Automatic Polytime Reductions of
NP Problems into a Fragment of STRIPS

Aldo Porco
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
aldo@gia.usb.ve

Alejandro Machado
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
alejandro@gia.usb.ve

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Abstract

We present a software tool that is able to automatically translate an NP problem into a STRIPS problem such that the former problem has a solution iff the latter has one, a solution for the latter can be transformed into a solution for the former, and all this can be done efficiently. Moreover, the tool is built such that it only produces problems that belong to a fragment of STRIPS that is solvable in non-deterministic polynomial time, a fact that guarantees that the whole approach is not an overkill. This tool has interesting applications. For example, with the advancement of planning technology, it can be used as an off-the-shelf method to solve general NP problems with the help of planners and to automatically generate benchmark problems of known complexity in a systematic and controlled manner. Another interesting contribution is related to the area of Knowledge Engineering in which one of the goals is to devise automatic methods for using the available planning technology to solve real-life problems.

Introduction

Deciding plan existence for STRIPS is PSPACE-complete (Bylander 1994). This means that any decision problem that can be solved by a deterministic algorithm that uses polynomial space can be reduced in polynomial time to deciding plan existence of a STRIPS problem. However, although such reductions exist, there is no known algorithm that automatically generates a STRIPS problem from an arbitrary PSPACE problem (up to our knowledge).

Such an algorithm would be a valuable tool for a number of reasons. First and most importantly, the algorithm would provide the basis for developing a General Problem Solver able to tackle PSPACE problems, suitably described in a high-level declarative language, by translating them into STRIPS and then applying one of the many available planners. Second, given the recent (and future) advancements in the area, the algorithm would be of practical interest too because it would offer an easy way to solve concrete instances of difficult problems and, in some cases, competitively with other specialized algorithms. Finally, by generating STRIPS problems in a controlled manner, one could design benchmark problems in order to evaluate the heuristics or algorithms used in planning. All these applications assume in

one way or another that the input is a declarative description of a PSPACE problem, that a solution for the input problem can be efficiently computed from a solution to the planning problem, and that solving the latter problem is no more difficult than solving the former problem.

In this paper we take a first step in constructing such a tool by considering the class NP instead of PSPACE. Nonetheless, the result is an interesting tool that is able to translate a given NP problem, expressed as a decision problem, into a STRIPS problem that satisfies above properties. We present the formal translation and its properties, and evaluate it on classical NP-complete problems such as satisfiability and the computation of Hamiltonian paths on digraphs.

The input problem is specified using the existential fragment of second-order logic that is known to capture NP. This is a fundamental result in the area of Descriptive Complexity Theory (DCT) on which the tool is firmly grounded. On the other hand, the guarantee that the planning problem is no more difficult than the input (in the worst case) is achieved by targeting a class of STRIPS problems for which plan existence can be decided in NP.

Part of the contribution is related to the area of Knowledge Engineering for Planning and Scheduling (KEPS) that focuses on the practical deployment of planning resources for solving real-life problems. The tool presented in this work produces a planning problem that can be fed into a planner from a declarative description of a NP problem. Thus, the tool can be thought as a KEPS tool that permits the use of planning technology for solving real-life problems that are not directly specified in PDDL. However, differently from other tools, our framework permits to formally characterize the scope, soundness and completeness of the tool, and to obtain worst-case guarantees on the complexity of solving the generated problems.

Our tool is not the first such tool. Cadoli and Schaerf (2005) develop one that translates a DATALOG-like specification of an NP problem, called NP-SPEC, into a SAT problem that is then fed into a solver to find a solution to the input problem. Hence, our proposal is quite similar to NP-SPEC, yet we target STRIPS instead of SAT. Furthermore, unrestricted STRIPS as well as specifications based on second-order logic go well beyond NP, and thus we have a clear direction for extending the tool in the future.

Another related work is that of Mitchell and Ternovska

(2005) that proposes a general framework for describing problems, in NP and beyond, that is based on the Model Extension (MX) problem. A simple parameterization of MX renders the decision problem in NP, while fully unrestricted MX is in NEXPTIME. Mitchell and Ternovska propose a language for expressing general problems, but do not present a practical solver based on their ideas.

In the following, we revise the relevant concepts from DCT, describe the tool, give the formal translation from NP problems into STRIPS, and study its properties. At the end, we present experiments and conclude with a discussion.

Descriptive Complexity Theory

It is a field of research, lying in the intersection of mathematics and computer science, that studies complexity theory from a pure mathematical viewpoint without committing to a model of computation such as Turing machines. DCT began with the seminal work of Fagin (1974) on NP. Today, the major complexity classes had been characterized and important results had been obtained (Immerman 1998).

In DCT, a decision problem like SAT[1] is denoted as a collection of finite (first-order) structures that satisfy a second-order existential sentence. In this section, we review the fundamental concepts from logic and the most important results from DCT that are relevant to this work.

Languages

Every logical language is constructed from a set of symbols or vocabulary. The symbols are typically partitioned into pure logical symbols such as '$\wedge$', '$\exists$', etc., punctuation symbols such as '(' and ')', and relational, functional and constant symbols. The logical and punctuation symbols belong to every language while the relations, functions and constants change from one language to another. Hence, it is convenient to define the *signature* of the language as the finite set of relations, functions and constants that are allowed in the formulae. Signatures are denoted by tuples like $\sigma = \langle P^1, Q^2, f^1, A, B \rangle$ that contains two relational symbols P and Q of arities 1 and 2 respectively, one functional symbol f of arity 1, and two constants A and B. In DCT, functional symbols can be avoided altogether and thus will not be considered in the rest of the paper. We denote with FOL(σ) and SOL(σ) the sets of all first-order and second-order formulae built from σ. In general, if $\mathcal{L}$ denotes a logic or fragment of a logic, $\mathcal{L}(\sigma)$ denotes the set of all formulae belonging to $\mathcal{L}$ built from σ.

SOL differs from FOL in that quantification on relational symbols is permitted. SOL formulae are constructed as

– any FOL(τ) formula, $\tau \supseteq \sigma$, is a SOL(σ) formula,

– if Φ and Ψ are SOL(σ) formulae, then so are '(Φ)', '$\neg\Phi$', '$\Phi \wedge \Psi$', '$\Phi \vee \Psi$', etc., and

– if $R^a \notin \sigma$ and Φ is a SOL(σ) formula, then '($\exists R)\Phi$' and '($\forall R)\Phi$' are SOL(σ) formulae.

We typically denote FOL formulae with lowercase Greek letters, and other formulae with capital letters. For the rest

---

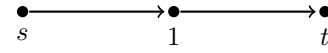[1]In this paper, SAT is the subset of satisfiable CNF formulae.

Figure 1: the digraph that corresponds to the structure $\mathcal{A} = \langle |\mathcal{A}|, E^{\mathcal{A}}, s^{\mathcal{A}}, t^{\mathcal{A}} \rangle$ where $|\mathcal{A}| = \{0, 1, 2\}$, $E^{\mathcal{A}} = \{(0, 1), (1, 2)\}$, $s^{\mathcal{A}} = 0$ and $t^{\mathcal{A}} = 2$.

of the paper, we only consider (first- and second-order) formulae without free variables, also called sentences, and for second-order formulae, those that comply with the form

$$\Phi = (\mathcal{Q}_1 R_1^{a_1})(\mathcal{Q}_2 R_2^{a_2}) \cdots (\mathcal{Q}_n R_n^{a_n})\psi$$

where each $\mathcal{Q}_i$ is a quantifier in $\{\exists, \forall\}$, and ψ is a first-order sentence over $\sigma \cup \{R_1^{a_1}, \ldots, R_n^{a_n}\}$. This form is universal because for every second-order sentence there is an equivalent of this form. If all $\mathcal{Q}_i$s are existential quantifiers, then we say that Φ is a second-order existential sentence. The class of all second-order existential sentences, also called the existential fragment of SOL, is denoted by SO$\exists$; similarly, one defines SO$\forall$. If there is $1 < k < n$, such that $\mathcal{Q}_i = \exists$ for all $i < k$ and $\mathcal{Q}_i = \forall$ for all $i \geq k$, then the sentence belongs to the fragment SO$\exists\forall$, and so on.

First-Order Structures

Logical formulae are interpreted with respect to first-order structures. A first-order structure over signature $\sigma = \langle R_1^{a_1}, \ldots, R_s^{a_s}, c_1, \ldots, c_t \rangle$, where R_i is an a_i-ary relational symbol and c_j is a constant, is a tuple $\mathcal{A} = \langle |\mathcal{A}|, R_1^{\mathcal{A}}, \ldots, R_s^{\mathcal{A}}, c_1^{\mathcal{A}}, \ldots, c_t^{\mathcal{A}} \rangle$ with non-empty universe $|\mathcal{A}|$ where each $R_i^{\mathcal{A}} \subseteq |\mathcal{A}|^{a_i}$ is a subset of a_i-tuples from $|\mathcal{A}|$ (called the interpretation of R_i) and each $c_j^{\mathcal{A}} \in |\mathcal{A}|$ is an element of $|\mathcal{A}|$ (called the interpretation of c_j). Without loss of generality, we assume that the universe is always of the form $|\mathcal{A}| = \{0, 1, \ldots, n-1\}$. DCT is only interested in finite structures (i.e. structures of finite universe); the class of all finite structures for signature σ is denoted by STRUC[σ].

We do not have enough space to formally present the semantic interpretation of formulae with respect to structures. We only say that a formula ϕ in FOL(σ) (or SOL(σ)) holds (or is satisfied) in structure $\mathcal{A} \in$ STRUC[σ] iff the formula holds when each relation R_i and constant c_j is interpreted by $R_i^{\mathcal{A}}$ and $c_j^{\mathcal{A}}$ respectively. If ϕ holds in $\mathcal{A}$, we write $\mathcal{A} \models \phi$. For example, consider $\sigma = \langle E^2, s, t \rangle$ and $\mathcal{A} = \langle |\mathcal{A}|, E^{\mathcal{A}}, s^{\mathcal{A}}, t^{\mathcal{A}} \rangle$ where $|\mathcal{A}| = \{0, 1, 2\}$, $E^{\mathcal{A}} = \{(0, 1), (1, 2)\}$, $s^{\mathcal{A}} = 0$ and $t^{\mathcal{A}} = 2$. Then, $\mathcal{A} \models (\exists x)(E(s, x) \wedge E(x, t))$ and $\mathcal{A} \nvDash (\forall x)(\exists y)E(x, y)$. Notice that σ can describe digraphs with designated vertices s and t; e.g., $\mathcal{A}$ corresponds to the graph shown in Fig. 1.

Second-order formulae are also interpreted with respect to first-order structures. For example, the sentence

$$\Phi_{2\text{COL}} = (\exists R^1)(\forall xy)[E(x, y) \rightarrow \neg(R(x) \leftrightarrow R(y))]$$

holds in $\mathcal{A}$ since the unary relation $R = \{1\}$ makes it true. Indeed, this formula holds in a structure $\mathcal{A} \in$ STRUC[σ] iff the digraph encoded by $\mathcal{A}$ is 2-colorable.

The class of finite structures in STRUC[σ] that satisfy a sentence $\Phi \in \mathcal{L}(\sigma)$ is denoted by MOD[Φ]; e.g., MOD[$\Phi_{2\text{COL}}$] is the class of all finite 2-colorable digraphs.

DCT requires some basic numeric relations and constants that have fixed interpretation at each structure; the relations are $<^2$, SUC^2, BIT^2, PLUS^3 and TIMES^3 while the constants are 0 and max. For the interpretations, $x < y$ iff x is less than y, $\text{SUC}(x, y)$ iff $y = x + 1$, $\text{BIT}(x, y)$ iff the yth-bit in the number x is 1, $\text{PLUS}(x, y, z)$ iff $z = x + y$, and $\text{TIMES}(x, y, z)$ iff $z = x \times y$. The constants 0 and max are mapped into 0 and $\|\mathcal{A}\| - 1$ respectively.

Complexity Classes

The last example shows that a sentence can describe a collection of finite discrete structures (such as digraphs) that satisfy a certain property (such as 2-colorability).

In DCT, a decision problem corresponds to a class of first-order structures that satisfy a sentence. The seminal work of Fagin (1974) established that every decision problem in NP can be characterized by the class of structures that satisfy a second-order existential sentence; in symbols, NP = SO∃. Consider SAT, for example. An instance of SAT is a CNF with m clauses over n propositional variables, where a clause is a subset of positive and negative literals. Two relational symbols $\sigma_{\text{SAT}} = \langle N^2, P^2 \rangle$ suffice to describe the positive and negative occurrences of propositions in clauses: $N(x, y)$ (resp. $P(x, y)$) expresses that the proposition x appears negatively (resp. positively) in clause y; e.g., $(p \vee \neg q \vee r) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee q)$ is encoded with $\mathcal{A} = \langle |\mathcal{A}|, N^{\mathcal{A}}, P^{\mathcal{A}} \rangle$ where $|\mathcal{A}| = \{0, 1, 2\}$, $N^{\mathcal{A}} = \{(1, 0), (0, 1), (2, 1), (0, 2)\}$ and $P^{\mathcal{A}} = \{(0, 0), (2, 0), (1, 2)\}$. On the other hand, a truth-assignment is a subset of true propositions and the existence of a truth assignment (satisfiability) can be expressed with the SO∃ sentence Φ_{SAT}:[2]

$$(\exists T^1)(\forall y)(\exists x)[(P(x, y) \wedge T(x)) \vee (N(x, y) \wedge \neg T(x))]$$

The following list shows the major results of DCT on the characterization of complexity classes (Immerman 1998):

- non-deterministic log-space (NL) equals FOL extended with a transitive-closure operator (FO(TC)),
- P equals second-order Horn sentences (SO-Horn),
- NP equals SO∃ and coNP equals SO∀,
- the polynomial-time hierarchy (PH) equals SO, and
- PSPACE equals SOL extended with a transitive-closure operator (SO(TC)).

A transitive-closure operator is a *syntactic construct* whose interpretation coincides with the transitive closure of a relation. Thus, it is not surprising that NL equals FO(TC) because checking the existence of a path from node s to node t in a digraph with designated vertices s and t is NL-complete (Sipser 2005), and this property holds whenever s is related to t in the transitive closure of the edge relation.

Syntactic Abbreviations

Quite often one needs to quantify over a k-ary function f^k instead of a relation. This can be accomplished by quantifying over a $(k + 1)$-ary relation F^{k+1} and adding first-order

---

[2]This sentence assumes that $m \geq n$. If not, add tautological clauses to the CNF.

---

formulae that guarantees that F represents f. For example, a unary function f can be represented by the binary relation F and the formula

$$\psi_{\text{fun}} = (\forall xyy')(F(x, y) \wedge F(x, y') \rightarrow y = y').$$

Likewise, in need of an injective function, one quantifies over a relation F and adds above formula plus

$$\psi_{\text{inj}} = (\forall xx'y)(F(x, y) \wedge F(x', y) \rightarrow x = x').$$

Finally, if the function needs to be total, then one should use $\psi_{\text{tot}} = (\forall x)(\exists y)F(x, y)$. We use the abbreviations $(\exists F \in \text{Fun})\phi$ and $(\exists F \in \text{Inj})\phi$ to denote $(\exists F^2)(\phi \wedge \psi_{\text{fun}} \wedge \psi_{\text{tot}})$ and $(\exists F^2)(\phi \wedge \psi_{\text{fun}} \wedge \psi_{\text{inj}} \wedge \psi_{\text{tot}})$ respectively, and 'PFun' instead of 'Fun' and 'PInj' instead of 'Inj' if the function does not need to be total. For example, the following sentence defines digraphs with directed Hamiltonian paths

$$\Phi_{\text{DHP}} = (\exists F \in \text{Inj})(\forall x)[x < \max \rightarrow$$
$$(\exists x'yz)(E(y, z) \wedge F(x, y) \wedge \text{SUC}(x, x') \wedge F(x', z))].$$

To see how it works, observe that a directed Hamiltonian path over the vertices $|\mathcal{A}| = \{0, \ldots, n-1\}$ can be thought of as a permutation f on the vertices such that $E(f(x), f(x + 1))$ for each $0 \leq x < n - 1$.

The Tool

The input to the tool is a signature σ, a SO∃ sentence Φ describing a property of interest, and a first-order structure $\mathcal{A}$ describing an object on which to test the property Φ. The output is a PDDL domain and instance that have a valid plan if and only if $\mathcal{A}$ satisfies Φ. Moreover, a *certificate* for the satisfaction of the property, in the form of values for the second-order variables in Φ, can be recovered in linear time from the plan.

Hence, we can think of the tool as a generator of reductions among problems. Recall that a reduction from problem A into problem B is a computable function f such that for each instance ω, $\omega \in A$ iff $f(\omega) \in B$.

In our case, the reduction decomposes in two functions

$$\mathfrak{D} : \text{Signatures} \times \text{SO∃} \rightarrow \text{PDDL Domains},$$
$$\mathfrak{I} : \text{Signatures} \times \text{SO∃} \times \text{STRUC} \rightarrow \text{PDDL Instances}$$

such that $\text{dom} = \mathfrak{D}(\sigma, \Phi)$ is a PDDL domain and $\text{ins} = \mathfrak{I}(\sigma, \Phi, \mathcal{A})$ is a PDDL instance.

In order to get something of theoretical and practical interest, the reduction should run in polynomial time (so that it would not be able to check whether $\mathcal{A}$ satisfies Φ and then produce a trivial planning problem) and its output should be solvable in NP (so that complexity of solving the output problem is no bigger than the complexity of solving the input problem). However, the plan-existence decision problem for PDDL is not in NP because 1) the number of grounded fluents and actions may be exponential in the input size, and 2) a minimum-length plan may be of exponential size in the number of grounded fluents and actions. Thus, not every reduction works for our purposes and we must be careful with its design. In the following two sections, we present an acceptable reduction and study its formal properties.

Reductions

A (grounded) STRIPS planning problem is a tuple $P = \langle F, I, G, O \rangle$ where F is a collection of fluents (propositions), $I \subseteq F$ denotes the initial state, $G \subseteq F$ denotes the goal states and O is a collection of actions. Each action $a \in O$ is defined by three subsets of fluents $pre(a)$, $add(a)$ and $del(a)$ that stand for the precondition, and the positive and negative effects of the action. As usual, action a is applicable at state s iff $pre(a) \subseteq s$, and the result of applying a is $res(s, a) = (s \setminus del(a)) \cup add(a)$. A plan for state s is a sequence of actions applicable from the initial state I that achieves the goal condition.

A PDDL planning problem is a pair $\langle \mathsf{dom}, \mathsf{ins} \rangle$ made of a domain and instance description in the PDDL language (McDermott et al. 1998). In this paper, we only consider the simplest fragment of PDDL which is equivalent to ungrounded STRIPS, and hence the *grounding* of $\langle \mathsf{dom}, \mathsf{ins} \rangle$ results in an STRIPS problem P. The size of the grounding is polynomial for *fixed* domain, but exponential for unrestricted domain and instance.

Getting an acceptable reduction is not obvious at the beginning, but once you get one, others become apparent. For lack of space, we present only one reduction that is aimed at SAT-based planners. We are aware of other reductions, including one designed for optimal sequential planners that produce a delete-free problem together with a length bound.

The first step in the translation is to transform the formula by eliminating the implications and bi-implications, and moving the negations to the literal level. Further, constants other than 0 and max are removed by introducing unary relational symbols with interpretations consisting only of the unique element that interprets each constant. After the formula is preprocessed in this manner, the domain and problem are generated as follows.

Domain

$\mathfrak{D}(\sigma, \Phi)$ produces a domain for signature σ and sentence $\Phi \in \mathrm{SOL}(\sigma)$ of the form $(\exists R_1^{a_1}) \cdots (\exists R_n^{a_n}) \psi$. The actions in the domain are divided in three groups: actions for setting the truth-value of the second-order variables, actions for proving the sentence ψ and two other actions.

Actions for variables For each second-order variable R_i or arity a_i, there is an action `set_Ri_true` with a_i parameters that sets the fluent `Ri` and removes `not-Ri` which is *initially true*; these fluents are used to denote the truth value of R_i. For example, the action for the relation T^1 in SAT is

```
(:action set_T_true
  :parameters (?x)
  :precondition (and (guess) (not-T ?x))
  :effect (and (T ?x) (not (not-T ?x)))))
```

The fluent `guess` is used to create two phases within plans: a 'guess' phase for setting the value of quantified relations, and a 'proof' phase for showing the validity of ψ.

Actions for formulae These actions are designed following the structure of ψ and make use of fluents that denote the validity of the subformulae in ψ.

For each subformula θ, there is a fluent $\mathfrak{F}[\theta]$ that denotes its validity in the structure and actions to add it. The fluent $\mathfrak{F}[\theta]$ has parameters that match the free variables in θ. The function $\mathfrak{F}[\cdot]$ is called the fluent translation and it is closely related to the Tseitin translation (Tseitin 1968).

The actions are generated by recursing over the subformulae θ of ψ in a depth-first manner as follows:[3]

- if $\theta(\bar{x}) = \bigwedge_{i=1}^n \theta_i(\bar{x}_i)$ with $\bar{x} = \cup_{i=1}^n \bar{x}_i$, then generate the action $\mathrm{prove}[\theta]$ with parameters $\bar{x}$, precondition $\bigwedge_{i=1}^n \mathfrak{F}[\theta_i](\bar{x}_i)$ and unique add effect $\mathfrak{F}[\theta](\bar{x})$,

- if $\theta(\bar{x}) = \bigvee_{i=1}^n \theta_i(\bar{x}_i)$ with $\bar{x} = \cup_{i=1}^n \bar{x}_i$, then generate n actions of the form $\mathrm{prove}[\theta]_i(\bar{x}_i)$ with precondition $\mathfrak{F}[\theta_i](\bar{x}_i)$ and unique add effect $\mathfrak{F}[\theta](\bar{x})$,

- if $\theta(\bar{x}) = (\exists y) \theta'(\bar{x}, y)$, then generate $\mathrm{prove}[\theta](\bar{x}, y)$ with precondition $\mathfrak{F}[\theta'](\bar{x}, y)$ and unique add effect $\mathfrak{F}[\theta](\bar{x})$,

- if $\theta(\bar{x}) = (\forall y) \theta'(\bar{x}, y)$ then generate two actions. The idea is to prove $\theta(\bar{x})$ by varying y over all objects.

  The first action $\mathrm{prove}[\theta]_0(\bar{x})$ shows $\theta'(\bar{x}, 0)$. The action has parameters $\bar{x}$, precondition $\mathfrak{F}[\theta'](\bar{x}, 0)$ and unique effect $\mathfrak{F}[(\forall y \leq z)\theta'(\bar{x}, y)](\bar{x}, 0)$. (Observe that the fluent translation is applied to a different formula in which the quantification is bounded by z.)

  The second action $\mathrm{prove}[\theta]_1(\bar{x}, z', z'')$ inductively proves $(\forall y \leq z)\theta'(x, y)$ once $(\forall y < z)\theta'(x, y)$ holds. The action has parameters $\bar{x}, z', z''$, precondition $\mathfrak{F}[(\forall y \leq z)\theta'(\bar{x}, y)](\bar{x}, z') \wedge \mathfrak{F}[\theta'](\bar{x}, z'') \wedge \mathrm{SUC}(z', z'')$ and unique add effect $\mathfrak{F}[(\forall y \leq z)\theta'(\bar{x}, y)](\bar{x}, z'')$.

All these actions have as additional precondition the fluent `proof`. Also, notice that there are no actions for literals as such are taken care by the fluent translation as follows:

- $\mathfrak{F}[Q(\bar{x})](\bar{x}) = Q(\bar{x})$,

- $\mathfrak{F}[\neg Q(\bar{x})](\bar{x}) = \mathtt{not\text{-}}Q(\bar{x})$,

- $\mathfrak{F}[(\forall y)\theta'(\bar{x}, y)](\bar{x}) = \mathfrak{F}[(\forall y \leq z)\theta'(\bar{x}, y)](\bar{x}, \max)$,

- in all other cases, $\mathfrak{F}[\theta](\bar{x}) = \mathtt{holds\_{<}id{>}}(\bar{x})$ where `<id>` is a unique identifier for θ.

Other actions Two other actions are required. One for switching the phase from 'guess' to 'proof' called `begin-proof` that has precondition `guess`, adds `proof` and removes `guess`, and another action called `prove-goal` that has precondition $\mathfrak{F}[\psi]$ and unique add effect `holds_goal`. Figure 2 shows the domain for Φ_{SAT}.

Abbreviations In the presence of abbreviations, the operators for the second-order variables are extended in order to make the translations more efficient. For $(\exists F \in \mathrm{Fun})$, the precondition and delete of `set_F_true` are extended with the fluent `(free_F_dom ?x)` so that there can be at most one fluent $F(x, y)$ true for each x and thus there is no need to include the subformula ψ_{fun}. Similarly, for $(\exists F \in \mathrm{Inj})$ the precondition and delete are further extended with the fluent `(free_F_ran ?y)`.

---

[3] $\theta(\bar{x})$ means that the free variables in θ are among those in $\bar{x}$.

```
(define (domain SAT)
  (:constants zero max)
  (:predicates
    (holds_and_2 ?x ?y) (holds_and_6 ?x0 ?x1)
    (holds_exists_8 ?x0) (holds_forall_9 ?x0)
    (holds_or_7 ?x0 ?x1) (holds_goal)
    (N ?x ?y) (P ?x ?y) (T ?x) (not-T ?x)
    (suc ?x ?y) (guess) (proof)
  )
  (:action set_T_true
    :parameters (?x)
    :precondition (and (guess) (not-T ?x))
    :effect (and (T ?x) (not (not-T ?x))))

  (:action prove_forall_9_1
    :precondition (and (proof)
                       (holds_exists_8 zero))
    :effect (holds_forall_9 zero))

  (:action prove_forall_9_2
    :parameters (?y1 ?y2)
    :precondition (and (proof)
                       (suc ?y1 ?y2)
                       (holds_forall_9 ?y1)
                       (holds_exists_8 ?y2))
    :effect (holds_forall_9 ?y2))

  (:action prove_exists_8
    :parameters (?y ?x)
    :precondition (and (proof)
                       (holds_or_7 ?y ?x))
    :effect (holds_exists_8 ?y))

  (:action prove_or_7_0
    :parameters (?y ?x)
    :precondition (and (proof)
                       (holds_and_2 ?y ?x))
    :effect (holds_or_7 ?y ?x))

  (:action prove_or_7_1
    :parameters (?y ?x)
    :precondition (and (proof)
                       (holds_and_6 ?y ?x))
    :effect (holds_or_7 ?y ?x))

  (:action prove_and_2
    :parameters (?y ?x)
    :precondition (and (proof)
                       (P ?x ?y) (T ?x))
    :effect (holds_and_2 ?y ?x))

  (:action prove_and_6
    :parameters (?y ?x)
    :precondition (and (proof)
                       (N ?x ?y) (not-T ?x))
    :effect (holds_and_6 ?y ?x))

  (:action prove-goal
    :precondition (holds_forall_9 max)
    :effect (holds_goal))

  (:action begin-proof
    :precondition (guess)
    :effect (and (proof) (not (guess)))) )
```

Figure 2: Full domain translation for $\Phi_{\text{sat}} = (\exists T^1)(\forall y)(\exists x)$ $[(P(x,y) \wedge T(x)) \vee (N(x,y) \wedge \neg T(x))]$.

Problem

The PDDL problem is generated by the call $\Im(\sigma, \Phi, \mathcal{A})$. The objects in the problem correspond to the elements in the universe $|\mathcal{A}| = \{0, \ldots, n-1\}$: 0 is mapped to the object zero, $n-1$ to the object max, and the other elements $0 < i < n-1$ to objects obj_i. The goal is to achieve holds_goal, and the initial situation consists of fluents describing the truth-value of all the relations in $\mathcal{A}$ and the predefined relations such as $<$, SUC, etc. that are mentioned in Φ. Also, for each second-order variable R, the initial situation has fluents to denote false values for R, and in cases where R is a function, the initial situation has fluents of the type free_R_dom and/or free_R_ran.

Formal Properties

The most important properties to care about are soundness, completeness and the complexity guarantees. Soundness and completeness mean that the translation function actually implements a reduction between decision problems, while the complexity guarantees refer to the time to compute the reduction and the complexity of deciding plan existence on the generated problem. In this section we show that the tool is a polytime reduction from the NP problem MOD[Φ] into a fragment of STRIPS that is decidable in NP.

It is well known that checking plan existence for STRIPS problems without deletes is in NP (Bylander 1994). The proof relies on the fact that an optimal plan does not repeat actions and thus is of linear size. A similar complexity result for STRIPS can be obtained if each action with non-empty delete list can be applied at most once.

Definition 1. *A STRIPS problem $P = \langle F, I, G, O \rangle$ is at-most-once iff the operators can be partitioned into $O = O_0 \cup O_1$ such that all operators in O_0 are delete-free, and for each $a \in O_1$, there is a fluent $p \in pre(a) \cap del(a)$ that is added by no action; i.e., $p \notin add(a)$ for all $a \in O$. The class of all at-most-once problems is denoted by STRIPS-1.*

Consider now the *grounding* function $\mathfrak{G}$ that maps a pair $\langle \text{dom}, \text{ins} \rangle$ of PDDL domain and instance into a STRIPS problem $P = \mathfrak{G}(\text{dom}, \text{ins})$. For fixed dom, the function ins $\rightsquigarrow \mathfrak{G}(\text{dom}, \text{ins})$ runs in polytime $\mathcal{O}(\|\text{ins}\|^k)$ for some k that only depends on dom. Likewise, the translation function $\Im$ runs in polytime in the size of the structure $\mathcal{A}$, but exponential in the largest arity of a second-order existential quantifier in Φ. Therefore, the function $f_{\sigma, \Phi} : \text{STRUC}[\sigma] \rightarrow$ STRIPS defined by

$$f_{\sigma, \Phi}(\mathcal{A}) = \mathfrak{G}(\mathfrak{D}(\sigma, \Phi), \Im(\sigma, \Phi, \mathcal{A}))$$

is a polytime function that maps σ-structures into grounded STRIPS problems. This function is a reduction.

Theorem 2. *The function $f_{\sigma, \Phi}$ is a polytime reduction from the decision problem MOD[Φ] into STRIPS-1.*

Proof. (Sketch.) The proof is by structural induction on the subformulae θ of ψ, starting from literals and building up towards more complex subformulae. The statement to show is that $\langle \mathcal{A}, R_1^\pi, \ldots, R_n^\pi \rangle \vDash \theta(\bar{x})$ iff there is a plan π that achieves $\mathfrak{F}[\theta](\bar{x})$ and defines interpretations R_i^π for the second-order variables R_i. Here, $\langle \mathcal{A}, R_1^\pi, \ldots, R_n^\pi \rangle$ denotes
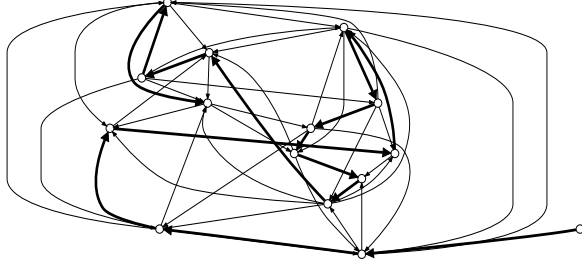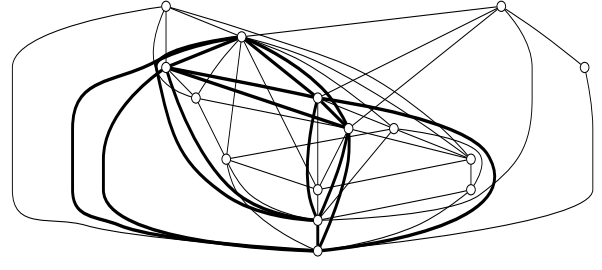
a) directed Hamiltonian path   *b*) 6-clique

Figure 3: Two examples of NP-Complete problems reduced to STRIPS and the solutions computed by an off-the-shelf planner. The left panel shows a digraph of 15 vertices with a Hamiltonian path and the right panel a graph of 15 vertices with a 6-clique.

the extension of $\mathcal{A}$ with the relations R_i^π that interpret the symbols R_i.

For $\mathcal{A} \in \text{STRUC}[\Phi]$, $f_{\sigma,\Phi}(\mathcal{A})$ is a STRIPS-1 problem because all operators are delete-free except `begin-proof` and the `set_R_true` operators, but each one of these deletes a precondition that is added by no operator. □

Corollary 3. *The plan-existence decision problem for STRIPS-1 is NP-complete.*

Proof. For inclusion, note that every action $a \in O_1$ can be applied at most once. Because each such action deletes fluents, then some (or all) actions in O_2 may be required before applying another action from O_1. Thus, the size of a plan is at most quadratic in the total number of actions. For hardness, $f_{\sigma_{\text{SAT}}, \Phi_{\text{SAT}}}$ reduces SAT into STRIPS-1 in polytime. □

Therefore, our tool translates any given problem in NP, encoded by a SO∃ sentence, into PDDL/STRIPS. Such a sentence specifies the properties of a solution in a declarative manner. The solution of the problem is obtained from the valuation of the second-order variables that make the sentence true and that is contained in any valid plan for the planning problem. For example, the assignment that satisfies the CNF encoded by a structure $\mathcal{A}$ corresponds to the values of the second-order variable T.

It is important to say that although SO∃ captures the whole class NP, there are problems that are easier to encode as sentences than others. For instance, there are succinct and clear sentences for SAT, Hamiltonian path, k-colorability, vertex cover and other problems, yet we do not have at this moment sentences for most of the benchmark problems used in planning. On the other hand, toy problems such as Blocksworld are not interesting and the exercise of abstracting relevant aspects of a real-world task into SO∃ sentences may reveal the core difficulties involved in a task.

In the rest of this section, we derive tight bounds on the length of parallel plans. These bounds are used with SAT-based planners to show that a given problem has no solution and also to improve performance.

Horizon Windows

A horizon window for a STRIPS problem P is an interval of the form $[s, f]$ such that P has a plan iff it has a plan of length $\ell \in [s, f]$. A window is a parallel-horizon window if ℓ refers to the makespan of a parallel plan. Horizon windows can be effectively used to prune the search space.

The recursive structure of the generated problem permits the calculation of non-trivial horizon windows and of tight parallel-horizon windows. Indeed, since all set operators can be applied concurrently, a parallel plan needs at most one time step to execute them. The plan also requires the operators `begin-proof` and `prove-goal`. Thus, the parallel-horizon window is $[2, 3]$ (the lower bound 2 applies when there is a plan that uses no set operators) plus the parallel-horizon window $mkspw(\psi)$ of the sentence ψ. The parallel-horizon window is inductively defined as

– $mkspw(\theta) \doteq [0, 0]$ if θ is a literal,

– $mkspw(\wedge_{i=1}^{n} \theta_i) \doteq 1 + \bigvee_{i=1}^{n} mkspw(\theta_i)$,

– $mkspw(\vee_{i=1}^{n} \theta_i) \doteq 1 + \bigwedge_{i=1}^{n} mkspw(\theta_i)$,

– $mkspw((\exists y)\theta(\bar{x}, y)) \doteq 1 + mkspw(\theta)$, and

– $mkspw((\forall y)\theta(\bar{x}, y)) \doteq \|\mathcal{A}\| + mkspw(\theta)$,

where $\mathcal{A}$ is the structure associated to the problem, and the operations between windows and scalars are $[a, b] \vee [a', b'] \doteq [\max\{a, a'\}, \max\{b, b'\}]$, $[a, b] \wedge [a', b'] \doteq [\min\{a, a'\}, \max\{b, b'\}]$ and $c + [a, b] \doteq [c + a, c + b]$. SAT, for example, has the window $[\|\mathcal{A}\| + 5, \|\mathcal{A}\| + 6]$ which means that the CNF encoded by a structure $\mathcal{A}$ is satisfiable iff there is a parallel plan of makespan ℓ such that $\|\mathcal{A}\| + 5 \leq \ell \leq \|\mathcal{A}\| + 6$.

By bounding the upper limit of parallel horizon windows, we obtain the following.

Theorem 4. *Consider a signature σ, $\Phi \in SO\exists(\sigma)$ and $\mathcal{A} \in STRUC[\sigma]$. Then, to decide $\mathcal{A} \vDash \Phi$, it is enough to consider parallel plans of makespan linear on $\|\mathcal{A}\|$ for fixed Φ but* independently *of the arities in σ and Φ. More precisely, it is enough to consider plans of makespan $\max_b q_b(\|\mathcal{A}\| - 1) + h_b + 3$ where q_b is the number of universal quantifiers along branch b in the parse tree of ψ, h_b is the height of branch b, and ψ is the FOL part of Φ.*

Proof. Let $n = \|\mathcal{A}\|$ and T the parse tree for ψ. For a maximal branch $b \in T$, let q_b be the number of universal quantifiers in b, h_b its height, and $u(b)$ the upper limit of the parallel horizon window along b. The upper limit $u(\psi)$ of $mkspw(\psi)$ is $\max_{b \in T} u(b)$, and $u(b) = q_b n + h_b - q_b = q_b(n-1) + h_b$. End by adding 3 to the upper limit $u(\psi)$. $\square$

This bound is tight for SAT. This result is surprising because one would expect the need to consider parallel plans of makespan $\mathcal{O}(\|\mathcal{A}\|^k)$ for some k. However, note that a linear makespan does not mean a linear number of operators.

Finally, observe that by composing the translation $f_{\sigma,\Phi}$ with any translation from STRIPS to SAT, and using the upper bound of the window, one obtains a polytime reduction from the problem expressed by Φ into SAT.

Experiments

We performed experiments on the NP-complete problems SAT, CLIQUE, DIRECTEDHAMILTONIANPATH, 3-DIMENSIONALMATCHING, 3-COLORABILITY and k-COLORABILITY. Also, we computed the chromatic number of random graphs using the tool as an oracle.

The instances for SAT were taken from the SATLIB repository,[4] the instances for graph problems were randomly generated according to the $G(n, p)$ model (Bollobás 2001), and the instances for matching were generated by randomly choosing triplets from $\{0, \ldots, n-1\}^3$ with probability p.

The experiments were performed on an Intel Xeon processor running at 1.86 GHz with 2 GB of RAM. The planners were run for 30 minutes with a limit of 1 GB of memory. The planners that solved more instances are M and Mp that support lower and upper bounds for time horizons (Rintanen 2010b; 2010a), and a num2sat (Hoffmann et al. 2007) modified to handle upper bounds on time horizons. Among these, M was the one that solved more instances.

Figure 3 shows two examples with solutions: the left panel shows a random digraph of 15 vertices that has a directed Hamiltonian path, and the right shows a random graph of 15 vertices with a 6-clique. These structures were discovered by M on the problems obtained from the sentences and the structures encoding the graphs.

Table 2 shows a summary of results for M. In total, we ran the planner on 1,920 instances from which 1,614 were solved: 706 on the positive side meaning that the input structure satisfies the property, and the rest 908 instances on the negative side. The problems of type uf20, uf50 and uf75 are random satisfiable 3CNF instances from the phase transition region with 20, 50 and 75 propositional variables respectively, while the problems uuf50 and uuf75 are random unsatisfiable instances. The instances of type n-k in CLIQUE refer to graphs with n vertices on which to look for cliques of size k, those n-k in k-COLORABILITY refer to graphs with n vertices on which to test k-colorability, and those of type n in other problems refer to graphs with n vertices.

The table contains information about the total number of instances of each type (N), the number of instances solved by M (N^*), the number of instances solved in the positive

[4]http://www.satlib.org

instance	χ	k-colorability						
		1	2	3	4	5	6	7
10-0.75-1	5	2	2	6	101	**3**		
10-0.75-2	5	1	2	2	6	**4**		
10-0.85	7	2	2	3	6	4	1,265	**4**
15-0.25	2	27	**62**					
15-0.60	5	27	29	54	118	**72**		
15-0.70	6	28	28	33	47	329	**67**	
20-0.10	3	214	350	**705**				
20-0.25	4	211	272	1,261	**837**			

Table 1: Results for M on the computation of chromatic numbers on random graphs. For each instance, the table shows the chromatic number χ, and the time (in seconds) to prove/disprove k-colorability for increasing values of k. The last value for k, for each instance, is the chromatic number.

and negative, and the average time that M took per instance. As it can be seen, we tried to generate a balanced set of problems with positive and negative instances. Overall, we think that M behaves very good as it solves 84.06% of the benchmark which is made of NP-complete problems of varying size and difficulty.

Chromatic Numbers

The chromatic number of a graph $G = (V, E)$ is the least k such that G is k-colorable. It is NP-hard to compute the chromatic number of a graph, but we can do it by testing for k-colorability for increasing values of $k = 1, \ldots, |V|$.[5] Table 1 shows results for the computation of chromatic numbers on random graphs. For each instance, it shows the chromatic number χ and the time to prove/disprove the existence of a k-coloring for increasing values of k.

Discussion

We presented a "black box" that given as input a signature σ, a second-order existential sentence Φ and a structure $\mathcal{A} \in \text{STRUC}[\sigma]$, outputs a STRIPS problem P that is solvable in non-deterministic polynomial time and has a plan iff $\mathcal{A} \models \Phi$. The black box is fully automated and runs in polynomial time in the size $\|\mathcal{A}\|$, and thus can be thought of as an efficient method to generate polytime reductions from NP into STRIPS.

The choice of SO∃ as the input language is arbitrary. However, SO∃ is a widely accepted formalism for expressing problems because it is *declarative* and not *tied* to any particular problem. In theory, one could choose any NP-Complete problem, such as SAT or Hamiltonian Path, as the input language for representing NP problems. This would make the translation much easier, but then the user would have to express his problems as instances of them, rendering the approach uninteresting.

We have not compared our approach with direct translations of problems into SAT, tools such as NP-SPEC from other areas, or specialized solvers. We expect to perform some of these comparisons in the near future. Specially,

[5]One can do better by performing a binary search on k.

with tools that generate SAT instances from problem descriptions, because our tool combined with M can be thought as a tool that reduces SO∃ to SAT.

More ambitiously, we would like to consider complexity classes beyond NP by exploiting known results in DCT. For example, PSPACE equals SO(TC) and thus any SO(TC) formula can be mapped into STRIPS. Unfortunately, the reduction is not as easy as the one for NP. The general reductions that we know consists of going from the formula to a polyspace TM that decides the validity of the formula in the input structure and then to simulate the TM with STRIPS. Instead, we would like more meaningful and practical reductions.

References

Bollobás, B. 2001. *Random Graphs*. Cambridge University Press, second edition.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69:165–204.

Cadoli, M., and Schaerf, A. 2005. Compiling problem specifications into SAT. *Artificial Intelligence* 162:89–120.

Fagin, R. 1974. Generalized first-order spectra and polynomial-time recognizable sets. *American Mathematical Society* 7:27–41.

Hoffmann, J.; Gomes, C.; Selman, B.; and Kautz, H. A. 2007. SAT encodings of state-space reachability problems in numeric domains. *Proc. 20th Int. Conf. on Automated Planning and Scheduling*, 1918–1923.

Immerman, N. 1998. *Descriptive Complexity*. Springer.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.

Mitchell, D. G., and Ternovska, E. 2005. A framework for representing and solving NP search problems. *Proc. 20th National Conf. on Artificial Intelligence*, 430–435.

Rintanen, J. 2010a. Heuristic planning with SAT: Beyond strict depth-first search. *Proc. 23rd Australasian Joint Conf. on Artificial Intelligence*, 415–424.

Rintanen, J. 2010b. Heuristics for planning with SAT. *Proc. 16th Int. Conf. on Principles and Practice of Contraint Programming*, 414–428.

Sipser, M. 2005. *Introduction to Theory of Computation, 2nd Edition*. Boston, MA: Thomson Course Technology.

Tseitin, G. S. 1968. On the complexity of derivation in propositional calculus. In Slisenko, A. O., ed., *Studies in Constructive Mathematics and Mathematical Logic, Part 2*. Springer. 115–125.

	N^*/N	#pos	#neg	avg. time
SAT: $mkspw = [n+5, n+6]$				
uf20	40/40	40	0	1.7
uf50	40/40	40	0	146.7
uf75	15/40	15	0	362.1
uuf50	40/40	0	40	548.5
uuf75	1/40	0	1	1,746.4
CLIQUE: $mkspw = [2n+4, 3n+7]$				
10-3	40/40	22	18	1.2
10-4	40/40	12	28	2.2
10-5	40/40	1	39	32.3
15-3	40/40	22	18	10.5
15-4	40/40	11	29	36.6
15-5	39/40	4	35	74.3
15-6	37/40	1	36	79.4
20-3	40/40	25	15	40.2
20-4	40/40	17	23	72.6
20-5	39/40	10	29	159.6
20-6	34/40	4	30	185.2
25-3	40/40	30	10	111.9
25-4	40/40	18	22	231.0
25-5	39/40	10	29	387.5
25-6	36/40	8	28	394.1
DIRECTEDHAMILTONIANPATH: $mkspw = [n+3, n+10]$				
10	40/40	15	25	1.1
15	39/40	18	21	63.7
20	31/40	20	11	70.0
25	29/40	20	9	202.1
30	22/40	20	2	629.1
3-DIMENSIONALMATCHING: $mkspw = [3n+4, 3n+6]$				
10	40/40	36	4	9.6
15	40/40	40	0	251.5
20	13/40	13	0	1,191.0
25	0/40	0	0	—
3-COLORABILITY: $mkspw = [2n+4, 2n+7]$				
10	40/40	18	22	0.1
15	40/40	24	16	0.9
20	40/40	12	28	3.0
25	40/40	30	10	8.9
30	40/40	9	31	20.9
40	40/40	4	36	75.1
50	40/40	1	39	196.7
k-COLORABILITY: $mkspw = [2n+4, 3n+6]$				
10-2	40/40	9	31	1.9
10-3	40/40	18	22	2.8
10-4	40/40	27	13	11.0
15-2	40/40	7	33	33.5
15-3	40/40	16	24	46.5
15-4	40/40	24	16	91.7
20-2	40/40	3	37	254.9
20-3	40/40	12	28	395.9
20-4	40/40	20	20	497.3
25-2	0/40	0	0	—
25-3	0/40	0	0	—
25-4	0/40	0	0	—
Total	1,614/1,920	706	908	180.9

Table 2: Results for M. For each problem type, the table shows number of solved instances (N^*), total number of instances (N), number of solved instances that satisfy the property (#pos), number of solved instances that do not satisfy the property (#neg), and the average time in seconds.