and scaled it by dividing by their sum. We then selected all problems, where this scaled ratio was bigger than 0.15, which we call deriving from the diagonal. Any value close to 1 is clearly dominated by either translation. As the values come closer to 0, they are closer to the diagonal. We used the same ratio and selection for f-evaluations. We present a few comparisons in Table 4.

We quantitatively analysed the points deviating from the diagonal. Our analysis focused on four attributes, namely on the number of binary FDR variables, number of FDR variables, number of FDR actions and number of FDR facts (atoms). We found that the deviating points belong to many different domains and that there are no directly obvious similarities between them. It is notable, that the two domains blocks and psr-small are the most prominent. Of the 15 domains of block that were translated and then solved for both algorithms, we found 13 that deviated from the diagonal. For psr-small, 47 were translated and solved for both algorithms and 24 of them deviated from the diagonal. However, for no heuristic did A^* perform consistently better for either translation.

When examining the block translations, we found, that the FDR representation by S-MIS had consistently fewer FDR facts, FDR actions and FDR variables. Both the examples in Table 4 reflect that. The translation by G-IRIS only achieved a smaller number of binary FDR variables for a few problems.

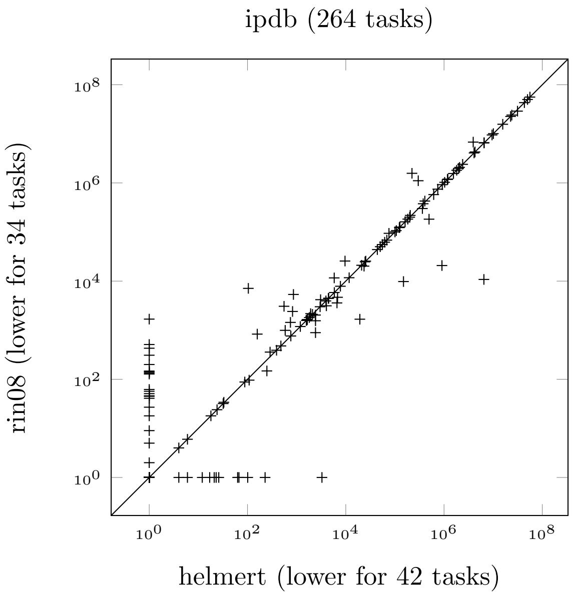

Figure 3: Number of f-evaluations for A^* with ipdb on optimal track.

However, for many instances, the translation of G-IRIS actually had fewer f-evaluations and took less time. Even for a few where it had a higher number for all the four attributes.

For psr-small we found that the translation by G-IRIS mostly had fewer binary FDR variables. However, it only performed significantly better for 5 of the problems across all four configurations. Some of these were solved better on the translation by S-MIS for other configurations. In terms of FDR facts, FDR actions and FDR variables, the translation by S-MIS often has fewer. However, there are a few problems where the FDR representation resulting from G-IRIS has fewer of all four attributes.

heuristic		ipdb				m&S
domain		termes	psr-small	psr-small	blocks	blocks
problem		p18	p48	p46	pB-8-1	pB-8-2
time	h	13.34	28.11	0.02	0.02	0.12
	Δ	10.09	-28.08	0.01	0	-0.1
	r	23.43	0.03	0.03	0.02	0.02
f-evaluations	h	3925448	6482493	5818	6803	44964
	Δ	2854084	-6471684	5808	-2122	-41685
	r	6779532	10809	11626	4681	3279
binary vars	h	1	35	27	9	9
	Δ	3	-4	-4	-1	3
	r	4	31	23	8	12
variables	h	13	36	92	17	17
	Δ	3	-1	0	8	12
	r	16	35	92	25	29
facts	h	58	91	67	90	90
	Δ	6	0	-5	21	27
	r	64	91	62	111	117
actions	h	468	188	28	128	128
	Δ	54	0	-3	10	14
	r	522	188	25	138	142

Table 4: Comparison of the A* search with the noted heuristic on both translations, h for tranlated by S-MIS and r for G-IRIS respectively, and the difference $\Delta = h - r$. We shortened "termes-opt18-strips" to "termes", "probBLOCKS" to "pB" and reduced the problem names for termes from "p48-s101-n5-l3-f30" and "p46-s97-n5-l2-f30" to their index, i.e. "p48".

5 Related Work

Bernardini and Smith [3] implemented a version of the algorithm proposed by Helmert[9] which uses a decision tree to guide the guess-check-repair process on a generalization to numerical and temporal planning tasks.

Bernardini et al. [4] improved further on the approach by applying the decision tree on the lifted representation of the planning task. They do report some improvements in number of identified invariants compared to the algorithm used in temporal fast downward.

5.1 Alcázar and Torralbas Invariant Synthesis Algorithm in 2015

Alcázar and Torralba [1] proposed the idea of using the regression direction to produce invariants which can be used for pruning during the forward search. The consequent step they take is to argue that the opposite direction works as well. Consequently they propose alternation of directions to use invariants form one direction to improve the other direction and continue using new knowledge as long as it can be found.

The Algorithm 11 is their proposed fix point computation of grounded invariants defined on SAS^+[2] problems. The h^n heuristic was proposed by Geffner and Haslum [6] in 2000. In *computeH2* mutexes of size 2 are computed as a side effect of a reachability analysis in a state space reduced to size 2. The function returns true if new invariants where found. Meanwhile *disambiguate-Actions* computes which actions are consistent with invariants and returns true if the set changed.

Algorithm 11: Fix point computation of invariants.

```
$fw := True, updatedFW := True, updatedBw := True$
**while** $updatedFW \lor updatedBW$ **do**
    **if** $fw \land updatedBW$ **then**
        $updatedFW$ := computeH2 (fw) $\lor$ disambiguateActions ()
    **if** $\neg fw \land updatedFW$ **then**
        $updatedBW$ := computeH2 (fw) $\lor$ disambiguateActions ()
    $fw := \neg fw$
```

Notice how *computeH2* receives a boolean *fw* that is false when updating backwards.

6 Discussion

In this thesis we implemented G-IRIS and used it for finding mutex groups in the translation of Fast Downward. In that role, we compared it against S-MIS. In terms of speed of the translation, S-MIS performs better. For our experimental evaluation, it outperformed our implementation of G-IRIS by more than two magnitudes for all but the simplest problems. Another benefit of S-MIS is that it can stop early and still deliver results, which is not the case for G-IRIS.

Our implementation of G-IRIS could conceivably be improved upon. Setting aside smaller optimizations, it might be possible to find systematic ways of speeding up the algorithm. In the unit resolution we test the formula resulting from the regression against the set of candidates from the last iteration. We could, once per iteration, precompute all the unit clauses which will fail the unit resolution on their own. If these appear in a set of unit clauses, the whole set can already be dismissed. While the unit resolution was implemented with smart data structures for clause selection, this optimization could still reduce the runtime of many of the 2-SAT tests.

We made a few observations about the performance of A^* on the translations. There was no obvious causal link between the representation size and the performance of A^*. It would be interesting to examine the few oddities we found further. There might be some similarities between the outliers which we did not yet discover.

A comparison between the implemented algorithms and the algorithm proposed by Alkazar and Torralba could also be interesting. They report to outperform S-MIS for many problems. While their algorithm most likely will also outperform G-IRIS in terms of invariant search speed, it would be interesting to compare the mutexes these two algorithms find.

References

[1] Vidal Alcázar and Alvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *Twenty–Fifth International Conference on Automated Planning and Scheduling*, pages 2–6, 2015.

[2] Christer Bäckström and Bernhard Nebel. Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[3] Sara Bernardini and David E Smith. Finding mutual exclusion invariants in temporal planning domains. *Seventh International Workshop on Planning and Scheduling for Space*, 2011.

[4] Sara Bernardini, Fabio Fagnani, and David E Smith. Extracting mutual exclusion invariants from lifted temporal planning domains. *Artificial Intelligence*, 258:1–65, 2018.

[5] Richard E Fikes and Nils J Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

[6] Héctor Geffner and Patrik Haslum. Admissible heuristics for optimal planning. In *Fifth International Conference of AI Planning Systems*, pages 140–149, 2000.

[7] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[8] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, Sven Koenig, et al. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *Twenty–Second AAAI Conference on Artificial Intelligence*, volume 7, pages 1007–1012, 2007.

[9] Malte Helmert. Concise finite–domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5–6):503–535, 2009.

[10] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: what's the difference anyway? In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2010.

[11] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[12] Drew M McDermott. The 1998 AI planning systems competition. *AI magazine*, 21(2):35–35, 2000.

[13] Jussi Rintanen. Compact representation of sets of binary constraints. In *ECAI*, volume 141, pages 143–147, 2006.

[14] Jussi Rintanen. Regression for classical and nondeterministic planning. In *ECAI*, volume 178, pages 568–572, 2008.

[15] Jussi Rintanen. Schematic invariants by reduction to ground invariants. In *Thirty–First AAAI Conference on Artificial Intelligence*, pages 3644–3650, 2017.

[16] Silvan Sievers, Martin Wehrle, and Malte Helmert. Generalized label reduction for merge-and-shrink heuristics. In *Twenty–Eighth AAAI Conference on Artificial Intelligence*, pages 2358–2366, 2014.

[17] Silvan Sievers, Martin Wehrle, and Malte Helmert. An analysis of merge strategies for merge-and-shrink heuristics. In *Twenty–Sixth International Conference on Automated Planning and Scheduling*, pages 294–298, 2016.

[18] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

A Search Plots

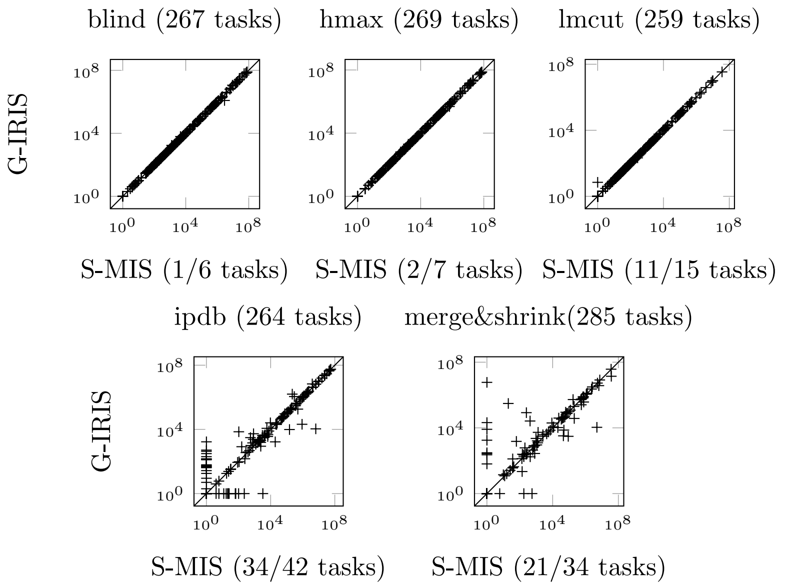

Figure 4: Number of evaluations until reaching the last f-layer using the heuristic in the title on the optimal track. The numbers (Y/X tasks) describe how many tasks where solved with fewer evaluations by the translation from G-IRIS (Y) or S-MIS (X).

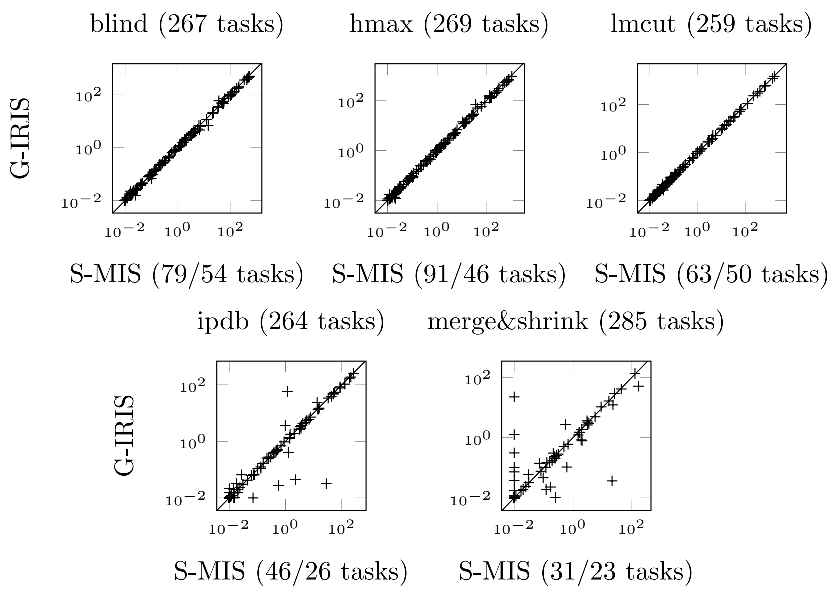

Figure 5: Search time using the heuristic in the title on the optimal track. The numbers (Y/X tasks) describe how many tasks where solved with less time by the translation from G-IRIS (Y) or S-MIS (X).

Universität Basel

Philosophisch-Naturwissenschaftliche Fakultät

Erklärung zur wissenschaftlichen Redlichkeit
(beinhaltet Erklärung zu Plagiat und Betrug)

Masterarbeit

Titel der Arbeit *(Druckschrift)*:
A Comparison of Invariant Synthesis Methods

Name, Vorname *(Druckschrift)*: Wyss, Severin

Matrikelnummer: 2013-053-707

Mit meiner Unterschrift erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe.

Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Diese Erklärung wird ergänzt durch eine separat abgeschlossene Vereinbarung bezüglich der Veröffentlichung oder öffentlichen Zugänglichkeit dieser Arbeit.

☐ ja [✓ nein]

Ort, Datum: Basel, 5. Oktober 2020

Unterschrift: [illegible]

Dieses Blatt ist in die Bachelor-, resp. Masterarbeit einzufügen.