Reachlunch Entering The Unsolvability IPC 2016

Tomáš Balyo*
Karlsruhe Institute of Technology
Karlsruhe, Germany
biotomas@gmail.com

Martin Suda†
Vienna University of Technology
Vienna, Austria
msuda@forsyte.tuwien.ac.at

Abstract

Reachlunch is a sequential portfolio planner designed to recognize unsatisfiable planning instances. In the first stage it runs blind depth-first search. If the problem is not solved by DFS then it is encoded into propositional satisfiability using the Compact Reinforced encoding. The encoded problem is handed to our (non)reachability solver based on the PDR/IC3 algorithm and implemented on top of the SAT solver Minisat.

Introduction

Reachlunch is a planner designed to recognize unsatisfiable planning instances. Although the main focus of the planning community has traditionally been on satisfiable (solvable) problems only, more recently the importance of detecting unsatisfiable (unsolvable) instances is getting recognized and addressed (see, e.g., Bäckström, Jonsson, and Ståhlberg, 2013; Hoffmann, Kissmann, and Álvaro Torralba, 2014). This is also testified by the emergence of the Unsolvability planning competition (Muise and Lipovetzky, 2016).

The main motivation behind Reachlunch is to explore the potential for the detection of unsatisfiable planning instances of Property Directed Reachability (PDR), also called IC3, a very successful algorithm developed in the model checking community (Bradley, 2011; Eén, Mishchenko, and Brayton, 2011). As explained by Suda (2014), PDR is designed for deciding reachability in symbolically represented transition systems, which is a representation to which a PDDL planning benchmark can be translated in a straightforward way by using most of the standard encoding schemes of the planning as satisfiability paradigm (Kautz and Selman, 1996). Reachlunch uses Reinforced Encoding (Balyo, Barták, and Trunda, 2015) for that purpose.

Reachlunch is a portfolio system and complements the power of PDR with another engine (actually executed first, for a limited amount of time) based on blind depth first search. In the following sections we describe the individual ingredients behind the design of Reachlunch.

*Supported by DFG project SA 933/11-1.

†Supported by the ERC Starting Grant 2014 SYMCAR 639270 and the Austrian research project FWF RiSE S11409-N23.

Symbolic Transition Systems

By a Symbolic Transition System (STS) we mean a finite state transition system described using the language of propositional logic, namely conjunctive normal form (CNF). A transition system is a graph having states as vertices and transitions between states as edges. There is a distinguished subset of vertices for initial states and another for goal states. We are interested in answering whether there exists a path from an initial state leading to a goal state. As a symbolic description, an STS can be exponentially more succinct than the explicit enumeration of the transition system's states. However, basic questions concerning the existence of an initial (or goal) state or the task of enumerating state's successors need to be in general delegated to a SAT solver.

Formally, an STS is a tuple $S = (\Sigma, I, U, G, T)$, where $\Sigma = \{x, y, \ldots\}$ is a finite signature, i.e. a finite set of propositional variables, I, G, U are sets of clauses over Σ, and T is a set of clauses over $\Sigma \cup \Sigma'$, where $\Sigma' = \{x', y', \ldots\}$ is the set of variables for the next state, a distinct copy of Σ. The set of states of S is formed by all the Boolean valuations over Σ which satisfy the U-clauses. Of these, those that also satisfy I are the initial states and those that also satisfy G are the goal states. There is a transition between states s and t iff $(s, t') \models T$, where t' is the valuation that works on the variables of Σ' in the same way as t works on those of Σ, i.e. $t'(x') = t(x)$ for any $x \in \Sigma$.

It is easy to observe that a planning problem can be translated into an STS. In fact, most of the standard encoding schemes of the planning as satisfiability paradigm (Kautz and Selman, 1996) can be used for this purpose. At the same time, STS can be used as an input of many reachability checking algorithms developed by the hardware model checking community, in particular PDR.

The SAT Encoding used by Reachlunch

To obtain an STS, Reachlunch uses the Reinforced Encoding (Balyo, Barták, and Trunda, 2015), which is a combination of the traditional Direct Encoding (Kautz and Selman, 1992), which encodes state variable values, and SASE (Huang, Chen, and Zhang, 2010), which encodes the transitions between the values of state variables in the planning problem. The Reinforced Encoding encodes both, which is redundant in the sense of Boolean variables, but on other hand it reduces the number of clauses in the formula

and enhances unit propagation. This helps SAT solvers to solve the formulas faster.

Property Directed Reachability

PDR is best understood as a hybrid between an explicit and a symbolic search of the given STS S. It explicitly constructs a path consisting of concrete states, starting from a goal state and regressing it step by step towards an initial state. (The opposite direction of traversal is also possible.) At the same time, it maintains symbolic reachability information, which is locally refined whenever the current path cannot be extended further. The reachability information guides the path construction and is bound to eventually converge to a certificate of non-reachability, if no path of arbitrary length exists.

In more detail, the reachability information takes the form of a sequence F_i of sets of clauses. The first set in the sequence, F_0, is fixed to be equal to I. Each of the following sets F_i *over-approximates* the image of F_{i-1} with respect to the transition relation. These clause sets play a role similar to an admissible heuristic. They represent a lower bound estimate for the distance of a state to the initial state and thus provide a means to guide the search towards it. However, while a heuristic value of a particular state is normally computed only once and it remains constant during the search for a plan, the clause sets in PDR are refined continually. The refinement happens on demand, driven by the states encountered during the search.

Since the path construction happens in the context of a concrete encoding, PDR can be likened to an instance of the planning as satisfiability approach in which the construction of the assignment is controlled to grow only in one direction. PDR also proceeds iteratively, gradually disproving existence of plans of length $k = 0, 1, 2, \ldots$. After each iteration, however, a special *clause propagation* phase tries to bring as many clauses as possible from F_i to F_{i+1} while preserving their logical relation. If it is achieved during this phase that $F_i = F_{i+1}$ for some $i < k$, the algorithm terminates having shown that there is no path connecting the goal and initial state. In a nutshell, the proof is inductive and consists of the following three claims:

$$I \models F_i, \quad F_i \wedge T \models F_i', \quad F_i \wedge G \models \bot.$$

The individual single-step reachability queries within the STS are typically implemented by a call to a SAT solver. However, for a simple encoding of STRIPS problems an explicit polynomial time procedure can be devised. More details on this interesting algorithm presented from the planning perspective can be found in (Suda, 2014).

The Other Engine: Blind Depth First Search

Our depth first search algorithm traverses the search space of the planning instance in a depth-first fashion by systematically trying each applicable action in each reachable state while avoiding revisiting already explored states. The ordering of the actions is guided by a trivial heuristic that prefers actions leading to states that are similar to the goal state. The heuristic simply counts the number of state variables that have the same values as the goal values of the particular variables.

The overall architecture

Reachlunch is a portfolio system running in two stages. The first stage executes the just described simple brute-force depth-first-search (DFS) for a limited amount of time. If the DFS cannot traverse all the reachable states within its time limit and prove that no solution exists then the second stage is executed.

The second stage is relies on Property Directed Reachability. The input problem is encoded into an STS and then handed over to an implementation of PDR. We designed a new file format for this exchange, which we call DIMSPEC (Suda, 2016). It is a simple modification of the well-known DIMACS CNF format used by most SAT solvers extended to define the four individual clause sets of an STS – I, U, G, T.

Implementation Details

Our planner takes input in the SAS+ format (Bäckström and Nebel, 1995). For benchmarks provided in the PDDL format we use Fast Downward (Helmert, 2006) to translate them into the SAS+ format.

The actual implementation of PDR is called minireachIC3 and is freely available (Suda, 2013). It relies on Minisat version 2.2 (Eén and Sörensson, 2003) as the backend SAT solver and is implemented in the C++ language.

The depth first search and the translation of SAS+ to SAT is implemented withing the Freelunch planning library (Balyo, 2016) which is written in Java. Hence the name of our planner – Reachlunch – Freelunch combined with reachability reasoning.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for sas+ planning. *Computational Intelligence* 11:625–656.

Bäckström, C.; Jonsson, P.; and Ståhlberg, S. 2013. Fast detection of unsolvable planning instances using local consistency. In Helmert, M., and Röger, G., eds., *SOCS*. AAAI Press.

Balyo, T.; Barták, R.; and Trunda, O. 2015. Reinforced encoding for planning as sat. In *Acta Polytechnica CTU Proceedings*.

Balyo, T. 2016. Freelunch, an open-source java planner and planning library. Web site, `http://freelunch.eu`.

Bradley, A. R. 2011. SAT-based model checking without unrolling. In Jhala, R., and Schmidt, D. A., eds., *VMCAI*, volume 6538 of *LNCS*, 70–87. Springer.

Eén, N., and Sörensson, N. 2003. An extensible SAT-solver. In Giunchiglia, E., and Tacchella, A., eds., *SAT*, volume 2919 of *Lecture Notes in Computer Science*, 502–518. Springer.

Eén, N.; Mishchenko, A.; and Brayton, R. K. 2011. Efficient implementation of property directed reachability. In Bjesse, P., and Slobodová, A., eds., *FMCAD*, 125–134. FMCAD Inc.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)* 26:191–246.

Hoffmann, J.; Kissmann, P.; and Álvaro Torralba. 2014. "Distance"? Who cares? Tailoring Merge-and-Shrink heuristics to detect unsolvability. In *ICAPS 2014 Workshop on Heuristics and Search for Domain-independent Planning (HSDIP)*. To appear.

Huang, R.; Chen, Y.; and Zhang, W. 2010. A novel transition based encoding scheme for planning as satisfiability. In Fox, M., and Poole, D., eds., *AAAI*. AAAI Press.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of ECAI*, 359–363.

Kautz, H. A., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic and stochastic search. In Clancey, W. J., and Weld, D. S., eds., *AAAI/IAAI, Vol. 2*, 1194–1201. AAAI Press / The MIT Press.

Muise, C., and Lipovetzky, N. 2016. Unsolvability international planning competition. http://unsolve-ipc.eng.unimelb.edu.au/.

Suda, M. 2013. minireachIC3, a Minisat-based implementation of PDR. Web site, `https://github.com/quickbeam123/minireachIC3`.

Suda, M. 2014. Property directed reachability for automated planning. *J. Artif. Intell. Res. (JAIR)* 50:265–319.

Suda, M. 2016. DIMSPEC, a format for specifying symbolic transition systems. Web site, `http://forsyte.at/dimspec/`.