Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

Automated Planning for Production Routines in Semiconductor Manufacturing

Gianluca Zavan[1], **Konstantin Schekotihin**[1], **Thomas Langreiter**[2]

[1]Alpen Adria Universität Klagenfurt,
[2]Infineon Technologies AG,
{gianluca.zavan, konstantin.schekotihin}@aau.at, thomas.langreiter@infineon.com

Abstract

Semiconductor device manufacturing involves complex production routines comprising hundreds of sophisticated steps, e.g., Lithography, Epitaxy, and Ion Implantation. Formulating these routines is challenging, as it requires a deep understanding of the technological interdependencies between operations and the ability to anticipate the consequences of long operation sequences. Currently, this process remains largely manual, with minimal tool support. This paper presents a knowledge-driven architecture for automated manufacturing routine composition, integrating expert knowledge acquisition through ontologies and Knowledge Graphs. The system enables the automatic generation of classical and hierarchical planning domain specifications, which are then used by automated planners for routine construction. Domain experts utilised the developed prototype to effectively model real-world semiconductor manufacturing use cases and compute corresponding routines, thereby demonstrating the practical applicability of the proposed approach.

Introduction

Semiconductor manufacturing is a highly complex process comprising numerous specialised production tasks—*Unit Processes* (UPs)—typically categorised into front-end and back-end phases. The former focuses on fabricating semiconductor devices directly on silicon wafers, involving the creation of individual device components and their interconnections. Whereas the back-end encompasses subsequent tasks dedicated to packaging the fabricated devices into integrated circuits suitable for shipment to customers.

Manufacturing a semiconductor device requires executing hundreds of UPs in a carefully orchestrated sequence—*Processflow*—where each UP has complex interactions with others and significantly impacts the final outcome. Moreover, most manufacturing objectives, like placing a metallization layer or applying a photoresist, can be achieved through multiple alternative sequences of UPs—*Process Building Blocks* (PBBs). Alternative PBBs having similar prerequisites and effects are further organized into abstract *Process Building Block Families* (PBBFs). Selecting the most suitable PBBF or PPB demands deep expertise in understanding the manufacturing goals, the prerequisites and effects of each sequence, and their interdependencies. Consequently, defining new Processflows is an extremely challenging and time-consuming task, often requiring weeks or months of expert effort. Moreover, Processflows can further be composed into *Processflow chains* which span over front-end and back-end manufacturing.

Example 1. Lithography *is a PBBF where light is used to transfer a pattern onto photoresist material deposited on the wafer. Key parameters characterise this macro-step, such as the* photomask*—an opaque plate with transparent regions that control light exposure on the wafer. A PBB for Lithography specifies the exact photomask to be used, whereas a UP sequence specifies low-level details such as photomask positioning, wafer orientation, and other operational parameters necessary for task execution.*

Given the structure of the domain, *hierarchical planning* is a natural choice to model the PBBFs, PBBs and UPs to enable the automatic construction of new Processflows and the evaluation of existing ones. However, applying planning in this context is not trivial for multiple reasons: (1) There is no formal model that can be exploited to systematically construct new Processflows based on the domain experts' goals. (2) The available knowledge of the Processflow steps is very sparse and not structured. (3) There is a lack of specialised tools for Processflow construction integrated into the experts' workflow.

In this paper, we address these issues with the following contributions: (1) A formalisation of the Processflow construction task as a planning problem, using both classical and hierarchical representations; (2) an ontological representation of the semiconductor manufacturing domain, along with a data pipeline to systematically build a Knowledge Graph (KG) from available data sources; (3) a prototype web application for the collection of planning knowledge and the interactive construction of Processflows using automated planners; and (4) a proof-of-concept implementation of the proposed approach, which was successfully used by domain experts to model and solve real-world use cases from semiconductor manufacturing.

Background

Let F be a finite set of ground atoms (i.e., propositional symbols). Following (Ghallab, Nau, and Traverso 2004), a *plan-*

ning domain is defined as $D = (S, A, \gamma)$, where $S \subseteq 2^F$ is the set of *ground states*; A is the set of *ground actions*, each of the form $(pre_a^+, pre_a^-, eff_a^+, eff_a^-)$, specifying positive and negative preconditions and effects respectively; and γ is a state *transition function*. An action $a \in A$ is *applicable* in state $s \in S$ if its preconditions are satisfied: $pre_a^+ \subseteq s$ and $s \cap pre_a^- = \emptyset$. The transition function $\gamma(s, a)$ computes the successor state $(s \cup eff_a^+) \setminus eff_a^-$ if a is applicable in s; otherwise it is undefined. Moreover, γ can be overloaded to take sequences of actions $\mathbf{a} = \langle a_1, a_2, \ldots, a_n \rangle$ as $\gamma(s, \mathbf{a}) = \gamma(\ldots \gamma(\gamma(s, a_1), a_2) \ldots, a_n)$, being undefined if the execution of an action in the sequence is not possible. Furthermore, given a sequence of actions $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$, we denote with $\mathbf{a}_{i,j}$ the subsequence $\langle a_i, \ldots, a_j \rangle$. A *planning problem description* is denoted as $\mathcal{P} = (D, s_0, G)$, where D is a domain description, $s_0 \in S$ is an initial state, and $G \subseteq S$ is a set of ground *goal states*. The planning problem is to find $\mathbf{a} \in F^*$ such that $\gamma(s_0, \mathbf{a}) \in G$.

Different from classical planning, hierarchical planning allows the distinction of *tasks*, *methods* and *primitive actions*. Each task can be realised by one or more methods, and each method encloses a sequence of tasks and primitive actions. Primitive actions differ from tasks in that they do not have any methods that refine them. Following the notation of (Marthi, Russell, and Wolfe 2007), a *hierarchical planning problem* is given by a planning domain D and an action hierarchy $\mathcal{H} = (\mathcal{A}, I, \mathcal{T})$. $\mathcal{A}$ is the set of tasks, i.e., *high-level actions*. For each $a \in \mathcal{A}$ there is a set $I(a)$ of methods, i.e., *immediate refinements*, each of which is a sequence $\mathbf{a} \in (A \cup \mathcal{A})^*$. Finally, $\mathcal{T} \subseteq (A \cup \mathcal{A})$ is the set of tasks and actions allowed to be used to compose high-level plans. A sequence $\mathbf{b}$ is a refinement of a sequence $\mathbf{a}$ iff $\mathbf{b}$ can be obtained from $\mathbf{a}$ by repeated replacement of $a_i \in \mathbf{a}$ by one of its immediate refinements $I(a_i)$. A refinement $\mathbf{b}$ of $\mathbf{a}$ is *primitive* iff $\mathbf{b}$ comprises only primitive actions $a_j \in A$. Moreover, $R^*(\mathbf{a})$ is defined as the set of all primitive refinements of the action sequence $\mathbf{a}$. Given an initial state s_0 and a goal G, a (partially or totally) ordered sequence of actions $\mathbf{t} \in \mathcal{T}^*$ is a *plan* or *solution* for G if there exists a primitive refinement $\mathbf{r} \in R^*(\mathbf{t})$ such that $\gamma(s_0, \mathbf{r}) \supseteq G$. Then, $\mathbf{t}$ is called a *high-level solution* while $\mathbf{r}$ is a *low-level solution*.

In the rest of the section, some definitions which will be used later in the paper are introduced.

Definition 1 (Upward-solution property (Tenenberg 1988, p. 64))**.** *A low-level solution to a problem is also a high-level solution to the problem when stated at that level.*

Definition 2 (Precondition Relaxation (Olz, Biundo, and Bercher 2021, Def. 8))**.** *Let $D = (S, A, \gamma)$ be a planning domain. Its* precondition-relaxation *is the domain $D^\emptyset = (S, A^\emptyset, \gamma)$ with $A^\emptyset = \{(\emptyset, \emptyset, eff_a^+, eff_a^-) \mid a = (pre_a^+, pre_a^-, eff_a^+, eff_a^-) \in A\}$.*

Given an action $a \in A$, let $a^\emptyset \in A^\emptyset$ be its counterpart in the precondition-relaxed domain $D^\emptyset$. Similarly, given an action sequence $\mathbf{a} = \langle a_1, \ldots, a_n \rangle$, its precondition-relaxed counterpart is denoted with $\mathbf{a}^\emptyset$.

To avoid introducing new notation, we reformulate the concepts of *precondition-relaxed effects* and *executability-relaxed preconditions* given in (Olz et al. 2025) as follows.

Definition 3 (Precondition-Relaxed Effects (Olz et al. 2025, Def. 5))**.** *Let $a \in \mathcal{A}$ be a task and $R^*(a)$ the set of all its primitive refinements. The sets of* possible *and* guaranteed precondition-relaxed effects *of a are given by:*

$$peff_a^{\emptyset+} := \bigcup_{\mathbf{a} \in R^*(a)} \gamma(\emptyset, \mathbf{a}^\emptyset)$$

$$peff_a^{\emptyset-} := \bigcup_{\mathbf{a} \in R^*(a)} F \setminus \gamma(F, \mathbf{a}^\emptyset)$$

$$eff_a^{\emptyset+} := \bigcap_{\mathbf{a} \in R^*(a)} \gamma(\emptyset, \mathbf{a}^\emptyset)$$

$$eff_a^{\emptyset-} := \bigcap_{\mathbf{a} \in R^*(a)} F \setminus \gamma(F, \mathbf{a}^\emptyset)$$

That is, the cumulative effects that result from the execution of the actions starting from the empty state, ignoring if the actions can be actually executed or not.

Definition 4 (Executability-Relaxed Precondition (Olz et al. 2025, Def. 6))**.** *Let $a \in \mathcal{A}$ be a task and $R^*(a)$ the set of all its primitive refinements. The sets of the* executability-relaxed preconditions *of a are defined follows:*

$$pre_a^{\emptyset+} := \bigcap_{\mathbf{a} \in R^*(a)} \bigcup_{i \in [1, |\mathbf{a}|]} pre_{a_i}^+ \setminus \gamma(\emptyset, \mathbf{a}_{1,i-1}^\emptyset)$$

$$pre_a^{\emptyset-} := \bigcap_{\mathbf{a} \in R^*(a)} \bigcup_{i \in [1, |\mathbf{a}|]} pre_{a_i}^- \setminus (F \setminus \gamma(F, \mathbf{a}_{1,i-1}^\emptyset))$$

That is, for all primitive refinements, we choose all the atoms that appear as a precondition of an action but are not satisfied by another action before it.

The usual definition of hierarchical domains and problems does not consider the inclusion of preconditions and effects in the task descriptions. If instead such a description is allowed, care is needed in determining its semantics. In the literature, many *legality criteria* to determine correct task descriptions have been proposed, and to the authors' knowledge, there is no current standard (Bercher et al. 2016). Here we present the criterion formulated in (Yang 1990) because it fits best the domain of interest.

Definition 5 (Legal method (Yang 1990, p. 14))**.** *Given two actions $a, b \in \mathcal{A} \cup A$, where A is the set of primitive actions and $\mathcal{A}$ the set of tasks, $a \prec b$ means that a is constrained to occur before b in a plan, and $a \prec_{im} b$ means that a is immediately before b. Given a task $a \in \mathcal{A}$, each method $M = \langle a_1, \ldots, a_n \rangle$ in $I(a)$ is* legal *if the following holds:*

1. *$\forall e \in eff_a^+$ (resp. eff_a^-), there exists an action $a_i \in M$ such that $e \in eff_{a_i}^+$ (resp. $eff_{a_i}^-$) and $\forall b \in M$, if $e \in eff_b^-$ (resp. eff_b^+) then $b \prec a_i$. That is, every effect of a task is asserted by at least one of the actions in the method M.*
2. *$\forall e \in pre_a^+$ (resp. pre_a^-), there exists an action $a_i \in M$ such that $e \in pre_{a_i}^+$ (resp. $pre_{a_i}^-$) and $\forall b \in M$, if $t \in eff_b^+$ (resp. pre_b^-) then $b \prec a_i$. That is, every precondition of a task is asserted by at least one of the actions in*

the method, and further, it is not the effect of some earlier subaction.

3. *$\forall a_i \in M$, $\forall p \in pre^+_{a_i}$ (resp. $pre^-_{a_i}$), $\forall b \in M$ such that $a_i \not\prec b$, if $p \in eff^-_b$ (resp. eff^+_b) then there exists $c \in M$ such that $c \prec a_i$ and $b \prec c$. This means that the sequence of actions in M must be free of conflicts.*

Planning Problem Definition

As already mentioned, domain experts first reason at an abstract level about the Processflow outcomes and then refine their ideas incrementally. This translates to first defining a PBBF sequence and then identifying the suitable PBBs and UPs that can realise it.

We can draw a one-to-one mapping between PBBFs, PBBs and UPs and hierarchical planning concepts. In particular, PBBFs are tasks, PBBs are methods, and UPs are primitive actions. For convenience, in the following we will use PBBF (resp. PBB, UP) and task (resp. method, primitive action) interchangeably.

On one side, at the PBBF level, we are only interested in modeling the *kind* of the relevant manufacturing parameters for the step, not exactly *which* parameters are used. This can be modelled by using lifted planning actions for PBBFs. On the other side, PBBs and UPs are executed using specific tools and resources, which can be represented as constant symbols. This means that PBBFs are lifted representations, while PBBs and UPs are grounded.

Furthermore, we can make the following assumptions:

Assumptions. *(1) All the action sequences are totally ordered, both at the PBBF and UP levels. (2) A PBBF is only refined by UPs, and refining a UP any further is impossible, so the action hierarchy is not recursive (also,* acyclic*). (3) Any PBBF has at least one UP sequence that refines it. Hence, there are no tasks without a primitive refinement. (4) The hierarchy tree has a finite, constant height.*

To further clarify these assumptions, mind that a PBBF without any UP-level refinement never appears in the domain, because the purpose of a PBBF is to simplify the work of experts by abstracting away alternative sequences of UPs. Therefore, the action hierarchy is acyclic by definition, i.e., it is a tree of constant height.

Since a PBBF abstracts multiple sequences of UPs, it makes sense to impose restrictions on which preconditions and effects can be used to describe them. To make the abstraction useful both from a planning standpoint and as a way to summarise the behaviour of the Processflow steps, we define the concept of *well-defined PBBF*.

Definition 6 (Well-defined PBBF task)**.** *The planning task for a PBBF is* well-defined *if the following conditions hold: (1) The PBBFs descriptions must be* legal *according to Definition 5. (2) The existence of a UP-level plan must imply the existence of a PBBF-level plan that abstracts it. Hence, the upward solution property (Definition 1) must hold. (3) The non-existence of a PBBF level plan must imply the non-existence of a UP level plan.*

This notion is primarely meant to support experts in specifying new PBBFs. A well-defined PBBF is guaranteed to be implementable in the semiconductor fab through corresponding UPs.

Condition (1) gives as a guarantee that there are no preconditions or effects in the action of a PBBF that do not appear in its refinements. If this does not hold, then the high-level description does not represent the low-level UP sequences. As a consequence, the only way to actually understand what a PBBF does when executed is to inspect its refinements. The legality of a PBBF description can be checked automatically at modeling time by checking that Definition 5 holds for all its sequences of UPs.

Condition (2) is crucial for the planning model to be correct. It is expected that a PBBF-level plan represents all possible UP-level plans obtainable by a refinement. Of course, if there exists a UP-level plan that cannot be abstracted, this does not hold, and it is considered a modeling error.

Finally, Condition (3) states that PBBFs must be consistent overapproximations. That is, if there exists a PBBF-level plan to reach a given goal, it represents all valid UP-level plans that reach the same goal, but also spurious ones. If we have the guarantee that a PBBF-level plan doesn't exclude any UP-level plan able to reach the given goal, then, in the case that it is not possible to build a PBBF-level plan, we are sure that no refinement can achieve the goal.

It is difficult to have all three conditions at the same time. By enforcing Condition (1), a PBBF might not be an overapproximation of its refinements, as shown in the following example.

Example 2. *Let $f = (\{p\}, \emptyset, \{q\}, \{p\})$ be the task for a PBBF, with two methods $m_1 = \langle(\{p\}, \emptyset, \{q\}, \{p\})\rangle$ and $m_2 = \langle(\{p\}, \emptyset, \{q, r\}, \{p\})\rangle$. According to Definition 5, m_1 and m_2 are legal, but f is not an overapproximation of both m_1 and m_2, because m_2 can achieve the goal $\{r\}$ while f cannot. If we also add r in the effects of f, then the legality criterion is violated because r is not an effect of m_1.*

Therefore, we require Condition (1) to hold at *modeling time*, while the actions for the PBBFs are described, and Conditions (2) and (3) to hold at *planning time*. At modeling time, we consider the preconditions of a PBBF as *mandatory*, meaning that they must be present in all refinements. Similarly, the effects are considered as *guaranteed*. No refinement of a PBBF can be executed if the mandatory preconditions are not fulfilled, and the state produced by the execution of any refinement always contains the guaranteed effects. This helps domain experts to understand exactly what the key properties of a PBBF are.

At planning time, we enforce Conditions (2) and (3) by adding to the PBBF actions the executability-relaxed preconditions (Definition 4) and the precondition-relaxed effects (Definition 3). The added preconditions reduce the number of states in which the PBBF can be executed, but without excluding any state in which the refinements are executable. To make it so that the precondition-relaxed effects realise the desired overapproximation, we compile away all the negative preconditions by adding an atom np for each atom p in the domain, which are usually never added to a state at the same time by any action (Gazen and Knoblock 1997, Sec. 2.6). Instead, by computing the precondition-

relaxed effects as by Definition 3, we allow the inclusion of all the atoms of the kind p, np to the positive effects of the PBBF, and do not include any negative effect.

Example 3. *Extending Example 2 the two methods* $m_1 = \langle(\{p\}, \emptyset, \{q, np\}, \{p\})\rangle$ *and* $m_2 = \langle(\{p, r\}, \emptyset, \{q, np, nr\}, \{p, r\})\rangle$ *are abstracted by the task* $f = (\{p\}, \emptyset, \{q, np, nr\}, \emptyset)$ *which correctly overapproximates them.*

This way, if there exists a UP sequence u_1 for a PBBF such that when executed it enables the execution of another sequence u_2, the state produced by the execution of the PBBF still makes u_2 executable. By allowing potentially "inconsistent" states in the sense of predicates p, np to be present at the same time, we realise the desired overapproximation. There are, of course, many spurious plans that are introduced by this kind of overapproximation. For the case of interest, this representation is useful to prevent gross mistakes during Processflow construction. Also, it is expected that the effects of the PBBFs seldom overlap.

Due to the fixed structure of the hierarchy, computing the executability-relaxed preconditions and the precondition-relaxed effects for a PBBF requires traversing the hierarchy tree with the PBBF as the root and the UPs as leaves. Let k be the number of UP sequences a PBBF can be implemented with, let l be the number of UPs in a sequence and let m be the number of preconditions and effects of a UP. Then, a naïve algorithm takes $\mathcal{O}(klm)$ steps.

Having the descriptions in place, it is possible to actually perform planning. At first, a classical planner can be used to compute a PBBF-level Processflow. If no plan is found for the given goals, thanks to Condition (3), it is not required to refine the PBBFs, because no UP-level plan exists. If, instead, a PBBF-level plan P is found, a hierarchical planner is employed to refine the PBBFs to get their UP refinements. In practice, the hierarchical problem $\mathcal{H} = (\mathcal{A}, I, \mathcal{T})$ is such that $\mathcal{A}$ is the set of the available PBBFs, I defines the UP sequences in which a PBBF can be refined, and $\mathcal{T} = \{p \mid p \in P\}$. Moreover, we are interested in finding the refinements for the sequence $\mathbf{t} \in \mathcal{T}^*$ s.t. $\mathbf{t} = P$.

Planning with a Predefined Plan Structure

Usually, a new Processflow is not defined from scratch. Instead, it is obtained by iteratively modifying an already available one to adapt it to the new requirements. The domain expert would then add or remove some Processflow steps, while ensuring that the base structure remains the same. This way of building Processflows from known ones doesn't exactly align with the way classical planning tasks are modelled and performed. Using a language like PDDL (Ghallab et al. 1998), modellers only define the actions, the initial state and the goal state of the problem, and the structure of the resulting plan is completely dependent on the strategy of the planner. There are cases, like for Processflow construction, in which the user might want to constrain the structure of the final plan, even if this could result in a non-optimal solution. This could happen if there are some constraints that the user knows, but that are not encoded in the planning domain. In particular, we are interested in giving a domain expert the possibility of specifying the relative order of the PBBFs in the Processflow, and to preserve such order while planning.

Let $\mathcal{P} = (D, s_0, G)$ be a planning problem for a domain $D = (S, A, \gamma)$. Let F be the set of all the ground atoms that define S, and let $P = \langle a_1, \ldots, a_n\rangle$ denote a *plan structure*, a sequence of actions where $a_i \in A$ for all $i \in \{1, \ldots, n\}$. Then, the objective is to define a new problem $\mathcal{P}' = (D', s_0', G')$ such that it must ensure that every plan $P' = \langle a_1', \ldots, a_m'\rangle$, where $a_i' \in A$ for all i and $m \geq n$, that achieves the goal $G' \supseteq G$ is such that all actions in P must appear in P' in the same relative order as in P. Formally, there must exist an increasing sequence of indices $1 \leq i_1 < i_2 < \cdots < i_n \leq m$ such that $a_1 = a_{i_1}'$, $a_2 = a_{i_2}'$, …, $a_n = a_{i_n}'$.

Note that the actions in P are not required to appear consecutively in P', leaving to the planner the possibility to add other actions that are needed to reach the goal.

Drawing the relation with *temporally extended goals* (Bacchus and Kabanza 1996), for each $a_i \in P$ the plan must satisfy the temporal formula $\Diamond a_i \wedge \Box(a_i \rightarrow \bigcirc\Diamond a_{i+1})$, where $\bigcirc$ stands for the LTL operator "next", $\Box$ for "always" and $\Diamond$ for "finally". Since we focus only on this specific kind of temporally extended goal, what follows is its direct representation with a classical planning domain and a problem. Let $\{t_0, \ldots, t_n\}$ be a set of ground atoms with arity 0, intuitively representing the steps in the plan structure P, with an additional "step zero". Let $check$ be an atom representing that the plan structure is being respected. Let $\{c_1, \ldots, c_n\}$ be a set of actions where for all $i \in \{1, \ldots, n\}$, $c_i = (\{check, t_i\}, \emptyset, \emptyset, \{check\})$.

Let $step_i(a) : A \rightarrow A$ be a function defined as $step_i(a) = (pre_a^+ \cup \{t_{i-1}\}, pre_a^- \cup \{check\}, eff_a^+ \cup \{t_i, check\}, eff_a^- \cup \{t_{i-1}\})$. Then, $\mathcal{P}' = (D', s_0', G')$ is such that:

- $D' = (S', A', \gamma)$ where
  - $A' = A \cup \{c_1, \ldots, c_n\} \cup \{step_i(a) \mid a_i \in P\}$
  - $S' \subseteq 2^{F'}$ is the set of states defined by the set of ground atoms $F' = F \cup \{t_0 \ldots t_n\} \cup \{check\}$
- $s_0' = s_0 \cup \{t_0\}$
- $G' = G \cup \{t_n\}$

Lemma 1. *Any plan fulfilling* G' *also fulfils* G *and preserves the plan structure.*

Proof. (Sketch) Since the goal G' is achievable only if t_n is added by an action in the plan and never removed, the only way to reach it is by executing actions obtained with the $step_i$ function. Each of such actions has as *negative* precondition the $check$ atom, which is removed only by the actions in $\{c_1, \ldots, c_n\}$: this means that when a $step_i$ is executed, the only way to enable the action at index $i + 1$ in the structure is to execute c_i, thus enforcing the order of the actions in the final plan. A plan for G' also satisfies G by definition. □

Note that even if the original problem P is solvable, there is no guarantee that P' is solvable as well. A plan that achieves G' can easily be transformed into a plan that

achieves G by simply removing all the c_i actions in it and by substituting each $step_i(a)$ with a.

Application Overview

Having defined the planning problem, we need to take into account what is needed to actually represent PBBFs, PBBs and UPs as planning actions. The major bottleneck in defining planning models is knowledge acquisition, which becomes even more relevant in industrial settings. In many cases, no single person in a company is able to fully describe the domain, and this is also the case for semiconductor manufacturing. Moreover, when different knowledge sources are available, there is the problem of integrating them and building a single source of truth that can be later exploited by domain experts and software systems.

For this reason, we propose a system that makes use of multiple technologies, summarised by the Data Flow Diagram in Figure 1. The system is built on three pillars: knowledge collection, the computation of the planning model, and the construction of new plans while interacting with domain experts. Knowledge collection starts by fetching the available data sources for Processflow data, goes through a preprocessing phase and ends with the construction of an initial KG. Each piece of data in the KG is an instance of a concept in a reference ontology, which formally describes how Processflows are structured and how they relate with other concepts of the domain (e.g. production facilities). The KG is used to generate PDDL (Ghallab et al. 1998) specifications on demand, which are used to construct new Processflows at the PBBF level using classical planning. Given these plans, the KG is used again to generate an HDDL (Höller et al. 2019) specification to perform hierarchical planning and construct Processflows at the UP level. The generated plans are shown to the domain experts via a GUI, which can also be used to collect new descriptions of the Processflow steps, enriching the KG.

Implementation

The KG comprises two major components: an ontology and data. We chose to model the structure of the Processflows using an OWL ontology for multiple reasons: (1) it allows the schema definition in a form that can unambiguously be interpreted by both machines and domain experts, (2) experts can collaboratively describe complex domains, and (3) supports efficient execution of various reasoning tasks.

We developed a *Wafer Manufacturing Ontology* to model the structure of the Processflows. Its core classes and relations are depicted in Figure 2, out of 71 class definitions and 48 relations in total. Furthermore, to integrate it with planning-related knowledge, the *Planning Ontology* presented in (Bharath et al. 2023) has been extended to enable the generation of PDDL and HDDL specifications.

Currently, we operate with 15 PBBFs, 1821 PBBs (defined on approx. 3000 UPs) that are used in at least one of the 7064 Processflows, with each Processflow containing at least one PBB.

Additionally, we implemented a pipeline (Figure 3) to build a KG from the ontology and real-world data. MorphKGC (Arenas-Guerrero et al. 2024a,b) was chosen to perform data mapping to produce the initial KG because of its stability and high performance. The actual merging of the KG and the ontology happens using Owlready2 (Lamy 2017). In our implementation, the pipeline runs offline and is used as a preliminary data collection mechanism. The domain KG is currently in the construction stage, focusing on using LLMs to extract data from existing databases. In addition to the domain concepts presented in the paper, the KG also contains auxiliary nodes and instances of domain classes. However, given the numbers above, we can estimate that the final KG will contain a number of nodes in the order of 10^9. To better decouple the data model from the KG, we implemented a custom library that maps Python objects to ontology class instances.

The system has been designed with modularity and scalability in mind, ensuring that components such as the KG, planning engine, and user interface can be independently developed, maintained, and replaced if necessary. The application includes various services to create, read, update, and delete data in the KG, generate PDDL and HDDL files, and interact with off-the-shelf planners. These services are exposed via a REST API. Given the large variety of available planning systems, a dedicated service wraps the system of choice and runs independently from the rest of the application. A standalone REST API has been developed to handle planning tasks, where planning domains and problems can

Listing 1: Plan structure. PDDL encoding example for an action which appears in 3rd and 6th position. The encoding makes it possible to provide as a goal in the planning problem predicates such as `(step-5o p q r)`, meaning that it is mandatory for the plan to include the plan structure up to the 5th step, and it shall be executed with p, q, r in place of its parameters.

```
(:action check-constraint
  :parameters ()
  :precondition ((check))
  :effect (and
    (when (step-0)
      (not (check)))
    (when (step-1)
      (not (check)))))

(:action action_123
  :parameters (?param_92 - type1)
  :precondition (and
    (not (check)))
  :effect (and
    (when (step-2)
      (and
        (step-3)
        (step-3o ?param_92)
        (not (step-2))))
    (when (step-5)
      (and
        (step-6)
        (step-6o ?param_92)
        (not (step-5))))))
```

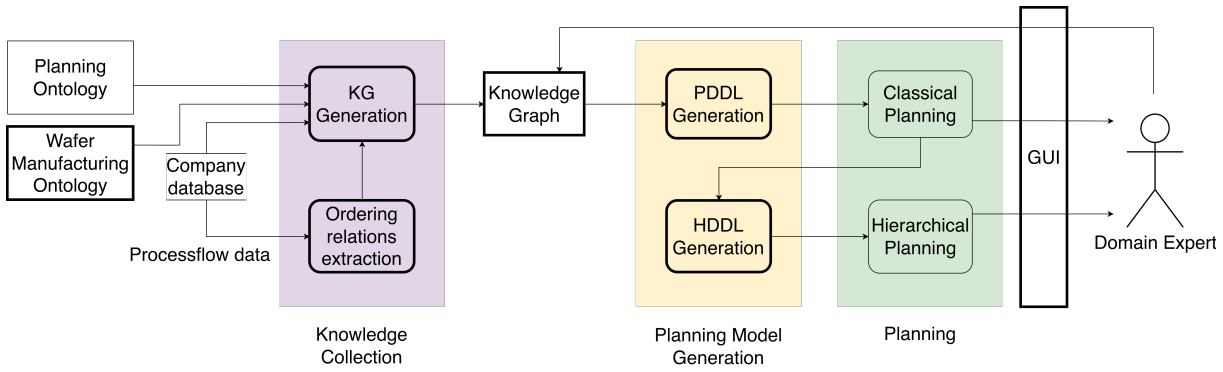

Figure 1: Application's Data Flow Diagram. Rectangular boxes represent the entities of the system that produce or consume the data, while rounded rectangles represent the system's data processes. The components with bold borders have been implemented from the ground up.

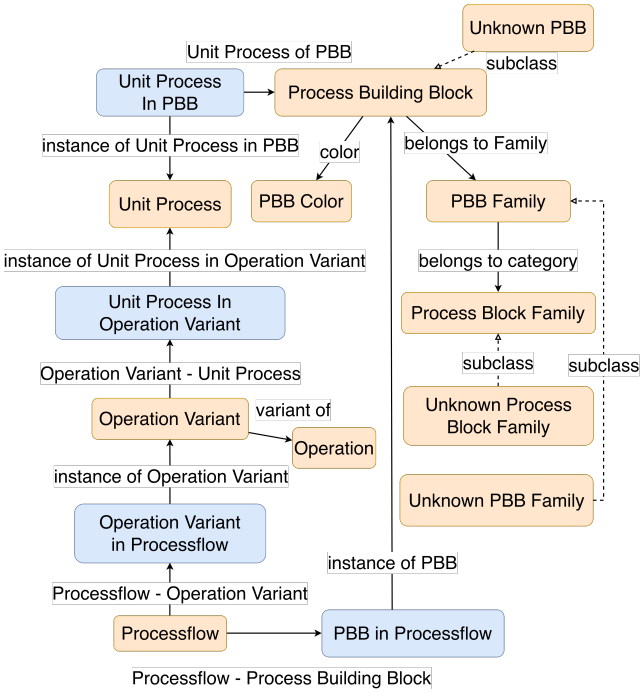

Figure 2: Main classes of the Wafer Manufacturing Ontology.

be submitted, and the computed plans are returned.

Users can describe the PBBFs, PBBs and UPs without writing PDDL and HDDL code, and can interactively construct new Processflows by providing a plan structure via the interface shown in Figure 4. Here, users can drag and drop the PBBFs from the left section to the centre one, and get live feedback based on the collected knowledge. The yellow arrows between two PBBFs indicate that no Processflow is used in the Facility in which such a sequence of PBBFs appears, while a green one means that there exists a matching sequence. Once a user clicks on the "Plan" button, the sequence of actions in the central view is substituted with

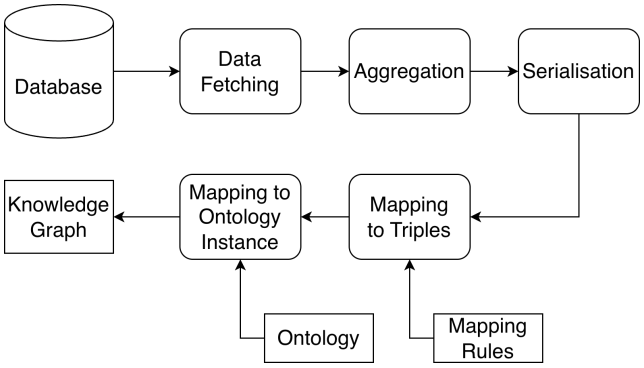

Figure 3: KG construction pipeline. Data is fetched from the database, aggregated and serialised to JSON. MorphKGC maps the JSON data to (Subject, Predicate, Object) triples integrated with the Ontology.

the computed plan. In case no plan can be computed, an error message is shown. The PDDL domain and problem encodings to include a given plan structure can be generated automatically. In fact, no PDDL or HDDL domains and problems are actually stored by the application: all the relevant data is contained in the KG, thus allowing a user to run queries over the defined actions. The encoding generation happens straightforwardly because of the classes and relations in the aforementioned Planning Ontology, which model concepts such as actions, preconditions and effects.

To allow users to define a plan structure when planning, each action a_i in the structure $\langle a_1, \ldots, a_n \rangle$, two predicates are added to the generated PDDL domain: `(step-i)` and `(step-io ?o_1 ?o_2 ... ?o_m)` where $o_1, \ldots, o_m$ are the action's parameters as found in its definition. Besides, a predicate named `(check)` is also added along with an action called `check-constraint`. The actions that are part of the plan structure are encoded as in Listing 1.

While PDDL3 (Gerevini and Long 2005) introduces plan constraints and preferences, which can be used to express a plan structure, the language is not fully supported by industry-grade planners. Instead, the proposed encod-

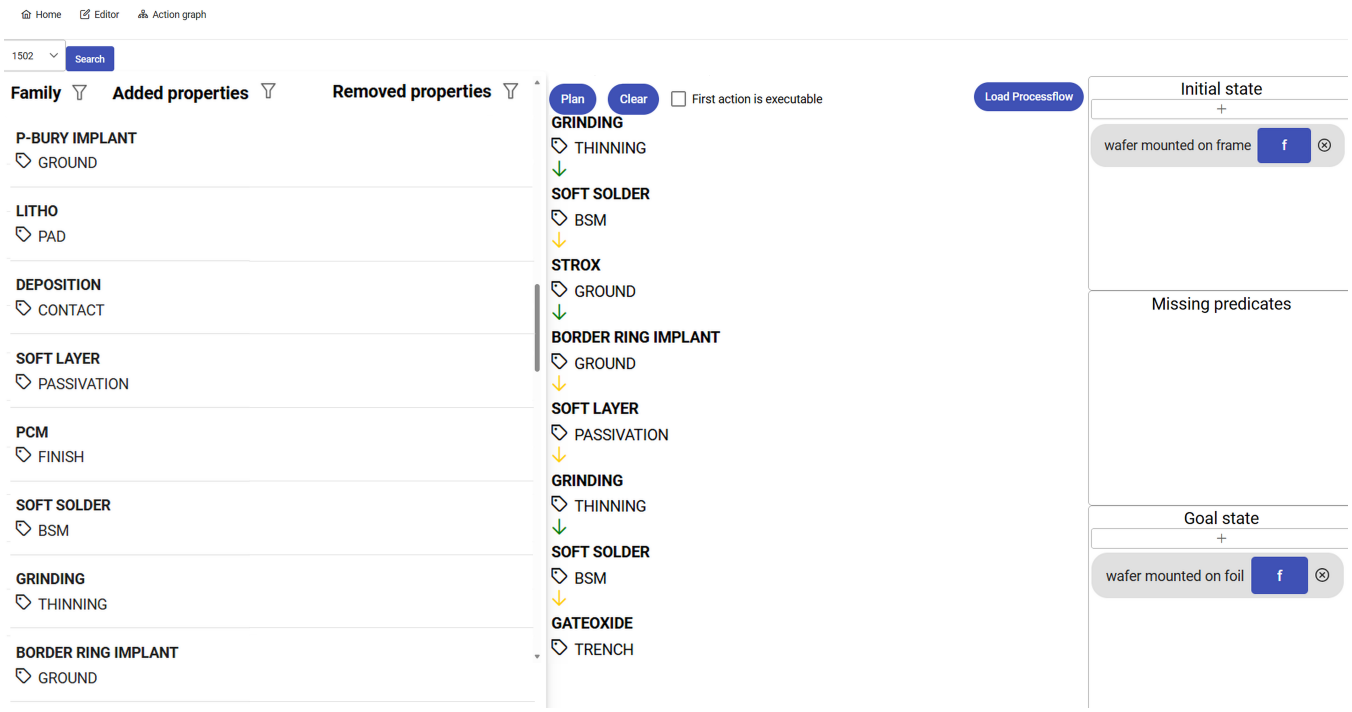

Figure 4: Processflow construction GUI.

Listing 2: Example PDDL encoding of a PBBF

```
(:action wafer-dicing
  :parameters (?w - wafer ?bsm_stack -
      stack ?si_thickness -
      silicon-thickness ?dicing_tape -
      dicing_tape)
  :precondition (and
    (frontside-flow-complete ?w)
    (backside-flow-complete ?w)
    (silicon-thickness ?w ?si_thickness)
    (bs-met-stack-deposited ?w ?
        bsm_stack))
  :effect (and
    (wafer-diced ?w)
    (mounted-on-tape ?w ?dicing_tape)
    (mounted-on-frame ?w)))
```

ing can be used by planners that support PDDL1.2 (Ghallab et al. 1998) using the `:negative-preconditions` and `:conditional-effects` requirements. We used Fast Downward for classic planning due to its stable performance in our experiments (Helmert 2006, 2009), and Panda$_\pi$ (Bercher, Meneguzzi, and Behnke 2020) for hierarchical planning, since it supports the HDDL language.

Example Use Case

As an example, we provide the encoding for the *Dicing* PBBF in Listing 2. The encoding is defined by a domain expert via the GUI and generated on demand for planning.

Die separation occurs between the wafer production and the chip assembly phases. Die separation by so-called *mechanical dicing* is the most widespread separation method in the semiconductor industry, which mechanically removes the silicon material between the dies. This implies that the wafer is mounted on a supporting frame and on an adhesive dicing tape that holds the dies during and after the separation process in place. A key parameter for dicing is the thickness of the wafer, which determines the specific UP sequence to use. The Dicing PBBF can be implemented by different sequences of UPs. Listing 3 shows one of such sequences. We distinguish the phase in which the wafer is mounted on the dicing frame and tape in `UP-mow10001` and in which the actual dicing is performed in `UP-mechanical-dicing-fast`. As can be seen, the UPs use specific parameters, such a dicing tape with ID `d-189`, which can be further described in the ontology. Another parameter is the exact backside metallization materials being used: in the example, the PBB implementing `UP-mow10001` and `UP-mechanical-dicing-fast` were developed for a wafer with a silicon thickness of $250\mu m$ and a backside metallization stack consisting of aluminium, titanium, nickel-vanadium, and gold-tin. From the modeling standpoint, we do not add the preconditions `mounted-on-tape` and `mounted-on-frame` to PBBF because they are fulfilled in the UP sequence itself.

Related Work

Up to our knowledge, at the time of writing the paper, there are no published planning approaches for semiconductor manufacturing. Nevertheless, specific components were ad-

Listing 3: Example HDDL encoding

```
(:task wafer-dicing
  :parameters (?w - wafer ?bsm_stack -
      stack ?si_thickness -
      silicon-thickness ?dicing_tape -
      dicing_tape))

(:method PBB-Dicing-Fast-process
  :parameters (?w - wafer)
  :task (wafer-dicing ?w Al-Ti-NiV-AuSn
      250um d-189)
  :ordered-subtasks (and
    (UP-mow10001 ?w)
    (UP-mechanical-dicing ?w)))

(:action UP-mow10001
  :parameters (?w - wafer)
  :precondition (and
    (frontside-flow-complete ?w)
    (backside-flow-complete ?w)
    (bs-met-stack-deposited ?w
       Al-Ti-NiV-AuSn))
  :effect (and
    (mounted-on-tape ?w d-189)
    (mounted-on-frame ?w)))

(:action UP-mechanical-dicing-fast
  :parameters (?w - wafer)
  :precondition (and
    (silicon-thickness ?w 250um)
    (mounted-on-tape ?w d-189)
    (mounted-on-frame ?w))
  :effect (and
    (wafer-diced ?w)))
```

dressed in previous work. Thus, modeling the semiconductor manufacturing domain with ontologies has been done by Mönch and Stehli (2003). Their ontology is quite general and, therefore, we developed a more specific one. More general approaches include the *Process Specification Language* (Schlenoff et al. 2000) and the MASON ontology (Lemaignan et al. 2006). The key concepts used by the partner company, for instance PBBFs, PBBs, UPs, and their relations, were not modeled by the ontologies from the literature that we analyzed. Therefore, we opted for a custom solution for the product- and process-related data.

The relevance of ontologies in planning is further explored in multiple studies (Gayathri and Uma 2018; Ko, Lee, and Lee 2012; Teixeira et al. 2023), and practical frameworks for the integration of the two formalisms have been proposed (McNeill, Bundy, and Walton 2005; Cioffi and Thompson 2007; Bouillet et al. 2007; Freitas et al. 2014; Behnke et al. 2015; Louadah et al. 2021; Malburg, Klein, and Bergmann 2023). The architecture of the system presented in this paper is similar to the work of (Pham and Stacey 2011), where the authors introduce an architecture that integrates OWL ontologies with planning systems. In this framework, planning domains and problems are embedded directly into the ontology.

Finally, we refer to (Marthi, Russell, and Wolfe 2007; Bercher et al. 2016) for their contributions on the topic of including preconditions and effects to high-level tasks in hierarchical planning domains, which can be considered a non-standard practice. Moreover, the topic of automatic inference of such high-level descriptions has been recently addressed (Olz, Biundo, and Bercher 2021; Olz et al. 2025).

Conclusions And Future Work

The paper proposes a foundation for knowledge-driven Processflow formulation in semiconductor manufacturing, but several areas remain open for future exploration and improvement. One of the primary limitations is the current scope of data collection. We built a KG using only a subset of Processflows, and expanding the data pipeline to include more comprehensive datasets would significantly enhance the system's knowledge base. The proposed web application offers basic functionality for planning-based Processflow construction and knowledge collection, but its user experience and productivity could be significantly improved. More adaptive interfaces with contextual guidance would better support users with different expertise levels. Enabling multi-user simultaneous editing of Processflows with conflict resolution would also support collaborative design common in industrial settings.

Several fundamental research questions have emerged from this work important for further investigation and advancement. *First*, manual definition of action models in automated planning is a significant bottleneck, particularly in complex domains such as semiconductor manufacturing. Unfortunatly, due to the insufficient data quality we were unable to apply approached to learning action models from available plans (Arora et al. 2018), to obtain results as reported in (Balyo et al. 2024; Gösgens, Jansen, and Geffner 2024; Lamanna et al. 2025). *Second*, it is important to explore ways to provide users with reasons why the goals of the Processflow under development cannot be reached, and what the possible corrections or alternative goals that could satisfy them. Additionally, designing interactive explanation systems that enable users to iteratively refine their requirements for improved usability would enhance the user experience. *Third*, it is important to develop algorithms to detect and resolve conflicts between concurrent processes, optimize resource allocation, and balance objectives such as cost, time, and quality. Additionally, mechanisms must be established to maintain consistency and reliability when Processflows are modified or updated.

A preliminary user-study showed that the experts have a great interest in the prototype adoption. Therefore, the company is now implementing a tool for industry-scale deployment, which will replace the existing tools. The research will be advanced by the partner company that provided funding for the continuation of the work over the next three years, with multiple sites worldwide involved.

Acknowledgements

This research was funded in part by the Austrian Science Fund (FWF) 10.55776/PAT1599524, Austrian Research Promotion Agency (FFG) 930480ATRIA (Eureka la-

beled ATRIA project), and Infineon Technologies AG. We thank Alice Tarzariol, Martin Gebser, and Agostino Dovier for their support and insights.

References

Arenas-Guerrero, J.; Chaves-Fraga, D.; Toledo, J.; Pérez, M. S.; and Corcho, Ó. 2024a. Morph-KGC: Scalable knowledge graph materialization with mapping partitions. *Semantic Web*, 15(1): 1–20.

Arenas-Guerrero, J.; Espinoza-Arias, P.; Bernabé-Díaz, J. A.; Deshmukh, P.; Sánchez-Fernández, J. L.; and Corcho, Ó. 2024b. An RML-FNML module for Python user-defined functions in Morph-KGC. *SoftwareX*, 26: 101709.

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A review of learning planning action models. *Knowl. Eng. Rev.*, 33: e20.

Bacchus, F.; and Kabanza, F. 1996. Planning for Temporally Extended Goals. In Clancey, W. J.; and Weld, D. S., eds., *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2*, 1215–1222. AAAI Press / The MIT Press.

Balyo, T.; Suda, M.; Chrpa, L.; Šafránek, D.; Gocht, S.; Dvořák, F.; Barták, R.; and Youngblood, G. M. 2024. Planning Domain Model Acquisition from State Traces without Action Parameters. 21(1): 812–822.

Behnke, G.; Bercher, P.; Biundo, S.; Glimm, B.; Ponomaryov, D.; and Schiller, M. 2015. Integrating Ontologies and Planning for Cognitive Systems.

Bercher, P.; Höller, D.; Behnke, G.; and Biundo Susanne. 2016. More than a Name? On Implications of Preconditions and Effects of Compound HTN Planning Tasks. In *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Bercher, P.; Meneguzzi, F.; and Behnke, G. 2020. https://github.com/panda-planner-dev/pandaPIengine.

Bharath, M.; Vishal, P.; Biplav, S.; Raghava, M.; Michael, N. H.; and Vignesh, N. 2023. A Planning Ontology to Represent and Exploit Planning Knowledge for Performance Efficiency. *CoRR*, abs/2307.13549.

Bouillet, E.; Feblowitz, M.; Liu, Z.; Ranganathan, A.; and Riabov, A. 2007. A Knowledge Engineering and Planning Framework Based on OWL Ontologies. *Proceedings of the Second International Competition on Knowledge Engineering*, 191.

Cioffi, M.; and Thompson, S. 2007. Planning with the Semantic Web by Fusing Ontologies and Planning Domain Definitions. In Bramer, M.; Coenen, F.; and Tuson, A., eds., *Research and Development in Intelligent Systems XXIII*, 289–302. Springer. ISBN 978-1-84628-663-6.

Freitas, A.; Schmidt, D.; Panisson, A.; Meneguzzi, F.; Vieira, R.; and Bordini, R. 2014. Semantic Representations of Agent Plans and Planning Problem Domains. In *Engineering Multi-Agent Systems - Second International Workshop*, volume 8758 of *Lecture Notes in Computer Science*. ISBN 978-3-319-14483-2.

Gayathri, R.; and Uma, V. 2018. Ontology Based Knowledge Representation Technique, Domain Modeling Languages and Planners for Robotic Path Planning: A Survey. 4(2): 69–74.

Gazen, B. C.; and Knoblock, C. A. 1997. Combining the Expressivity of UCPOP with the Efficiency of Graphplan. In Steel, S.; and Alami, R., eds., *Recent Advances in AI Planning, 4th European Conference on Planning, ECP'97, Toulouse, France, September 24-26, 1997, Proceedings*, volume 1348 of *Lecture Notes in Computer Science*, 221–233. Springer.

Gerevini, A.; and Long, D. 2005. Plan Constraints and Preferences in PDDL3. Technical report, Technical Report 2005-08-07, Department of Electronics for Automation ....

Ghallab, M.; Howe, A.; Knoblock, C.; McDermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. Pddl—the Planning Domain Definition Language. *Technical Report, Tech. Rep.*

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier. ISBN 978-1-55860-856-6.

Gösgens, J.; Jansen, N.; and Geffner, H. 2024. Learning Lifted STRIPS Models from Action Traces Alone: A Simple, General, and Scalable Solution. *CoRR*, abs/2411.14995.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.*, 26: 191–246.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artif. Intell.*, 173(5-6): 503–535.

Höller, D.; Behnke, G.; Bercher, P.; Biundo, S.; Fiorino, H.; Pellier, D.; and Alford, R. 2019. HDDL - A Language to Describe Hierarchical Planning Problems. *CoRR*, abs/1911.05499.

Ko, R. K.; Lee, E.; and Lee, S. 2012. Business-OWL (BOWL)—A Hierarchical Task Network Ontology for Dynamic Business Process Decomposition and Formulation. 5(2): 246–259.

Lamanna, L.; Serafini, L.; Saetti, A.; Gerevini, A. E.; and Traverso, P. 2025. Lifted Action Models Learning from Partial Traces. 339: 104256.

Lamy, J. 2017. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artif. Intell. Medicine*, 80: 11–28.

Lemaignan, S.; Siadat, A.; Dantan, J.-Y.; and Semenenko, A. 2006. MASON: A Proposal for an Ontology of Manufacturing Domain. In *IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06)*, 195–200. IEEE.

Louadah, H.; Papadakis, E.; Mccluskey, T.; Tucker, G.; Hughes, P.; and Bevan, A. 2021. Translating Ontological Knowledge to PDDL to Do Planning in Train Depot Management Operations. In *Proceedings of PlanSIG 2021*.

Malburg, L.; Klein, P.; and Bergmann, R. 2023. Converting Semantic Web Services into Formal Planning Domain Descriptions to Enable Manufacturing Process Planning and Scheduling in Industry 4.0. *Engineering Applications of Artificial Intelligence*, 126: 106727.

Marthi, B.; Russell, S.; and Wolfe, J. A. 2007. Angelic Semantics for High-Level Actions. In Boddy, M. S.; Fox, M.; and Thiébaux, S., eds., *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, 232–239. AAAI.

McNeill, F.; Bundy, A.; and Walton, C. 2005. Planning from Rich Ontologies through Translation between Representations. In *Workshop on The Role of Ontologies in Planning and Scheduling*.

Mönch, L.; and Stehli, M. 2003. An Ontology for Production Control of Semiconductor Manufacturing Processes. In Schillo, M.; Klusch, M.; Müller, J.; and Tianfield, H., eds., *Multiagent System Technologies*, 156–167. Springer. ISBN 978-3-540-39869-1.

Olz, C.; Biundo, S.; and Bercher, P. 2021. Revealing Hidden Preconditions and Effects of Compound HTN Planning Tasks – A Complexity Analysis. 35(13): 11903–11912.

Olz, C.; Lodemann, A.; Jutz, B.; Schmautz, M.; Borowiec, M.; Biundo, S.; and Bercher, P. 2025. An Extensive Empirical Evaluation of Inferring Preconditions and Effects of Compound Tasks in Ground HTN Planning Problems. 82: 1407–1444.

Pham, H.; and Stacey, D. 2011. Practical Goal-Based Reasoning in Ontology-Driven Applications. In *Proceedings of the International Competition on Knowledge Engineering and Ontology Development*, 99–109. SciTePress.

Schlenoff, C.; Gruninger, M.; Tissot, F.; Valois, J.; and Lee, J. 2000. The Process Specification Language (PSL) Overview and Version 1.0 Specification. Technical Report NIST IR 6459, National Institute of Standards and Technology.

Teixeira, M. S.; Welt, M.; Chis, R.; and Glimm, B. 2023. Challenges on Deriving Planning Problems from Ontologies. In *PLATO@ICAPS*, volume 3493 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Tenenberg, J. D. 1988. *Abstraction in planning*. Ph.D. thesis, University of Rochester, USA. Order No: GAX88-16885.

Yang, Q. 1990. Formalizing Planning Knowledge for Hierarchical Planning. *Computational Intelligence*, 6(1): 12–24.