DLPlan: Description Logics State Features for Planning

Dominik Drexler, Jendrik Seipp

Linköping University, Linköping, Sweden
{dominik.drexler, jendrik.seipp}@liu.se

Abstract

Description logics are a family of knowledge representation languages that has become increasingly popular in planning. The main purpose of description logics in planning is to define abstract state features in a domain-general way for allowing to learn human-interpretable classifiers or regression models. After a brief review of the literature that uses description logics features in planning, we present DLPlan, a C++ and Python library for constructing and evaluating state features for planning based on description logics.

Introduction

Knowledge derived from a planning instance or a planning domain can be used to substantially speed up the solving process or even make instances or domains solvable without any search. We distinguish two types of knowledge: instance-specific and domain-general knowledge. Instance-specific knowledge is meaningful for only the single instance for which it was learned, and domain-general knowledge is meaningful for all instances from a domain.

In this paper, we use description logics (Baader et al. 2003) to define domain-general knowledge (Jiménez, Segovia-Aguas, and Jonsson 2019). Description logics were used in planning to learn generalized policies (Martín and Geffner 2000, 2004; Fern, Yoon, and Givan 2004; Yoon, Fern, and Givan 2008; Francès, Bonet, and Geffner 2021; Ståhlberg, Bonet, and Geffner 2022a,b), abstract actions (Bonet and Geffner 2018), unsolvability heuristics (Ståhlberg, Francès, and Seipp 2021), progress states for greedy best-first search (Ferber et al. 2022), generalized heuristics (Francès et al. 2019; Yoon, Fern, and Givan 2008; de Graaff, Corrêa, and Pommerening 2021), and policy sketches (Drexler, Seipp, and Geffner 2022). These works use description logics to define abstract domain-general state features. To provide a solid, unified code base for future research in this direction, we present a C++ and Python library for defining and evaluating state features based on description logics.

Background

We describe *planning*, *state spaces*, and *description logics* drawing on the notation from Drexler, Seipp, and Geffner (2022) and Baader et al. (2003).

Planning

A *planning problem* or *instance*, is a pair $P = \langle D, I \rangle$ where D is a first-order domain consisting of a set of predicates F and a set of action schemas over F, and I is instance-specific information consisting of a set of objects O, a description of the initial situation $Init$ and the goal $Goal$. Instantiating the predicates in F with objects from O induces the set of atoms A. A state s is a set of atoms, and we say that an atom a is true in s iff $a \in s$. The action schemas and atoms A induce a set of ground actions.

Each planning problem P induces a state space $S(P) = \langle S, s_0, G, L, T \rangle$, where S is the set of all states over atoms A, $s_0 \in S$ is the initial state describing $Init$, G is the set of goal states where $Goal$ is true, L is the set of labels for ground actions, and $T \subseteq S \times L \times S$ is a set of transitions, i.e., $\langle s, l, s' \rangle \in T$ iff applying the action with label l in s results in state s'.

Description Logics

Description logics (Baader et al. 2003) are a family of languages to represent knowledge. This paper uses description logics to represent knowledge derived from states in planning problems. In a description logic, there is a set of *objects* Δ, a set of *concepts*, and a set of *roles*.

The concepts and roles are expressions describing unary and binary relations over Δ, respectively. Both types of expressions are built recursively, starting from a small set of *primitive* concepts and roles, giving rise to *composite* concepts and roles. Most description logics are more expressive than propositional logic but less general than two-variable first-order logic.

The DLPlan Library

The DLPlan library (Drexler, Francès, and Seipp 2022) implements the basic types for predicates, objects, atoms, states, and state spaces, as well as commonly used description logics concepts and roles, and additional Boolean and numerical features. A Boolean feature maps a state into the

Boolean domain, and a numerical feature maps a state into a natural number $\mathbb{N}_0$. From now on, we refer to concepts, roles, Booleans and numericals features simply as *features*. The interpretation of a feature depends on a given state s under the following semantics. Consider nullary predicate p_0, unary predicate p_1, binary predicate p_2, concepts C, D, and roles R, S, and either concept or role X.

Concepts

- *Universe* $\triangle$ with $\triangle^s = O$,
- *Primitive* p_1 with $(p_1)^s = \{a \in \Delta \mid p_1(a) \in s\}$,
- *Top* $\top$ and *bottom* $\bot$ with $\top^s = \Delta^s$, $\bot^s = \emptyset$,
- *Intersection* $C \sqcap D$, *union* $C \sqcup D$, and *negation* $\neg C$ with $(C \sqcap D)^s = C^s \cap D^s$, $(C \sqcup D)^s = C^s \cup D^s$, and $(\neg C)^s = \Delta \setminus C^s$,
- *Value restriction* $\forall R.C$ with $(\forall R.C)^s = \{a \mid \forall b : (a,b) \in R^s \to b \in C^s\}$,
- *Existential quantification* $\exists R.C$ with $(\exists R.C)^s = \{a \mid \exists b : (a,b) \in R^s \land b \in C^s\}$,
- *Role-value-map* $R \subseteq S$ and $R = S$ with $(R \subseteq S)^s = \{a \mid \forall b : (a,b) \in R^s \to (a,b) \in S^s\}$, $(R = S)^s = \{a \mid \forall b : (a,b) \in R^s \leftrightarrow (a,b) \in S^s\}$,
- *Nominal* a with $a^s = \{a\}$.

Roles

- *Primitive* p_2 with $(p_2)^s = \{(a,b) \mid p_2(a,b) \in s\}$,
- *Top* $\top$ with $\top^s = \Delta^s \times \Delta^s$,
- *Intersection* $R \sqcap S$, *union* $R \sqcup S$, and *negation* $\neg R$ with $(R \sqcap S)^s = R^s \cap S^s$, $(R \sqcup S)^s = R^s \cup S^s$, $(\neg R)^s = \top^s \setminus R^s$,
- *Inverse* R^{-1} with $(R^{-1})^s = \{(b,a) \mid (a,b) \in R^s\}$,
- *Composition* $R \circ S$ with $(R \circ S)^s = \{(a,c) \mid (a,b) \in R^s \land (b,c) \in S^s\}$,
- *Identity* $id(C)$ with $(id(C))^s = \{(a,a) \mid a \in C^s\}$,
- *Transitive (reflexive) closure* R^+, R^* with $(R^+)^s = \bigcup_{n \geq 1}(R^s)^n$, $(R^*)^s = \bigcup_{n \geq 0}(R^s)^n$, $(R^s)^0 = (id(\triangle))^s$, $(R^s)^{n+1} = (R^s)^n \circ R^s$,
- *Restrict* $R|_C$ with $(R|_C)^s = R^s \sqcap (\Delta \times C^s)$.

Example 1. *In the Delivery planning domain, there is a set of packages distributed over a fully-connected grid. The objective is to move all packages one-by-one to a single target location. Consider binary predicates* at *and* at_g *where* $at(p,l)$ *is true iff package* p *is at location* l*, and* $at\_g(p,l)$ *is true iff package* p *has goal location* l*. The concept* f *describing the set of undelivered packages can be defined as* $f \equiv (\neg(at = at\_g)) \sqcap package$.

Booleans

- $Empty(X)$ with $Empty(X)^s$ is true iff $|X^s| = 0$,
- $Nullary(p_0)$ with $Nullary(p_0)^s$ is true iff $p_0() \in s$.

Numericals

- $Count(X)$ with $Count(X)^s \equiv |X^s|$,
- $Distance(C,R,D)$ with $Distance(C,R,D)^s$ is the smallest number $n \in \mathbb{N}_0$ such that there are objects $c_1, \ldots, c_n$ with $c_1 \in C^s$, $c_n \in D^s$, and $(c_i, c_{i+1}) \in R^s$ for $i = 1, \ldots, n-1$. If no n exists then the result is ∞.

Example 2. *Consider Example 1. The numerical* f' *desribing the number of undelivered packages can be defined as* $f' \equiv Count((\neg(at = at\_g)) \sqcap package)$.

The DLPlan library provides functionality for parsing features from plain text and manually building them incrementally using the feature constructors. The features are uniquely stored in a forest, in the sense that there are never two syntactically identical features. The uniqueness allows for caching feature evaluations for each state to drastically speeds up the evaluation when (sub-)features are reused.

Use Cases

We now describe the three main use cases of the library.[1,2]

Generate First-order State Spaces

DLPlan can generate and construct the full state spaces from a set of input files writting in the planning domain definition language (PDDL).

```
state_space = generate_state_space(
    "domain.pddl", "problem.pddl")
```

Evaluate Description Logics Features

DLPlan can construct, parse, and efficiently evaluate domain-general state features based on description logics.

```
factory = SyntacticElementFactory()
numerical = factory.parse_numerical("n_count
    (r_and(r_primitive(at,0,1),r_not(
    r_primitive(at_g,0,1))))")
value = numerical.evaluate(State(...))
```

Generate Description Logics Features

For a given set of instances from a planning domain, DLPlan can generate a pool $\mathcal{F}$ of interesting domain-general features. To do so, DLPlan generates the state space and automatically derives $\mathcal{F}$ consisting of features with complexity up to k. The complexity of an feature is the number of applied grammar rules. DLPlan constructs features incrementally by generating all possible syntactic compositions. To reduce the exponential blowup, it uses a collection of states to prune redundant features. A feature f is redundant if a previously generated feature f' has the same feature valuation as f for all the given states.

```
feature_strings = generate_features(factory,
     state_space.get_states().values(), k,
    ...)
```

Conclusions

We presented a brief overview of use cases for description logics state features in planning and showed how to address them with our simple and powerful DLPLan library.

[1] Link to demonstration video: tinyurl.com/2p96j4hj

[2] Link to Colab notebook: https://tinyurl.com/2p8x3kdv

References

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Bonet, B.; and Geffner, H. 2018. Features, Projections, and Representation Change for Generalized Planning. In Lang, J., ed., *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, 4667–4673. IJCAI.

de Graaff, R.; Corrêa, A. B.; and Pommerening, F. 2021. Concept Languages as Expert Input for Generalized Planning: Preliminary Results. In *ICAPS 2021 Workshop on Knowledge Engineering for Planning and Scheduling*.

Drexler, D.; Francès, G.; and Seipp, J. 2022. Description Logics State Features for Planning (DLPlan). https://doi.org/10.5281/zenodo.5826139.

Drexler, D.; Seipp, J.; and Geffner, H. 2022. Learning Sketches for Decomposing Planning Problems into Subproblems of Bounded Width. In (Thiébaux and Yeoh 2022).

Ferber, P.; Cohen, L.; Seipp, J.; and Keller, T. 2022. Learning and Exploiting Progress States in Greedy Best-First Search. In De Raedt, L., ed., *Proceedings of the 31th International Joint Conference on Artificial Intelligence (IJCAI 2022)*. IJCAI.

Fern, A.; Yoon, S. W.; and Givan, R. 2004. Learning Domain-Specific Control Knowledge from Random Walks. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 191–198. AAAI Press.

Francès, G.; Bonet, B.; and Geffner, H. 2021. Learning General Planning Policies from Small Examples Without Supervision. In Leyton-Brown, K.; and Mausam, eds., *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021)*, 11801–11808. AAAI Press.

Francès, G.; Corrêa, A. B.; Geissmann, C.; and Pommerening, F. 2019. Generalized Potential Heuristics for Classical Planning. In Kraus, S., ed., *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, 5554–5561. IJCAI.

Jiménez, S.; Segovia-Aguas, J.; and Jonsson, A. 2019. A review of generalized planning. *The Knowledge Engineering Review*, 34: e5.

Martín, M.; and Geffner, H. 2000. Learning Generalized Policies from Planning Examples Using Concept Languages. In Cohn, A. G.; Giunchiglia, F.; and Selman, B., eds., *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR 2000)*, 667–677. Morgan Kaufmann.

Martín, M.; and Geffner, H. 2004. Learning Generalized Policies from Planning Examples Using Concept Languages. *Applied Intelligence*, 20(1): 9–19.

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022a. Learning General Optimal Policies with Graph Neural Networks: Expressive Power, Transparency, and Limits. In (Thiébaux and Yeoh 2022).

Ståhlberg, S.; Bonet, B.; and Geffner, H. 2022b. Learning Generalized Policies without Supervision Using GNNs. In *Proceedings of the Nineteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2022)*.

Ståhlberg, S.; Francès, G.; and Seipp, J. 2021. Learning Generalized Unsolvability Heuristics for Classical Planning. In Zhou, Z.-H., ed., *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021)*, 4175–4181. IJCAI.

Thiébaux, S.; and Yeoh, W., eds. 2022. *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*. AAAI Press.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning Control Knowledge for Forward Search Planning. *Journal of Machine Learning Research*, 9: 683–718.