Proceedings of the Federated Conference on Computer Science and Information Systems pp. 587–596
DOI: 10.15439/2017F154
ISSN 2300-5963 ACSIS, Vol. 11

Answer Set Programming for Modeling and Reasoning on Modular and Reconfigurable Transportation Systems

Walter Terkaj
Istituto di Tecnologie Industriali e Automazione
via A.Corti 12, 20131 Milano, Italy
Email: walter.terkaj@itia.cnr.it

Marcello Urgo, Daniela Andolfatto
Politecnico di Milano,
Mechanical Engineering Department,
via La Masa 1, 20156 Milano, Italy
Email: marcello.urgo@polimi.it,
daniela.andolfatto@mail.polimi.it

***Abstract*—This paper addresses the modeling of modular and reconfigurable transportation systems, aiming at developing tools to support the planning and control. Answer Set Programming (ASP) is employed to formalize rules modeling the characteristic of a transportation system and describing its dynamics. Then, automatic reasoning can be exploited to find solutions in different use cases, including the generation of optimal or alternative paths, the generation and validation of control sequences. The proposed methodology is applied to a reconfigurable industrial transportation system consisting of multiple linear conveyor modules with actuators enabling longitudinal and transversal movements of pallets.**

I. Introduction

RECONFIGURABLE Manufacturing Systems (RMS) [1], [2] aim at tackling current market challenges such as frequent product changes, product customization, demand variability, rapid changes in technologies and regulations. RMS are conceived as a composition of elements, machines and material handling systems, whose systemic configuration can be easily changed. Reconfigurability should be met at both hardware and software level to reduce reconfiguration time (ramp-up), effort and cost.

Material handling systems (MHS) represent a key component in an RMS, being the pieces of equipment devoted to handling, storing, and controlling materials and parts. When addressing the design of an RMS, the MHS plays a relevant role, since it must support the reconfigurability at system level. A possible solution are Reconfigurable Transportation Systems (RTS), usually based on modular mechatronic transportation units that can be combined to implement a logistic system layout. Carpanzano et al. [3] identified three main challenges to be addressed when designing and implementing RTSs:

(a) hardware reconfigurability, i.e., the design of transportation systems that meet the physical reconfigurability requirement. Mechanical and mechatronic interfaces are a relevant issue when tackling this matter.
(b) software development of the control systems, including the appropriate sensing solutions. The suggested architecture is a distributed one, since all the modules should be independent and considered as an autonomous control block.
(c) real-time management of the production system, considering routings, dispatching policies, planning and scheduling, maintenance.

In this work the main focus will be on (b) and (c). The control problem (b) is particularly relevant when mechatronic systems are involved, due to the difficulty of integrating and managing all the components. The control system of an automated MHS must be able to elaborate plans and instructions considering objectives such as throughput maximization, response time and execution time optimization. Haneyah et al. [4] presented an extensive study of generic planning and control requirements for automated material handling systems. These requirements include starvation, blocking and deadlock avoidance, saturation and buffer balancing, urgencies, disruptions and operational flexibility.

Cataldo and Scattolini [5] presented a methodology to optimize the control of a transportation system where pallets are moved along modular conveyors. The control system was implemented by adopting Model Predictive Control (MPC) with a multi-layer approach. In the low level control (LLC) of the system there are the local Programmable Logic Controllers (PLC) that manage each module sensors and actuators, while at the highest level the MPC controller is implemented and manages the flows of the pallets in real-time providing proper control sequences. The High-Level Control (HLC) System was implemented by representing the transportation system as a directed graph, where nodes are machines and buffer zones, whereas the arcs are the possible movements of the pallet. Based on the graph, a mathematical representation of the system was developed through a Mixed Linear Dynamical (MLD) model and embedded in the MPC controller.

Hegny et al. [6] presented a non-traditional control ap-

This work has has been partially funded from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 636966 (Customer-driven design of product-services and production networks to adapt to regional market requirements - ProRegio) and from the Italian research project Smart Manufacturing 2020 within the Cluster Tecnologico Nazionale Fabbrica Intelligente.

proach, implementing a two-levels control architecture based on Multi Agents System (MAS) technology.

In both cases, the need to divide the control system into a low and high level is the solution to the reconfigurability objective. This structure, in fact, allows easier reconfiguration of the software system since the LLC is implemented on each unit and the HLC is designed independently from the physical system. This allows to separate the control of the functioning of the elements composing the system from the high-level control of the system as a whole and from the implementation of planning and routing algorithms.

The planning problem (c) for an automated transportation systems consists in defining routing and dispatching policies regulating the movements of the processed pieces by defining the sequence they visit machines and workstations. Routing and dispatching policies are implemented in the high-level control layer of the control system.

The goal of this paper is the development of an elaboration tool to support the modeling of a transportation system and reasoning on the system dynamics to derive its properties. Possible applications of the tool include:

(i) the generation of paths and control sequences to implement movements of the objects along the transportation system;
(ii) the generation of a reachability graph showing how the positions in the transportation system can be connected;
(iii) validation of the control sequences generated by a planning algorithm;
(iv) configuration of a transportation system and its devices enabling to meet the required functionalities expressed in terms of movements in the system.

In particular this work will focus on the applications (i) and (ii), with the aim of supporting the following users:

(1) system designers in the analysis and validation of alternative transportation system configurations;
(2) control designers addressing an existing hardware design of a transportation system;
(3) control designers tackling an existing transportation system through the extraction of knowledge, e.g., from the analysis of PLCs, hardware components, etc.

Herein, the elaboration tool is conceived as a logic program that adopts the Answer Set Programming (ASP) language to represent the system in terms of rules and obtain solutions in the form of stable models (i.e. *answer set*). Problems related to the modeling of automation systems to support their control are typically addressed using other techniques like automata, finite state machine and Petri Nets (PN) [7]. In particular, PN is in principle able to support the applications (i)-(iv) previously defined. However, it must be noted that an approach based on PN requires a quite verbose formalism that leads to overly complex models with an explosion in terms of number of *places* and *transitions*. More advanced PN extensions can only partially reduce such complexity. Therefore, even if a PN model can be used to easily derive relevant properties of a system (e.g. reachability, liveness, boundedness, deadlocks) using general purpose tools, still the generation of the PN model itself is a relevant problem that requires skilled modeling operations and thus represents a bottleneck of the approach. This problem is particularly relevant when dealing with reconfigurable transport systems since a physical reconfiguration demands also a reconfiguration of the model. Even if a modular PN approach can be employed (and it is not immediate in the general case of non-identical transport modules), anyhow it can be cumbersome to identify which are the transitions linking the *places* in different PN sub-graphs.

The proposed ASP-based approach aims first and foremost to support the generation of a formal model for a specific transportation system by taking as input the description of the physical system (*facts*) and the general description of the dynamics and interactions between the basic components of the system (*rules*). Therefore, ASP can be seen as complementary to PN for the generation of the formal model, whereas regarding the reasoning over a formal model of the system there can be an overlapping between ASP and PN, as it will be discussed in the next sections.

The paper is organized as follows. Sect.II will briefly present ASP and some relevant application of this language. Sect.III will introduce the details of the reference transportation problem that is considered in this work. Sect.IV presents the details of the model based on ASP rules. Sect.V shows which type of reasoning can be performed and finally Sect.VI provides an application example.

II. Reasoning with Answer Set Programming

Answer Set Programming [8], [9], [10] is a logic programming language for knowledge representation and reasoning. An ASP program consists of a set of rules of the form:

$$a \leftarrow b_1, ..., b_m, \, not\, c_1, ..., not\, c_n \tag{1}$$

where *a,* b_i, c_j are atoms. The left-hand side of the rule is the *head*, whereas the right-hand side is the *body*. The head is derived to be true if all the literals in the body are true. If a rule has no body, then it is a *fact*. A rule without head is a *constraint*. The symbol *not* in (1) represents a *default negation* (or *weak negation*), that is a key feature of ASP and more generally of non-monotonic reasoning. The solution of an ASP program is an answer set (or stable model), i.e. the smallest set of literals that satisfies the program. An ASP solver can return zero, one or many answer sets.

Another useful feature of the ASP language and its extensions is represented by *cardinality atom*. For instance, the form $l\{a_1, ..., a_n\}k$ means that at least l and at most k atoms in the set $\{a_1, ..., a_n\}$ are true. If l or k are missing, then the corresponding side is unbounded.

ASP has been successfully applied in several domains, e.g., product configuration, aerospace, data management, music and planning [10]. ASP is also a declarative language, i.e. a program written in this language does not specify how to search a solution (this is a key difference between ASP and Prolog); indeed, the solution is independently found by the

solver also thanks to the availability of general purpose ASP solvers.

Taking in consideration the scope of automation and control, it is relevant to mention the work of [11] that discussed how ASP can be generically employed to generate plans as alternative stable models. More specifically, [12] addressed how ASP can be applied to support the planning of collaborative robot. [13] proposed an approach to generate paths for robots in a dynamic environment. Similarly, [14] focused on a cost-based robot planning taking by formalizing the rigid knowledge, time-dependent knowledge, action knowledge and incomplete information. However, to the best of our knowledge, this paper represents the first application of ASP in the domain of modular transportation systems.

The relation between ASP and PN has been studied by Anwar et al. [15], [16] and they demonstrated that PN models can be actually encoded in ASP language. Once the model is encoded in ASP, then it is possible to use ASP solvers to analyze typical properties of the system as in the case of a PN (e.g. reachability of a state, basic liveness). Simulations can be run using either a PN or ASP approach, but ASP offers the following advantages:

- ASP may return the enumeration of all possible evolutions of a simulation [15]
- the formal model can be enriched with additional and customized reasoning about the simulations [15]

Therefore, ASP can be used not only to generate a formal model of the system, but also to analyze the system behavior with a reasoning power that is not lower than what can be done with PN. Given a specific transportation system to be analyzed, the advantage of ASP is that it supports both the generation of the formal model and also the reasoning about its properties using the same language and solvers.

III. Problem Statement

In this work the attention is focused on MHS that consist of conveyors. These transportation systems are bound to specific operational routes and require the installation of fixed tracks. The material is loaded on pallets or specific carriers transported along the conveyors (by means of chains, rollers, belts, etc.). The structure of conveyor systems is usually obtained by combining together standardized modular pieces. Modularity requires considerable design and investment efforts, but it offers the possibility to expand and reconfigure the transportation system according to the variable needs of the plant. Moreover, automation also entails a high level of real-time flexibility, thanks to the possibility of developing complex planning and control software tools. An automated conveyor-based transportation system also contains sensing devices and actuators as well as control tools to manage all these elements. Therefore, the generic transportation system that is taken as a reference is defined as follows:

- the system consists of M linear conveyor modules;
- pallets are characterized by a rectangular shape and can be moved along the system;
- each module is equipped with 1) sensing elements to detect the presence of a pallet, 2) conveyor belts, or any other similar movement device, 3) stacker cranes for cross movement of the pallet along a direction orthogonal to the movement of the main conveyor belt, 4) blocking actuators to stop the movement of the pallet when it is flowing along the conveyors;
- each module hosts N discrete positions for pallets, numbered from 1 to N, discretizing the space available on the module on the basis of the pallet size and shape. Each position can host only one pallet;
- each position is characterized by four sides, numbered from 1 to 4 clock-wise (see Fig.1). Sides are used to describe how positions are connected to each other (see Sect.IV-B);
- the direction of movement of the conveyor belt is *forward* if it causes the pallet to move along a direction with increasing number id of the positions, *backward* in the opposite case;
- the direction of movement of the stacker crane is *left* if it causes the pallet to move on the left when looking from position 1 to N, *right* in the opposite case;
- the blocking actuators are always coupled with sensing elements. The actuators are designed so that the pallet can move toward a position with an actuator even if it is activated, whereas it can move out of the position only if the actuator is not activated. An actuator can work both when the pallet is moving forward or backward.

The transportation system consists of a set elements connected together to form a complex system characterized by the properties of reconfigurability, scalability and flexibility. Reconfigurability is obtained through varying the arrangement of modules in the plant in order to face changing functional requirements. Similarly, the system is easily scalable by adding or removing modules. Flexibility can be reached by allowing different pallet routings.

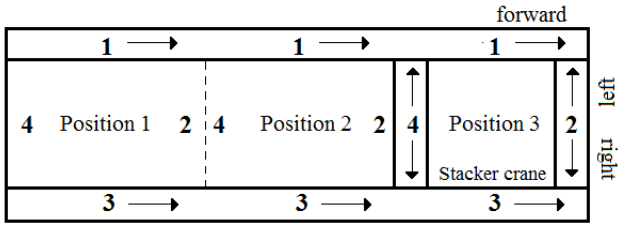

Fig. 1. Representation of the sides and feasible movements in a 3-position transportation module with one stacker crane.

IV. Modeling Transportation Modules using ASP

This section presents the logic program written in ASP language able to capture the basic characteristics and dynamics of the transportation system and elaborate its evolution in time in terms of the sequence of states reached by the system. The state of the system is defined by the position of the pallets

and the state of actuators and sensors. The dynamic behavior of the system is subject to the following constraints:

- placement of the modules (system physical layout);
- characteristics of the modules in terms of hosted positions and installed actuators;
- characteristics of the modules and cross elements in terms of supported direction of movement.

The addressed problem requires the logic program to be modular, clustering the rules. Each cluster addresses a characteristic of the system or a specific modeling issue and is independent from the others. In general, modularity allows leaner development of the rules and faster maintenance. The main clusters are described in the following subsections: *Input Facts* describing the topological and physical characteristics of the system (Sect.IV-A), the *Rigid Knowledge* defining how the interactions between the system components determine the possible system behavior (Sect.IV-B), and the *Time-dependent Knowledge* defining how the state of the system changes along time because of the control actions (Sect.IV-C).

The clusters of rules are presented by adopting the syntax of `clingo` [17], including its extensions to the basic ASP language. Like in most logic programming languages, the symbol `:-` represents the leftwards arrow that separates the head and body of a rule, as show in (1).

A. Input Facts

The physical and geometric characteristics of a specific transportation system are defined in terms of input facts. The following predicates (fluents) need to be properly instantiated:

- `module(M)` defines the transportation module `M`.
- `pos(Y,M)` defines a position `Y` on transportation module `M`.
- `cross(Y,M)` defines the presence of a stacker crane element on position `Y` of module `M`.
- `act_stop(Y,M)` defines that there is a blocking actuator on position `Y` of module `M`.
- `conn_mod(Y1,M1,S1,Y2,M2,S2)` defines that the side `S1` of position `Y1` in module `M1` is adjacent to the side `S2` of position `Y2` in module `M2`.
- `conv(M,D1,D2)` defines the feasible movement directions of the conveyor belt for module `M`. If the conveyor belt supports both forward and backward direction, then `D1=f` and `D2=b`, respectively. If the conveyor supports only one movement direction, then corresponding variable is set to `0`.
- `cross_conv(Y,M,D1,D2)` defines the feasible directions of movement of a stacker crane that is placed on position `Y` of module `M`. If it supports both the left and right directions, then `D1=l` and `D2=r`, respectively. If the stacker crane supports only one movement direction, then the corresponding variable is set to `0`.

B. Rigid Knowledge

Rigid knowledge refers to the information that is not subject to change during the execution of the program and its reasoning, i.e. spatial properties of the system, reachable positions in the system, buffer zones. This knowledge is general purpose because it can be exploited for any specific system characterized by its corresponding *input facts*.

1) Spatial properties of the system: The possibility to move a pallet between two adjacent positions must consider the availability of proper actuators. Figure 2 shows some of the types of connections. Specifically, the pallet can be moved from a position to another only if the two positions are adjacent and if the connecting elements are consistent in terms of direction of movement. The feasibility of a transfer between

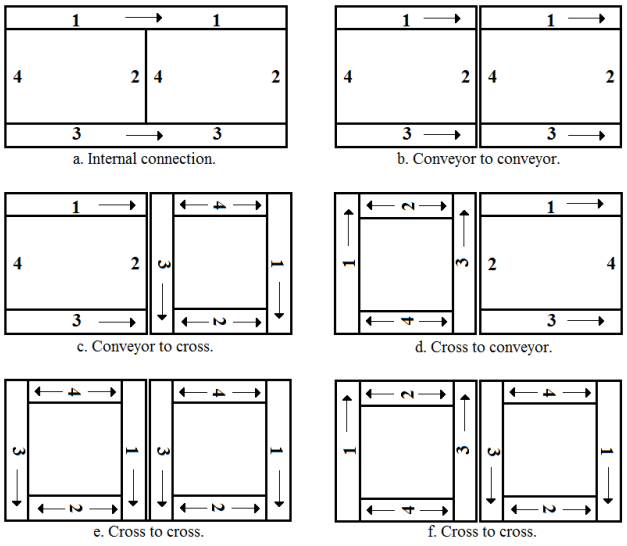

Fig. 2. Examples of adjacent positions enabling a movement.

two positions is defined by the following predicates:

- `conv_to_conv(Y1,M1,Y2,M2,D1,D2)` between position `Y1` of module `M1` and position `Y2` of module `M2` if the conveyor belt of `M1` is moving in direction `D1` (i.e. `f` or `b`) and the conveyor belt of `M2` is moving in direction `D2` (see cases a. and b. in Fig.2);
- `conv_to_cross(Y1,M1,Y2,M2,D1,D2)` between position `Y1` of module `M1` and position `Y2` of module `M2` if the conveyor belt of `M1` is moving in direction `D1` and the stacker crane on position `Y2` is moving in direction `D2` (see case c. in Fig.2);
- `cross_to_conv(Y2,M2,Y1,M1,D1,D2)` between position `Y1` of module `M1` and position `Y2` of module `M2` if the stacker crane on position `Y1` is moving in direction `D1` and the conveyor belt of `M2` is moving in direction `D2` (see case d. in Fig.2);
- `cross_to_cross(Y1,M1,Y2,M2,D1,D2)` between position `Y1` of module `M1` and position `Y2` of module `M2` if the stacker crane on position `Y1` is moving in direction `D1` and the stacker crane on position `Y2` is moving in direction `D2` (see cases e. and f. in Fig.2).

All the four types of connections can be derived from the input facts. For example rules (2) and (3) defines feasible `conv_to_conv` connections for adjacent positions belonging to the same module (see Fig.2.a). Rule (2) deals with forward movement, whereas rule (3) with backward movement. The underscore symbol in `conv(M,f,_)` and `conv(M,_,b` is a wildcard standing for any type of movement supported by the conveyor belt for the case of backward and forward movement, respectively.

```
conv_to_conv(Y1,M,Y2,M,f,f) :- Y2=Y1+1,
       pos(Y1,M), pos(Y2,M),conv(M,f,_).
```
(2)

```
conv_to_conv(Y1,M,Y2,M,b,b) :- Y1=Y2+1,
        pos(Y1,M),pos(Y2,M),conv(M,_,b).
```
(3)

If the adjacent positions belong to different modules (see Fig.2.b), then other four rules can be defined to specify the required movements of the two conveyor belts (forward or backward) depending on the relative placement between two position in terms of adjacent sides. For example, rule (4) defines the feasible `conv_to_conv` connection from position `Y1` of module `M1` to position `Y2` of module `M2` when the side 2 of `Y1` is adjacent to side 4 of `Y2`. The cardinality atom `1{conn_mod(Y1,M1,S1,Y2,M2,S2); conn_mod(Y2,M2,S2,Y1,M1,S1)}1` takes into account the fact that `conn_mod` is not considering an order between two adjacent positions, i.e. `conn_mod(Y1,M1,S1,Y2,M2,S2)` is equivalent to `conn_mod(Y2,M2,S2,Y1,M1,S1)`.

```
conv_to_conv(Y1,M1,Y2,M2,f,f):- S1=2,S2=4,
    M1!=M2,pos(Y1,M1),pos(Y2,M2),
    conv(M1,f,_),conv(M2,f,_),
    1{conn_mod(Y1,M1,S1,Y2,M2,S2);
    conn_mod(Y2,M2,S2,Y1,M1,S1)}1.
```
(4)

The movement between not aligned modules is operated by stacker crane actuators. Rules similar to (4) can be defined taking in consideration all the possible couplings (some of which are shown in Fig.2.c-Fig.2.f) to derive the predicates `conv_to_cross`, `cross_to_conv` and `cross_to_cross`. If a connection is not feasible, then the rules are not satisfied and the corresponding predicate is not derived. Due to space limitations, most of these rules are omitted and only rule (5) is shown as an example of `conv_to_cross` connection. In this case the connection from position `Y1` of module `M1` to position `Y2` of module `M2` is feasible if the side 2 of `Y1` is adjacent to side 3 of `Y2` and there is a stacker crane element on the second position (`cross(Y2,M2)`).

```
conv_to_cross(Y1,M1,Y2,M2,f,l) :- S1=2,
    S2=3,M1!=M2,pos(Y1,M1),pos(Y2,M2),
    cross(Y2,M2),conv(M1,f,_),
    cross_conv(Y2,M2,l,_),
    1{conn_mod(Y1,M1,S1,Y2,M2,S2);
    conn_mod(Y2,M2,S2,Y1,M1,S1)}1.
```
(5)

2) Reachable positions: Grounding on the previous rules, the predicate `rch(Y1,M1,Y2,M2)` representing the reachability between two positions is derived if position `pos(Y2,M2)` can be reached from position `pos(Y1,M1)` via a feasible connection. Rule (6) shows an example in case of `conv_to_conv` connection. Similar rules can be written for the other possible connections (see Fig.2).

```
rch(Y1,M1,Y2,M2) :- pos(Y1,M1),pos(Y2,M2),
          conv_to_conv(Y1,M1,Y2,M2,_,_).
```
(6)

3) Buffer zones: A buffer zone is defined by the predicate `b_zone(Y,M)` if the pallet can be stopped in position `Y` of module `M` thanks to the presence of a blocking actuator. The blocking actuator is associated with a sensor able to detect the presence of the pallet and/or the status of the actuator (on/off), hence, a buffer zone is also an observable position. The buffer zones can be derived from rule (7).

```
b_zone(Y,M) :- pos(Y,M),act_stop(Y,M).
```
(7)

C. Time-dependent Knowledge

The assumption is made that the evolution of the transportation system along time can be represented by defining discrete time intervals ranging from 0 to `max_time`. The following predicates specify the evolution of the system along the time horizon. For sake of simplicity, the case of a single pallet is considered, but the rules can be extended to represent also the general case of multi-pallet systems. The following predicates are used to define the state of the system:

- `p_pos(Y1,M1,T)` defines that the pallet is in position `pos(Y1,M1)` at time `T`;
- `move(Y1,M1,Y2,M2,T)` defines that at time `T` the pallet has been moved from `pos(Y1,M1)` to `pos(Y2,M2)`;
- `conv_on(M,D,T)` defines that the conveyor of module `M` is active at time `T` and moving in direction `D` (i.e. `f` or `b`);
- `cross_conv_on(Y,M,D,T)` defines that the stacker crane hosted in pos(Y,M) is active at time `T` and moving in direction `D` (i.e. `l` or `r`);
- `stop_on(Y,M,T)` defines that the blocking actuator installed in `pos(Y,M)` is active at time `T`.

1) Dynamics: The following rules generates the sequence of movements of the pallet along the transportation system. Rule (8) states that if a pallet is in a buffer zone, then the following time step it can remain in the same position or move to a reachable position; on the other hand, rule (9) states that if a pallet is in a position that is not a buffer zone, then the following time step it must be moved to a reachable position. Rule (10) makes explicit the movement of a pallet. Rule (11) guarantees that the same pallet cannot be in two different positions at the same time step. Finally, rule (12) is optional and can be enabled if the pallet is forbidden to visit twice the same position within the same control sequence.

```
{p_pos(Y3,M3,T+1): rch(Y2,M2,Y3,M3);
    p_pos(Y2,M2,T+1)}1 :- p_pos(Y2,M2,T),
    b_zone(Y2,M2),T<max_time.                    (8)

{p_pos(Y3,M3,T+1): rch(Y2,M2,Y3,M3)}1 :-
    p_pos(Y2,M2,T),not b_zone(Y2,M2),
    T<max_time.                                  (9)

move(Y1,M1,Y2,M2,T+1) :-
   p_pos(Y1,M1,T), p_pos(Y2,M2,T+1),
   pos(Y1,M1)!=pos(Y2,M2),T<max_time.           (10)

:- p_pos(Y1,M1,T), p_pos(Y2,M2,T),
  pos(Y1,M1)!=pos(Y2,M2),T<=max_time.           (11)

:- move(Y,M,_,_,A),move(Y,M,_,_,B),A!=B.
                                               (12)
```

2) Control Actions: Given the movements of the pallet, rules can be used for deriving the required sequence of control actions, (e.g. activation of actuators). Rules (13) and (14) derive the activation of the conveyor belt in case of a connection `conv_to_conv`. Other rules are defined (omitted due to space limitations) to derive the activation of the conveyor belt and/or the stacker crane for the other types of connections (`conv_to_cross`, `cross_to_conv` and `cross_to_cross`) as introduced in Sect.IV-B.

```
conv_on(M,D,T) :-
    conv_to_conv(Y,M,Y1,M1,D,D1),
    move(Y,M,Y1,M1,T),T<=max_time.              (13)

conv_on(M1,D1,T) :-
    conv_to_conv(Y,M,Y1,M1,D,D1),
    move(Y,M,Y1,M1,T),T<=max_time.              (14)
```

In addition, rules (15)-(17) define if and when the blocking actuator must be activated. Rules (18) and (19) guarantee that both the conveyor belt and the stacker crane can activate only one direction of movement at the same time.

```
-stop_on(Y,M,T) :- move(Y,M,Y2,M2,T),
            b_zone(Y,M),T<=max_time.           (15)

stop_on(Y2,M2,T) :- move(Y,M,Y2,M2,T),
             b_zone(Y2,M2),T<max_time.
                                               (16)

stop_on(Y,M,T+1) :- p_pos(Y,M,T),
    b_zone(Y,M),not move(Y,M,_,_,T+1),
    T<=max_time.                               (17)

:- conv_on(M,D1,T),conv_on(M,D2,T),D1!=D2.
                                               (18)

   :- cross_conv_on(Y,M,D1,T),D1!=D2,
         cross_conv_on(Y,M,D2,T).              (19)
```

V. Reasoning

This section shows how the ASP formalization (Sect.IV) can be exploited to perform some reasoning related to:

- possible paths between two positions and generation of the corresponding control actions (Sect.V-A);
- which are the buffer zones that can be directly reached from a given position (Sect.V-B.)

The following predicates are used to characterize the first and last position in a sequence of movements of a pallet:

- `start(Y,M)` defines that the pallet will start from position `pos(Y,M)`;
- `end(Y,M)` defines that the pallet will end at position `pos(Y,M)`;

Given the additional predicates, Rule (20) specifies the starting condition, i.e. the position of the pallet at time 0. Rule (21) is a constraint imposing to reach the end position at any time. Rule (22) terminates the generation of further movements as soon as the pallet reaches the end position.

```
p_pos(Y,M,0) :- start(Y,M).                    (20)

:- not p_pos(Y,M,_),end(Y,M).                  (21)

:- p_pos(Y,M,Q), end(Y,M), p_pos(_,_,Q+1).
                                               (22)
```

A. Path generation

The generation of a path requires as input facts the starting position `start(Y,M)`, the target position `end(Y,M)` and the constant `max_time` as the maximum number of time steps. In this way the ASP solver may potentially generate one or more paths to link the `start` and `end` position. Rule (23) can be added to introduce an objective function in the ASP program that will minimize the number of time steps

needed to reach the end position. In this case the ASP solver will return only one feasible path (if existing).

```
#minimize{Q@1: p_pos(Y,M,Q),end(Y,M)}.
```
(23)

B. Graph generation

A directed graph (or reachability graph) can be used to represent the feasible transitions of a pallet between two adjacent buffer zones, as presented in [5]. The arcs can be generated by solving the ASP program consisting of the rules in Sect.IV together with rules (24)-(27). The number of arcs coming out of the buffer zones will be equal to the number of stable models generated by the ASP solver. Rules (24) and (25) impose the start and end position to be in a buffer zone, respectively. Rule (26) guarantees that the start and end positions are different, whereas rule (27) sets the end position as soon as the pallet visits a position corresponding to a buffer zone.

```
1{start(Y,M): b_zone(Y,M)}1.
```
(24)

```
1{end(Y,M): b_zone(Y,M)}1.
```
(25)

```
:- end(Y,M),start(Y,M).
```
(26)

```
end(Y,M) :- p_pos(Y,M,T),b_zone(Y,M),
                    T>0,T<=max_time.
```
(27)

VI. Experiments

This sections demonstrates how the reasoning capabilities presented in Sect.V can be exploited when addressing a realistic modular transportation system. The ASP solver `clingo` was used to run the experiments.

A. Path generation

The generation of a path and the corresponding control sequence (cf. Sect.V-A) is tested taking in consideration the transportation system represented in Fig.3 that consists of two identical modules (`Module 1` and `Module 2`) with three positions each, a stacker crane and blocking actuators in the first and last positions. The conveyor belt and the stacker cranes allow both directions of movement.

The input facts representing the system characteristics are:

```
module(1). module(2). pos((1;2;3),1).
pos((1;2;3),2). cross((1;3),1).
cross((1;3),2). act_stop((1;3),1).
act_stop((1;3),2). conn_mod(1,1,3,3,2,3).
conn_mod(2,1,3,2,2,3).
conn_mod(3,1,3,1,2,3).
conv(1,f,b). conv(2,f,b).
cross_conv(1,1,1,r). cross_conv(3,1,1,r).
cross_conv(1,2,1,r). cross_conv(3,2,1,r).
#const max_time=5.
```

The rules elaborating the layout derive the following feasible connections:

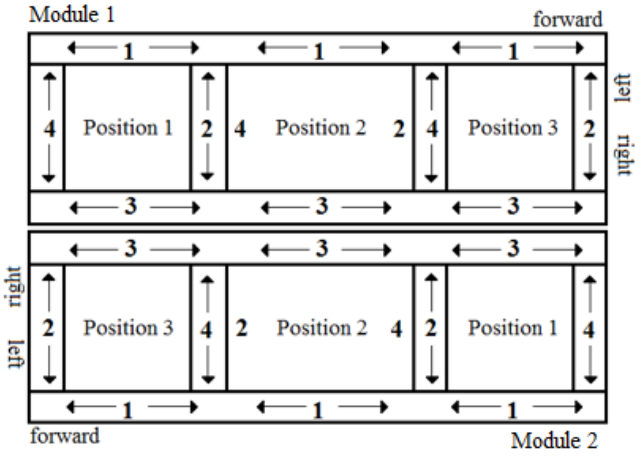

Fig. 3. Test case 1.

```
rch(1,1,2,1) rch(2,1,3,1) rch(1,2,2,2)
rch(2,2,3,2) rch(2,1,1,1) rch(3,1,2,1)
rch(2,2,1,2) rch(3,2,2,2) rch(1,1,3,2)
rch(3,1,1,2) rch(3,2,1,1) rch(1,2,3,1)
```

If the rule (23) is disabled and the start and end positions are defined as `start(1,1)` and `end(3,2)`, then the program generates six stable models:

```
Answer: 1
p_pos(1,1,0) p_pos(3,2,1) move(1,1,3,2,1)
cross_conv_on(1,1,r,1)
cross_conv_on(3,2,1,1)
Answer: 2
p_pos(1,1,0) p_pos(1,1,1) p_pos(3,2,2)
move(1,1,3,2,2) cross_conv_on(1,1,r,2)
cross_conv_on(3,2,1,2)
Answer: 3
p_pos(1,1,0) p_pos(1,1,1) p_pos(1,1,2)
p_pos(3,2,3) move(1,1,3,2,3)
cross_conv_on(1,1,r,3)
cross_conv_on(3,2,1,3)
Answer: 4
p_pos(1,1,0) p_pos(1,1,1) p_pos(1,1,2)
p_pos(1,1,3) p_pos(3,2,4)
move(1,1,3,2,4) cross_conv_on(1,1,r,4)
cross_conv_on(3,2,1,4)
Answer: 5
p_pos(1,1,0) p_pos(1,1,1) p_pos(1,1,2)
p_pos(1,1,3) p_pos(1,1,4) p_pos(3,2,5)
move(1,1,3,2,5) cross_conv_on(1,1,r,5)
cross_conv_on(3,2,1,5)
Answer: 6
p_pos(1,1,0) p_pos(2,1,1) p_pos(3,1,2)
p_pos(1,2,3) p_pos(2,2,4) p_pos(3,2,5)
move(1,1,2,1,1) move(2,1,3,1,2)
move(3,1,1,2,3) move(1,2,2,2,4)
```

```
move(2,2,3,2,5)
conv_on(1,f,1) conv_on(1,f,2)
cross_conv_on(3,1,r,3)
cross_conv_on(1,2,l,3)
conv_on(2,f,4) conv_on(2,f,5)
SATISFIABLE Models: 6, Time: 0.115s
```

`Answer 1` and `Answer 6` are the relevant solutions and are graphically represented in Fig.4 with a dashed line and a continuous line, respectively. Indeed, `Answer 2`-`Answer 5` are equivalent to `Answer 1` with a delay in the starting position. If the minimization objective in rule (23) is enabled, then the program returns `Answer 1` as the best solution.

It can be noticed that the solutions define how and when the actuators must be activated. For instance, `Answer 1` requires only `cross(1,1)` and `cross(3,2)` to be activated at `time 1` moving in direction `r` and `l`, respectively.

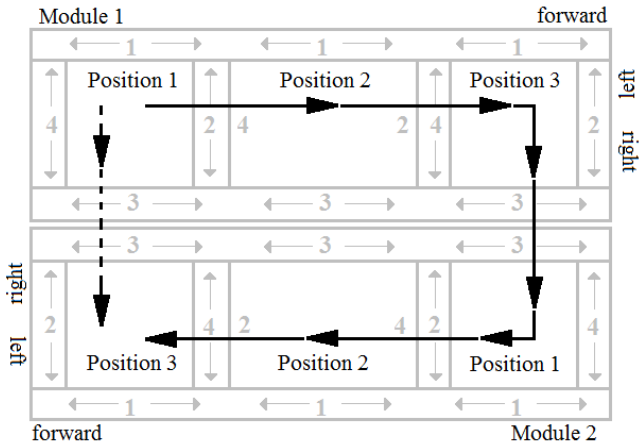

Fig. 4. Test case 1 and possible paths.

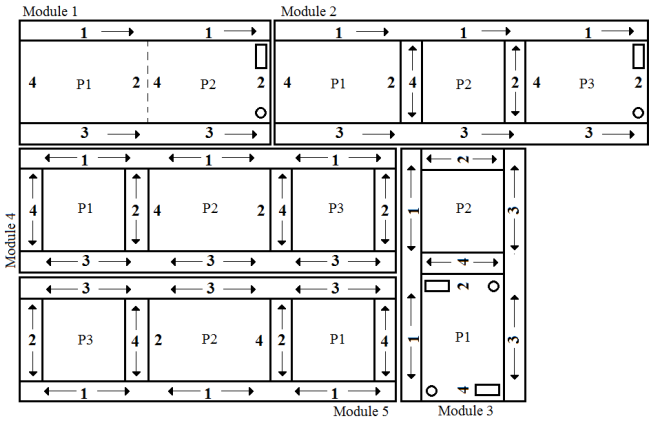

Fig. 5. Test case 2.

B. Graph generation

The generation of a graph to represent the feasible transitions of a pallet between two adjacent buffer zones (cf. Sect.V-B) is tested taking in consideration the transportation system represented in Fig.5 that consists of five different modules with two or three positions each. Given the position of the blocking actuators and the the stacker cranes, the buffer zones consist in the positions `pos(2,1)`, `pos(2,2)`, `pos(3,2)`, `pos(1,4)`, `pos(3,4)`, `pos(2,3)`, `pos(3,5)`, `pos(1,5)`, `pos(1,3)`.

The input facts representing the system characteristics are:

```
module(1).pos((1;2),1).act_stop(2,1).
conv(1,f,0).
module(2).pos((1;2;3),2).act_stop(2,2).
act_stop(3,2).cross(2,2).conv(2,f,0).
cross_conv(2,2,l,r).
module(3).pos((1;2),3).act_stop((1;2),3).
cross(2,3).conv(3,f,b).
cross_conv(2,3,l,r).
module(4).pos((1;2;3),4).cross((1;3),4).
cross_conv(1,4,l,r).cross_conv(3,4,l,r).
act_stop((1;3),4).conv(4,f,b).
module(5).pos((1;2;3),5).cross((1;3),5).
act_stop((1;3),5).conv(5,f,b).
cross_conv(1,5,l,r).cross_conv(3,5,l,r).
conn_mod(2,1,2,1,2,4).
conn_mod(2,3,2,2,2,3).
conn_mod(1,4,1,1,1,3).
conn_mod(1,4,3,3,5,3).
conn_mod(2,4,1,2,1,3).
conn_mod(2,4,3,2,5,3).
conn_mod(3,4,1,1,2,3).
conn_mod(3,4,2,2,3,1).
conn_mod(3,4,3,1,5,3).
conn_mod(1,5,4,1,3,1).
```

The graph can be generated if a program with the rules presented in Sect.V-B is run:

```
Answer: 1
start(2,3) end(1,3)
Answer: 2
start(2,3) end(3,4)
Answer: 3
start(1,5) end(3,4)
Answer: 4
start(2,2) end(3,2)
Answer: 5
start(3,4) end(2,3)
Answer: 6
start(1,3) end(2,3)
Answer: 7
start(2,2) end(2,3)
Answer: 8
start(2,3) end(2,2)
Answer: 9
```

```
start(3,4) end(1,5)
Answer: 10
start(1,4) end(3,5)
Answer: 11
start(3,5) end(1,4)
Answer: 12
start(1,5) end(3,5)
Answer: 13
start(3,4) end(1,4)
Answer: 14
start(3,5) end(1,5)
Answer: 15
start(2,1) end(2,2)
Answer: 16
start(1,4) end(3,4)
SATISFIABLE    Models: 16, Time: 0.113s
```

Since 16 stable models are obtained, it means that 16 directed arcs must be added to the reachability graph representing the considered transportation system, as shown in Fig.6. For instance, since the position `pos(2,2)` can be reached from the position `pos(2,1)`, then an arc from node `pos(2,1)` to node `pos(2,2)` is added to the graph.

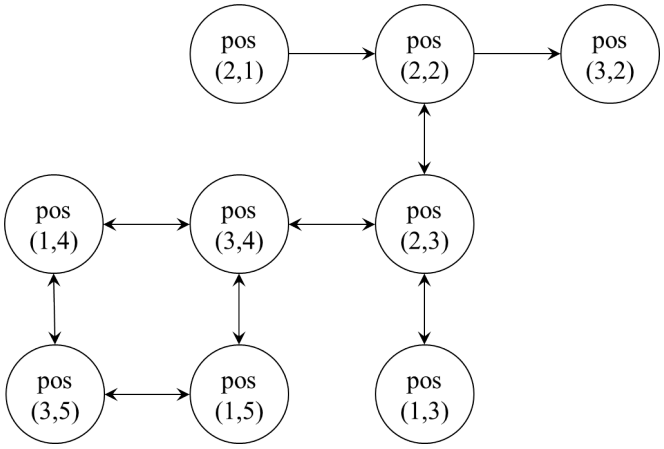

Fig. 6. Reachability graph of Test case 2.

VII. Conclusions

This paper presented an initial study showing the use of ASP to enable automatic generation of a formal model representing a modular and reconfigurable transportation systems. In addition, the reasoning capabilities of ASP can be exploited to derive system properties and control sequences. Further developments will address:

- an extension to the multi-pallet case. This will require to modify and the rules in Sect.IV-C to specify the position of each pallet. Moreover, it will be needed to constrain that in a position there can only one pallet.
- the addition of rules to support the applications (iii) and (iv) defined in Sect.I
- testing the programs on problems of larger size. The rules have already been tested on the plant presented in [18] showing an acceptable performance.
- the integration with a semantic representation of the system for an automatic generation of the input facts and the results of the reasoning to better support the interoperability with other tools [19], [20].
- a more extended comparison between ASP and Petri Nets. Moreover, given the input facts characterizing a system (cf. Sect.IV-A), then the rigid knowledge (cf. Sect.IV-B) with additional rules can be actually used to automatically generate a formal model of the system as a Petri Net. This would be the reciprocal of what developed by Anwar et al. [15] and would pave the way to a synergistic use of ASP and Petri Nets.

Acknowledgment

The authors thank Dr. Andrea Cataldo for his support in the definition of the problem statement and the formalization and analysis of the case studies.

References

[1] Y. Koren, U. Heisel, F. Jovane, T. Moriwaki, G. Pritschow, G. Ulsoy, and H. V. Brussel, "Reconfigurable manufacturing systems," *CIRP Annals - Manufacturing Technology*, vol. 48, no. 2, pp. 527–540, 1999.

[2] A. Gola and A. Swic, "Reconfigurable manufacturing systems as a way of long-term economic capacity management," *Actual Problems of Economics*, vol. 166, no. 4, pp. 15–22, 2015. cited By 0.

[3] E. Carpanzano, A. Cesta, A. Orlandini, R. Rasconi, and A. Valente, "Intelligent dynamic part routing policies in plug&produce reconfigurable transportation systems," *CIRP Annals - Manufacturing Technology*, vol. 63, no. 1, pp. 425–428, 2014.

[4] S. Haneyah, J. Schutten, P. Schuur, and W. Zijm, "Generic planning and control of automated material handling systems: Practical requirements versus existing theory," *Computers in Industry*, vol. 64, no. 3, pp. 177 – 190, 2013.

[5] A. Cataldo and R. Scattolini, "Modeling and model predictive control of a de-manufacturing plant," in *2014 IEEE Conference on Control Applications (CCA)*, pp. 1855–1860, Oct 2014.

[6] I. Hegny, O. Hummer, A. Zoitl, G. Koppensteiner, and M. Merdan, "Integrating software agents and iec 61499 realtime control for reconfigurable distributed manufacturing systems," in *2008 International Symposium on Industrial Embedded Systems*, pp. 249–252, June 2008.

[7] C. A. Petri, *Kommunikation mit Automaten*. PhD thesis, UniversitÃd't Hamburg, 1962.

[8] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proceedings of International Logic Programming Conference and Symposium* (R. Kowalski, Bowen, and Kenneth, eds.), pp. 1070–1080, MIT Press, 1988.

[9] V. Lifschitz, "What is answer set programming?," in *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pp. 1594–1597, AAAI Press, 2008.

[10] G. Brewka, T. Eiter, and M. Truszczyński, "Answer set programming at a glance," *Commun. ACM*, vol. 54, pp. 92–103, Dec. 2011.

[11] V. Lifschitz, "Answer set programming and plan generation," *Artificial Intelligence*, vol. 138, no. 1, pp. 39 – 54, 2002.

[12] E. Aker, V. Patoglu, and E. Erdem, "Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics," *IFAC Proceedings Volumes*, vol. 45, no. 22, pp. 77 – 83, 2012.

[13] J. J. Portillo, C. L. Garcia-Mata, P. R. Márquez-Gutiérrez, and R. Baray-Arana, "Robot platform motion planning using answer set programming," in *LA-NMR*, 2011.

[14] F. Yang, P. Khandelwal, M. Leonetti, and P. Stone, "Planning in answer set programming while learning action costs for mobile robots," in *AAAI Spring 2014 Symposium on Knowledge Representation and Reasoning in Robotics (AAAI-SSS)*, March 2014.

[15] S. Anwar, C. Baral, and K. Inoue, "Encoding petri nets in answer set programming for simulation based reasoning," *CoRR*, vol. abs/1306.3542, 2013.

[16] S. Anwar, C. Baral, and K. Inoue, *Encoding Higher Level Extensions of Petri Nets in Answer Set Programming*, pp. 116–121. Springer Berlin Heidelberg, 2013.

[17] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele, "Potassco User Guide," 2015. Available online: http://potassco.sourceforge.net (Last accessed on 11 September 2017).

[18] A. Cataldo, R. Scattolini, and T. Tolio, "An energy consumption evaluation methodology for a manufacturing plant," *{CIRP} Journal of Manufacturing Science and Technology*, vol. 11, pp. 53 – 61, 2015.

[19] M. R. Blackburn and P. O. Denno, "Using semantic web technologies for integrating domain specific modeling and analytical tools," *Procedia Computer Science*, vol. 61, pp. 141 – 146, 2015. Complex Adaptive Systems San Jose, CA November 2-4, 2015.

[20] W. Terkaj, T. Tolio, and M. Urgo, "A virtual factory approach for in situ simulation to support production and maintenance planning," *{CIRP} Annals - Manufacturing Technology*, vol. 64, no. 1, pp. 451 – 454, 2015.