Catching Label Subsets for Relaxed Bisimulation: An Abstraction Refinement Approach

Marcel Steinmetz
Saarland University
Saarbrücken, Germany
s9mrstei@stud.uni-saarland.de

Jörg Hoffmann and **Michael Katz**
Saarland University
Saarbrücken, Germany
{hoffmann, katz}@cs.uni-saarland.de

Abstract

Merge-and-Shrink abstraction (M&S) is an approach for constructing admissible heuristic functions. A key issue in M&S abstractions is which states are mapped to the same abstract state: That decision directly controls the trade-off between the accuracy of the resulting heuristic function on the one hand, and the size of the abstract state space on the other hand. A recent approach towards tackling this issue introduced the notion of K*-catching bisimulation*. This class of abstractions is a bisimulation – preserving transition behavior exactly – but only for a subset K of the planning operators, ignoring all others. It has been shown that this form of relaxed bisimulation is invariant over the M&S process, and that choosing K appropriately may reduce abstraction size exponentially while still delivering a perfect heuristic. Determining those exact operator subsets K is, however, highly intractable, and practical approximations so far did not yield convincing results. Thus the question remains: How to select K?

We propose to answer that question by a counter-example guided abstraction refinement (CEGAR) approach. Given a K-catching bisimulation, we analyze its optimal solutions and identify new operators to be added to K, ultimately excluding all spurious solutions. We design, and experiment with, practical criteria to terminate the refinement process before that happens.

Introduction

One of the currently most successful approaches to solve problems in optimal planning is to use the A^* search algorithm with an admissible heuristic function. Heuristic functions estimate the cost of the cheapest operator sequence that, when applied on the initial state, leads to the goal. In addition, admissible heuristic functions give the guarantee that this estimation is always a lower bound on the cheapest cost of real solutions, and the better this estimation is, i.e., the smaller the difference to this cost, the faster will the search algorithm find a solution for the problem. Therefore, the main question becomes how good admissible heuristic functions can be computed automatically, for any given problem.

One approach to construct an admissible heuristic function is based on abstractions. Abstractions ignore the difference between certain states and consequently reduce the total size of the state space, while preserving all operator sequences that are possible in the concrete state space. With fewer states, the analysis of this abstract state space can become feasible so that the heuristic value for some state can be computed by computing the cost of the optimal solution of its representative in the abstract version of the state space.

Currently, two methods for building an abstraction are used in planning: Pattern databases (Haslum et al. 2007) and Merge-and-Shrink abstractions (Helmert, Haslum, and Hoffmann 2007), and the latter one, Merge-and-Shrink abstraction, strictly generalizes pattern databases. M&S builds an abstraction by iteratively combining, called *merging*, and further reducing the size of, called shrinking, basic parts of the state space. To preserve the entire behavior of the original state space in the abstract state space, Nissim et al. (2011) used the well-known notion of bisimulation (Milner 1990). They observe that the bisimulation requirement is unnecessarily strict for the purpose of computing heuristic functions. This has been addressed by two different relaxations of bisimulation: greedy bisimulation (Nissim, Hoffmann, and Helmert 2011), and K-catching bisimulation (Katz, Hoffmann, and Helmert 2012). Both are based on the same idea, ignoring the difference between more states by considering only a subset of transitions during the test for the bisimulation property. Greedy bisimulation ignores transitions based on a local condition, i.e., on a per-transition basis (abstract goal distances at the transition's end points). By contrast, K-catching bisimulation fixes a global criterion throughout the M&S process, simply catching a transition if its label – the planning operator inducing it – is contained in a label subset K fixed a priori. The key advantage of that criterion is its invariance over the M&S construction (**?**), providing direct control over the final abstraction in terms of the choice of K.

Katz et al. identify two different types of operators that, if caught by K, lead to a perfect heuristic, while the resulting K-catching bisimulation can be exponentially smaller than any general bisimulation. However, computing these sets K exactly is highly intractable: They consist of all operators that form part of an optimal solution for any state in (part of) the state space. Katz et al. devise some simple approximation methods, with mediocre empirical results. Herein, we instead explore the possibility to design K via *counterexample-guided abstraction refinement (CEGAR)*.

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

CEGAR was originally introduced in the context of model checking (Clarke et al. 2003) for the purpose of proving (un)reachability of states in transition systems. It computes an abstraction of a transition system by an incremental procedure that analyzes intermediate abstractions to extract information – counterexamples – that can be used to improve the abstraction. One starts with a simple initial abstraction of the transition system that is improved by incrementally distinguishing more states, in the so called *refinement loop*, as long as certain properties are not fulfilled. CEGAR consists of the three general components:

(1) *Initial abstraction.* Specification of the abstraction that is used to start the refinement. (In our case: K-catching bisimulation for the empty set K.)

(2) *Analysis of the abstract transition system.* Determining unintended behavior of the abstract transition system of the current abstraction. (In our case: Optimal abstract solutions that are spurious, failing to solve the original planning task.)

(3) *Refine the abstraction.* Computing a new abstraction, by using the information obtained by (2), that excludes the unintended behavior. (In our case: Including new operators into K.)

(2) together with (3) are called the refinement step and they are executed as long as the abstract transition system does not fulfill some specific criteria. In the following we show how (1), (2) and (3) are instantiated for the purpose of building a K-catching bisimulation.

Background

A **planning task** is a 5-tuple $\Pi = (\mathcal{V}, \mathcal{O}, c, s_0, s_*)$. $\mathcal{V}$ is a finite set of **variables** v, each $v \in \mathcal{V}$ associated with a finite domain $\mathcal{D}_v$. A **partial state** over $\mathcal{V}$ is a function s on a subset $\mathcal{V}_s$ of $\mathcal{V}$, so that $s(v) \in \mathcal{D}_v$ for all $v \in \mathcal{V}_s$; s is a **state** if $\mathcal{V}_s = \mathcal{V}$. The **initial state** s_0 is a state. The **goal** s_* is a partial state. $\mathcal{O}$ is a finite set of **operators**, each being a pair $(\mathit{pre}, \mathit{eff})$ of partial states, called its **precondition** and **effect**. Each $o \in \mathcal{O}$ is also associated with its **cost** $c(o) \in \mathbb{R}_0^+$.

The **state space** of a planning task is given by a **transition system**. Such a system is a 6-tuple $\Theta = (S, L, c, T, s_0, S_*)$ where S is a finite set of **states**, L is a finite set of **transition labels**, each associated with a **label cost** $c(l) \in \mathbb{R}_0^+$, $T \subseteq S \times L \times S$ is a set of **transitions**, $s_0 \in S$ is the **start state**, and $S_* \subseteq S$ is the set of **goal states**. We define the **remaining cost** $h^* : S \to \mathbb{R}_0^+$ as the minimal cost of any path (the sum of costs of the labels on the path), in Θ, from a given state s to any $s_* \in S_*$, or $h^*(s) = \infty$ if there is no such path.

In the state space of a planning task, S is the set of all states. The start state s_0 is the initial state of the task, and $s \in S_*$ if $s_* \subseteq s$. The transition labels L are the operators $\mathcal{O}$, and $(s, (\mathit{pre}, \mathit{eff}), s') \in T$ if s complies with pre, and $s'(v) = \mathit{eff}(v)$ for all $v \in \mathcal{V}_{\mathit{eff}}$ while $s'(v) = s(v)$ for all $v \in \mathcal{V} \setminus \mathcal{V}_{\mathit{eff}}$. The solution of a planning task, called **plan**, is a path from s_0 to any $s_* \in S_*$. The plan is **optimal** if its summed-up cost is equal to $h^*(s_0)$.

A **heuristic** is a function $h : S \to \mathbb{R}_0^+ \cup \{\infty\}$. The heuristic is **admissible** if, for every $s \in S$, $h(s) \leq h^*(s)$; it is **consistent** if, for every $(s, l, s') \in T$, $h(s) \leq h(s') + c(l)$; it is **perfect** if h coincides with h^*. The A^* algorithm expands states by increasing value of $g(s) + h(s)$ where $g(s)$ is the accumulated cost on the path to s. If h is admissible, then A^* returns an optimal solution. If h is consistent then A^* does not need to re-open any nodes.

One way of automatically constructing admissible heuristics is based on **abstractions**. This is a function α mapping S to a set of **abstract states** S^α. The **abstract state space** Θ^α is defined as $(S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$, where $T^\alpha := \{(\alpha(s), l, \alpha(s')) \mid (s, l, s') \in T\}$, $s_0^\alpha := \alpha(s_0)$, and $S_*^\alpha := \{\alpha(s_*) \mid s_* \in S_*\}$. The **abstraction heuristic** h^α maps each $s \in S$ to the remaining cost of $\alpha(s)$ in Θ^α; h^α is admissible and consistent. The **pre-image** of $s^\alpha \in S^\alpha$ under the abstraction α is the set $\mathit{Pre}_\alpha(s^\alpha) = \{s \in S \mid \alpha(s) = s^\alpha\}$. We will sometimes consider the **induced equivalence relation** $\sim^\alpha$, defined by setting $s \sim^\alpha t$ iff $\alpha(s) = \alpha(t)$.

How to construct a good α in general? Helmert et al. (2007) propose M&S abstraction as a method allowing fine-grained abstraction design, selecting individual pairs of (abstract) states to aggregate. The approach builds the abstraction in an incremental fashion, iterating between *merging* and *shrinking* steps. In detail, an abstraction α is an **M&S abstraction over** $V \subseteq \mathcal{V}$ if it can be constructed using these rules:

(i) For $v \in \mathcal{V}$, $\pi_{\{v\}}$ is an M&S abstraction over $\{v\}$.

(ii) If β is an M&S abstraction over V and γ is a function on S^β, then $\gamma \circ \beta$ is an M&S abstraction over V.

(iii) If α_1 and α_2 are M&S abstractions over disjoint sets V_1 and V_2, then $\alpha_1 \otimes \alpha_2$ is an M&S abstraction over $V_1 \cup V_2$.

Rule (i) allows to start from **atomic projections**. These are simple abstractions $\pi_{\{v\}}$ (also written π_v) mapping each state $s \in S$ to the value of one selected variable v. **Rule (ii)**, the **shrinking step**, allows to iteratively aggregate an arbitrary number of state pairs, in abstraction β. Formally, this simply means to apply an additional abstraction γ to the image of β. In **rule (iii)**, the **merging step**, the merged abstraction $\alpha_1 \otimes \alpha_2$ is defined by $(\alpha_1 \otimes \alpha_2)(s) := (\alpha_1(s), \alpha_2(s))$.

To implement M&S in practice, we need a **merging strategy** deciding which abstractions to merge in (iii), and a **shrinking strategy** deciding which (and how many) states to aggregate in (ii). Throughout this paper, we use the same merging strategy as presented by Nissim et al. (2011). To obtain an abstraction that preserves the behavior of the original transition system, i.e., to obtain a perfect heuristic, Nissim et al. devise a shrinking strategy that computes a *bisimulation* of the state space.

Definition 1 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system. An equivalence relation $\sim$ on S is a **bisimulation** for Θ if for every $s \sim t$ holds: (1) either $s, t \in S_*$ or $s, t \notin S_*$; (2) for every transition label $l \in L$, $\{[s'] \mid (s, l, s') \in T\} = \{[t'] \mid (t, l, t') \in T\}$.*

As usual, $[s]$ for a state s denotes the equivalence class of s. An abstraction α is a bisimulation iff the induced equivalence relation $\sim^\alpha$ is. Nissim et al. have shown that the bisimulation property is preserved throughout the merging steps:

If α_1 and α_2 are bisimulations for $\Theta^{\pi_{V_1}}$ and $\Theta^{\pi_{V_2}}$, where $V_1 \cap V_2 = \emptyset$, then $\alpha_1 \otimes \alpha_2$ is a bisimulation for $\Theta^{\pi_{V_1 \cup V_2}}$. Therefore, if α is constructed such that, in every application of (ii), γ is a bisimulation for Θ^{β}, then the overall M&S abstraction α will be a bisimulation for Θ^{π_V} (Nissim, Hoffmann, and Helmert 2011).

Unfortunately, bisimulations are exponentially big even in trivial examples. Katz et al. (2012) address this by the notion of *K-catching bisimulation*. This relaxes the definition of bisimulation by applying constraint (2) of Definition 1 to only a subset of the transitions in T, selected by a subset of transition labels K:

Definition 2 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and $K \subseteq L$. An equivalence relation $\sim$ on S is a **K-catching bisimulation** for Θ if it is a bisimulation for the transition system (S, K, c, T^K, s_0, S_*), where $T^K = \{(s, l, t) \mid (s, l, t) \in T, l \in K\}$.*

Katz et al. (2012) identify two theoretical classes of such label subsets that, when considered in a K-catching bisimulation, lead to abstract transition systems that still provide strong properties. First, if the label subset contains (at least) each label that is used in some optimal path from any state to a goal state, then the resulting abstraction heuristic will still be perfect, while possibly obtaining an abstract transition system that is exponentially smaller than the abstract transition system of any general bisimulation. Such labels are called globally relevant, i.e., a label $l \in L$ is called **globally relevant** if there is a transition $(s, l, s') \in T$ such that $h^*(s) = h^*(s') + c(l)$.

Second, K does not even have to contain all globally relevant labels. If K contains each label that is used in some optimal solution for any state *with cost less or equal than the remaining cost of the initial state*, then A^*, using the abstraction heuristic, will still only expand a number of states linear in the length of the plan returned (under the assumption that the planning task does not contain 0-cost operators). Such labels are called $h^*(s_0)$-relevant, i.e., a label $l \in L$ is called R**-relevant** if there is a transition $(s, l, s') \in T$ such that $h^*(s) \leq R$ and $h^*(s) = h^*(s') + c(l)$.

Unfortunately, computing either of these label subsets K involves solving the problem in first place. We now introduce a method to select K based on counterexample guided abstraction refinement.

Abstraction Refinement

The general algorithm is depicted in Figure 1. The whole refinement procedure is based on a *current label subset*. This label subset is initially empty. For the purpose of refining the K-catching bisimulation, we add new labels to this label subset and recompute the K-catching bisimulation with the updated label set. To catch the right labels, we compare the optimal solutions of the abstract transition system with solutions of the original transition system. If these abstract solutions are spurious, i.e., they do not correspond to solutions of the original transition system, then we extract labels to be added to the label subset, which will eliminate these solutions from the abstract transition system.

$K \leftarrow \emptyset$
$\alpha \leftarrow K$-catching bisimulation for Θ
$\Theta^\alpha \leftarrow$ Abstract transition system of α
while the given criterion is not satisfied **do**
  $K' \leftarrow$ *analyze*(Π, Θ^α)
  $K \leftarrow K \cup K'$
  $\alpha \leftarrow K$-catching bisimulation for Θ
  $\Theta^\alpha \leftarrow$ Abstract transition system of α
endwhile
return α

Figure 1: General abstraction refinement procedure for computing K-catching bisimulations, based on a given termination criterion.

This step is repeated until the abstract transition system fulfills certain constraints. As an example, one could require that every optimal solution of the abstract transition system is a solution of the original transition system.

In the following sections we show how the abstract transition system is analyzed, that is how spurious solutions are identified, and which labels have to be considered to exclude these paths from the abstract transition system.

Refinement Step

We distinguish between *forward propagation*, where optimal solutions of the abstract transition system are compared to paths in the original transitions system, and *backward propagation*, where backward optimal solutions, i.e. the cheapest paths from some abstract goal state to the abstract initial state, obtained by inverting all transitions of the abstract transition system, are considered.

Depending on the termination criterion, the abstract solutions should only partially correspond to paths of the original transition system. Therefore, the distinction between these two cases allows to determine whether the abstraction should be more precise around the initial state, or more precise around the goal states, i.e., whether the labels added to the label subset should be closer to the initial state, or closer to the goal states.

Forward Propagation

Forward propagation analyzes the abstract transition system by finding, for each optimal solution, an equivalent path in the original transition system that starts in the initial state and which is labeled with the same label sequence. Therefore, we say that a label sequence is *forward-applicable* in a state s if there is a path in the transition system that starts in s and contains the given labels. Formally it is defined as follows:

Definition 3 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system, $\vec{a} = \langle a_1, \ldots, a_n \rangle \in L^n$ be a label sequence. Then $\vec{a}$ is **forward-applicable in** $t_0 \in S$ if there are states $t_1, \ldots, t_n \in S$ so that $(t_{i-1}, a_i, t_i) \in T$, for all $1 \leq i \leq n$. Moreover, t_n is called the **end-state**. If it further holds that for all $t'_i \in S \setminus \{t_i\}$, $(t_{i-1}, a_i, t'_i) \notin T$, for every $1 \leq i \leq n$, then $\vec{a}$ is called **deterministic forward-applicable in** t_0.*

A solution of the abstract transition system $s^\alpha_0, a_1, s^\alpha_1, \ldots, a_n, s^\alpha_n$ is called **spurious**, if its label

sequence $\langle a_1, \ldots, a_n \rangle$ is not forward-applicable in s_0, or if it is forward-applicable in s_0, leading to the path $s_0, a_1, s_1, \ldots, a_n, s_n$, but s_n is not a goal state, $s_n \notin S_*$.

We distinguish between three different classes of spurious solutions in order to find the label that must be added to the current label subset to remove the considered path from the abstract transition system.

If the label sequence of a spurious solution is deterministic forward-applicable in the abstract initial state, but it is not forward-applicable in the initial state of the original transition system, then it is enough to add only one label to the label subset in order to remove the considered path from the abstract transition system:

Theorem 1 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and let α be a K-catching bisimulation for Θ, with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. Then it holds for any state $t_0 \in S$ and for any action sequence $\langle a_1, \ldots, a_n \rangle \in L^n$ so that $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in $\alpha(t_0)$: If $\langle a_1, a_2, \ldots, a_n \rangle$ is forward-applicable in $\alpha(t_0)$, and $\langle a_1, a_2, \ldots, a_{n-1} \rangle$ is forward-applicable in t_0, but $\langle a_1, a_2, \ldots, a_n \rangle$ is not forward-applicable in t_0, then $a_n \notin K$.*

Proof: Assume for contradiction that $a_n \in K$. Because $\langle a_1, a_2, \ldots, a_{n-1} \rangle$ is forward-applicable in t_0 in Θ, there must be states $t_1, \ldots, t_{n-1} \in S$, s.t. $(t_{i-1}, a_i, t_i) \in T$, for all $1 \leq i < n$. Then it follows by the definition of the abstract transition system, for the states $t_0^\alpha = \alpha(t_0)$, $t_i^\alpha := \alpha(t_i)$, that $(t_{i-1}^\alpha, a_i, t_i^\alpha) \in T^\alpha$, for all $1 \leq i < n$. By assumption $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in t_0^α, implying that $t_0^\alpha, a_1, \ldots, a_{n-1}, t_{n-1}^\alpha$ is the *only* path in Θ^α with this label sequence and starting in t_0^α. Now, we know that $\langle a_1, \ldots, a_n \rangle$ is forward-applicable in t_0^α. As $t_0^\alpha, a_1, \ldots, a_{n-1}, t_{n-1}^\alpha$ is the only path matching the prefix $\langle a_1, \ldots, a_{n-1} \rangle$, we must have a transition $(t_{n-1}^\alpha, a_n, t_n^\alpha) \in T^\alpha$, for some abstract state $t_n^\alpha \in S^\alpha$. We can conclude by definition of Θ^α that there must be states $t'_{n-1}, t'_n \in S$ with $\alpha(t'_{n-1}) = t_{n-1}^\alpha$, $\alpha(t'_n) = t_n^\alpha$ and $(t'_{n-1}, a_n, t'_n) \in T$. Since α is a K-catching bisimulation for Θ and $a_n \in K$, it follows from (2) of Definition 1 that for every $t \in S$ with $\alpha(t) = t_{n-1}^\alpha$, there is a state $t' \in S$ with $\alpha(t') = t_n^\alpha$ such that $(t, a_n, t') \in T$. So in particular there is a state $t_n \in S$ with $\alpha(t_n) = t_n^\alpha$ such that $(t_{n-1}, a_n, t_n) \in T$. Therefore $\langle a_1, a_2, \ldots, a_n \rangle$ is actually forward-applicable in t_0 in Θ, which contradicts the assumption. It follows that a_n cannot be contained in K. So $a_n \notin K$. ∎

Let $s_0^\alpha, a_1, \ldots, a_m, s_m^\alpha$ be a spurious solution whose label sequence $\langle a_1, \ldots, a_m \rangle$ is not forward-applicable in the initial state of the original transition system. To refine the current label subset based on this path, we find the index $1 \leq n \leq m$ such that the sub-sequence $\langle a_1, \ldots, a_{n-1} \rangle$ is forward-applicable in s_0, but $\langle a_1, \ldots, a_n \rangle$ is not. Under the assumption that $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in s_0^α, Theorem 1 implies that adding a_n to the current label subset will either remove the entire path from the abstract transition system, or it will at least destroy the deterministic forward-applicability of $\langle a_1, \ldots, a_{n-1} \rangle$ in s_0^α.

If the label sequence of the spurious path is actually forward-applicable in the initial state of the original transition system, then it can only be spurious if its execution in the original transition system does not end in a goal state.

Theorem 2 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and let α be a K-catching bisimulation for Θ, with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. It holds for any state $t_0 \in S$ and for any action sequence $\langle a_1, \ldots, a_n \rangle \in L^n$, such that $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in $\alpha(t_0)$: If $\langle a_1, \ldots, a_n \rangle$ is forward-applicable in $\alpha(t_0)$ with end-state $t_n^\alpha \in S_*^\alpha$, and $\langle a_1, a_2, \ldots, a_n \rangle$ is forward-applicable in t_0, but for all possible end-states $t_n \notin S_*$, then $a_n \notin K$.*

Proof: Let $t_1, \ldots, t_n \in S$ be states so that $(t_{i-1}, a_i, t_i) \in T$, for all $1 \leq i \leq n$. Such states must exist since $\langle a_1, a_2, \ldots, a_n \rangle$ is applicable in t_0. By definition of Θ^α follows that $(\alpha(t_{i-1}), a_i, \alpha(t_i)) \in T^\alpha$, for all $1 \leq i \leq n$. By assumption $\langle a_1, \ldots, a_{n-1} \rangle$ is deterministic forward-applicable in $\alpha(t_0)$, implying that $\alpha(t_0), a_1, \ldots, a_{n-1}, \alpha(t_{n-1})$ is the only path in Θ^α with this label sequence and starting in $\alpha(t_0)$. But this means that t_n^α can only be an end-state of a forward-application of $\langle a_1, \ldots, a_n \rangle$ in $\alpha(t_0)$ if there is a transition $(\alpha(t_{n-1}), a_n, t_n^\alpha) \in T^\alpha$. By definition of Θ^α follows that there are states $t'_{n-1}, t'_n \in S$ with $\alpha(t'_{n-1}) = \alpha(t_{n-1})$ and $\alpha(t'_n) = t_n^\alpha$ such that $(t'_{n-1}, a_n, t'_n) \in T$. Rule (1) of Definition 1, together with $t_n^\alpha \in S_*^\alpha$ imply that $t'_n \in S_*$, which by assumption means that $t'_{n-1} \neq t_{n-1}$. Moreover, this rule implies that $\alpha(t_n) \neq t_n^\alpha = \alpha(t'_n)$. Therefore, we have found two states t_{n-1}, t'_{n-1} with $\alpha(t_{n-1}) = \alpha(t'_{n-1})$ such that there is a transition $(t'_{n-1}, a_n, t'_n) \in T$, but for all transition $(t_{n-1}, a_n, t_n) \in T$ holds that $\alpha(t_n) \neq \alpha(t'_n)$. Now, rule (2) of Definition 1 implies that $a_n \notin K$. ∎

Let $s_0^\alpha, a_1, s_1^\alpha \ldots, a_m, s_m^\alpha$ be a spurious solution such that $\langle a_1, \ldots, a_m \rangle$ is forward applicable in s_0, meaning that the execution of this label sequence in s_0 does not end in a goal state. Under the assumption that $\langle a_1, \ldots, a_{m-1} \rangle$ is deterministic forward-applicable in s_0^α, we can conclude with Theorem 2 that adding a_m to the current label subset will either remove the entire path from the abstract transition system, or it will at least destroy the deterministic forward-applicability of $\langle a_1, \ldots, a_{n-1} \rangle$ in s_0^α.

Finally, if the label sequence of a spurious path is not deterministic forward-applicable in the initial state of the abstract transition system, then we remove the non-deterministic choice along the execution of the considered label sequence in the abstract transition system.

Definition 4 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and let $l \in L$. Θ is **non-deterministic in** l if there are transitions $(s, l, t) \in T$ and $(s, l, t') \in T$, for some states $s, t, t' \in S$, such that $t \neq t'$.*

If the label sequence of the considered spurious solution is not deterministic forward-applicable in the abstract initial state, then the abstract transition system must be non-deterministic in at least one label, contained in this sequence. If such a label has been identified, then it is enough to add

it to the current label subset, in order to remove the non-determinism from the abstract transition system:

Theorem 3 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and let α be a K-catching bisimulation for Θ with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. If Θ^α is non-deterministic in $l \in L$, but Θ is not, then $l \notin K$.*

Proof: Because of Θ^α is non-deterministic in l, there must be states $s^\alpha, t_0^\alpha, t_1^\alpha \in S^\alpha$ with $t_0^\alpha \neq t_1^\alpha$ and $(s^\alpha, l, t_0^\alpha) \in T^\alpha$, $(s^\alpha, l, t_1^\alpha) \in T^\alpha$. By definition of Θ^α follows that there are states $s, s', t_0, t_1 \in S$ with $\alpha(s) = \alpha(s') = s^\alpha$, $\alpha(t_0) = t_0^\alpha$, $\alpha(t_1) = t_1^\alpha$ and $(s, l, t_0) \in T$, $(s', l, t_1) \in T$. By assumption Θ is deterministic in l, so $s = s'$ cannot hold, since $t_0 \neq t_1$. Therefore there are transitions $(s, l, t) \in T$ and $(s', l, t') \in T$ for $s \neq s'$, but $\alpha(s) = \alpha(s')$ and $\alpha(t) \neq \alpha(t')$. This means that $l \notin K$, otherwise α would not be a K-catching bisimulation. ∎

Let $s_0^\alpha, a_1, s_1^\alpha \ldots, a_m, s_m^\alpha$ be a spurious solution such that the label sequence $\langle a_1, \ldots, a_m \rangle$ is not deterministic forward-applicable in s_0^α. We find the first index $1 \leq n \leq m$, so that Θ^α is non-deterministic in a_n, and add this label a_n to the current label subset. Since the state space of a planning task is by definition deterministic, it follows by Theorem 3 that adding a_n to the label subset is sufficient to exclude the non-determinism of a_n from the abstract transition system.

Backward Propagation

Backward propagation analyzes the abstract transition system in a similar way as forward propagation. It considers also the optimal solutions of the abstract transition system, but instead of starting with the initial state and testing for the forward-applicability of the labels, backward propagation starts with the goal states of the original transition system and tries to *regress* these states with the label sequences of the solutions, until the initial state has been reached. Formally, regression is defined as follows:

Definition 5 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system, $S' \subseteq S$ be a set of states, and $l \in L$ be a label. Then **regressing** S' **with** l is defined by $regr(S', l) = \{s \in S \mid (s, l, s') \in T, s' \in S'\}$. For a label sequence $\vec{a} = \langle a_1, \ldots, a_n \rangle \in L^n$, $regr(S', \vec{a}) = regr(regr(\ldots.(regr(S', a_n), \ldots), a_1)$. Moreover, $\vec{a}$ is called **reverse-applicable** in S' if $regr(S', \vec{a}) \neq \emptyset$.*

For $S' = \{s\}$ we often write $regr(s, l)$ instead of $regr(S', l)$. Then a solution $s_0^\alpha, a_1, s_1^\alpha, \ldots, a_n, s_n^\alpha$ of the abstract transition system is called **spurious** if $s_0 \notin regr(S_*, \langle a_1, \ldots, a_n \rangle)$.

First, observe that if all labels of a label sequence are contained in the label subset, then regressing an abstract state t^α is actually equivalent to regressing the states t of the original transition system, with $\alpha(t) = t^\alpha$:

Lemma 1 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system, and α be a K-catching bisimulation for Θ with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. Moreover, let $t^\alpha \in S^\alpha$ be an abstract state, and $l \in K$ be label. If $s^\alpha \in regr_\alpha(t^\alpha, l)$, then for every state $s \in Pre_\alpha(s^\alpha)$, there is a state $t \in Pre_\alpha(t^\alpha)$ with $s \in regr(t, l)$.*

Proof: Let $s^\alpha \in regr_\alpha(t^\alpha, l)$. By definition of $regr_\alpha$, there is a transition $(s^\alpha, l, t^\alpha) \in T^\alpha$. Then, by definition of Θ^α follows that there are states $s, t \in S$ with $\alpha(s) = s^\alpha$ and $\alpha(t) = t^\alpha$, so that $(s, l, t) \in T$. Since α is a K-catching bisimulation for Θ and $l \in K$, it holds for all $s' \in S$, such that $\alpha(s') = \alpha(s)$, that there is a state $t' \in S$ with $\alpha(t') = \alpha(t)$ so that $(s', l, t') \in T$, and therefore $s' \in regr(t', l)$. So for every state $s \in Pre_\alpha(s^\alpha)$, there is a state $t \in Pre_\alpha(t^\alpha)$ with $s \in regr(t, l)$. ∎

Lemma 1 implies that if no goal state of the original transition system can be regressed with the label sequence of a spurious path, then there must be some label on this sequence that is not contained in the current label subset.

Theorem 4 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and α be a K-catching bisimulation for Θ with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. Then for any $\langle a_1, \ldots, a_n \rangle \in L^n$ and state $t^\alpha \in S^\alpha$ holds: If $\langle a_1, \ldots, a_n \rangle$ is reverse-applicable in t^α, and $\langle a_2, \ldots, a_n \rangle$ is reverse-applicable in some $t \in Pre_\alpha(t^\alpha)$, but $\langle a_1, \ldots, a_n \rangle$ is not reverse-applicable in any $t \in Pre_\alpha(t^\alpha)$, then there must be a label a_i with $a_i \notin K$, for some $1 < i \leq n$.*

Proof: Assume for contradiction that all a_i are already contained in K, i.e. $a_i \in K$, for all $1 < i \leq n$. Let $r^\alpha \in regr_\alpha(t^\alpha, \langle a_1, \ldots, a_n \rangle)$. Such an abstract state must exist because this label sequence is reverse-applicable in t^α. By definition of $regr_\alpha$, there must be a transition $(r^\alpha, a_1, s^\alpha) \in T^\alpha$, for some state $s^\alpha \in regr_\alpha(t^\alpha, \langle a_2, \ldots, a_n \rangle)$. By definition of Θ^α, there must be states $r, s \in S$, $\alpha(r) = r^\alpha$, $\alpha(s) = s^\alpha$, so that $(r, a_1, s) \in T$. Now, when recursively applying Lemma 1, then it follows by $s^\alpha \in regr(t^\alpha, \langle a_2, \ldots, a_n \rangle)$ that there is a state $t \in Pre_\alpha(t^\alpha)$ such that $s \in regr(t, \langle a_2, \ldots, a_n \rangle)$. Therefore, $r \in regr(t, \langle a_1, \ldots, a_n \rangle)$ which contradicts the assumption that $\langle a_1, \ldots, a_n \rangle$ is not reverse-applicable in t. Hence, there must be at least one $1 < i \leq n$, such that $s_i \notin K$. ∎

Let $s_0^\alpha, a_1, s_1^\alpha, \ldots, a_n, s_n^\alpha$ be a spurious solution such that the label sequence $\langle a_1, \ldots, a_n \rangle$ is not reverse-applicable in all goal states $s_* \in S_*$. We find the index $1 \leq i < n$, so that $\langle a_{i+1}, \ldots, a_n \rangle$ is reverse-applicable in some $s_* \in S_*$, but $\langle a_i, \ldots, a_n \rangle$ is not reverse-applicable in any $s_* \in S_*$. Theorem 5 implies that adding the labels $a_{i+1}, \ldots, a_n$ to the current label subset is sufficient to exclude the considered spurious solution from the abstract transition system.
Moreover, for spurious paths whose label sequences are actually reverse-applicable in some goal state of the original transition system, but the regression does not end in the initial state, it follows immediately by Lemma 1 that there must be at least one label in this sequence that is not already contained in the current label subset.

Theorem 5 *Let $\Theta = (S, L, c, T, s_0, S_*)$ be a transition system and α be a K-catching bisimulation for Θ with abstract transition system $\Theta^\alpha = (S^\alpha, L, c, T^\alpha, s_0^\alpha, S_*^\alpha)$. Then for any state $t^\alpha \in S^\alpha$ and $\langle a_1, \ldots, a_n \rangle \in L^n$ holds: If $s_0^\alpha \in regr_\alpha(t^\alpha, \langle a_1, \ldots, a_n \rangle)$, and $s_0 \notin regr(t, \langle a_1, \ldots, a_n \rangle)$, for any $t \in Pre_\alpha(t^\alpha)$, then there must be an action a_i with $a_i \notin K$, for some $1 \leq i \leq n$.*

procedure *analyze*(Π, Θ^α)
  $P \leftarrow$ compute the first S optimal solutions of Θ^α
  $labels \leftarrow \emptyset$
  for $p \in P$ **do**
    if p is spurious **then**
      $labels \leftarrow labels \cup$ *exclude*(p)
    endif
  done
  return $labels$

Figure 2: Analyze procedure, as called by the abstraction refinement procedure (Figure 1) within each refinement step.

Proof: Assume for contradiction that $a_i \in K$, for all $1 \leq i \leq n$. Since $s_0^\alpha \in regr_\alpha(t^\alpha, \langle a_1, \ldots, a_n \rangle)$ and $\alpha(s_0) = s_0^\alpha$, by definition of Θ^α, it follows by recursively applying Lemma 1 that $s_0 \in regr(t, \langle a_1, \ldots, a_n \rangle)$, for some $t \in Pre_\alpha(t^\alpha)$. But this contradicts the assumption. Therefore, it must hold $a_i \notin K$, for at least one $1 \leq i \leq n$. ∎

Let $s_0^\alpha, a_1, s_1^\alpha, \ldots, a_n, s_n^\alpha$ be a spurious solution such that the label sequence $\vec{a} = \langle a_1, \ldots, a_n \rangle$ is reverse-applicable in some goal state $s_* \in S_*$. This means that $s_0 \notin regr(s_*, \vec{a})$ for all $s_* \in S_*$, and it follows from Theorem 5 that adding the labels $a_1, \ldots, a_n$ to the current label subset is sufficient to remove the considered spurious solution from the abstract transition system.

A trivial consequence of our results is that, if we keep running abstraction refinement and adding labels as described, then eventually all abstract solutions will actually be plans for the original planning task. Of course, we can stop as soon as that is the case for at least one abstract solution. As one would expect, such a stopping criterion is impractical: The abstractions required for it to occur are, in most cases, infeasibly large. In our implementation, described next, we use earlier cut-offs.

Implementation

Our techniques are implemented in *Fast Downward* (Helmert 2006), as an extension of the M&S approach. The overall refinement procedure was already depicted in Figure 1. It has 4 input parameters: (1) forward vs. backward propagation; (2) the termination criterion; (3) the number S of abstract solutions considered in any refinemrnt step; and (4) a size limit L on the abstract transition system. Parameters (1) and (3) are used in the analyze function, described next. Parameters (2) and (4) are used in the termination criterion.

Analysis Procedure

The *analyze* function is depicted in Figure 2. It computes the labels *that will be added to* K, in order to refine the K-catching bisimulation. Our current code is optimized for readability rather than efficiency, storing the entire set of optimal abstract solutions in memory, prior to analyzing them (instead, one could generate and analyze each solution one-by-one). As there can be a huge number of such solutions, this can cause serious memory issues. For the moment, we control this via the parameter S: As soon as S abstract solutions have been generated, we stop and proceed to the analysis step.

During the analysis each abstract solution is analyzed in either forward or backward manner, as specified by parameter (1). Whenever the considered solution is spurious, then *exclude* will apply the theorems shown above and returns the corresponding labels.

Termination Criteria

A practical termination criterion cannot require that an optimal solution of the abstract transition system matches exactly to a solution of the original transition system. However, to still obtain a useful abstraction heuristic, one has to require that the abstract optimal solutions correspond at least somewhat to paths of the original transition system. We experimented with 3 different kinds of termination criteria:

- **All.** This criterion is satisfied if none of the extracted optimal abstract solutions is spurious. This extreme criterion just serves to illustrate what happens when putting all computational effort into the abstraction.
- **Lower Bounding (LB).** This criterion requires a lower bound on the solution cost h_{min}, i.e., $h_{min} \leq h^*(s_0)$. When the cost of the optimal abstract solution is greater or equal than this value, then the refinement loop will be terminated. In other words, as soon as for the current K-catching bisimulation α holds that $h^\alpha(s_0) \geq h_{min}$, the refinement loop will be stopped.
- **Cost Increase Threshold (CIT).** This criterion requires a threshold $\beta \in [1, \infty)$. It essentially tests whether the increase of the estimated goal distance between the K-catching bisimulation α_i, and the K'-catching bisimulation α_{i+1} of two consecutive refinement steps i and $i+1$ is higher than the minimum allowed increase. In other words, the refinement of the abstraction will be stopped as soon as the quotient $h^{\alpha_{i+1}}(s_0)/h^{\alpha_i}(s_0)$ is less than the given β.

Empirical Evaluation

We ran a total of 32 different variants of our abstraction refinement approach, 7 other M&S configurations, two competing heuristics, and a blind search instance on a total of 280 instances of the 14 benchmarks from the track of optimal planners at IPC'11. The experiments are performed on an Intel Core i7-3770K processor, limiting the run time to 5 minutes and the memory usage to 2 GB.

We ran the abstraction refinement procedure with both forward propagation and backward propagation. The termination criterion is set to either *All*, or *LB* with $h_{min} = h^1(s_0)$ (Haslum and Geffner 2000), or *CIT* with $\beta \in \{1.40, 2.0\}$. For (3), we extract a maximum of either $S = \infty$, or $S = 10K$ optimal solutions from the abstract transition system. For (4), we set a bound on the number of states of the abstract transition system: $L = 100K$, or $L = \infty$.

We ran also BJOLP (Domshlak et al. 2011) and LM-cut (Helmert and Domshlak 2009), two methods that were used in the portfolio winning the 1st prize in the track of optimal planners at IPC 11.

Refinement	Backward propagation							
Termination	CIT $\beta = 1.4$		CIT $\beta = 2.0$		LB h^1		All	
L	∞	$100K$	∞	$100K$	∞	$100K$	∞	$100K$
barman	AM=4, SM=8, AT=8	AM=4, SM=8, AT=8	AM=4, SM=8, AT=5, ST=3	AM=4, SM=8, AT=5, ST=3	AM=12, AT=7, ST=1	AM=12, AT=7 ST=1	AM=12, AT=8	AM=12, AT=8
elevators	C=**10**, AM=5, SM=5	C=**10**, AM=5, SM=5	C=**10**, AM=5, SM=5	C=**10**, AM=5, SM=5	C=7, AM=13	C=8, AM=9, SM=3	C=3, AM=17	C=8, AM=9, SM=3
floortile	C=**2**, AM=18	C=**2**, AM=18	C=**2**, AM=18	C=**2**, AM=18	C=**2**, SM=18	C=**2**, SM=18	C=**2**, AM=18	C=**2**, AM=18
nomystery	C=**18**, SM=2	C=**18**, SM=2	C=**18**, SM=2	C=**18**, SM=2	C=12, SM=8	C=12, SM=8	C=8, AM=12	C=14, AM=4, SM=2
openstacks	C=0, AM=15, AT=5	C=**3**, AM=15, AT=2	C=0, AM=15, AT=5	C=**3**, AM=14, AT=3	C=**3**, AM=14, AT=3	C=**3**, AM=15, AT=2	C=0, AM=15, AT=5	C=**3**, AM=14, AT=3
parcprinter	C=10, AM=9, SM=1	C=10, AM=9, SM=1	C=10, AM=9, SM=1	C=10, AM=9, SM=1	C=**11**, SM=9	C=**11**, SM=9	C=6, AM=10, AT=4	C=9, AM=9, SM=1, AT=1
parking	C=1, AM=7, SM=9, ST=3	C=1, AM=7, SM=8, ST=4	C=1, AM=7, SM=8, ST=4	C=1, AM=7, SM=8, ST=4	C=**6**, SM=14	C=**6**, SM=14	C=0, AM=20	C=0, AM=20
pegsol	C=1, AM=14, AT=5	C=4, AM=14, SM=2	C=3, AM=15, AT=2	C=4, AM=14, SM=2	C=**9**, AM=9, SM=2	C=**9**, AM=9, SM=2	C=1, AM=15, AT=4	C=4, AM=14, SM=2
scanalyzer	C=4, AM=15	C=4, AM=15	C=4, AM=15	C=4, AM=15	C=**9**, AM=3, SM=7	C=**9**, AM=3, SM=7	C=3, AM=16	C=3, AM=16
sokoban	C=**9**, AM=11	C=**9**, AM=11	C=**9**, AM=11	C=**9**, AM=11	C=6, AM=14	C=6, AM=14	C=0, AM=20	C=0, AM=20
tidybot	C=1, AM=19	C=1, AM=19	C=**11**, AM=6, ST=3	C=10, AM=6, ST=4	C=3, AM=17	C=3, AM=17	C=1, AM=19	C=1, AM=19
transport	C=**6**, SM=14	C=**6**, SM=14	C=**6**, SM=14	C=**6**, SM=14	C=5, AM=14, SM=1	C=**6**, AM=9, SM=5	C=4, AM=14, AT=2	C=**6**, AM=9, SM=5
visitall	C=8, AM=3, AT=9	C=8, AM=3, AT=9	C=8, AM=3, AT=9	C=8, AM=3, AT=9	C=**16**, SM=4	C=**16**, SM=4	C=8, AM=2, AT=10	C=8, AM=3, AT=9
woodworking	C=2, AM=17, SM=1	C=3, AM=16, SM=1	C=3, AM=16, SM=1	C=3, AM=16, SM=1	C=**5**, SM=15	C=**5**, SM=15	C=2, AM=18	C=3, AM=17
$\sum$	C=72, AM=137, SM=40, AT=27, ST=3	C=79, AM=136, SM=41, AT=19, ST=4	C=85, AM=124, SM=39, AT=21, ST=10	C=88, AM=122, SM=41, AT=17, ST=11	C=94, AM=96, SM=78, AT=10, ST=1	C=**96**, AM=88, SM=85, AT=9, ST=1	C=38, AM=208, AT=33	C=61, AM=184, SM=13, AT=21

Table 1: Experiment data for the abstraction refinement variants. S is always set to 10K. C represents the number of completed tasks. The other values represent the number of violations of the requirements, split into violations of the memory requirement (during refinement process: AM, during the search: SM), and into violations of the run time requirement (during the refinement process: AT, during the search: ST). The highest amount of solved task per domain is highlighted in bold.

We ran 4 M&S configurations based on K-catching bisimulations. The label subset is either computed by the approximation technique *Backward* h^1, or by the approximation technique *IntAbs* (Katz, Hoffmann, and Helmert 2012), and the size limit is set to either $N = 100K$, or $N = \infty$. Additionally, we ran 2 variants of greedy-bisimulation (Nissim, Hoffmann, and Helmert 2011), one with size limit $N = 100K$, and the other one with $N = \infty$. Finally, we ran another M&S instance using full bisimulation, without using a size limit ($N = \infty$).

The size limit N influences the size of the abstract transition system in a different way than L does. If the amount of states of the abstract transition system reaches this bound N, then the shrinking strategy of the M&S abstraction is forced to aggregate more states by dropping the bisimulation requirement. In contrast, whether the size limit L is exceeded is only tested between the refinement steps. If it is, then the refinement process is aborted, so one catches less labels instead of dropping the bisimulation requirement.

Backward h^1 computes the label subset by collecting all labels that occur within the radius, given by the product of $h^1(s_0)$ and some parameter $\beta \in [0, 1]$, around the goal states of the actual state space, i. e. all labels that are used in an optimal path to some goal state, with cost less or equal than this radius. If $\beta = 0$, then the smallest β is considered that leads a non-empty label subset. *IntAbs* computes the label subset by computing the standard bisimulation until some size limit M is reached. Afterwards, the exact label subset in the resulting abstract transition system, i. e. either all globally relevant, or all $h^\alpha(s_0)$-relevant labels, is computed. *Greedy bisimulation* relaxes the bisimulation criterion by ignoring all transitions (s, l, t) with $h^*(s) < h^*(t) + c(l)$, i.e., transitions that are not used in any optimal solution.

The number of solved tasks, as well as the number of violations of the run time and memory requirement for selected abstraction refinement variants are shown in Table 1. We show data only for backward refinement and for $S = 10K$, as the respective other settings are dominated consistently ($S = \infty$, resulted, in most of the cases, in a violation of the memory requirement within the first few refinement steps). In Table 2, we show the average ratio of considered labels for solved tasks, as well as the average amount of refinement steps for solved tasks.

No variant of abstraction refinement could solved any task of *barman*. The backward propagation variant, using *All* and $L = \infty$, shows that excluding all (extracted) spurious so-

Refinement	Backward propagation							
Termination	CIT $\beta = 1.4$		CIT $\beta = 2.0$		LB h^1		All	
L	∞	$100K$	∞	$100K$	∞	$100K$	∞	$100K$
elevators	9.4, 2.1	9.4, 2.1	9.4, 2.1	9.4, 2.1	15.4, 2.9	12.1, 2.5	49.0, 12.3	17.0, 3.2
floortile	25.0, 1.0	25.0, 1.0	25.0, 1.0	25.0, 1.0	0.0, 0.0	0.0, 0.0	50.7, 2.0	50.7, 2.0
nomystery	14.4, 1.2	14.4, 1.2	1.8, 1.0	1.8, 1.0	0.0, 0.0	0.0, 0.0	51.1, 3.4	30.7, 2.6
openstacks	–	57.3, 1.3	–	57.3, 1.3	45.8, 1.0	45.8, 1.0	–	57.3, 1.3
parcprinter	11.3, 1.0	11.3, 1.0	11.3, 1.0	11.3, 1.0	0.0, 0.0	0.0, 0.0	37.3, 14.8	26.7, 8.0
parking	0.3, 1.0	0.3, 1.0	0.3, 1.0	0.3, 1.0	0.0, 0.0	0.0, 0.0	–	–
pegsol	55.1, 4.0	10.8, 2.0	48.6, 3.0	10.8, 2.0	2.8, 1.1	2.8, 1.1	55.1, 4.0	10.8, 2.0
scanalyzer	29.6, 1.2	29.6, 1.2	20.1, 1.0	20.1, 1.0	0.0, 0.0	0.0, 0.0	58.5, 2.0	39.6, 1.7
sokoban	5.7, 1.1	5.7, 1.1	5.7, 1.1	5.7, 1.1	0.0, 0.2	0.0, 0.2	–	–
tidybot	0.1, 1.0	0.1, 1.0	0.1, 1.0	0.1, 1.0	0.0, 0.7	0.0, 0.7	0.1, 1.0	0.1, 1.0
transport	9.3, 2.3	9.3, 2.3	7.4, 2.0	7.4, 2.0	16.7, 7.4	14.9, 5.2	26.5, 15.0	16.5, 5.8
visitall	60.8, 1.1	60.8, 1.1	53.5, 1.0	53.5, 1.0	0.0, 0.0	0.0, 0.0	81.9, 2.4	76.0, 1.6
woodworking	24.2, 1.0	20.1, 1.0	20.1, 1.0	20.1, 1.0	0.0, 0.0	0.0, 0.0	38.1, 3.0	25.8, 1.3
⌀	20.4, 1.5	19.5, 1.3	16.9, 1.4	17.1, 1.3	6.2, 1.0	5.8, 0.8	44.8, 6.0	31.9, 2.8

Table 2: Results for backward propagation. S is always set to 10K. The first value is the average ratio of caught labels (%), for solved tasks. The last value is the average amount of refinement steps, for solved tasks.

lutions from the abstract transition system is not feasible in practice. It can only solve 38 instances within the given memory and run time limits. The main reason for not finishing a task was the violation of the memory requirement during the construction of the abstractions. This implies that the resulting abstract transition systems are too big to store in memory. Bounding the size of the abstract transition system, i.e., $L = 100K$, improves the performance of *All*, while reducing the number of violations of the memory, and run time constraints (during the refinement process). Overall, it can solve 23 tasks more than the version without size limit. As can be observed in Table 2, bounding the size of the abstract transition system for *All* reduces the average amount of refinement steps, and consequently reduces the overall amount of caught labels. This does generally hold, independent of the used termination criterion. However, the difference between $L = 100K$ and $L = \infty$ for the other termination criteria is not as big as for *All*. This implies that the termination criteria *CIT* ($\beta \in \{1.4, 2.0\}$), and *LB* h^1 are often fulfilled even before the size limit is reached.

Comparing $\beta = 1.4$ and $\beta = 2.0$ for the termination criterion *CIT*, the latter is considerably better. For $L = \infty$, *CIT* with $\beta = 2.0$ is equally good in 11 domains, and it is better in 3 domains. Overall, it can solve 13 tasks more than *CIT* with $\beta = 1.4$, while reducing the number of violations of the memory requirement during the refinement process. When using $\beta = 2.0$, one terminates also slightly faster, i.e., one catches less labels, than when using $\beta = 1.4$.

Consider the termination criterion *LB* h^1. Surprisingly, many entries of Table 2 are 0. This implies that the first abstract transition system, i.e., the abstract transition system of a K-catching bisimulation with empty K, does already fulfill the termination criterion *LB* h^1 in many cases; evidently, in these cases $h^1(s_0)$ is very small. But if the termination criterion holds initially, then there is actually no refinement step executed, which results in an empty label subset. However, in our experiments, using backward propagation, *LB* h^1 and $L = 100K$ was the best abstraction refinement variant.

Comparing the per domain results of *LB* h^1 and *CIT* with $\beta = 2.0$, without a size limit, the former one can solve more tasks in 7 domains, is equally good in 2 domains, and worse in 5 domains. The comparison of the reasons for not completing a task shows that the *CIT* variant mainly fails due to a violation of the memory, or run time constraint during the refinement process, whereas *LB* h^1 often fails due to a violation of the memory constraint during the search. That is only logical: A $\emptyset$-catching bisimulation is very cheap to compute – it only distinguishes non-goal states from goal states – but does not provide a lot of search guidance.[1]

Table 3 compares the coverage data of the best abstraction refinement variant with the results of the non abstraction-refinement configurations.

In general, for all other M&S variants, using the size limit $N = 100K$ can solve more tasks than the equivalent variants without a size limit. *Greedy bisimulation* with bounded size is almost identical to *Backward* h^1 with bounded size. Moreover, comparing *IntAbs* and *Backward* h^1, both with $N = 100K$, the former one can solve slightly more tasks.

Comparing the best abstraction refinement variant, i.e., using backward propagation with the termination criterion LB h^1, $L = 100K$, and $S = 10K$, with *IntAbs*, using $N = 100K$, and $M = 10K$, then the abstraction refinement approach is better in 1 domain, equally good in 4 domains, and worse in 9 domains. It gives almost identical results in 3 domains. Overall, the best configuration of this experiment is LM-cut. Comparing it to the best M&S configuration, *IntAbs* with $N = 100K$ and $M = 10K$, LM-cut can solve more tasks in 9 domains, it is equally good in 4 domains, and worse in 1 domain.

Conclusions

In theory, abstraction refinement appears to be a suitable way to extract meaningful label subsets for K-catching bisimulations. Practical results, thus far, are disappointing. Despite this, we believe the approach is not without hope. As our empirical data vividly demonstrates, our current strategies

[1]Note that the failures during abstraction, for LB, are almost exclusively due to domains where K is *not* empty.

Approach	Abstraction Refinement Backward, *LB* h^1		IntAbs		Backward h^1		Greedy bisim.		Full bisim.	LM-cut	BJOLP	Blind
L/N	∞	$100K$	∞	$100K$	∞	$100K$	∞	$100K$	∞			
$S/M/\beta/M$	$10K$	$10K$	$10K$	$10K$	0	0.25	∞	$100K$	∞			
barman	0	0	**4**	**4**	**4**	**4**	0	**4**	0	**4**	**4**	**4**
elevators	7	8	9	11	9	11	4	11	4	**15**	14	9
floortile	2	2	2	2	2	2	2	2	2	**6**	2	2
nomystery	12	12	15	14	13	**20**	12	**20**	12	14	**20**	8
openstacks	3	3	**14**	**14**	0	**14**	3	**14**	3	12	11	**14**
parcprinter	11	11	12	12	11	12	8	12	8	**13**	10	6
parking	**6**	**6**	5	**6**	0	0	0	0	0	1	1	0
pegsol	9	9	**17**	**17**	13	**17**	15	**17**	14	**17**	**17**	**17**
scanalyzer	9	9	9	9	3	8	3	8	3	**10**	3	9
sokoban	6	6	19	19	17	20	4	19	5	**20**	**20**	17
tidybot	3	3	5	5	8	0	0	0	0	13	**14**	6
transport	5	**6**	**6**	**6**	**6**	**6**	4	**6**	4	**6**	**6**	**6**
visitall	**16**	**16**	9	9	8	9	8	9	8	10	10	9
woodworking	5	5	6	6	5	6	3	7	4	**10**	9	2
$\sum$	94	96	132	134	99	129	66	129	67	**151**	141	109

Table 3: Comparison of solved tasks over the selected domains. Best results are highlighted in bold.

spend way too much time and memory in the abstraction process. There are many possibilities to control the practical CEGAR process differently, reducing that overhead.

A major point regards greedier termination criteria. We have already seen that setting β in CIT to 2 rather than 1.4 reduces abstraction effort considerably. A first step thus simply is to systematically experiment with largr values of β. For LB, following one of the methods proposed by Katz et al., one could also introduce a β parameter and terminate as soon as the heuristic value has reached $h^1(s_0) * \beta$.

Another issue is the amount of abstract solutions considered, which was huge in many cases. It appears that many spurious solutions share the same error label, and therefore it would be enough to consider only a subset of representative solutions in the refinement process. How to find such good subsets efficiently is an interesting open question.

Finally, label reduction has not been covered in this work, but in some cases can reduce the size of the abstract transition system drastically. It remains to investigate how label reduction can be integrated into our approach.

Acknowledgments. We thank the anonymous HSDIP'13 reviewers, whose comments helped to improve the paper.

References

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the Association for Computing Machinery* 50(5):752–794.

Domshlak, C.; Helmert, M.; Karpas, E.; Keyder, E.; Richter, S.; Röger, G.; Seipp, J.; and Westphal, M. 2011. BJOLP: The big joint optimal landmarks planner. In *IPC 2011 planner abstracts*, 91–95.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, R.; and Knoblock, C., eds., *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS-00)*, 140–149. Breckenridge, CO: AAAI Press, Menlo Park.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In Howe, A., and Holte, R. C., eds., *Proceedings of the 22nd National Conference of the American Association for Artificial Intelligence (AAAI-07)*, 1007–1012. Vancouver, BC, Canada: AAAI Press.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169. AAAI Press.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In Boddy, M.; Fox, M.; and Thiebaux, S., eds., *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07)*, 176–183. Providence, Rhode Island, USA: Morgan Kaufmann.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Katz, M.; Hoffmann, J.; and Helmert, M. 2012. How to relax a bisimulation? In Bonet, B.; McCluskey, L.; Silva, J. R.; and Williams, B., eds., *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press.

Milner, R. 1990. Operational and algebraic semantics of concurrent processes. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*. Elsevier and MIT Press. 1201–1242.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh, T., ed., *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11)*, 1983–1990. AAAI Press/IJCAI.