Compilability between Generalized Representations for Classical Planning

Master's thesis

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

Examiner: Prof. Dr. Malte Helmert, Dr. Gabriele Röger
Supervisor: Dr. Gabriele Röger

Claudia S. Grundke
claudia.grundke@unibas.ch
17-058-538

22 December, 2022

Acknowledgments

First, I would like to thank Prof. Dr. Helmert for the opportunity to write my Master's thesis in his research group and for the ideas he offered regarding my topic. And secondly, I am very grateful to my supervisor Dr. Gabriele Röger because of her support and the feedback she provided during our many meetings.

Abstract

In generalized planning the aim is to solve whole classes of planning tasks instead of single tasks one at a time. Generalized representations provide information or knowledge about such classes to help solving them. This work compares the expressiveness of three generalized representations, generalized potential heuristics, policy sketches and action schema networks, in terms of compilability. We use a notion of equivalence that requires two generalized representations to decompose the tasks of a class into the same subtasks. We present compilations between pairs of equivalent generalized representations and proofs where a compilation is impossible.

Table of Contents

1 Introduction

In automated planning the aim is to find plans that solve planning tasks. Take for example the problem of building a tower of blocks from multiple building blocks lying on a table. This task can be solved by picking up and placing the blocks onto each other such that a single tower remains. Now, what if there is not a single task that asks for a single tower but there are many different tasks that ask for different configurations of blocks where different numbers of blocks are involved? The plan from before is not sufficient because there might be configurations with more than one tower to be built. This is what generalized planning is concerned with, solving not only single tasks but whole classes of tasks.

Generalized representations provide information about such classes of tasks that can be used to solve them. Heuristics for example give an estimation for each state of a planning task that tells how far or how close the state is to the goal. Generalized heuristics, like generalized potential heuristics which we consider in this work, can provide these estimates not only for each state of a single task but for all states of all tasks in a class. Policy sketches are the second generalized representation we evaluate and by defining subgoals for each state they decompose the tasks of a class into subtasks. Solving the subtasks of a task then leads to a solution for the whole task. Lastly, we consider action schema networks which yield policies to solve the tasks of a class. These policies tell for each state which action should be chosen to get closer to the goal.

In this work we take a look at these three generalized representations, generalized potential heuristics, policy sketches and action schema networks, and investigate whether they can express the same information about a class of tasks. We compare them in terms of compilability, that means we provide compilations from one generalized representation into another or proof that no compilation exists.

This work is structured as follows. In Chapter 2 we begin with introducing the concepts needed to understand our compilations results. Most importantly we explain the basics of each of the three generalized representations we are going to compare. Chapter 3 then is our main contribution and presents compilations and counterexamples where a compilation is impossible. We first show that generalized potential heuristics can be compiled into policy sketches (Section 3.1) but the reverse direction, that means compiling policy sketches into generalized potential heuristics, is usually not possible (3.2). Afterwards we show that action schema networks and generalized potential heuristics can in general not be compiled into each other (Sections 3.3 and 3.4). The following Chapter 4 then is concerned with work related to our compilation results. Lastly, we discuss our results and provide an outlook on potential future work in Chapter 5.

Summary of Compilation Results

The following is a summary of our compilation results and is shown here as an overview. It will be discussed in the last chapter.

Compilation into policy sketch:

Theorem 1 Compilation from non-negative integer-valued GP heuristic into **equivalent** sketch

Theorem 2 Compilation from non-negative **real-valued** GP heuristic into **equivalent** sketch

Theorem 3 Compilation from integer-valued (including **negative values**) GP heuristic into **equivalent** sketch

Theorem 4 Compilation is **not possible** from (non-negative integer-valued) GP heuristic into **equivalent** sketch with **same features**

Compilation from policy sketch:

Theorem 5 Compilation **not possible** from sketch into **equivalent** GP heuristic (in general and thus also not with same features)

Theorem 6 Compilation from sketch into **similar** GP heuristic

Theorem 7 Compilation **not possible** from sketch into **similar** GP heuristic with **same features**

Compilation from and into action schema network parameters:

Theorem 8 Compilation **not possible** from ASNet parameters into **equivalent** GP heuristic

Theorem 9 Compilation **not possible** from GP heuristic into **equivalent** ASNet parameters

Theorems 5 and 8 hold for heuristics in general and Theorems 1, 2 and 3 can be altered to hold for heuristics in general.

2 Background

This chapter introduces the notation and concepts which form the basis of the compilation results presented in Chapter 3. Section 2.1 paves the way with the overall setting of generalized classical planning in which our results are settled. The following three sections introduce the three generalized representations we want to compile into each other: generalized potential heuristics in Section 2.2, policy sketches in Section 2.3 and action schema networks in Section 2.4.

2.1 Generalized Classical Planning

Classical planning is concerned with deterministic planning tasks in which a single agent starts in some initial state and tries to reach a goal state by altering its current state with some action. The aim is to find a sequence of actions, a plan, that leads from the initial state to some goal state. We consider the special case of classical planning where each action has the same cost, i. e. unit-cost planning, because all three considered generalized representations are based on this.

Generalized planning reasons over whole classes of planning tasks. That means, trying to find solutions for multiple, possibly infinitely many, related planning tasks at once instead of finding a solution for each task separately.

Generalized classical planning is therefore concerned with classes of classical planning tasks. We begin though, with introducing the basic notions of classical planning. The foundation are *state spaces* which describe a problem by specifying its states and actions. A state space can be considered as a graph where each node corresponds to a state and each directed edge represents an action between two nodes. It is defined as follows.

Definition 1 (State space). *A* state space *is a tuple* $\mathcal{S} = \langle S, A, T, s_0, S_* \rangle$ *where*

- S *is a finite set of* states*,*
- A *is a finite set of* actions*,*
- $T \subseteq S \times A \times S$ *is the* transition relation*,*
- $s_0 \in S$ *is the* initial state*, and*

- $S_* \subseteq S$ *is the set of* goal states*.*

Because we are in the setting of classical planning the transition relation T is considered to be deterministic. That means, for each pair $\langle s, a\rangle \in S \times A$ there is at most one state $s' \in S$ such that $\langle s, a, s'\rangle \in T$ holds. So, given a state s and an action a that is applied in this state, we can predict with certainty what the resulting state s' will be.
Action schema networks are actually able to handle stochastic transitions because they were introduced for *stochastic shortest path problems* (SSPs). With stochastic transitions, applying an action a in a state s can lead to different successor states s' with certain probabilities. But since our focus is on classical planning, we only consider the less general case of deterministic transitions (which is just the special case where for each state-action pair there is at most one transition and this transition has probability 1).
Next, we formally define the terms *applicability* and *successor* which are important for reasoning about planning tasks.

Definition 2 (Applicable action, successor state)**.** *Let T be a transition relation.*
An action a is applicable *in state s if there exists a transition $\langle s, a, s'\rangle \in T$ for some state s'.*
The successors *of a state s are all states s' such that $\langle s, a, s'\rangle \in T$ holds for some action a.*

Applying actions over and over leads to sequences from which we can derive the current state. These action sequences are called *paths* and are defined as follows.

Definition 3 (Path)**.** *Let $\mathcal{S}$ be a state space with transition relation T.*
A path *between two states s_0 and s_n (for some finite $n \in \mathbb{N}_{\geq 0}$) is a sequence of actions $\pi = \langle a_1, \dots, a_n\rangle$ such that there exist states $s_1, \dots, s_{n-1}$ with $\langle s_{i-1}, a_i, s_i\rangle \in T$ for $1 \leq i \leq n$.*
The length of a path π *is $|\pi| = n$ which is the number of actions it includes.*

Paths that start in the initial state and end in a goal state are most important in planning as they describe *solutions* for a state space:

Definition 4 (Plan, solution for a state space)**.** *Let $\mathcal{S} = \langle S, A, T, s_0, S_*\rangle$ be a state space.*
A plan *for state $s \in S$ is a path from s to some goal state $s_* \in S_*$.*
A solution *for state space $\mathcal{S}$ is a plan for s_0. If a state space has a solution it is called* solvable*, otherwise it is* unsolvable*.*
An optimal plan *for a state $s \in S$ is a plan of s with minimal length among all plans for s and an* optimal solution *for state space $\mathcal{S}$ is an optimal plan for s_0.*

With these definitions we can now specify state spaces and their solutions. The following definition introduces terms to differentiate certain states from each other such that we can reason further about state spaces.

Definition 5 (Reachable state, solvable state, alive state, dead-end state)**.** *A state s is* reachable *if there exists a path from the initial state to s.*
A state s is solvable *if there exists a path from s to some goal state and it is* unsolvable *if it is not solvable.*
A state s is alive *if it is reachable, solvable and not a goal state.*
A state s is a dead-end *if it is reachable and unsolvable.*

Note that the initial state is trivially reachable and all goal states are trivially solvable by the empty path $\langle\rangle$.
Because the number of states can be huge and thus can be difficult to store explicitly, the core concept of automated planning are *planning tasks* that compactly represent state spaces. The following definition of a planning task complies with the STRIPS formalism [6].

Definition 6 (Planning task)**.** *A* planning task *is a tuple $P = \langle F, A, I, G\rangle$ where*

- *F is the finite set of propositions, also called* ground atoms*, or just* atoms*,*
- *A is the finite set of actions where for each action $a \in A$ the following is defined:*
  - *$Pre(a) \subseteq F$ is the set of* preconditions *of a,*
  - *$Add(a) \subseteq F$ is the set of* add effects *of a, and*
  - *$Del(a) \subseteq F$ is the set of* delete effects *of a,*
- *$I \subseteq F$ represents the initial state,*
- *$G \subseteq F$ represents the set of goal states.*

Comparing state spaces with planning tasks, the parallels of actions, initial state and goal states are easy to see. The relation of states and propositions is not so direct and calls for more explanation. Each subset of the propositions of a planning task, $s \subseteq F$, represents the state where the propositions in s are true and all other propositions, $F \setminus s$, are false.
With this we can formally define how a state space can be related to a planning task:

Definition 7 (Induced state space)**.** *A planning task $P = \langle F, A, I, G\rangle$* induces a state space *$\mathcal{S} = \langle S, A, T, s_0, S_*\rangle$ where*

- *$S = 2^F$ (power set of F),*
- *for states s and s' and action a we have $\langle s, a, s'\rangle \in T$ if $Pre(a) \subseteq s$ and $s' = (s \setminus Del(a)) \cup Add(a)$ hold, that means $\langle s, a, s'\rangle$ is a transition if s satisfies the preconditions of a and s' is s where the delete effects of a are applied and then the add effects,*
- *$s_0 = I$, and*
- *for a state s we have $s \in S_*$ if $G \subseteq s$, so state s is a goal state if the propositions in G are true in s.*

Using this definition we can take over some terminology from state spaces to planning tasks. A *solution of a planning task* is a plan that solves the initial state of the planning task, or analogously a solution of the state space that is induced by a planning task is also a solution for the planning task itself. Likewise, a planning task is *solvable* if there exists a plan that solves its initial state and it is *unsolvable* otherwise.
We have now covered the necessary concepts from classical planning and will in the following introduce concepts from generalized planning. As mentioned before, in generalized planning we do not try to find solutions for single tasks but for multiple planning tasks at once. It is usually assumed that these tasks are defined over the same *planning domain* and that they share a common underlying structure. A domain in this case is defined as follows.

Definition 8 (Planning domain)**.** *A* planning domain *(or* domain *for short) is a tuple $\langle \mathcal{P}, \mathcal{A} \rangle$ where*

- *$\mathcal{P}$ is a finite set of* predicates*, and*
- *$\mathcal{A}$ is a finite set of* action schemas.

This definition is based on lifted SSPs from Toyer et al. [23]. The domain definition of Bonet and Geffner [3] slightly differs from ours but is analogous. Francès et al. [7] additionally use the concept of a first-order vocabulary which makes their framework slightly more expressive. We will only consider generalized potential heuristics in the context of domains as defined above because the use of concept-based features in Francès et al. [7] leads to similar restrictions compared to the usage of above domain definition.
To obtain a planning task from a domain the predicates and action schemas need to be *grounded* with a set of objects. Grounding a predicate from $\mathcal{P}$ means instantiating it with a tuple of objects which yields a proposition (called ground atom). Similarly, grounding an action schema from $\mathcal{A}$, that means instantiating it with a tuple of objects, yields an action. Take for example the blocksworld domain where we have blocks on a table that should be brought from some configurations into other configurations. We might have a predicate *on* with arity 2 that specifies whether some block lies on another block. When grounding this predicate with two block-objects a and b for example we obtain the proposition $on(a, b)$ which is true if block a lies on block b.
In order to fully specify a planning task we furthermore need an initial state and goal states. The following definition summarizes what is needed to fully specify a set of tasks for a domain.

Definition 9 (Class of planning tasks)**.** *A* class of planning tasks *is a set of planning tasks over the same domain $\langle \mathcal{P}, \mathcal{A} \rangle$ where each task has its own* instance information $\langle \mathcal{O}, I, G \rangle$ *with*

- *$\mathcal{O}$ a set of* objects*,*
- *$I \subseteq F$ the initial state (as defined for planning tasks), and*
- *$G \subseteq F$ the set of goal states (as defined for planning tasks)*

where propositions F are obtained from grounding the predicates $\mathcal{P}$ with the objects $\mathcal{O}$. Grounding the predicates $\mathcal{P}$ and action schemas $\mathcal{A}$ with the objects in $\mathcal{O}$ yields a planning task with initial state I and goal states G.

This definition is based on Toyer et al. [24] and Bonet and Geffner [3] use the term class of planning tasks analogously. Francès et al. [7] on the other hand define this terminology differently. What we call a class of planning tasks, they call generalized planning domain and as mentioned earlier they additionally use a first-order vocabulary.
We have now introduced all basic concepts of our work. In the following three sections we will introduce each of the three generalized representations we want to compile into each other.

2.2 Generalized Potential Heuristics

Generalized potential heuristics (or GP heuristics for short) were introduced by Francès et al. [7] and are weighted sums of features. They are a generalized planning variation of potential heuristics from Pommerening et al. [14]. Their features are mappings from states to integers while the weights are real-valued and fixed for each feature. Francès et al. [7] use a certain kind of features that are based on concept languages also called description logics [1] but for our work the general notion of features as mappings from states to integers suffices. Generalized potential heuristics are formally defined as follows.

Definition 10 (Generalized potential heuristic, [7]). *Let S be a set of states and $\mathcal{F}$ a set of features $f : S \rightarrow \mathbb{Z}$. Let $w : \mathcal{F} \rightarrow \mathbb{R}$ be a* weight function *mapping features to weights. The value of the* generalized potential heuristic *with features $\mathcal{F}$ and weights w on state $s \in S$ is*

$$h(s) = \sum_{f \in \mathcal{F}} w(f) \cdot f(s).$$

The states of S do not have to be from the same task but for the generalized potential heuristic to be well-defined they should be from tasks from the same class of planning tasks. A generalized potential heuristic is well-defined on any state of any task of a class of planning tasks if all its features are well-defined. Because of this Francès et al. [7] derive their features from the predicates and constants that are shared among all tasks in a class. They use concept languages [1] to build these features.
Francès et al. [7] use GP heuristics to guide greedy search algorithms and because of that they are interested in *descending* and *dead-end avoiding* GP heuristics. Heuristics that are descending and dead-end avoiding have no local minima and thus effectively guide greedy searches [18].
We begin with descending heuristics which are heuristics that assign a lower heuristic value to at least one successor of each alive state. That means each alive state has at least one successor with improving heuristic value which a search algorithm can choose as next state. Descending heuristics are formally defined as follows.

Definition 11 (Descending heuristic, [18]). *A heuristic h is* descending *in task P if every alive state s of P has a successor state s' with $h(s') < h(s)$.*
A heuristic is descending in a class $\mathcal{Q}$ of planning tasks if it is descending in every task $P \in \mathcal{Q}$.

While a descending heuristic guarantees that there is a successor that looks promising, we also need dead-end avoidance to guarantee that this successor actually is an improvement. With a dead-end avoiding heuristic we know that each improving successor of an alive state is solvable. This is formally defined as follows.

Definition 12 (Dead-end avoiding heuristic, [18]). *A heuristic is* dead-end avoiding *in task P if all successors s' with $h(s') < h(s)$ for alive states s are solvable.*
It is dead-end avoiding in a class $\mathcal{Q}$ of planning tasks if it is dead-end avoiding in every task $P \in \mathcal{Q}$.

So, with a heuristic that is descending and dead-end avoiding there always is an improving successor and all improving successors are solvable and thus good choices. A heuristic with these properties can enable polynomial runtime for greedy search algorithms over planning tasks. Simple hill-climbing with a descending and dead-end avoiding heuristic for example directly finds a path to a goal which means that the runtime depends on the length of the discovered plan. Thus in our compilations we aim for the compiled GP heuristics to be descending and dead-end avoiding.

2.3 Policy Sketches

Policy sketches were introduced by Bonet and Geffner [3] in the context of general policies. The follow-up work by Drexler et al. [4] presents hand-crafted sketches for certain domains and shows experimentally that sketches help solving problems of these domains. The second follow-up work by Drexler et al. [5] presents a method for learning sketches automatically.
Policy sketches describe changes that must happen on the path towards the goal. In contrast to general policies these changes do not have to happen within a single action application. To be able to generalize over multiple planning tasks, sketches (as well as general policies) use features similar to the features of generalized potential heuristics. These features map states to the natural numbers, i.e. non-negative integers. The features allowed for GP heuristics in contrast can take values in the full integer numbers.
The changes a policy sketch describes are specified in terms of *sketch rules.* They tell how certain features must change if a state satisfies the feature conditions of the rule. Formally, the syntax of sketch rules is defined as follows.

Definition 13 (Sketch rule)**.** *For a set S of states from the same class $\mathcal{Q}$ of planning tasks, let Φ be a set of features that are either boolean, $p : S \rightarrow \{\bot, \top\}$, or non-negative integers, $n : S \rightarrow \mathbb{N}_{\geq 0}$.*
A sketch rule *$C \rightarrow E$ then consists of set C with* feature conditions *of the form p, $\neg p$, $n > 0$ or $n = 0$, and set E with* feature effects *of the form p, $\neg p$, $p?$, $n\downarrow$, $n\uparrow$ or $n?$.*

This definition is based on Bonet and Geffner [3] as well as the following definition of policy sketches which are just sets of sketch rules.

Definition 14 (Policy sketch)**.** *A* policy sketch *(or* sketch *for short) for class $\mathcal{Q}$ of planning tasks is a set R_Φ of sketch rules $C \rightarrow E$ over features Φ that are defined for the states of class $\mathcal{Q}$.*

In contrast to the definition of sketches from Bonet and Geffner [3] we do not require goal-separating features for a sketch to be well-defined. The reason for omitting goal-separation is discussed at the end of this section.
Before turning to the semantics of feature conditions and effects we present a short example to illustrate the syntax of policy sketches and sketch rules: For a class of planning tasks in the blocksworld domain where we have towers of blocks built on a table and want to place all blocks directly on the table we can take for example the feature set $\Phi = \{H, n\}$. Boolean feature H tells whether a block is currently held and numerical feature n represents the number of blocks currently lying on the table. Then a sketch could look like this:

$R_\Phi = \{\{\neg H\} \rightarrow \{H\}, \{\} \rightarrow \{H?, n\uparrow\}\}$. This sketch has two rules $\{\neg H\} \rightarrow \{H\}$ and $\{\} \rightarrow \{H?, n\uparrow\}$. The semantics of sketch rules are discussed in the following.

Intuitively, a sketch rule $C \rightarrow E$ describes changes that must happen on the path towards the goal. If a state s satisfies the conditions in C then the effects in E determine which states are subgoal states that we want to reach from s. For a boolean feature p, a state s satisfies the feature condition p if $p(s)$ is true and it satisfies the condition $\neg p$ if $p(s)$ is false. For a numerical feature n, a state s satisfies $n > 0$ if $n(s)$ is greater zero and it satisfies $n = 0$ if $n(s)$ is exactly zero. Satisfying a feature effect is defined as follows:

Definition 15 (Satisfy feature effect and sketch compatibility[1]). *A state pair* $\langle s, s' \rangle$ satisfies a feature effect *set E when the following holds:*

1. *if, for a boolean feature p, feature effect p (respectively $\neg p$) is in E, then $p(s') = \top$ (respectively $p(s') = \bot$) must hold,*
2. *if, for a numerical feature n, feature effect $n\downarrow$ (respectively $n\uparrow$) is in E, then $n(s) > n(s')$ (respectively $n(s) < n(s')$) must hold, and*
3. *if boolean feature p (respectively numerical feature n) is not mentioned at all in E, then $p(s) = p(s')$ (respectively $n(s) = n(s')$) must hold.*

The state pair $\langle s, s' \rangle$ is compatible *with sketch R_Φ if there is a sketch rule $C \rightarrow E$ such that s makes the conditions in C true and $\langle s, s' \rangle$ satisfies the effects in E.*

Additionally, the effect set E can also contain effects $p?$ and $n?$, as mentioned in Definition 13 of sketch rules. These effects do not put any restrictions on the values of the features p and n. That means while features not mentioned in an effect set E must not change their value (third rule in above definition), features mentioned as $p?$ or $n?$ can take any value.

Returning to the example from before, the first sketch rule of the sketch $R_\Phi = \{\{\neg H\} \rightarrow \{H\}, \{\} \rightarrow \{H?, n\uparrow\}\}$ states that if boolean feature H is false in a state, then we should reach a state where H is true while the value of n remains unchanged. This rule describes that it is good to pick up a block that is not lying on the table. Picking up a block changes H from false to true and doing this while keeping n, the number of blocks on the table, unchanged is only possible if a block not lying on the table is picked up. The second sketch rule, $\{\} \rightarrow \{H?, n\uparrow\}$ tells that in all states it is good to increase n, the number of blocks directly on the table, no matter if or how the value of H changes.

Sketches were introduced as a tool for the greedy search algorithm SIW to be able to handle problems of large width. For the variant SIW$_R$ that uses sketches Bonet and Geffner [3] can guarantee polynomial runtime if sketch R has certain properties. Both SIW and SIW$_R$ use the IW algorithm which is a breadth-first search that prunes states based on a measure of novelty related to the width of a problem. SIW$_R$ calls multiple IW searches to solve the subproblems induced by sketch R. Solving the overall problem can be done in polynomial time and space if R is terminating, has linear features and there is a fixed bound on the

[1] This definition is based on Definition 5 of Bonet and Geffner [3] but uses state pairs instead of transitions. Furthermore, Bonet and Geffner [3] defined this notion in the context of policy rules and later adjusted it to sketch rules, which is why their definition uses transitions.

sketch width. Bonet and Geffner [3] furthermore require the sketch features to be goal-separating which we do not aim for though as will be discussed at the end of this section. The other properties we aim for in our compilations and therefore we introduce them now. We begin with *feature valuations* and *boolean feature valuations* though because these are needed for the following definitions. Given a finite set of features $\Phi = \{f_1, \ldots, f_N\}$, a feature valuation f_Φ maps each feature f_i to a value of it. Furthermore, the feature valuation determined by a state s is the feature valuation f_Φ where $f_\Phi(f_i) = f_i(s)$ for all features f_i in Φ, that means f_Φ maps all features to the values they take in state s. A condition or effect on feature f_i then, is true under f_Φ if $f_\Phi(f_i)$ satisfies it. Thus, like for state pairs, we can also say that a feature valuation pair $\langle f_\Phi, f'_\Phi \rangle$ is compatible with a sketch.
Boolean feature valuations b map each value of a feature valuation to true or false. The values originally from boolean features remain $\top$ (true) or $\perp$ (false). The value of a numerical feature is mapped to true if it is equal to zero (i. e. if $n = 0$) and it is mapped to false if it is larger than zero (i. e. $n > 0$). This leads to $2^{|\Phi|}$ possible boolean feature valuations for a feature set Φ because we have for each feature f_i in Φ either true or false in the boolean feature valuation.
The first property we are going to define is *termination* which is crucial for polynomial runtime of SIW_R as it guarantees that SIW_R does not get stuck in infinite loops. To define it though we first need to introduce *policy graphs.*

Definition 16 (Policy graph)**.** *The* policy graph $G(R_\Phi)$ *for policy sketch* R_Φ *has nodes* b*, one for each of the* $2^{|\Phi|}$ *boolean feature valuations over* Φ*, and edges* $b \to b'$ *labeled with* E *if* $\langle b, b' \rangle$ *is compatible with a rule* $C \to E$ *in the policy sketch.*

This definition is based on Bonet and Geffner [3] and slightly different because when labeling the edges it does not ignore pairs $\langle b, b' \rangle$ of which b is a so-called goal valuation. This change is a result of our omission of goal-separating features which are needed to enable goal valuations.
A policy sketch R_Φ is terminating if following its rules cannot lead to infinitely long paths. Since the set of rules is finite, infinitely long paths can only occur if the sketch rules induce cycles in the policy graph.

Definition 17 (Termination, [3])**.** *A policy* π_Φ *and a policy graph* $G(\pi_\Phi)$ *are* terminating *if for every cycle in the graph* $G(\pi_\Phi)$*, i.e., any sequence of edges* $b_i \to b_{i+1}$ *with labels* E_i *that start and end in the same node, there is a numerical feature* n *in* Φ *that is decreased along some edge and increased in none. That is,* $n\downarrow \in E_k$ *for some edge in the cycle, and* $n\uparrow \notin E_j$ *and* $n? \notin E_j$ *for all others edges in the cycle.*

This definition is implicitly stricter compared to that of Bonet and Geffner [3] because we defined policy graphs differently. With our definition of policy graphs we cannot differentiate whether a cycle in the graph is before the goal or not. Thus, for a sketch to be terminating we require that even irrelevant cycles within or behind the goal must comply with Definition 17. Accepting this stricter termination definition of termination though enables us to drop the requirement of goal-separating features. This is a beneficial trade-off for us because on the one hand, our compilation results are not hindered by this stricter definition and they also

do not exploit the special case where infinite loops behind the goal make a sketch non-terminating by our definition. Goal-separating features on the other hand are a broader restriction to sketches than our version of termination. The omission of goal-separation is further discussed at the end of the section.
The next property needed to bound the runtime of SIW$_R$ is a fixed bound on the *sketch width*. First, we define *width* in general.

Definition 18 (Width, [3])**.** *The* width $w(P)$ *of a problem P is the minimal k for which there exists a sequence $t_0, t_1, \dots, t_m$ of atom tuples t_i, each with at most k atoms, such that:*

1. *t_0 is true in the initial state of P,*
2. *any optimal plan for t_i, with $i \in \{0, 1, \dots, m-1\}$, can be extended into an optimal plan for t_{i+1} by adding a single action,*
3. *any optimal plan for t_m is an optimal plan for P.*

The width is $w(P) = 0$ iff the initial state of P is a goal state. For convenience, we set $w(P)$ to 0 if the goal of P is reachable in a single step, and to $w(P) = N + 1$ if P has no solution where N is the number of atoms in P.

The last part implies that there is no fixed bound on the width for unsolvable problems because for these the width grows with the number of atoms. Now we can define sketch width which is in a sense the maximal width of all subproblems that a sketch can induce.

Definition 19 (Sketch width, [3])**.** *Let R_Φ be a sketch for a class of planning tasks $\mathcal{Q}$ over features Φ, and let s be a reachable state in some instance P of $\mathcal{Q}$.*
The width of sketch *R_Φ at state s of problem P, $w_R(P[s])$, is the width of the subproblem $P[s]$ that is like P but with initial state s and goal states s' such that s' is a goal state of P or the pair $\langle f_\Phi(s), f_\Phi(s')\rangle$ of feature valuations is compatible with a sketch rule $C \to E$.*
For a class $\mathcal{Q}$ of planning tasks, the width of sketch R_Φ, $w_R(\mathcal{Q})$, is the maximal width $w_R(P[s])$ for any reachable state s in any problem P of $\mathcal{Q}$.

A caveat of this definition is that a fixed bound on the sketch width is impossible for terminating sketches over tasks with dead ends. This is the case because the sketch width is determined by the maximal width of the subproblems described in Definition 19. So if one such subproblem has maximal width $N + 1$ (N is the number of atoms) which is not a fixed bound on the width, then this width determines the sketch width for the whole task. This sketch width in turn also has no fixed bound. Extending this argument, this also means there is no fixed bound on the sketch width for a whole class of planning tasks as soon as it includes a problem with a dead end.
Consider a terminating sketch R and such a subproblem $P[s]$ with initial state s (a reachable state within P) that is a dead end in the original task P. Since s is a dead end this directly implies that we cannot reach any goal of P from s. It follows that any reachable goal state of $P[s]$ must be a state s' such that $\langle s, s'\rangle$ is compatible with$R$. If no such state existed the subproblem $P[s]$ would be unsolvable and would have width $N + 1$ which then also means that R would have width $N + 1$ which is not a fixed bound.

Hence for a fixed bound on the sketch width to exist in such a case, sketch R must have a rule with which $\langle s, s' \rangle$ is compatible and state s' must be reachable from s (then s' is a reachable goal state of subproblem $P[s]$). This state s' however, is also a dead end in the original problem P because s already is a dead end. Let without loss of generality $s = s'$ hold, so $\langle s, s \rangle$ satisfies a rule in sketch R. It follows that in the policy graph of R there is a self-loop at the boolean feature valuation corresponding to state s. This self-loop though does not comply with the definition of a terminating sketch (Definition 17). We know that in this self-loop no feature value is decreased because $\langle s, s \rangle$ satisfies the sketch rule corresponding to the only edge of the loop and a feature value cannot be decreased in this loop if before and after rule application the feature valuation (that of state s) remains exactly the same. Therefore, with a terminating sketch such a (self-)loop is not possible and it follows that a fixed bound on the sketch width of a terminating sketch is only possible for tasks without dead ends. That means however, we can never get polynomial runtime of SIW_R for tasks with dead ends because polynomial runtime requires both termination and a fixed bound on the sketch width.
Drexler et al. [5] adjusted the definition of sketch width by restricting the subproblems $P[s]$ to have initial states s that are reachable and solvable in P. This change enables a fixed bound on the sketch width even for terminating sketches over tasks with dead ends, given that the dead ends cannot be reached when following the sketch. This altered definition of sketch width is useful for us as it allows us to give a fixed bound on the width of the sketch compiled from a generalized potential heuristic (Section 3.1, Theorem 1). To differentiate it from the definition of Bonet and Geffner [3] we call the sketch width defined by Drexler et al. [5] *alive sketch width* and to define it we first need to introduce the *set of subgoal states* $G_R(s)$ and the notion of a *closed* class of tasks.

Definition 20 (Subgoal states, [5]). *For sketch R, the set of* subgoal states *in state s of problem $P \in \mathcal{Q}$ is $G_R(s)$. It contains the goal states of P and the states s' for which the state pair $\langle s, s' \rangle$ satisfies a sketch rule from R.*
Furthermore, $G^*_R(s) \subseteq G_R(s)$ is the set that contains the states from $G_R(s)$ that are closest to s.

Class $\mathcal{Q}$ of tasks is *closed* if the following holds: if we have $P \in \mathcal{Q}$ then for all reachable and solvable states s in P we also have $P' \in \mathcal{Q}$ where P' is like task P but with initial state s [5]. Now we can define alive sketch width.

Definition 21 (Alive sketch width [5]). *The* alive sketch width *of sketch R over a closed class of tasks $\mathcal{Q}$ is $w_R(\mathcal{Q}) = \max_{P \in \mathcal{Q}} w(P')$ where P' is P but with goal states $G^*_R(s)$ and s is the initial state of both, provided that $G^*_R(s)$ does not contain unsolvable states.*

The definitions of sketch width (19) and alive sketch width are analogous in the sense that they both define the sketch width as the maximal width of the subproblems. They consider different sets of subproblems though. For (original) sketch width we have a subproblem for each state s reachable in the original problem and the goal states of each subproblem are the original goal states together with the subgoals defined by the sketch for state s. For alive sketch width we have a subproblem only for each state s that is reachable and solvable in the

original problem. The goal states of each subproblem are also defined slightly differently. From the set of original goal states and subgoal states of s only the closest ones which are also solvable are contained in the goal states of each subproblem.
As mentioned before, alive sketch width enables a fixed bound on the (alive) sketch width also for tasks with dead ends. This is the case because in the definition we now ignore subproblems that are unsolvable and which cannot be reached when following the sketch.
The last property Bonet and Geffner [3] require for polynomial runtime of SIW_R are *linear features* which are defined as follows.

Definition 22 (Linear feature assumption)**.** *The features f in Φ are either boolean or numerical, ranging in the latter case over the non-negative integers $\mathbb{N}_{\geq 0}$. The value $f(s)$ of feature f in a state s for problem P can be computed in time bounded by $O(bN)$ where N is the number of atoms and b bounds the branching factor in P. Numerical features can take up to $O(N)$ values.*

This definition is exactly the same as in Bonet and Geffner [3] except the last part about the number of values. Bonet and Geffner [3] require that numerical features can only take up to N values while we allow the number of values to be in $O(N)$. Only being able to use up to N values is rather restricting and does not appear necessary for polynomial runtime of SIW_R. The time and space complexity of SIW_R remains the same for $O(N)$ instead of N feature values. Thus, we relaxed this restriction and only assume features with up to $O(N)$ values. This change is relevant only for compilations from generalized potential heuristics into policy sketches, thus Theorems 1, 2 and 3. These theorems could be adjusted to the original definition by restricting the number of values of the heuristic itself to N, then the compiled sketches would have linear features in the sense of Bonet and Geffner [3] (Theorem 4 remains a counterexample even if we restricted the whole heuristic to have only up to N values).
As mentioned previously, we end this section with a discussion on why we do not require goal-separating features in contrast to Bonet and Geffner [3]. A features set is goal-separating if we can differentiate goal states from non-goal states using boolean feature valuations derived from the features set. This notion is defined formally as follows.

Definition 23 (Goal-separation, [3])**.** *Features Φ* separate goals *from non-goals in a task class $\mathcal{Q}$ iff there is a set of boolean feature valuations $\mathcal{K}$ such that for any problem P in $\mathcal{Q}$ and any reachable state s in P, state s is a goal state iff boolean feature valuation $f_{\mathcal{K}}(s)$ is in $\mathcal{K}$. The valuations in $\mathcal{K}$ are called* goal valuations*.*

Bonet and Geffner [3] ask for goal-separating features for the time and space bounds of SIW_R and, possibly because of this, also in their definitions of policy sketches and sketch width. However, we do not deem this property necessary. Search algorithms can usually detect goal-states on their own and do not have to rely on (generalized) representations to tell them when a goal state is reached. Hence, we should not require for sketches in general that their features must be goal-separating.
The feature set $\Phi = \{H, n\}$ for example, of our above mentioned sketch $R_\Phi = \{\{\neg H\} \rightarrow \{H\}, \{\} \rightarrow \{H?, n\uparrow\}\}$ is not goal-separating. Although numerical feature n counts the

number of blocks directly on the table, it is not goal-separating. With neither $n > 0$ nor $n = 0$ we can differentiate goal states from non-goal states because our goal is to place all blocks on the table where "all" can be a different number in each task of the same task class. To make Φ goal-separating we could for example add a third feature G that is true if all blocks lie directly on the table and it is false otherwise. We would then however, need to adjust the sketch rules as well.

Next to the definitions of sketches and sketch width, Bonet and Geffner [2][2] require goal-separating features for the proof of their Theorem 37 (they do not require goal-separation directly but well-definedness for which goal-separation is needed). This theorem is concerned with polynomial time and space bounds for the sketch-variant SIW_R of SIW. The proof however is based on the analogous proof for SIW with serializations which need goal-separating features. The proof itself does not need goal-separation, especially when considering it for sketches instead of serializations.

The proof shows on the one hand that SIW_R finds a solution and on the other hand that this can be achieved in polynomial time and space. The latter part is based on time and space bounds of the IW search algorithm (which is called repeatedly during SIW) and that the number of feature valuations for a task is bounded. Whether the sketch can distinguish goal states from non-goal states is not relevant here.

The former part, that SIW_R finds a solution, also does not require goal-separating features. We know that SIW_R cannot encounter unsolvable subproblems because sketch R has a fixed bound on the (alive) sketch width (this is a requirement for the polynomial time and space bounds). Thus the IW searches during SIW_R always find either a goal state or a solvable state that satisfies a sketch rule of R. Furthermore, since the sketch is terminating (this is also a requirement for the time and space bounds), SIW_R only consists of a finite number of IW searches. Hence, at some point a IW search must find a goal state and thus solve the whole task.

So, because SIW_R recognizes if a IW search found a goal state (and not a state that satisfies a sketch rule) on its own it is not necessary for the sketch to distinguish goal states from non-goal states. Hence goal-separation is not needed for SIW_R to have polynomial time and space bounds.

This line of reasoning for dropping the requirement of goal-separation is further affirmed by Drexler et al. [5], one of the follow-up papers of Bonet and Geffner [3]. This paper does not mention goal-separation at all and only asks for a terminating sketch with a fixed bound on the sketch width to guarantee polynomial time and space complexity.

2.4 Action Schema Networks

Action schema networks (ASNets) were introduced in Toyer et al. [23] and extended in Toyer et al. [24] which will use as basis for this work. ASNets are neural networks whose structure is adjusted to planning tasks. They are capable of handling (factored) stochastic shortest

[2] The proof for the time and space bounds of SIW_R is not included in Bonet and Geffner [3], only in the longer version: Bonet and Geffner [2].

path problems but this work only considers them in the context of deterministic planning tasks as mentioned in Section 2.1. In contrast to generalized potential heuristics and policy sketches a single ASNet can only solve a single task. Its parameters however can be shared among all tasks of a class and enable ASNets to generalize. So, for a class of planning tasks we have one set of ASNet parameters with which we initialize an ASNet for each task that we want to solve.

An ASNet is built of L *proposition layers* and $L+1$ *action layers* which alternate within the network. The first and last layer is always an action layer. All action layers contain an *action module* for each action of task $P \in \mathcal{Q}$ and all propositional layers contain a *proposition module* for each proposition in task P. As this structure depends on the propositions and actions of a specific task, it cannot be reused directly for other tasks. Instead action modules whose actions are derived from the same action schema share the same weight and proposition modules whose propositions are derived from the same predicate share the same weight. These weights are shared among the ASNets of all tasks over the same domain. This means that during training only a single set of weights needs to be learned and then these weights can be used to solve all tasks of a domain. Thus, the set of weights for a task class $\mathcal{Q}$ contains $L \cdot p + (L+1) \cdot a$ weights where L is the number of layers, p is the number of predicates of class $\mathcal{Q}$ and a is the number of action schemas of class $\mathcal{Q}$.

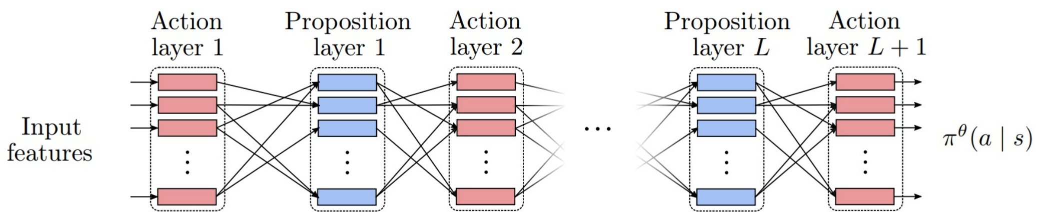


Figure 2.1: Illustration of an L-layer ASNet where each rectangle represents a module in the network. Image taken from Toyer et al. [24].

We say that an ASNet can be initialized for a task P of task class $\mathcal{Q}$ with *ASNet parameters* θ for class $\mathcal{Q}$. The ASNet parameters consist of the set of learned weights (including the bias), the number of layers L, the hidden dimension h_d, the activation function f, and the pooling function *pool*.

An ASNet initialized for a task takes a state (and potentially some other features) as input and yields a *policy* for the task. A policy $\pi : A \times S \to [0,1]$ is a probability distribution over the actions $a \in A$ in state $s \in S$ [24]. An agent following this policy could choose action a in state s with probability $\pi(a|s)$ but since we only consider non-probabilistic tasks it is not necessary for a policy to offer multiple possible actions for one state. Thus we assume that an agent following policy $\pi(a|s)$ always chooses the action a in state s for which $\pi(a|s)$ is maximal.

The connections between the modules in two layers are based on *relatedness* which relies on

the concept of *lifted propositions* [24].[3] A lifted proposition is a predicate with a specific combination of action parameters. So, for example for predicate *on* with arity 2 and action schema *moveFromBlockToBlock*($?toMove, ?currBelow, ?target$) a lifted proposition is $on(?toMove, ?target)$. Each unique lifted proposition has a position within an action schema which is determined by the first occurrence of it within the preconditions or effects of an action schema.[4]
Take for example the action schema *moveFromBlockToBlock*($?toMove, ?currBelow, ?tar$-$get$) with preconditions $\{clear(?toMove), clear(?target), on(?toMove, ?currBelow)\}$, add effects $\{clear(?currBelow), on(?toMove, ?target)\}$ and delete effects $\{clear($ $?target$), $on($ $?toMove$, $?currBelow)\}$. The first lifted proposition then is $clear(?toMove)$, the second is $clear(?target)$ and so on. The last lifted proposition with a position is $on(?toMove, ?target)$ at position 5 because $clear(?target)$ and $on(?toMove, ?currBelow)$ in the delete set are already accounted for in the preconditions and thus no new unique lifted propositions.
We can now define relatedness. A proposition p and an action a are related at position k, written as $R(a, p, k)$, if the lifted proposition corresponding to p is the kth unique lifted proposition mentioned in the action schema from which a is derived. Using the example above, for instance, the action $moveFromBlockToBlock(blockA, blockB, blockC)$ is related to the proposition $on(blockA, blockB)$ at position 3.
An action module in layer l for an action a takes as input vector u_a^l where the hidden representations (the outputs) of all proposition modules in the previous layer whose propositions p are related to a are concatenated. An exception are the action modules in the first layer of an ASNet which take as input vector u_a^1 that concatenates the truth values of the related propositions in current state s, the truth values of the related propositions in the goal and whether action a is applicable in s.[5]
Within an action module the input vector u_a^l is multiplied with weight W_a^l and then a bias b_a^l is added which both correspond to the lth layer of the ASNet and the action schema from which a is derived. The result is then passed through a non-linear activation function f, e. g. tanh, sigmoid or ReLU. So the output of an action module in layer l, corresponding to action a, called the *hidden representation*, is $\phi_a^l = f(W \cdot u_a + b_a^l)$.
The hidden representation of proposition modules are computed analogously, that means $\psi_p^l = f(W_p^l \cdot v_p)$ where p is the proposition corresponding to the module, W_p^l is the weight corresponding to layer l of the ASNet and the predicate from which p is derived and v_p^l is the input vector for the module. Unlike for action modules this input vector v_p^l is not just a concatenation of the hidden representations of action modules from the previous layer whose actions are related to p. All actions derived from the same action schema and related to p

[3] One extension to ASNets of Toyer et al. [24] are skip connections that connect two action layers directly. We do not consider ASNets with skip connections in this work as they do not change our results regarding the compilation between ASNets and generalized potential heuristics.

[4] Since, theoretically, the preconditions and effects are sets we would need to manually number the mentioned lifted propositions. However, in practice these are usually written one after another from which a numbering can be derived.

[5] Toyer et al. [24] extend these first input vectors with additional features about the task and the search to overcome the limited receptive field of ASNets. They use for example information derived from landmarks found by the LM-cut heuristic [8]. However in this work we only consider ASNets without these additional input features.

at the same position k are pooled together into a single vector to keep the input size fixed of each proposition module. Then these vectors are concatenated and used as input vector v_p^l. Possible pooling methods are for example mean pooling (using the mean of all hidden representations) or max pooling (using only the maximal hidden representation).
The last piece needed for an ASNet is the output of the final action layer to obtain a policy for task $P \in \mathcal{Q}$. Instead of a hidden representation vector, each action module with corresponding action a gives a probability $\pi(a|s)$ that action a should be selected in the current state s. To normalize these probabilities and to ensure that only applicable actions have a non-zero probability, a masked softmax activation function is used.

3 Compilability between four Generalized Representations

After introducing the background of our results in the last chapter, this chapter presents in which cases compilations are possible between certain generalized representations and in which cases they are not possible.
The overall aim of our work is to compile generalized representations into other generalized representations with *equivalent behaviour* or to show that such compilations are not possible. For the sake of readability we will often refer to generalized representations with equivalent behaviour simply as being equivalent. What we consider as equivalent behaviour between two generalized representations will be explained in each case. In general, we can think of each representation as a *subgoal generator* that defines subgoal states for each state s which should be reached from s. Equivalent behaviour then means that both representations mark the same states as subgoals for each state s. For policies and many heuristics the subgoals of a state are direct successor states of it. But with policy sketches and heuristics in general the subgoals of a state could include any state.
In Section 3.1 we begin with the compilation from generalized potential heuristics into policy sketches, followed by the opposite direction in Section 3.2. Afterwards, we talk about the compilation from action schema networks into generalized potential heuristics in Section 3.3 and lastly about the opposite direction in Section 3.4.

3.1 Generalized Potential Heuristics into Policy Sketches

The first results we present are about compilations from generalized potential heuristics into policy sketches. As our goal is to preserve equivalent behaviour though, we begin with defining when we consider a generalized potential heuristic and a policy sketch to have equivalent behaviour.
In general we mean with equivalent behaviour of two generalized representations that they mark for each state s the same states as subgoals. For policy sketches this is even formally defined (Definition 20) as the goal states and the states s' for which the state pair $\langle s, s' \rangle$ satisfies a sketch rule or in other words for which $\langle s, s' \rangle$ is compatible with the sketch. In the

context of equivalent behaviour though we use the term subgoal more loosely and usually do not include the goal states in the set of subgoal states of a state.

In contrast to policy sketches it is not defined when a (generalized potential) heuristic considers a state as subgoal of another state. Looking only at the successors of a state s for example, the subgoals could be all states with smaller heuristic value than s or all states that have a minimal heuristic value among all successors of s. The latter viewpoint we will use when comparing action schema networks with generalized potential heuristics in Section 3.3 and Section 3.4. Only looking at successor states however is not reasonable when comparing policy sketches with generalized potential heuristics as sketches can define any state as subgoal of another state. They do not have to be successors or even reachable from s. Likewise, comparing heuristic values is possible for any two states and not only for states and their successors.

Therefore, for our comparison of policy sketches and generalized potential heuristics we say that a GP heuristic h^{GP} defines a state s' as subgoal of s if it has a better (i. e. lower) heuristic value than s, that means if $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$ holds. We then define equivalence of a GP heuristic and a sketch as follows.

Definition 24 (Equivalent behaviour of a policy sketch and a generalized potential heuristic)**.** *A policy sketch R and a generalized potential heuristic h^{GP} show* equivalent behaviour *if for all states s and s' it holds that*

$$h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s') \textit{ iff } \langle s, s' \rangle \textit{ is compatible with } R.$$

So, for each state s the GP heuristic h^{GP} and the sketch R must define exactly the same states s' as subgoals of s. Note that we do not use the term subgoal in the formal definition and, as mentioned before, mean with it here only states for which the state pair $\langle s, s' \rangle$ satisfies a sketch rule and in general not goal states as well.

Instead of above equivalence definition we could also use a less strict one. Currently we consider all states when checking for equivalence. However, we could also take into account only reachable states s and from s reachable states s'. Then it would be irrelevant whether a state s' is a subgoal of a state s or not if we cannot reach s' from s anyway. We do not use this less strict definition though because the cases where a compilation is possible also work for the stricter definition (where all states are considered) and in the cases where a compilation is not possible the presented counterexamples remain counterexamples even if we were using the less strict definition (where only reachable states are considered).

With the above definition of equivalent behaviour we will see that in general we can compile a generalized potential heuristic into a policy sketch with equivalent behaviour. However, we will also show that the compilation is not possible when we require that the compiled sketch must use the same features as the GP heuristic.

Our first result, that the compilation from a generalized potential heuristic into a policy sketch is possible in general, is split into three theorems though. This is the case because sketches are defined only for non-negative integer-valued features while a GP heuristic might use the full real numbers. In the first of the three theorems, we consider the special case of a GP heuristic whose features and weights are in the natural numbers. As a result, the estimates of the GP heuristic also are natural numbers.

Theorem 1. *Let h^{GP} be a generalized potential heuristic for set $\mathcal{F}$ of linear features $f : S \to \mathbb{N}_{\geq 0}$ over the states S from class $\mathcal{Q}$ of planning tasks and weight function $w : \mathcal{F} \to \mathbb{N}_{\geq 0}$. Then there is a terminating policy sketch R_Φ over a suitably defined feature set Φ of linear features for $\mathcal{Q}$ such that R_Φ is equivalent to h^{GP}. That means it holds for all states s and s' in S that*

$$h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s') \textit{ iff } \langle s, s' \rangle \textit{ is compatible with } R_\Phi.$$

Let, furthermore, task class $\mathcal{Q}$ contain only solvable tasks and let h^{GP} be descending and dead-end avoiding in $\mathcal{Q}$.
Sketch R_Φ then has alive sketch width 0.
If, additionally, all tasks of $\mathcal{Q}$ have no dead-ends, then R_Φ also has sketch width 0.

Proof. For the sketch, we can use the heuristic function as the only feature, i.e. $\Phi = \{h^{\mathrm{GP}}\}$. This feature furthermore is linear because h^{GP} is a linear combination of linear features.
We show that $R_\Phi = \{\{\} \to \{h^{\mathrm{GP}}\downarrow\}\}$ has the desired properties, starting with termination.
A sketch is terminating if the rule-set interpreted as policy is terminating, i.e. for every cycle in the policy graph there is a numerical feature in Φ that is decreased along some edge and increased in none. With the given feature, the policy graph has exactly two nodes, one for $h^{\mathrm{GP}} = 0$ and one for $h^{\mathrm{GP}} > 0$ and the only cycle is a self-loop at the node for $h^{\mathrm{GP}} > 0$ corresponding to the only rule of R_Φ. Obviously, feature h^{GP} is decreased along this edge and never increased in the cycle since this is the only edge. Hence, the rule-set is terminating and thus the sketch is terminating as well.
Next, we need to show that R_Φ is equivalent to h^{GP}, i. e. it holds for all states s and s' of all tasks in class $\mathcal{Q}$ that $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$ iff $\langle s, s' \rangle$ is compatible with R_Φ. This holds trivially since the only sketch rule of R_Φ is $\{\} \to \{h^{\mathrm{GP}}\downarrow\}$. So, for each state s, the states s' with $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$ are exactly the states for which $\langle s, s' \rangle$ satisfies the sketch rule $\{\} \to \{h^{\mathrm{GP}}\downarrow\}$ and thus $\langle s, s' \rangle$ is compatible with R_Φ.
It remains to be shown that the alive sketch width of R_Φ is 0 if h^{GP} is descending and dead-end avoiding for class $\mathcal{Q}$ of solvable tasks. Observe that with a dead-end avoiding heuristic, all successor states s' of an alive state s with a lower heuristic estimate than s also must be alive or be a goal state. With a descending heuristic, there always is such a state s' if s is not already a goal state. Since the sketch rule asks for lower heuristic estimates, this means the closest subgoal states of any alive state must again be alive or goal states and thus solvable.
Consider an arbitrary reachable and solvable state s in some task $P \in \mathcal{Q}$. If s is a goal state, the width of the corresponding subproblem is 0 by definition. Otherwise, by the previous discussion, there is a subgoal reachable within one step, thus the width of the corresponding subproblem is again 0 by definition.
Since the subproblems of all reachable and solvable states have width 0, the alive sketch width is 0.
Lastly, we prove that if the tasks of $\mathcal{Q}$ additionally have no dead ends, R_ϕ also has sketch width 0. The argument is analogous to the one for alive sketch width.
Since all tasks in $\mathcal{Q}$ are solvable and have no dead ends, all reachable states of any task in $\mathcal{Q}$ are solvable, and hence also alive if they are not goal states. With a descending heuristic

each such alive state s has a successor s' with lower heuristic estimate. As the sketch rule of R_Φ asks for lower heuristic estimates, this means the closest subgoal states of any alive state must again be alive or goal states and thus be solvable. Note that a heuristic is always dead-end avoiding in a task without dead ends.
Consider an arbitrary reachable state s in some task $P \in \mathcal{Q}$. If s is a goal state, the width of the corresponding subproblem is 0 by definition. Otherwise, by the previous discussion, there is a subgoal reachable within one step, thus the width of the corresponding subproblem is again 0 by definition.
Since the subproblems of all reachable states have width 0, the sketch width of $_\Phi$ is 0. $\square$

The result of alive sketch width 0 can in general not be extended to normal sketch width. Consider for example a task P with three states, initial state s_0, a single goal state s_g and a dead end s_d, where the only two transitions are from s_0 to s_g and from s_0 to s_d. Let h^{GP} assign 0 to s_g and 1 to s_0 and s_d, then h^{GP} satisfies the preconditions of Theorem 1 and the resulting sketch $R_\Phi = \{\{\} \to \{h^{\mathrm{GP}}\downarrow\}\}$ with $\Phi = \{h^{\mathrm{GP}}\}$ has alive width of 0.[6] The sketch width of R_Φ however has no fixed bound because, by Definition 19, the subproblem $P[s_d]$ is unsolvable and thus its width grows with the number of atoms.
For a (alive) sketch width of 0 the tasks of class $\mathcal{Q}$ are assumed to be solvable because a (alive) sketch width of 0 can only be achieved for solvable tasks. Unsolvable tasks have maximal width (which is not fixed) by Definition 18 which then determines the bound on the (alive) sketch width to be maximal and thus not fixed as well. Furthermore h^{GP} needs to be descending and dead-end avoiding to guarantee an alive sketch width of 0. The latter restriction holds trivially for tasks without dead ends. But otherwise h^{GP} needs to be dead-end avoiding explicitly such that unsolvable subproblems cannot be encountered that would cause the alive sketch width to have no fixed bound. While task solvability and h^{GP} being dead-end avoiding is necessary for a fixed bound on the alive sketch width in general, being descending is only needed for the alive sketch width to be 0. With a descending heuristic each alive state has a successor with lower heuristic value which means that each subproblem can be solved within one step and we get an alive sketch width of 0. Dropping this restriction can still lead to a fixed bound on the alive sketch width but the specific value then depends on the task as well as on h^{GP}.
Although generalized potential heuristics are defined for features over the full integer numbers and real-valued weights, we restrict both to be non-negative integer-valued in Theorem 1. This is necessary because sketches are defined only for non-negative integer-valued features. So for the presented compilation h^{GP} must be non-negative integer-valued such that we can use it as feature for R_Φ. However with some adjustments we can get similar results for the full integer numbers and for real numbers.
We first consider the case of non-negative real-valued weights such that h^{GP} can take non-negative real values as well. There are two differences compared to Theorem 1. The first is the weight function which now assigns non-negative real values to the features instead of

[6] The width of $P[g_g]$ is 0 by definition because s_g is a goal state. Furthermore, the only goal state of $P[s_0]$ is s_g because s_d is not solvable and the width of $P[s_0]$ is 0 since its goal is one step away from s_0. No more subproblems are relevant for the alive sketch width in this case because s_d is unsolvable. So the maximal width of all subproblems is 0 and thus the alive sketch width is 0 as well.

non-negative integer values. The features are still in the natural numbers because they must remain non-negative and they are otherwise integer-valued by definition (Definition 10). The second difference is that we cannot guarantee that the features of the compiled sketch are linear.

Theorem 2. *Let h^{GP} be the generalized potential heuristic for set $\mathcal{F}$ of linear features $f : S \to \mathbb{N}_{\geq 0}$ over the states S from class $\mathcal{Q}$ of planning tasks and weight function $w : \mathcal{F} \to \mathbb{R}_{\geq 0}$. Then there is a terminating policy sketch R_Φ over a suitably defined feature set Φ of features for $\mathcal{Q}$ such that R_Φ is equivalent to h^{GP}. That means it holds for all states s and s' in S that*

$$h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s') \textit{ iff } \langle s, s' \rangle \textit{ is compatible with } R_\Phi.$$

Let, furthermore, task class $\mathcal{Q}$ contain only solvable tasks and let h^{GP} be descending and dead-end avoiding in $\mathcal{Q}$,sketch R_Φ then has alive sketch width 0.
If, additionally, all tasks of $\mathcal{Q}$ have no dead ends, then R_Φ also has sketch width 0.

The following proof exploits that we have a fixed class of tasks which also means that we have a fixed set S of states over this class. Then, although the values of h^{GP} can be in the real numbers, h^{GP} has at most $|S|$ different values because we have one value for each state. Because of this we can map h^{GP} to a sketch feature in the natural numbers.

Proof. For the sketch R_Φ we can use $\Phi = \{n\}$ as feature set where numerical feature n ranks each state the same as h^{GP} but in the natural numbers instead of in the non-negative real numbers. It is possible to map h^{GP} to n, that means from the non-negative real numbers into the natural numbers, which we explain in the following argument.
Observe that the state set S of task class $\mathcal{Q}$ is fixed because the class itself is fixed. Thus the GP heuristic h^{GP} (as well as the features of h^{GP}) can take only finitely many (at most $|S|$) different values in class $\mathcal{Q}$. Since h^{GP} is furthermore a subset of $\mathbb{R}_{\geq 0}$ it is a finite total-order (i. e. a well-order). That means there is a mapping from h^{GP} to the natural numbers which is order-preserving (even order-isomorphic). Let f be such a mapping, then $n = f \circ h^{\mathrm{GP}}$ ranks each state the same as h^{GP}. Note that if h^{GP} is descending and dead-end avoiding n interpreted as a heuristic over class $\mathcal{Q}$ is still descending and dead-end avoiding.
With this n as numerical feature then, the proof of Theorem 1 can be applied directly for sketch $R_\Phi = \{\{\} \to \{n\downarrow\}\}$ with one exception. The single feature n is not necessarily linear anymore because depending on f the mapping $n = f \circ h^{\mathrm{GP}}$ might not be computable in time $O(bN)$ as is required by Definition 22. □

This shows that we can compile a generalized potential heuristic into an equivalent policy sketch even if the heuristic uses values in the non-negative real numbers. This compilation though might be rather unpractical because of the mapping f from h^{GP} to the natural numbers which could be quite expensive. Furthermore, this mapping needs to be applied each time the feature n is evaluated. Thus there is room for improvement here which we leave as future work though.
The next theorem restricts the weights to the integers again but allows negative values for the features and weights of h^{GP}. Because of this h^{GP} as well can be negative (but still integer-valued).

Theorem 3. *Let h^{GP} be the generalized potential heuristic for set $\mathcal{F}$ of linear features $f : S \to \mathbb{Z}$ over the states S from class $\mathcal{Q}$ of planning tasks and weight function $w : \mathcal{F} \to \mathbb{Z}$. Then there is a terminating policy sketch R_Φ over a suitably defined feature set Φ of linear features for $\mathcal{Q}$ such that R_Φ is equivalent to h^{GP}. That means it holds for all states s and s' in S that*

$$h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s') \text{ iff } \langle s, s' \rangle \text{ is compatible with } R_\Phi.$$

Let, furthermore, task class $\mathcal{Q}$ contain only solvable tasks and let h^{GP} be descending and dead-end avoiding in $\mathcal{Q}$,sketch R_Φ then has alive sketch width 0.
If, additionally, all tasks of $\mathcal{Q}$ have no dead ends, then R_Φ also has sketch width 0.

Again we exploit that state set S over class $\mathcal{Q}$ is fixed because $\mathcal{Q}$ is fixed. Because of this we can compute a lower bound for the GP heuristic h^{GP} and by adding this lower bound we get a sketch feature (in the natural numbers) which ranks each state the same as h^{GP}.

Proof. For the sketch R_Φ we can use $\Phi = \{n\}$ as feature set where $n = h^{\mathrm{GP}} + |h^{\mathrm{GP}}_{min}|$ and h^{GP}_{min} is a lower bound on the minimal value that h^{GP} can take. Thus n is non-negative and can be used as feature for sketch R_Φ. Note that a possible lower bound h^{GP}_{min} can be computed as follows:
Let $\mathcal{F}$, the set of features of h^{GP}, be divided into the following two sets:

- $\mathcal{F}_1 \subseteq \mathcal{F}$ is the set of features $f \in \mathcal{F}$ with weights $w(f) \geq 0$, and
- $\mathcal{F}_2 \subseteq \mathcal{F}$ is the set of features $f \in \mathcal{F}$ with weights $w(f) < 0$.

A lower bound on the minimum of h^{GP} then is

$$h^{\mathrm{GP}}_{min} = \sum_{f \in \mathcal{F}_1} w(f) \cdot \min_{s \in S} f(s) + \sum_{f \in \mathcal{F}_2} w(f) \cdot \max_{s \in S} f(s).$$

Computing this lower bound can be rather expensive in general. But in practice the minima and maxima of the features are often known or easy to compute in and in these cases a (potentially very rough) lower bound on h^{GP} is feasible to compute. Furthermore, this lower bound needs to be computed only once. When evaluating $n = h^{\mathrm{GP}} + |h^{\mathrm{GP}}_{min}|$ as sketch feature the lower bound h^{GP}_{min} is fixed.
Since h^{GP} is a linear combination of a constant number of linear features it can be computed in linear time and it follows that $n = h^{\mathrm{GP}} + |h^{\mathrm{GP}}_{min}|$ can be computed in linear time as well because h^{GP}_{min} is constant. Thus the only feature of sketch R_Φ is linear.
Note that if h^{GP} is descending and dead-end avoiding n interpreted as a generalized potential heuristic over class $\mathcal{Q}$ is still descending and dead-end avoiding. Thus the remainder of the proof is analogous to the proof of Theorem 1 with $R_\Phi = \{\{\} \to \{n\downarrow\}\}$ instead of $R_\Phi = \{\{\} \to \{h^{\mathrm{GP}}\downarrow\}\}$. □

These last two results show that we can compile non-negative real-valued GP heuristics as well as integer-valued GP heuristics into equivalent policy sketches. Using SIW$_R$ with the resulting sketches leads to behaviour comparable to enforced hill-climbing [9] with the GP heuristic itself. Both algorithms use breadth first searches to find states with lower heuristic value than a current state.[7] While enforced hill-climbing aims explicitly at states with lower heuristic value, SIW$_R$ does this only implicitly. It aims at states that satisfy a sketch rule of sketch R which are just the states with lower heuristic value if R is a sketch compiled from a GP heuristic as described in the proofs of Theorems 1, 2 and 3.

The last two results can be combined to compile a GP heuristic over the full real numbers into an equivalent sketch. The lower bound described in the proof of Theorem 3 can be applied to real-valued GP heuristics as well and then the compilation described in the proof of Theorem 2 can be used. We do not prove this formally though.

In addition, all three of the presented compilations can be altered such that heuristics in general can be compiled into policy sketches. The restriction of linear features would need to be extended to the whole heuristic which limits the pool of applicable heuristics but is necessary if the runtime guarantees of the SIW$_R$ search algorithm should be kept. Furthermore, the compilations could become more expensive, since for example we would need to compute the minimal value of the whole heuristic for the states of task class $\mathcal{Q}$ instead of the minimal values of the features of a GP heuristic.

That the presented compilations rely on class $\mathcal{Q}$ being fixed is a rather unpleasant caveat. Thus, the question also remains open whether compilations are possible when not all conceivable tasks are included in the class. But instead of investigating these topics further we now turn to a different kind of compilation.

So far all presented compilations use the GP heuristic more or less directly as sole feature for the sketch although the GP heuristic itself has features that the sketch could use. So, instead of enveloping a GP heuristic into a sketch we can also ask whether a sketch can express equivalent behaviour if it uses the same features as the underlying GP heuristic.

We will now show that a generalized potential heuristic with feature set $\mathcal{F}$ cannot be compiled into a policy sketch that uses the same features $\mathcal{F}$. This holds even if we restrict the features to be non-negative integers as needed for sketches.

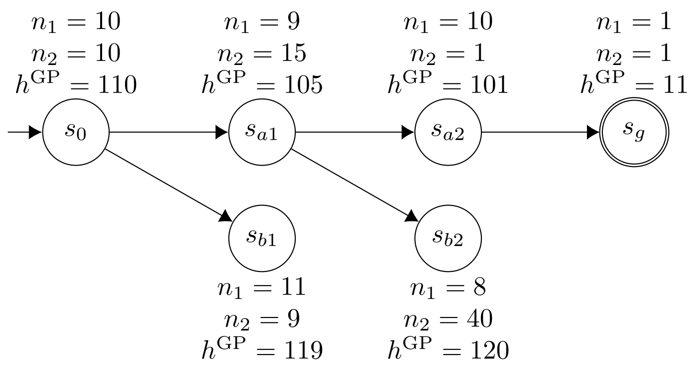
Theorem 4. *There exist a task class $\mathcal{Q}$ and a generalized potential heuristic h^{GP} with feature set $\mathcal{F}$ of linear features $f : S \to \mathbb{N}_{\geq 0}$ over the states S of class $\mathcal{Q}$ and with weight function $w : \mathcal{F} \to \mathbb{N}_{\geq 0}$ such that there is no sketch $R_{\mathcal{F}}$ with the same features $\mathcal{F}$ that is equivalent to h^{GP}. That means it does not hold for all states s and s' of all tasks in $\mathcal{Q}$ that*

$$h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s') \textit{ iff } \langle s, s' \rangle \textit{ is compatible with } R_{\mathcal{F}}.$$

The example proving this theorem in the following exploits the ability of generalized potential heuristics to weigh one feature against another. The policy sketch is not able to replicate this prioritization as it can only define that a numerical feature should increase or decrease but not by how much.

Proof. Let $h^{\mathrm{GP}}(s) = 10 \cdot n_1(s) + 1 \cdot n_2(s)$ be a generalized potential heuristic over task class $\mathcal{Q}$ with feature set $\mathcal{F} = \{n_1, n_2\}$ of non-negative integer-valued features. And let the following be the state space of a task $P \in \mathcal{Q}$ with the mentioned feature values in each state (action names are omitted for simplicity):

[7] SIW$_R$ uses IW instead of a standard breadth-first search like enforced hill-climbing.



We will now prove that for P there is no sketch $R_{\mathcal{F}}$ equivalent to h^{GP} by showing that no sketch $R_{\mathcal{F}}$ can define exactly the same states as subgoals as h^{GP}.
Observe that for state s_0 the successor s_{a1} is defined as a subgoal because we have $h^{\mathrm{GP}}(s_0) > h^{\mathrm{GP}}(s_{a1})$ while s_{b1} is not defined as a subgoal because of $h^{\mathrm{GP}}(s_0) < h^{\mathrm{GP}}(s_{b1})$. From this we can infer that policy sketch $R_{\mathcal{F}}$ must have a sketch rule whose feature effect set tells that n_1 must decrease or n_2 must increase. There are three possible effect sets that capture this idea: $\{n_1{\downarrow}, n_2{\uparrow}\}$, $\{n_1?, n_2{\uparrow}\}$ and $\{n_1{\downarrow}, n_2?\}$. So, to ensure that $R_{\mathcal{F}}$ defines s_{a1} as subgoal of s_0 but not s_{b1} we need a sketch rule with one of these effect sets.
Analogously, for state s_{a1} the successor s_{a2} is defined as a subgoal because $h^{\mathrm{GP}}(s_{a1}) > h^{\mathrm{GP}}(s_{a2})$ holds and s_{b2} is not defined as a subgoal because of $h^{\mathrm{GP}}(s_{a1}) < h^{\mathrm{GP}}(s_{b2})$. Hence, $R_{\mathcal{F}}$ must have a sketch rule whose effect set tells that n_1 must increase or n_2 must decrease. Again there are three possible effect sets to achieve this: $\{n_1{\uparrow}, n_2{\downarrow}\}$, $\{n_1?, n_2{\downarrow}\}$ and $\{n_1{\uparrow}, n_2?\}$. Note that this effect set triple is completely different compared to the triple mentioned before. They are even orthogonal in a sense because the first triple tells that n_1 must decrease while n_2 increases and the second triple tells the opposite, n_1 must increase while n_2 decreases.
Therefore, policy sketch $R_{\mathcal{F}}$ must have two rules, one with an effect set of the first triple, which we will call r_1, and one with an effect set of the second triple, which will call r_2. The feature conditions of both rules have to be $\{\}$, $\{n_1 > 0\}$, $\{n_2 > 0\}$, or $\{n_1 > 0, n_2 > 0\}$ but which one of these is chosen does not matter for P since all its states satisfy these conditions (the empty one is trivially satisfied by any state).
As $h^{\mathrm{GP}}(s_0) > h^{\mathrm{GP}}(s_{a1})$ requires, we now have that $\langle s_0, s_{a1}\rangle$ is compatible with $R_{\mathcal{F}}$ because of sketch rule r_1 and for $h^{\mathrm{GP}}(s_{a1}) > h^{\mathrm{GP}}(s_{a2})$ we have that $\langle s_{a1}, s_{a2}\rangle$ is compatible with $R_{\mathcal{F}}$ because of sketch rule r_2. However sketch $R_{\mathcal{F}} = \{r_1, r_2\}$ is not equivalent to h^{GP}. That is, we also have that $\langle s_0, s_{b1}\rangle$ is compatible with $R_{\mathcal{F}}$ because of sketch rule r_2. But this is not allowed since $h^{\mathrm{GP}}(s_0) < h^{\mathrm{GP}}(s_{b1})$.
Therefore, it does not hold for all states s and s' of $P \in \mathcal{Q}$ that $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$ iff $\langle s, s'\rangle$ is compatible with $R_{\mathcal{F}}$. And since we covered all options for sketch $R_{\mathcal{F}}$[8] this shows that there is no equivalent sketch for h^{GP} over task $P \in \mathcal{Q}$ that uses the same features as h^{GP}. $\square$

[8] We covered all options for the two rules r_1 and r_2 but adding more rules does not change the result. All further rules must use one of the mentioned effect sets and condition sets, so they will be like r_1 or r_2 and add no further information.

This shows that we cannot compile a generalized potential heuristic into an equivalent policy sketch when both must use the same features. The ability of GP heuristics to aggregate and weigh feature values against each other can in general not be replicated with policy sketches. So the first three presented compilations showed that policy sketches can express the same as generalized potential heuristics given the correct features. However, in all three cases the features were just the heuristics themselves. The compiled sketches used the GP heuristics directly instead of replicating their behaviour. The last result of this section in contrast showed that the compilation is not possible when the GP heuristic and the sketch must use the same features. Hence, given the same features a policy sketch is no more expressive than a generalized potential heuristic.
This raises the question whether a compilation is possible in the opposite direction. That means can we compile a policy sketch into an equivalent generalized potential heuristic that uses the same features? This would show that, given the same features, GP heuristics are more expressive than sketches. Among other results this question is answered in the next section.

3.2 Policy Sketches into Generalized Potential Heuristics

In this section we investigate if and how we can compile policy sketches into generalized potential heuristics. We will see that the compilation is only possible for a relaxed equivalence definition and is not possible when the same features must be used.
As before (Definition 24), we consider a policy sketch and a generalized potential heuristic equivalent if it holds for all states s and s' of all tasks in $\mathcal{Q}$ that

$$h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s') \text{ iff } \langle s, s' \rangle \text{ is compatible with } R_\Phi.$$

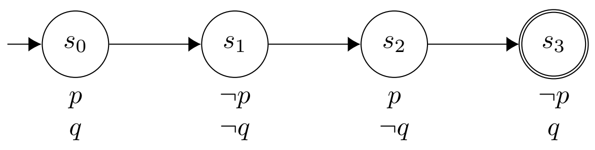
The first result of this section shows that in general we cannot compile a policy sketch into an equivalent generalized potential heuristic.

Theorem 5. *There exist a task class $\mathcal{Q}$ and a policy sketch R_Φ with feature set Φ over the states of class $\mathcal{Q}$ such that there is no generalized potential heuristic h^{GP} that is equivalent to R_Φ. That means it does not hold for all states s and s' of all tasks in $\mathcal{Q}$ that*

$$h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s') \textit{ iff } \langle s, s' \rangle \textit{ is compatible with } R_\Phi.$$

The example in the following proof exploits that a policy sketch can define a state as subgoal of some state and at the same time define it to not be a subgoal of another state. We will see that in the presented case a generalized potential heuristic is not able to replicate this behaviour.

Proof. Let $\mathcal{Q}$ be a class of planning tasks and let $R_\Phi = \{\{p, q\} \rightarrow \{\neg p, \neg q\}, \{\neg p, \neg q\} \rightarrow \{p\}\}$ be a sketch with features $\Phi = \{p, q\}$ (both p and q are boolean) over the states of class $\mathcal{Q}$. Furthermore, let the following be the state space of a task $P \in \mathcal{Q}$ where each state is annotated with the feature values of it ($p(s) = \top$ is shortened to p and $p(s) = \bot$ is shortened to $\neg p$ same as for q):



We will now show that there is no generalized potential heuristic h^{GP} for this task and sketch such that it holds for all states s and s that $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$ iff $\langle s, s'\rangle$ is compatible with R_Φ.
Observe that the state pair $\langle s_0, s\rangle$ is compatible with R_Φ only for $s = s_1$. That means, the only subgoal which is not a goal state of s_0 is s_1. Thus for the generalized potential heuristic h^{GP} to be equivalent to R_Φ it must hold that

$h^{\mathrm{GP}}(s_0) > h^{\mathrm{GP}}(s_1)$,

$h^{\mathrm{GP}}(s_0) \leq h^{\mathrm{GP}}(s_2)$, and

$h^{\mathrm{GP}}(s_0) \leq h^{\mathrm{GP}}(s_3)$.

Furthermore, since $\langle s_1, s\rangle$ is compatible with R_Φ only for $s = s_2$, we have

$h^{\mathrm{GP}}(s_1) > h^{\mathrm{GP}}(s_2)$,

$h^{\mathrm{GP}}(s_1) \leq h^{\mathrm{GP}}(s_0)$, and

$h^{\mathrm{GP}}(s_1) \leq h^{\mathrm{GP}}(s_3)$.

From $h^{\mathrm{GP}}(s_0) > h^{\mathrm{GP}}(s_1)$ and $h^{\mathrm{GP}}(s_1) > h^{\mathrm{GP}}(s_2)$ we can then infer that $h^{\mathrm{GP}}(s_0) > h^{\mathrm{GP}}(s_2)$ holds. But this is not possible since, as seen above, we must also have $h^{\mathrm{GP}}(s_0) \leq h^{\mathrm{GP}}(s_2)$. Hence, it does not hold for all states s and s' of P that $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$ holds iff $\langle s, s'\rangle$ is compatible with R_Φ. That means, for the above task P and sketch R_Φ there is no generalized potential heuristic that is equivalent to R_Φ. $\square$

So, in general we cannot compile a policy sketch into an equivalent generalized potential heuristic. From this it directly follows that we also cannot compile a sketch into a GP heuristic where both use the same features. This answers the question at the end of the last section whether GP heuristics are more expressive than policy sketches when both use the same features. Neither is more expressive than the other since, given the same features, we cannot compile them into each other.
Furthermore, this negative result for the compilation of sketches into GP heuristics even holds for heuristics in general. The contradiction derived in the proof does not use any specifics of generalized potential heuristics.
The proof exploits that with our definition of equivalence the subgoals must be preserved exactly. In some cases this is impossible for a heuristic as it might "accidentally" mark a state as subgoal for one state although the state was defined as subgoal for another state. This happened in the example of above proof. Because s_1 is a subgoal of s_0 (by the first rule of sketch R_Φ) the heuristic value must decrease from s_0 to s_1 and because s_2 is a subgoal of s_1 (by the second rule of R_Φ) its heuristic value must be even lower. So, s_2 also has a lower heuristic value than s_0 and is marked in a sense transitively as subgoal of s_0. This however,

is not allowed as no sketch rule explicitly defines s_2 as subgoal of s_0 and our equivalence definition requires that subgoals (and non-subgoals) are preserved exactly.
Heuristics cannot differentiate in this case in contrast to policy sketches which can declare a state as subgoal or non-subgoal depending on the current state. At least in the presented example, this differentiation however is not necessary. Defining s_2 as subgoal of s_0 makes sense in this case as it must be passed on the path towards the goal. So with sketches in general it might happen that a state close to the goal is declared as non-subgoal because of the current state. But this does not necessarily mean that a heuristic should give this state a high (i. e. "bad") heuristic value.
Because of that we relax the equivalence requirement in the following and do not anymore demand that the subgoals are preserved exactly. It now suffices that the heuristic value improves, i. e. decreases, between two states if those states satisfy a sketch rule. We can think of this as *similar behaviour* instead of equivalent behaviour. The following theorem and proof show that in this case a compilation is possible.

Theorem 6. *Let R_Φ be a terminating sketch over linear features Φ for class $\mathcal{Q}$ of tasks. Then there is a generalized potential heuristic h^{GP} over a suitably defined feature set $\mathcal{F}$ and weight function $w : \mathcal{F} \to \mathbb{R}$ such that it holds for all states s and s' of class $\mathcal{Q}$ that*

$$\textit{if } \langle s, s' \rangle \textit{ is compatible with } R_\Phi \textit{ then we have } h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s').$$

*Let additionally R_Φ have an alive sketch width of 0 and let for all alive states s the set $G^*_R(s)$ contain at least one s' such that the pair $\langle s, s' \rangle$ is compatible with R_Φ.
Then is h^{GP} also descending and dead-end avoiding.*

Proof. For the feature set $\mathcal{F}$ of the generalized potential heuristic h^{GP} we can use the following: for each possible feature valuation $f_{\Phi,i}$ over the sketch features Φ of R_Φ, the feature set $\mathcal{F}$ of h^{GP} contains a feature f_i that is 1 if $f_{\Phi,i}$ is true in a state s and 0 otherwise. Note that because the features in Φ are linear, the size of $\mathcal{F}$ is bounded by $O(N^{|\Phi|})$ where N is the maximal number of atoms over the tasks in $\mathcal{Q}$.
With this feature set $\mathcal{F}$ for h^{GP} exactly one feature $f_i \in \mathcal{F}$ is true in each state s because f_i is associated with the feature valuation $f_{\Phi,i} = f_\Phi(s)$. It follows that for every state s of a task from $\mathcal{Q}$, we have $h^{\mathrm{GP}}(s) = w(f_i)$ for feature $f_i \in \mathcal{F}$ associated with feature valuation $f_{\Phi,i} = f_\Phi(s)$.
We can determine the weights w of each feature $f_i \in \mathcal{F}$ with the following algorithm:

```
1:  Let $V$ be the set of all feature valuations $f_{\Phi,i}$
2:  Initialize $w : \mathcal{F} \to \mathbb{R}$
3:  for $n = 0, -1, -2, -3, \dots$ do
4:      $remove := \emptyset$
5:      for all $f_{\Phi,i} \in V$ do
6:          if there is no $f_{\Phi,j} \in V$ such that $\langle f_{\Phi,j}, f_{\Phi,i} \rangle$ satisfies a sketch rule of $R_\Phi$ then
7:              $w(f_i) := n$
8:              $remove := remove \cup \{f_{\Phi,i}\}$
9:      $V := V \setminus remove$
10:     if $V = \emptyset$ then
11:         return $w$
```

Note that in line 7, f_i is the feature from $\mathcal{F}$ associated with feature valuation $f_{\Phi,i}$. Intuitively, the above algorithm selects in each iteration all current "worst" feature valuations, that means feature valuations that the sketch considers to be no more beneficial than the remaining ones in V. After setting their weights to the current maximal value n these "worst" feature valuations are removed from V.

Observe that for a terminating sketch this algorithm finishes in a finite number of iterations. Since a terminating sketch induces a strict partial order over the feature valuations (Theorem 33 of Bonet and Geffner [3]), following such a sketch never leads to cycles of feature valuations. Thus, there are feature valuations that are *maximal* in the sense that they are never mentioned as second component of all feature valuation pairs that satisfy a sketch rule. In other words, the sketch never defines these feature valuations as subgoals.

In the first iteration of above algorithm the weights of the h^{GP}-features associated with these maximal feature valuations are set to 0 and the feature valuations are removed from V (maximal feature valuations are never mentioned as second component and thus make the condition in line 6 of above algorithm true). Removing feature valuations from V does not change the fact that a terminating sketch induces a strict partial order over the feature valuations remaining in V. Hence, there are again maximal feature valuations. The next iteration of the algorithm then sets the corresponding weights to -1 and removes these maximal feature valuations from V.

This argument can be repeated until V is empty because after each iteration there will be other feature valuations that are maximal and thus going be removed from V. Since there is a finite number of feature valuations in V this means that V becomes empty after a finite number of iterations and the algorithm will terminate.

We will now prove that with features and weights defined that way it holds for the resulting generalized potential heuristic h^{GP} and two states s and s' from a task in $\mathcal{Q}$ that $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$ if $\langle f_\Phi(s), f_\Phi(s')\rangle$ is compatible with R_Φ.

Let $f_{\Phi,i} = f_\Phi(s)$ hold, i. e. $f_{\Phi,i}$ is the feature valuation that is true in state s, for the feature valuation f_Φ over the features Φ from sketch R_Φ and for a state s from a task in $\mathcal{Q}$. Let analogously $f_{\Phi,j} = f_\Phi(s')$ hold and let $\langle f_{\Phi,i}, f_{\Phi,j}\rangle$ satisfy a sketch rule from R_Φ. With the compilation described above we have for $f_{\Phi,i}$ the associated feature $f_i \in \mathcal{F}$ and weight $w(f_i)$ and analogously for $f_{\Phi,j}$ we have $f_j \in \mathcal{F}$ and $w(f_j)$.

Since $\langle f_{\Phi,i}, f_{\Phi,j}\rangle$ satisfies a sketch rule, we know that with the algorithm described above $f_{\Phi,j}$ cannot be removed from V while $f_{\Phi,i}$ is still in V. (As long as there is any $f_{\Phi,i}$ still in V such that $\langle f_{\Phi,i}, f_{\Phi,j}\rangle$ satisfies a sketch rule, the condition in line 6 is not satisfied and $f_{\Phi,j}$ is not removed from V.) Thus $f_{\Phi,i}$ must be removed in an earlier iteration than $f_{\Phi,j}$ which also means that $w(f_i)$ is set to the value of an earlier iteration than $w(f_j)$. As we know from discussing the termination of the algorithm, in each iteration at least one feature valuation is removed from V if R_Φ is terminating. Thus, indeed both feature valuations are removed from V eventually and also $f_{\Phi,i}$ is removed before $f_{\Phi,j}$.

From this it follows that $w(f_i) > w(f_j)$ is true because earlier iterations are associated with larger numbers. Furthermore, as mentioned previously, with the described compilation we have $h^{\mathrm{GP}}(s) = w(f_i)$ for feature $f_i \in \mathcal{F}$ associated with feature valuation $f_{\Phi,i} = f_\Phi(s)$. Hence, since $w(f_i) > w(f_j)$ holds, also $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$ holds.

It remains to be shown that h^{GP} is descending and dead-end avoiding if R_Φ has an alive sketch width of 0 and $G^*_R(s)$ contains for all alive states s at least one state s' such that the pair $\langle f_\Phi(s), f_\Phi(s')\rangle$ satisfies a sketch rule of R_Φ. We begin with proving dead-end avoidance. A heuristic is dead-end avoiding if all successors s' of an alive state s with $h(s) > h(s')$ are solvable. Since we already showed that $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$ if $\langle f_\Phi(s), f_\Phi(s')\rangle$ satisfies a sketch rule of R_Φ, we need to prove that s' is solvable for s' being a successor of an alive state s and $\langle f_\Phi(s), f_\Phi(s')\rangle$ satisfying a sketch rule of R_Φ.
Let s and s' be such states, then we know that $s' \in G^*_R(s)$ because there cannot be subgoal states that are closer than successors of s (also s cannot be in $G^*_R(s)$ because R_Φ is terminating). Furthermore, we know that for all alive s the set of closest subgoals $G^*_R(s)$ does not contain unsolvable states because R_Φ has a alive sketch width of 0 (a fixed bound on the alive sketch width suffices for this argument).
Hence, because s' is in $G^*_R(s)$ of alive state s it follows that s' is solvable and thus h^{GP} is dead-end avoiding.
Lastly, we show that h^{GP} is descending which is the case if every alive state s has at least one successor s' with $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$. As for dead-end avoidance it suffices to show that every alive state s has at least one successor s' with $\langle f_\Phi(s), f_\Phi(s')\rangle$ satisfying a sketch rule of R_Φ.
Because R_Φ has an alive sketch width of 0, we know that for all alive states s the states in $G^*_R(s)$ are reachable within one step and $G^*_R(s)$ is not empty (the last part only requires bounded alive sketch width). So, the states in $G^*_R(s)$ are successors of alive state s. Furthermore, for all alive s the set $G^*_R(s)$ contains at least one state s' such that $\langle f_\Phi(s), f_\Phi(s')\rangle$ satisfies a sketch rule of R_Φ (this is a requirement of the theorem and prevents that $G^*_R(s)$ only contains goal states). Thus, there is a successor s' of alive state s such that $\langle f_\Phi(s), f_\Phi(s')\rangle$ satisfies a sketch rule of R_Φ which proves that h^{GP} is descending. □

This shows that when only requiring similar behaviour instead of equivalent behaviour we can compile a policy sketch into a generalized potential heuristic. But the presented compilation is rather expensive and the question arises whether there is a more efficient compilation. One approach to find a more efficient compilation than in Theorem 6 could be to use the boolean feature valuations of a sketch as the features of the GP heuristic instead of all feature valuations. This does not work, however. Take for example a sketch $R_\Phi = \{\{\} \rightarrow \{n\downarrow\}\}$ with single numerical feature n. Since there are only two boolean feature valuations $n > 0$ and $n = 0$ which are furthermore mutually exclusive, the GP heuristic would have only two different values and it would not be able to differentiate any two states with $n > 0$. So, for example a state with $n = 10$ would be treated the same as a state with $n = 5000$ although the sketch clearly marks the first state as a subgoal of the second one.
A more promising option might be a compilation where we keep the features of the sketch. In this case we could differentiate above states with $n = 10$ and $n = 5000$ because feature n (multiplied with some weight) is the whole GP heuristic. However, the next result shows that such a compilation also is not possible in general.

Theorem 7. *There exist a task class $\mathcal{Q}$ and a policy sketch R_Φ with feature set Φ over the states of class $\mathcal{Q}$ such that there is no generalized potential heuristic h^{GP} with the same* *features Φ for which the following holds for all states s and s' of $\mathcal{Q}$:*

$$\textit{if } \langle s, s' \rangle \textit{ is compatible with } R_\Phi \textit{ then } h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s').$$

The example in the following proof exploits that a policy sketch can declare a feature increase beneficial in some states and a decrease of the same feature beneficial in other states. We will show that a generalized potential heuristic cannot express this when using the same features as the sketch.

Proof. Let $\mathcal{Q}$ be a class of planning tasks and let $R_\Phi = \{\{p\} \to \{n{\downarrow}\}, \{\neg p, n = 0\} \to \{n{\uparrow}\}\}$ be a sketch with features $\Phi = \{p, n\}$ (p is boolean and n numerical) over the states of class $\mathcal{Q}$. Furthermore, let the following be the state space of a task $P \in \mathcal{Q}$ where each state is annotated with its feature values ($p(s) = \top$ is shortened to p and $p(s) = \bot$ is shortened to $\neg p$):

$$\begin{tikzcd}
{} \arrow[r] & s_0 \arrow[r] & s_1 \arrow[r] & s_2 \arrow[r] & s_3 \\
& \begin{matrix} p \\ n = 10 \end{matrix} & \begin{matrix} p \\ n = 0 \end{matrix} & \begin{matrix} \neg p \\ n = 0 \end{matrix} & \begin{matrix} \neg p \\ n = 100 \end{matrix}
\end{tikzcd}$$

We will now show that there is no generalized potential heuristic for this task and sketch such that it holds for all states s and s that if $\langle s, s' \rangle$ is compatible with R_Φ then we have $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$.
Since the GP heuristic h^{GP} must use the same features as R_Φ we know that it is of the form

$$h^{\mathrm{GP}}(s) = w_p \cdot p(s) + w_n \cdot n(s)$$

where w_p and w_n are the weights of the respective features that are yet to be determined and n maps states to non-negative integers (because it is derived from a sketch feature). Although we use the same letter here, the heuristic feature p is not exactly the same as the sketch feature p because we need a number for the heuristic while the sketch feature returns $\top$ or $\bot$ for a state. Common choices are 0 or -1 in case of $p = \bot$ and 1 in case of $p = \top$. For the following argument we just require that $p(s) > p(s')$ holds if we have $p = \top$ in s and $p = \bot$ in s' (the reverse would be fine as well since weight w_p can compensate this).
Observe that because $\langle s_0, s_1 \rangle$ is compatible with R_Φ it must hold that $h^{\mathrm{GP}}(s_0) > h^{\mathrm{GP}}(s_1)$. From this we can infer that $w_n \cdot 10 > w_n \cdot 0$ holds because we have $p(s_0) = p(s_1)$ (feature p is true in both states).[9] From $w_n \cdot 10 > w_n \cdot 0$ it follows that we have $w_n > 0$.
Similarly, because $\langle s_2, s_3 \rangle$ is compatible with R_Φ it must hold that $h^{\mathrm{GP}}(s_2) > h^{\mathrm{GP}}(s_3)$ and from that we know that $w_n \cdot 0 > w_n \cdot 100$ holds because we have $p(s_2) = p(s_3)$ (feature p is false in both states). Hence we must also have $w_n < 0$.
But this is a contradiction as we cannot have $w_n > 0$ and $w_n < 0$ at the same time. Therefore, for task P and sketch R_Φ there is no generalized potential heuristic over the same features as R_Φ such that it holds for all states s and s' that if $\langle s, s' \rangle$ is compatible with R_Φ then we have $h^{\mathrm{GP}}(s) > h^{\mathrm{GP}}(s')$. $\square$

[9] $h^{\mathrm{GP}}(s_0) = w_p \cdot p(s_0) + w_n \cdot n(s_0) = w_p \cdot p(s_0) + w_n \cdot 10$
$h^{\mathrm{GP}}(s_1) = w_p \cdot p(s_1) + w_n \cdot n(s_1) = w_p \cdot p(s_1) + w_n \cdot 0$

This shows that even with a less strict behaviour comparison than in Theorem 5 we cannot compile a policy sketch into a generalized potential heuristic that uses the same features.
The question whether there is a more efficient compilation than the one presented in Theorem 6 remains open though. However, as the other presented compilation results in this section are all negative there might not be an efficient compilation at all. Furthermore, as we proved equivalence impossible, we would only get a generalized potential heuristic with similar behaviour anyway. So whether such a compilation would be relevant is an open question as well. In the future it might be more promising to investigate under which restrictions we can efficiently compile a policy sketch into an equivalent generalized potential heuristic or equivalent heuristic in general.
Summing up our compilation results of the last two sections we can say that in general policy sketches are more expressive than generalized potential heuristics (Theorems 1, 2, 3 and 5) but when restricting both to use the same features neither is more expressive than the other (Theorems 4 and 5). A compilation from sketches into GP heuristics though becomes possible when asking only for similar behaviour and not requiring both to use the same features (Theorems 6 and 7).
With this we finish our comparison of policy sketches and generalized potential heuristics and move on to compilations between action schema networks and generalized potential heuristics.

3.3 Action Schema Networks into Generalized Potential Heuristics

While the last two sections were concerned with the compilation between generalized potential heuristics and policy sketches, the following two sections talk about the compilation between generalized potential heuristics and action schema networks (ASNets).
Since we aim for equivalent behaviour with our compilations we first need to define when some ASNet parameters and a generalized potential heuristic are equivalent. In the following we abuse notation for the sake of readability and denote the set $\{s' \mid \exists s, a : \arg\max_{\langle s,a,s'\rangle \in T} \pi(a|s)\}$ as $\arg\max_{s':\langle s,a,s'\rangle \in T} \pi(a|s)$.

Definition 25 (Equivalent behaviour of some ASNet parameters and a generalized potential heuristic)**.** *Given a task class $\mathcal{Q}$, we say that a generalized potential heuristic h^{GP} and some ASNet parameters θ have* equivalent behaviour *if for all states s of all tasks P in $\mathcal{Q}$ it holds that*

$$\arg\max_{s':\langle s,a,s'\rangle \in T} \pi(a|s) = \arg\min_{s':\langle s,a,s'\rangle \in T} h^{\mathrm{GP}}(s')$$

where T is the set of transitions of the state space induced by task P and π is the policy of the ASNet initialized for P with θ.

That means the successors s' with minimal h^{GP}-value must exactly be the states that can be reached with actions a for which $\pi(a|s)$ is maximal. So, policy π declares a successor as subgoal if its policy value is maximal and heuristic h^{GP} marks a successor as subgoal if its heuristic value is minimal among all successors.
In contrast to the comparison of sketches and GP heuristics, we only consider successor states and not arbitrary state pairs here. While we can compare the heuristic values of two

arbitrary states (as done in the last two sections), a (ASNet) policy is only concerned with the successors of each state. More precisely, it selects for each state the action to apply in this state. That is why we chose successors as basis for the comparison of GP heuristics and ASNets instead of arbitrary state pairs.
Furthermore, in the previous two section states with lower heuristic value were considered subgoals when comparing sketches and GP heuristics. Now instead, we consider the (successor) states with minimal heuristic value as subgoals and compare them against the states for which the policy is maximal. We chose this as basis for our comparison of ASNets and GP heuristics as it better reflects that there can be qualitative differences between the successors of a state. When looking at two successors of the same state the successor with lower heuristic value, or respectively with higher policy, value is preferable compared to the other successor.
Using the above equivalence definition, we will now show that in general it is not possible to compile some ASNet parameters into an equivalent generalized potential heuristic.

Theorem 8. *There exists a task class $\mathcal{Q}$ with a task $P \in \mathcal{Q}$ and ASNet parameters θ such that there is no generalized potential heuristic h^{GP} over $\mathcal{Q}$ that is equivalent to θ. That means there is no h^{GP} for which the following holds for all states s in task P:*

$$\arg \max_{s' : \langle s,a,s' \rangle \in T} \pi(a|s) = \arg \min_{s' : \langle s,a,s' \rangle \in T} h^{\mathrm{GP}}(s'),$$

where T are the transitions of the state space induced by P and π is the policy of the ASNet initialized for P with θ.

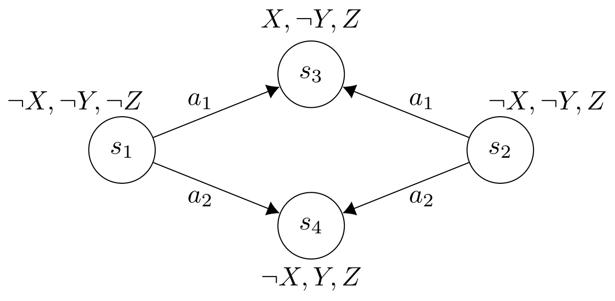
The following proof shows this with an example. If two states s_1 and s_2 have the same two successors s_3 and s_4 then the policy of an ASNet can select either s_3 or s_4 (expressed formally, it picks the actions with which these states can be reached) depending on whether the current state is s_1 or s_2. With a generalized potential heuristic the same state is picked no matter whether the current state is s_1 or s_2 because the heuristic value of the successors is not influenced by the current state (the current state only influences which states are successors).
For a single task this is might be irrelevant as we will always reach either s_1 or s_2 but never the other one when following a heuristic and always choosing the successor with minimal heuristic value, e. g. with steepest-ascent hill-climbing. So, we never observe the choice of the heuristic for the other state. But for a class of planning tasks this becomes relevant. Take for example a class with a task where s_1 is the initial state and another task where s_2 is the initial state. Then, we can observe for the first task which successor the heuristic chooses in s_1 and we can observe for the second task which successor the heuristic chooses for s_2.

Proof. We proof this with an example. Consider a task class $\mathcal{Q}$ over the domain $\langle \{X, Y, Z\}, \{A_1, A_2\} \rangle$ where the predicates X, Y and Z as well as both action schemas A_1 and A_2 are nullary (i. e. they do not take any arguments). The actions obtained from grounding A_1 and A_2 are

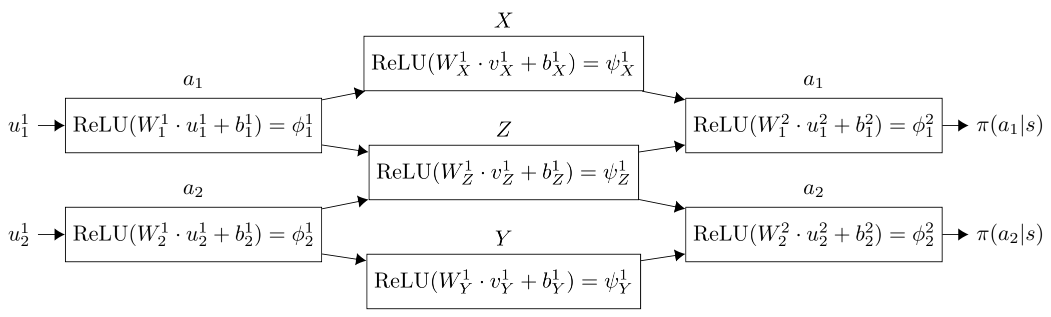
$A_1() = \langle \{\}, \{X(), Z()\}, \{\} \rangle$ and

$A_2() = \langle\{\}, \{Y(), Z()\}, \{\}\rangle$.

So, the actions derived from both action schemas A_1 and A_2 have no preconditions and their only effects are that $Z()$ is made true and $X()$ or respectively $Y()$ is made true.
Since all predicates and actions are nullary we slightly abuse notation in the following and use X, Y, Z directly as propositions. The actions $A_1()$ and $A_2()$ we will call a_1 and a_2.
Let P be a planning task in $\mathcal{Q}$ whose state space contains the following subgraph:



The ASNet for which we will show that its parameters cannot be compiled into a GP heuristic comes next. The idea of this ASNet is that whether action a_1 or a_2 is chosen only depends on the value of Z in the current state s. This is achieved by choosing the parameters such that the output for action a_1 is 0 if Z is true and it is 1 if Z is false. And the output for action a_2 is mirrored, that means it is 1 if Z is true and it is 0 if Z is false. We first show the structure of the ASNet and then list its notation and parameters.
Let the following be an ASNet for task P:


X
$\mathrm{ReLU}(W_X^1 \cdot v_X^1 + b_X^1) = \psi_X^1$
a_1
a_1
u_1^1
$\mathrm{ReLU}(W_1^1 \cdot u_1^1 + b_1^1) = \phi_1^1$
Z
$\mathrm{ReLU}(W_1^2 \cdot u_1^2 + b_1^2) = \phi_1^2$
$\pi(a_1|s)$
$\mathrm{ReLU}(W_Z^1 \cdot v_Z^1 + b_Z^1) = \psi_Z^1$
a_2
a_2
u_2^1
$\mathrm{ReLU}(W_2^1 \cdot u_2^1 + b_2^1) = \phi_2^1$
Y
$\mathrm{ReLU}(W_2^2 \cdot u_2^2 + b_2^2) = \phi_2^2$
$\pi(a_2|s)$
$\mathrm{ReLU}(W_Y^1 \cdot v_Y^1 + b_Y^1) = \psi_Y^1$

Notation

- $\mathrm{ReLU}(x) = max(0, x)$
- x, y and z are 1 if X, Y and Z, respectively, are true in current state s and 0 if they are false in s
- g_X, g_Y and g_Z are 1 if X, Y and Z, respectively, are true in the goal and 0 otherwise
- m_i is 1 if action a_i is applicable in current state s ($i \in \{1, 2\}$) and 0 if it is not applicable

First action layer

- $u_1^1 = \begin{pmatrix} x \\ z \\ g_X \\ g_Z \\ m_1 \end{pmatrix}$, $W_1^1 = \begin{pmatrix} 0 & -1 & 0 & 0 & 0 \end{pmatrix}$, $b_1^1 = 1$, thus $\phi_1^1 = -z + 1$
- $u_2^1 = \begin{pmatrix} y \\ z \\ g_Y \\ g_Z \\ m_2 \end{pmatrix}$, $W_2^1 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \end{pmatrix}$, $b_2^1 = 0$, thus $\phi_2^1 = z$

Proposition layer

- $v_X^1 = \phi_1^1$, $W_X^1 = 0$, $b_X^1 = 0$, thus $\psi_X^1 = 0$
- $v_Y^1 = \phi_2^1$, $W_Y^1 = 0$, $b_Y^1 = 0$, thus $\psi_Y^1 = 0$
- $v_Z^1 = \begin{pmatrix} \phi_1^1 \\ \phi_2^1 \end{pmatrix}$, $W_Z^1 = \begin{pmatrix} 0 & 1 \end{pmatrix}$, $b_Z^1 = 0$, thus $\psi_Z^1 = z$

Second (final) action layer

- $u_1^2 = \begin{pmatrix} \psi_X^1 \\ \psi_Z^1 \end{pmatrix}$, $W_1^2 = \begin{pmatrix} 0 & -1 \end{pmatrix}$, $b_1^2 = 1$, thus $\phi_1^2 = -z + 1$
- $u_2^2 = \begin{pmatrix} \psi_Y^1 \\ \psi_Z^1 \end{pmatrix}$, $W_2^2 = \begin{pmatrix} 0 & 1 \end{pmatrix}$, $b_2^2 = 0$, thus $\phi_2^2 = z$

Observe that the unique maximum of this ASNet's policy π occurs for action a_1 if Z is false in the current state and it occurs for action a_2 if Z is true in the current state because the output of the last a_1-action-module is $\phi_1^2 = -z + 1$ and the output of the last a_2-action-module is $\phi_2^2 = z$. In other words, since both actions a_1 and a_2 are always applicable, policy π chooses action a_1 if Z is true in the current state and otherwise, if Z is false, it chooses action a_2.
In particular, for above task P, the unique maximum of π in state s_1 is $\pi(a_1|s_1)$ because Z if false in s_1 and it is $\pi(a_2|s_2)$ in state s_2 because Z is true in s_2. That means, we have

$$\arg \max_{s':\langle s_1,a,s'\rangle \in T} \pi(a|s_1) = \{s_3\} \text{ and}$$
$$\arg \max_{s':\langle s_2,a,s'\rangle \in T} \pi(a|s_2) = \{s_4\}.$$

So, policy π chooses action a_1 in state s_1 leading to state s_3 but in state s_2 it chooses a_2 leading to s_4. Thus, for a generalized potential heuristic h^{GP} equivalent to the ASNet parameters θ it must hold that

$$\arg \min_{s':\langle s_1,a,s'\rangle \in T} h^{\text{GP}}(s') = \{s_3\} \text{ and}$$
$$\arg \min_{s':\langle s_2,a,s'\rangle \in T} h^{\text{GP}}(s') = \{s_4\}.$$

However, this is impossible, since both states s_1 and s_2 have the same two successors s_3 and s_4 whose heuristic values $h^{\mathrm{GP}}(s_3)$ and $h^{\mathrm{GP}}(s_4)$ are not influenced by whether s_1 or s_2 is the current state. So, we always have

$$\arg \min_{s':\langle s_1,a,s'\rangle \in T} h^{\mathrm{GP}}(s') = \arg \min_{s':\langle s_2,a,s'\rangle \in T} h^{\mathrm{GP}}(s')$$

which can be equal to either $\{s_3\}$ or $\{s_4\}$ but not both at the same time.
This shows that an equivalent generalized potential does not exist for all possible ASNet parameters and classes of planning tasks. $\square$

The presented example shows that we cannot compile ASNet parameters into a generalized potential heuristic in all cases. This even holds for heuristics in general and not only generalized potential heuristics. The above proof exploits that in the presented example a heuristic cannot choose different subgoals depending on whether the current state is s_1 or s_2. This applies to all heuristics and not only generalized potential heuristics.
This special case though, which we exploited in the proof and where the policy chooses a different successor depending on the current state, is unnecessary in the considered setting where all actions have the same action costs. No matter the current state, in our example either s_3 is the best successor, s_4 is the best successor or both are equally good. In the first two cases the policy should always prefer the same successor no matter whether the current state is s_1 or s_2. In the latter case it is not necessary that the policy differentiates s_3 and s_4 based on the current state.
Therefore, potential future work could be concerned with restrictions under which it becomes possible to compile an ASNet into a (generalized potential) heuristic. For now though, we move on and look at the opposite direction, whether we can compile a generalized potential heuristic into an action schema network.

3.4 Generalized Potential Heuristics into Action Schema Networks

The last compilation we consider in this work is the one from a generalized potential heuristic into ASNet parameters which we will discuss in this section.
As before (Definition 25), for a task class $\mathcal{Q}$ we consider a GP heuristic h^{GP} and some ASNet parameters θ equivalent if for all states s of all tasks P in $\mathcal{Q}$ it holds that

$$\arg \max_{s':\langle s,a,s'\rangle \in T} \pi(a|s) = \arg \min_{s':\langle s,a,s'\rangle \in T} h^{\mathrm{GP}}(s')$$

where T is the set of transitions of the state space induced by task P and π is the policy of the ASNet initialized for P with θ. That means the successors s' with minimal h^{GP}-value must exactly be the states that can be reached with actions a for which $\pi(a|s)$ is maximal.
Based on this our final result shows that in general we cannot compile a generalized potential heuristic into ASNet parameters.

Theorem 9. *There exist a task class $\mathcal{Q}$, a task $P \in \mathcal{Q}$ and a generalized potential heuristic h^{GP} over $\mathcal{Q}$ such that there are no ASNet parameters θ equivalent to h^{GP}. That means, for $\mathcal{Q}$ there are no ASNet parameters θ such that the following holds for all states s in task P:*

$$\arg \max_{s':\langle s,a,s'\rangle \in T} \pi(a|s) = \arg \min_{s':\langle s,a,s'\rangle \in T} h^{\mathrm{GP}}(s')$$

where T are the transitions of the state space induced by P.

To prove this we exploit that an ASNet cannot combine hidden representations if the corresponding propositions and actions are not related. The example is built such that a whole state must be taken into account to replicate the information of the GP heuristic. But the actions and propositions are constructed such that they are independent in the sense that all actions are related to exactly one proposition and no two actions share the related proposition. Because of this the corresponding action schema network is not able to combine or spread information and thus never has access to all information of a state at once.

Proof. Consider a task class $\mathcal{Q}$ over the domain $\langle\{X, Y\}, \{A, B\}\rangle$ where all predicates (X and Y) and all action schemas (A and B) are unary, i. e. they take a single argument. The action schemas are defined as follows:

$$A(o) = \langle\{\}, \{X(o)\}, \{\}\rangle$$

$$B(o) = \langle\{\}, \{Y(o)\}, \{\}\rangle$$

So, the actions derived from A and B have no preconditions and their only effects are making the propositions $X(o)$ and $Y(o)$, respectively, true.
Let $P \in \mathcal{Q}$ be the task that is derived from the domain of $\mathcal{Q}$ with instance information $\langle\{o_1, o_2\}, \{\}, \{X(o_1), X(o_2), Y(o_1), Y(o_2)\}\rangle$. That means, P describes the task of getting from the initial state $\{\}$ where no propositions are true to the goal $\{X(o_1), X(o_2), Y(o_1), Y(o_2)\}$ where all propositions are true. Task P has propositions $X(o_i)$ and $Y(o_i)$, actions $a_i = \langle\{\}, \{X(o_i)\}, \{\}\rangle$ derived from A and actions $b_i = \langle\{\}, \{Y(o_i)\}, \{\}\rangle$ derived from B for $i \in \{1, 2\}$.
Let, furthermore, $h^{\mathrm{GP}}(s) = f_d(s) - 3 \cdot f_X(s) - 2 \cdot f_Y(s)$ be a generalized potential heuristic over the states s of class $\mathcal{Q}$ with the following features over objects o:

$$f_X(s) = |\{o \mid s \models X(o)\}|,$$

$$f_Y(s) = |\{o \mid s \models Y(o)\}|,$$

$$f_d(s) = abs(f_X(s) - f_Y(s)).$$

Feature f_X counts the number of objects o for which proposition $X(o)$ is true in state s, feature f_Y does the same for proposition $Y(o)$ and feature f_d is the difference between the first two features. The following table gives an overview of the possible values of h^{GP} in task P. For states where it makes no difference whether o_1 or o_2 is chosen the rows are abbreviated and use o_i and o_j.
Thus, in task P, heuristic $h^{\mathrm{GP}}(s) = f_d(s) - 3 \cdot f_X(s) - 2 \cdot f_Y(s)$ prefers states where the number of true X-propositions and true Y-propositions is equal and if they are equal h^{GP} prioritizes making X-propositions true over making Y-propositions true.
To prove that there are no ASNet parameters θ that are equivalent to h^{GP}, we first take a look at the ASNets that can be initialized for P. The general structure of ASNets initialized for task P is shown in Figure 3.2.

Figure 3.1: Overview of the values of $h^{\mathrm{GP}}(s) = f_d(s) - 3 \cdot f_X(s) - 2 \cdot f_Y(s)$ in task P.

States s	$f_X(s)$	$f_Y(s)$	$f_d(s)$	$h^{\mathrm{GP}}(s)$
$\{\}$	0	0	0	0
$\{X(o_i)\}$	1	0	1	-2
$\{Y(o_i)\}$	0	1	1	-1
$\{X(o_1), X(o_2)\}$	2	0	2	-4
$\{Y(o_1), Y(o_2)\}$	0	2	2	-2
$\{X(o_i), Y(o_j)\}$	1	1	0	-5
$\{X(o_1), X(o_2), Y(o_i)\}$	2	1	1	-7
$\{X(o_i), Y(o_1), Y(o_2)\}$	1	2	1	-6
$\{X(o_1), X(o_2), Y(o_1), Y(o_2)\}$	2	2	0	-10

$$\begin{tikzcd}
 & a_1 & X(o_1) & & a_1 & \\
u^1_{a_1} \arrow[r] & \boxed{f(W^1_A \cdot u^1_{a_1} + b^1_A) = \phi^1_{a_1}} \arrow[r] & \boxed{f(W^1_X \cdot v^1_{X_1} + b^1_X) = \psi^1_{X_1}} \arrow[r] & \cdots \arrow[r] & \boxed{f(W^{L+1}_A \cdot u^{L+1}_{a_1} + b^{L+1}_A) = \phi^{L+1}_{a_1}} \arrow[r] & \pi(a_1|s) \\
 & a_2 & X(o_2) & & a_2 & \\
u^1_{a_2} \arrow[r] & \boxed{f(W^1_A \cdot u^1_{a_2} + b^1_A) = \phi^1_{a_2}} \arrow[r] & \boxed{f(W^1_X \cdot v^1_{X_2} + b^1_X) = \psi^1_{X_2}} \arrow[r] & \cdots \arrow[r] & \boxed{f(W^{L+1}_A \cdot u^{L+1}_{a_2} + b^{L+1}_A) = \phi^{L+1}_{a_2}} \arrow[r] & \pi(a_2|s) \\
 & b_1 & Y(o_1) & & b_1 & \\
u^1_{b_1} \arrow[r] & \boxed{f(W^1_B \cdot u^1_{b_1} + b^1_B) = \phi^1_{b_1}} \arrow[r] & \boxed{f(W^1_Y \cdot v^1_{Y_1} + b^1_Y) = \psi^1_{Y_1}} \arrow[r] & \cdots \arrow[r] & \boxed{f(W^{L+1}_B \cdot u^{L+1}_{b_1} + b^{L+1}_B) = \phi^{L+1}_{b_1}} \arrow[r] & \pi(b_1|s) \\
 & b_2 & Y(o_2) & & b_2 & \\
u^1_{b_2} \arrow[r] & \boxed{f(W^1_B \cdot u^1_{b_2} + b^1_B) = \phi^1_{b_2}} \arrow[r] & \boxed{f(W^1_Y \cdot v^1_{Y_2} + b^1_Y) = \psi^1_{Y_2}} \arrow[r] & \cdots \arrow[r] & \boxed{f(W^{L+1}_B \cdot u^{L+1}_{b_2} + b^{L+1}_B) = \phi^{L+1}_{b_2}} \arrow[r] & \pi(b_2|s)
\end{tikzcd}$$

Figure 3.2: General structure of an ASNet initialized for task P

The weight parameters W^l_i and b^l_i of all modules, the hidden dimension h_d, the activation function f and the number of layers L can be chosen freely for the following argument. The input vectors of all modules are as follows ($i \in \{1, 2\}$):

$$u^1_{a_i} = \begin{pmatrix} x \\ g_X \\ m_{a_i} \end{pmatrix}$$

$$u^1_{b_i} = \begin{pmatrix} y \\ g_Y \\ m_{b_i} \end{pmatrix}$$

$u^l_{a_i} = \psi^{l-1}_{X_i}$ for all $2 \leq l \leq L + 1$

$u^l_{b_i} = \psi^{l-1}_{Y_i}$ for all $2 \leq l \leq L + 1$

$v^l_{X_i} = \phi^l_{a_i}$ for all $1 \leq l \leq L$

$v^l_{Y_i} = \phi^l_{b_i}$ for all $1 \leq l \leq L$

Notation:

- x and y are 1 if X and Y, respectively, are true in current state s and 0 if they are false in the current state
- g_X and g_Y are 1 if X and Y, respectively, are true in the goal and 0 otherwise

- m_a is 1 if action a is applicable in current state s and 0 if it is not applicable ($a \in \{a_1, a_2, b_1, b_2\}$)

Observe that, because there are no connections between modules in different "rows" and because all actions are always applicable, the only thing that can change the result of $\pi(a_i|s)$ is the truth value of $X(o_i)$ in current state s and the only thing that can change the result of $\pi(b_i|s)$ is the truth value of $Y(o_i)$ in s. Furthermore, for all actions the output is the same if their inputs have the same truth value, i. e. $\pi(a_i|s) = \pi(a_j|s)$ for all i, j with $X(o_i) = X(o_j)$ in s and $\pi(b_i|s) = \pi(b_j|s)$ for all i, j with $Y(o_i) = Y(o_j)$ in s. These observations hold because all modules of actions a_i within a layer l share the same weight W^l_A, activation function f and hidden representation dimension h_d (the latter two are even shared among all modules in the ASNet). The same argument applies for the modules of actions b_i and propositions $X(o_i)$ and $Y(o_i)$.
To prove non-equivalence, we will now show that for state $s_1 = \{X(o_1)\}$ policy π and heuristic h^{GP} do not choose the same successors. That means we will show for s_1 that it does not hold that

$$\arg \max_{s':\langle s_1,a,s'\rangle \in T} \pi(a|s_1) = \arg \min_{s':\langle s_1,a,s'\rangle \in T} h^{\text{GP}}(s').$$

To obtain the successors that h^{GP} chooses, we take a look at all successors of $s_1 = \{X(o_1)\}$ and their heuristic values:

$s_2 = \{X(o_1), X(o_2)\}$ via action a_2 with $h^{\text{GP}}(s_2) = -4$

$s_3 = \{X(o_1), Y(o_1)\}$ via action b_1 with $h^{\text{GP}}(s_3) = -5$ and

$s_4 = \{X(o_1), Y(o_2)\}$ via action b_2 with $h^{\text{GP}}(s_3) = -5$.

As s_3 and s_4 have the minimal heuristic values, heuristic h^{GP} chooses these two states and we have

$$\arg \min_{s':\langle s_1,a,s'\rangle \in T} h^{\text{GP}}(s') = \{s_3, s_4\}.$$

Therefore, policy π must pick these states as well which means it must hold that

$$\arg \max_{s':\langle s_1,a,s'\rangle \in T} \pi(a|s_1) = \{s_3, s_4\}.$$

In the following we will show that the policy π of the ASNet initialized for task P cannot achieve this. As we are considering all possible options for the ASNet parameters θ we do not have concrete values that would tell us directly which successors policy π chooses in s_1 (or more formally which actions it chooses that lead to the successors). But we can compare π in state s_1 with π in state $s_0 = \{\}$ to determine for which actions π is maximal in s_1, that means which actions it chooses in s_1. To prove our argument we are going to derive a connection between the policy value of action a_2 in state s_1 and the policy value of the actions b_1 and b_2 in the same state, i. e. we aim for a connection between $\pi(a_2|s_1)$ and $\pi(b_i|s_1)$ ($i \in \{1, 2\}$).
Based on the above mentioned observations of the ASNet for P we know the following:

1. $\pi(a_1|s_0) = \pi(a_2|s_0)$ because $X(o_1) = X(o_2) = \bot$ in s_0,

2. $\pi(a_1|s_0) = \pi(a_2|s_0) = \pi(a_2|s_1)$ because $X(o_2) = \bot$ in s_1 like $X(o_1)$ and $X(o_2)$ in s_0, and

3. $\pi(b_1|s_0) = \pi(b_2|s_0) = \pi(b_1|s_1) = \pi(b_2|s_1)$ because $Y(o_1) = Y(o_2) = \bot$ in both s_0 and s_1.

We furthermore know that in s_0 policy π must assign a higher value to the a-actions than to the b-actions. This is the case because to be equivalent to the GP heuristic h^{GP} policy π must choose the same successors as h^{GP} which chooses state $\{X(o_1)\}$ or $\{X(o_2)\}$ reachable via actions a_1 or a_2 respectively. That means we have $\pi(a_i|s_0) > \pi(b_j|s_0)$ for $i, j \in \{1, 2\}$ or combined with 1. and 3. we can write it as $\pi(a_1|s_0) = \pi(a_2|s_0) > \pi(b_1|s_0) = \pi(b_2|s_0) = \pi(b_1|s_1) = \pi(b_2|s_1)$. With this we have now derived a connection between the policy values of the a-actions in state s_0 and the policy values of the b-actions in state s_1, that is we have $\pi(a_i|s_0) > \pi(b_j|s_1)$ $(i \in \{1, 2\})$. Our goal however is to derive such a connection in the same state s_1. We can achieve this by combining the result of the last step $\pi(a_i|s_0) > \pi(b_j|s_1)$ with 2. which yields $\pi(a_2|s_1) = \pi(a_i|s_0) > \pi(b_j|s_1)$. This can be shortened to $\pi(a_2|s_1) > \pi(b_j|s_1)$ and means that in state s_1 policy π chooses action a_2 over the actions b_1 and b_2.
Since a_2 leads to state s_2 while b_1 and b_2 lead to s_3 and s_4 respectively we now know that

$$\arg \max_{s':\langle s_1,a,s'\rangle \in T} \pi(a|s_1) \neq \{s_3, s_4\}$$

holds from which it directly follows that we have

$$\arg \max_{s':\langle s_1,a,s'\rangle \in T} \pi(a|s_1) \neq \arg \min_{s':\langle s_1,a,s'\rangle \in T} h^{\mathrm{GP}}(s').$$

Hence, in state s_1 heuristic h^{GP} and policy π do not choose the same successors. This means that the generalized potential heuristic h^{GP} and the ASNet parameters θ are not equivalent in class $\mathcal{Q}$. $\square$

Together with Theorem 8, this shows that ASNets and generalized potential heuristics cannot be compiled into each other in general. However, we again exploited a rather artificial special case (the task is built from independent subproblems) to show that a GP heuristic cannot be compiled into some ASNet parameters in general. On the one hand does h^{GP} unnecessarily require that P is solved by alternating a_i and b_i actions, and on the other hand is it rather unlikely in practice that there are absolutely no actions that are related to the same propositions.
Potential future work on this is thus to investigate under which restrictions the compilation might become possible and whether these restrictions are relevant in practice. One option for example, is to alter how equivalence is defined. Instead of comparing successors with minimal heuristic value and successors with maximal policy value we could compare successors that have a lower heuristic value than the current state with successors whose policy value is larger zero.
Here, we finish the part about our contributions though. In the next chapter we will take a closer look at the works from which GP heuristics, sketches and ASNets originate and other related work.

4 Related Work

This work lies in the area of generalized planning, so other work related to ours is for example Hu and De Giacomo [10], Levesque [11] and Srivastava et al. [21]. Our focus though is on generalized representations that appear comparable. Hence, we present work related to these in Section 4.1.
To the extend of our knowledge, this is the first work about compilability between generalized representations. But as discussed in Section 4.2 compilability in automated planning has already been considered.

4.1 Generalized Representations

The generalized representation most important to our work are generalized potential heuristics as all our compilations aim to compile from or into them. They were introduced by Francès et al. [7] and are based on potential heuristics from Pommerening et al. [14]. These original potential heuristics work only for single tasks and are sums over the potentials of all facts. A fact here is a tuple of a variable and a value of it. The potential function maps all facts to real numbers. Expressed in terms of generalized potential heuristics the potential function is a combination of a feature and its weight, while the fact is a component of the feature that maps a state to the current value of each variable. This dependency on the variables of a task prevents potential heuristics from generalizing.
Shortly after the introduction of potential heuristics, Seipp et al. [17] presented optimization functions that yield better informed potential heuristics. They also combined multiple potential heuristics to improve the heuristic guidance even further. Later, Pommerening et al. [15] extended potential heuristics such that they can handle more kinds of features. The features they consider are single facts, as before, or conjunctions of facts. This version of potential heuristics only distinguishes itself from generalized potential heuristics by the features it uses and that it does not use the feature values directly but whether a feature is present or not in a state. Pommerening et al. [15] furthermore evaluate potential heuristics with binary features, i. e. pairs of facts, and they present a hardness result for potential heuristics with higher-dimensional features.
When using features as abstractions over states, which GP heuristics do, they can generalize

over multiple tasks and potentially all infinitely many tasks of a domain. Francès et al. [7] use features based on concept languages (also called description logics [1]) and aim for generalized potential heuristics that are descending and dead-end avoiding to greedily solve planning tasks in polynomial time. They present some hand-crafted GP heuristics that are descending and dead-end avoiding as well as a method to learn such GP heuristics automatically.

Policy sketches were introduced by Bonet and Geffner [2, 3]. This theoretical work is largely concerned with general policies of which sketches are a generalization and the serializations underlying them. All three can be used to decompose problems into subproblems with small width such that the greedy search algorithm SIW can solve the overall problem in polynomial time and space. General policies and policy sketches both are sets of rules that define subgoals for each state depending on whether this state satisfies the conditions of a rule. The difference between general policies and policy sketches is that with the first the subgoals must be reachable within one step while it can take multiple steps with the latter. Both induce serializations though that decompose planning tasks into subtasks with potentially low width such that these can be solved efficiently. Same as the generalized potential heuristics of Francès et al. [7] policy sketches (and general policies and serializations) also use features as abstractions of states to be able to generalize over multiple tasks. Furthermore, Bonet and Geffner [3] also construct their considered features using concept languages.

The work on policy sketches of Bonet and Geffner [3] was extended by Drexler et al. [4]. They present hand-crafted policy sketches for certain domains and prove that these sketches have the properties needed such that SIW_R can solve problems of these domains in polynomial time and space. They furthermore empirically evaluate SIW_R with their presented sketches. The second follow-up work on policy sketches from Drexler et al. [5] is concerned with learning policy sketches automatically and proves for some resulting sketches that they have the properties needed for SIW_R to solve tasks in polynomial time and space.

The last generalized representation we evaluated are action schema networks (ASNets). They were introduced by Toyer et al. [23] as one of the first neural network architectures for planning tasks. Toyer et al. [24] then expanded their work on ASNets, most importantly by extending the architecture with skip connections and a more expressive pooling mechanism. ASNets are neural networks whose structure is based on the connections between related actions and propositions from a planning task. The relatedness of actions and propositions depends on the action schemas of a domain which enables ASNets to generalize over multiple tasks. When Toyer et al. [23] first introduced ASNets they considered a proposition and an action to be related if the proposition is mentioned in the preconditions of effects of the action. Toyer et al. [24] refined this concept by differentiating which kind of "role" a proposition takes within an action. They furthermore added so-called skip connections to the architecture of ASNets that directly connect action modules of different layers to better propagate information and thus mitigate the limited receptive field of ASNets.

Toyer et al. [24] also expand the experimental evaluation of ASNets from Toyer et al. [23] and they present an example to show how the interpretability of ASNets can be improved with sparsity regularisation.

Another generalized representation that appears interesting to compare against are STRIPS

hypergraph networks (STRIPS-HGNs) introduced by Shen et al. [20] based on the bachelor's thesis of Shen [19]. Same as ASNets also STRIPS hypergraph networks are a neural network structure for planning tasks. In contrast to the other three considered generalized representations STRIPS-HGNs are able to generalize over tasks from different domains and not only tasks from a single domain. They take as input hypergraphs representing the delete relaxation of a task to compute heuristic estimates. A STRIPS-HGN has three building blocks, one that encodes the hypergraph into a latent representation, one that processes this latent hypergraph, and a last block that decodes the processed hypergraph into a heuristic value. The processing block is applied multiple times and takes as input the original hypergraph and the latent hypergraph from its last application.
Since a STRIPS-HGN operates on hypergraphs induced by delete relaxations it can handle tasks independently of their domain. Furthermore, a single STRIPS-HGN works for all considered tasks in contrast to ASNets where a new ASNet must be initialized for each considered task. Because of that it seems unlikely that it is possible to compile STRIPS-HGNs into one of the other three representations. Nonetheless, it might be interesting to investigate whether GP heuristics, policy sketches or ASNets could be compiled into STRIPS-HGNs.

4.2 Compilability

Compilability can be used as a means of comparing expressiveness as we do in this work where we compare whether certain generalized representations can express the same information about a class of classical planning tasks.
Nebel [13] evaluate the expressiveness of different propositional planning formalisms such as STRIPS [6]. They compare the expressiveness of different formalisms by requiring that the size of the compilation result grows at most polynomially and they differentiate whether plan size in the new formalism is preserved exactly, linearly, or polynomially. Compared to this our work is rather strict because we aim for compilations that preserve not only plan size but even require that the same plans can be expressed.
Thiébaux et al. [22] argue that PDDL-like axioms improve expressiveness of planning formalisms by means of compilability. They use the compilation schemes of Nebel [13] to show that PDDL-like axioms cannot be compiled away if plan size and the size of domain descriptions may grow at most polynomially.
Also using the compilation schemes of Nebel [13], Röger et al. [16] describe a compilation from a subset of the Golog [12] agent programming language to the ADL fragment of PDDL. They discuss which restrictions are necessary such that a compilation remains possible and thus identify a maximal fragment that can be compiled into ADL.
While we compare three rather different concepts, a heuristic, a policy produced from a neural network, and a generalization of policies Helmert and Domshlak [8] compile different heuristics into each other. They present connections between heuristics based on delete relaxations, critical paths, abstractions, and landmarks by compiling them into each other and proving for some cases that a compilation is not possible. Based on that they then introduce the landmark cut heuristic (LM-cut heuristic) which is a good approximation of

the perfect heuristic over delete-relaxed planning tasks. Toyer et al. [23, 24] use features derived from this heuristic as additional input for ASNets to overcome the limited receptive field of ASNets.

5 Conclusions

The aim of this work was to find compilations or proofs that compilations are impossible between three generalized representations. In this last chapter we discuss our results and present potential future work.

5.1 Discussion

First, we present the summary of our compilation results:

Compilation into policy sketch:

Theorem 1 Compilation from non-negative integer-valued GP heuristic into **equivalent** sketch

Theorem 2 Compilation from non-negative **real-valued** GP heuristic into **equivalent** sketch

Theorem 3 Compilation from integer-valued (including **negative values**) GP heuristic into **equivalent** sketch

Theorem 4 Compilation is **not possible** from (non-negative integer-valued) GP heuristic into **equivalent** sketch with **same features**

Compilation from policy sketch:

Theorem 5 Compilation **not possible** from sketch into **equivalent** GP heuristic (in general and thus also not with same features)

Theorem 6 Compilation from sketch into **similar** GP heuristic

Theorem 7 Compilation **not possible** from sketch into **similar** GP heuristic with **same features**

Compilation from and into action schema network parameters:

Theorem 8 Compilation **not possible** from ASNet parameters into **equivalent** GP heuristic

Theorem 9 Compilation **not possible** from GP heuristic into **equivalent** ASNet parameters

Theorems 5 and 8 hold for heuristics in general and Theorems 1, 2 and 3 can be altered to hold for heuristics in general.

The first three theorems are concerned with the compilation of generalized potential heuristics into equivalent policy sketches in general. These results can even be adjusted to hold for heuristics in general because the compiled sketches just use the heuristics directly as feature. If we require however that the compiled sketch must use the same features as the GP heuristic (Theorem 4) then the compilation is not possible anymore. This means, sketches are at least as expressive as (GP-) heuristics given suitable features but when using the same features they cannot express everything a GP heuristic can express.
In the reverse direction compilation is even in general not possible (Theorem 5). We cannot compile a policy sketch into an equivalent (generalized potential) heuristic in general and thus especially not when requiring the GP heuristic to use the features from the sketch. A compilation from policy sketches into generalized potential heuristics becomes possible when we are less strict in our comparison (Theorem 6). But even in this case a compilation where the features remain the same is not possible (Theorem 7).
Combining the two directions of compilations shows that sketches are strictly more expressive than GP heuristics given suitable features. But when restricting both representations to use the same features the compilation is impossible in both directions. On the one hand are policy sketches able to differentiate subgoals based on the current state which GP heuristics are not capable of to the same degree. And on the other hand can GP heuristics weight features against each other which sketches cannot do. This indicates that there are (at least theoretical) scenarios where it is beneficial to use a generalized potential heuristic and there are scenarios where it is preferable to use a policy sketch.
The compilation between action schema networks and generalized potential heuristics is in both directions not possible in general. Theorem 8 even holds for heuristics in general and shows that we cannot compile ASNet parameters into an equivalent heuristic. Similar to policy sketches, ASNet policies can base their decisions on more local information compared to GP heuristics. ASNet policies can differentiate successor states that seem equally good choices based on the current state which GP heuristics cannot do. They, on the other hand, are capable of aggregating information even when actions and propositions are not explicitly related with each other which is not possible with ASNets as their structure is based on relatedness. So, as with sketches, there seem to be scenarios where it is better to use ASNets and there are scenarios where GP heuristics appear to be the better choice.

5.2 Future Work

Based on the compilation results we presented, there are multiple interesting lines of future work which we present in this section.
First of all, we investigated compilations between generalized potential heuristics and policy sketches and between generalized potential heuristics and action schema networks. Potential

compilations between policy sketches and action schema networks are not included in this work and thus researching these is a natural next step. Furthermore, STRIPS hypergraph networks are a fourth generalized representation which we discussed as related work but did not present compilations or counterexamples for. It might be worth investigating whether the three generalized representations we covered and STRIPS hypergaph networks can be compiled into each other.

We showed that in general it is not possible to compile ASNet parameters and generalized potential heuristics equivalently into each other. Potential future work is thus to find cases in which a compilation is possible. It might also be worth investigating whether a different notion of equivalent behaviour is useful. We compared whether the same successors have a minimal heuristic value and a maximal policy value. Instead one could compare whether the same successors have an improving heuristic value, i. e. a lower heuristic value than the current state, and a policy value larger than zero. On the one hand would we then consider only states that actually have a better heuristic value than the current state. But on the other hand would we take into account not only the best successors an ASNet policy chooses but all possible successors.

Similarly, one could investigate under which restrictions it is possible to compile policy sketches into generalized potential heuristics and when it is possible to compile sketches and GP heuristics with the same features into each other.

Also interesting would be to research whether all three generalized representations are able to solve the same task classes (equally well) instead of requiring equivalent behaviour. Similar to the work of Nebel [13] one could compare generalized representations by the size of the plans they yield for a task class and by the size of the representations themselves needed to express certain information about a class.

Bibliography

[1] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[2] Blai Bonet and Hector Geffner. General policies, serializations, and planning width. *arXiv preprint arXiv:2012.08033*, 2020.

[3] Blai Bonet and Hector Geffner. General policies, representations, and planning width. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 11764–11773, 2021.

[4] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Expressing and exploiting the common subgoal structure of classical planning domains using sketches. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 258–268, 11 2021.

[5] Dominik Drexler, Jendrik Seipp, and Hector Geffner. Learning sketches for decomposing planning problems into subproblems of bounded width. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 62–70, 2021.

[6] 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.

[7] Guillem Francès, Augusto B Corrêa, Cedric Geissmann, and Florian Pommerening. Generalized potential heuristics for classical planning. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 5554–5561, 2019.

[8] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 162–169, 2009.

[9] Jörg Hoffmann. A heuristic for domain independent planning and its use in an enforced hill-climbing algorithm. In *International Symposium on Methodologies for Intelligent Systems*, pages 216–227. Springer, 2000.

[10] Yuxiao Hu and Giuseppe De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 918–923, 2011.

[11] Hector J Levesque. Planning with loops. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 509–515, 2005.

[12] Hector J Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B Scherl. Golog: A logic programming language for dynamic domains. *The Journal of Logic Programming*, 31(1-3):59–83, 1997.

[13] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.

[14] Florian Pommerening, Malte Helmert, Gabriele Röger, and Jendrik Seipp. From non-negative to general operator cost partitioning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3335–3341, 2015.

[15] Florian Pommerening, Malte Helmert, and Blai Bonet. Higher-dimensional potential heuristics for optimal classical planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3636–3643, 2017.

[16] Gabriele Röger, Malte Helmert, and Bernhard Nebel. On the relative expressiveness of ADL and Golog: The last piece in the puzzle. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pages 544–550, 2008.

[17] Jendrik Seipp, Florian Pommerening, and Malte Helmert. New optimization functions for potential heuristics. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 25, pages 193–201, 2015.

[18] Jendrik Seipp, Florian Pommerening, Gabriele Röger, and Malte Helmert. Correlation complexity of classical planning domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 3242–3250. AAAI Press, 2016.

[19] William Shen. Learning heuristics for planning with hypergraph networks. Bachelor's thesis, The Australian National University, 2019.

[20] William Shen, Felipe Trevizan, and Sylvie Thiébaux. Learning domain-independent planning heuristics with hypergraph networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 574–584, 2020.

[21] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. Learning generalized plans using abstract counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 8, pages 991–997, 2008.

[22] Sylvie Thiébaux, Jörg Hoffmann, and Bernhard Nebel. In defense of pddl axioms. *Artificial Intelligence*, 168(1-2):38–69, 2005.

[23] Sam Toyer, Felipe Trevizan, Sylvie Thiébaux, and Lexing Xie. Action schema networks: Generalised policies with deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6294–6301, 2018.

[24] Sam Toyer, Sylvie Thiébaux, Felipe Trevizan, and Lexing Xie. Asnets: Deep learning for generalised planning. *Journal of Artificial Intelligence Research*, 68:1–68, 2020.