Information Systems Frontiers

Synthesis and Execution of Optimal Controllers for Multi-Robot Systems in Logistics

--Manuscript Draft--

Manuscript Number:	ISFI-D-17-00377
Full Title:	Synthesis and Execution of Optimal Controllers for Multi-Robot Systems in Logistics
Article Type:	SI: Recent Trends in Reuse and Integration
Keywords:	optimization modulo theories; robotics; high-level controllers; logistics
Corresponding Author:	Francesco Leofante Universita degli Studi di Genova and RWTH Aachen University ITALY
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Universita degli Studi di Genova and RWTH Aachen University
Corresponding Author's Secondary Institution:	
First Author:	Francesco Leofante
First Author Secondary Information:	
Order of Authors:	Francesco Leofante
	Erika Ábrahám
	Tim Niemueller
	Gerhard Lakemeyer
	Armando Tacchella
Order of Authors Secondary Information:	
Funding Information:	
Abstract:	Model-based synthesis allows to generate controllers that achieve high-level tasks while satisfying certain properties of interest. However, when controllers are executed on concrete systems, several modeling assumptions may be challenged, jeopardizing their real applicability. This paper presents an integrated system for generating, executing and monitoring optimal-by-construction controllers for multi-robot systems. This system unites the power of Optimization Modulo Theories with the flexibility of an on-line executive, providing optimal controllers for high-level task planning, and runtime feedback on their feasibility. After presenting how our system orchestrates static and runtime components, we demonstrate its capabilities using the RoboCup Logistics League as testbed. We do not only present our final solution but also its chronological development, and draw some general observations for the development of OMT-based approaches.

Noname manuscript No.
(will be inserted by the editor)

Synthesis and Execution of Optimal Controllers for Multi-Robot Systems in Logistics

Francesco Leofante[1,3†] · Erika Ábrahám[1] · Tim Niemueller[2] · Gerhard Lakemeyer[2] · Armando Tacchella[3]

the date of receipt and acceptance should be inserted later

Abstract Model-based synthesis allows to generate controllers that achieve high-level tasks while satisfying certain properties of interest. However, when controllers are executed on concrete systems, several modeling assumptions may be challenged, jeopardizing their real applicability. This paper presents an integrated system for generating, executing and monitoring optimal-by-construction controllers for multi-robot systems. This system unites the power of Optimization Modulo Theories with the flexibility of an on-line executive, providing optimal controllers for high-level task planning, and runtime feedback on their feasibility. After presenting how our system orchestrates static and runtime components, we demonstrate its capabilities using the RoboCup Logistics League as testbed. We do not only present our final solution but also its chronological development, and draw some general observations for the development of OMT-based approaches.

1 Introduction

With the advent of Industry 4.0, factories are moving from static process chains towards the introduction of *autonomous robots* in their *production* lines. As the abilities and the complexity of such systems increase, the problem of managing and optimizing the in-factory supply chain carried out by (fleets of) autonomous robots becomes crucial. This paradigm shift also opens up a number of new research challenges for the AI community. A significant challenge is to create methods to generate *controllers*[1] that can achieve high-level tasks while satisfying properties of interest.

The *RoboCup Logistics League* (*RCLL*) (Niemueller et al, 2015) has been proposed as a realistic testbed to study the above mentioned problems at a manageable scale. There, groups of robots need to maintain and optimize the material flow according to dynamic orders in a simplified factory environment.

Though there exist successful heuristic methods to solve the underlying planning and scheduling problem, *e.g.*, (Hofmann et al, 2016; Niemueller et al, 2013), a major disadvantage of these methods is that they provide *no guarantees* about the quality of the solution, as observed in (Bensalem et al, 2014). A promising solution to this issue is offered by the recently emerging field of *Optimization Modulo Theories* (*OMT*), where Satisfiability Modulo Theories (SMT) solving is extended with optimization capabilities – see, *e.g.*, (Nieuwenhuis and Oliveras, 2006; Sebastiani and Tomasi, 2015; Sebastiani and Trentin, 2015b).

In this paper we present an integrated system for generating, executing and monitoring optimal-by-construction controllers. We employ OMT to synthesize high-level controllers with optimality guarantees for multi-robot systems within the RCLL scope. Furthermore, we integrate our approach into an on-line execution and monitoring system based on `CLIPS` (Wygant, 1989), a rule-based production system using forward chaining inference. The system presented here extends the

† Corresponding author.
[1] Theory of Hybrid Systems
RWTH Aachen University, Germany
`{leofante,abraham}@cs.rwth-aachen.de`
[2] Knowledge-Based Systems
RWTH Aachen University, Germany
`{niemueller,gerhard}@cs.rwth-aachen.de`
[3] Università degli Studi di Genova, Italy
`armando.tacchella@unige.it`

[1] We will use the words (high-level) *controller* and *plan* interchangeably throughout this paper.

results obtained in (Leofante et al, 2017), where we showed how controllers could be synthesized for the *exploration phase*, during which robots roam the game arena and determine where the team's own machines are positioned. By rigorous experimental analysis we showed that naive encodings fail to cope with the complexity of the domain and we then detailed our findings and solutions adopted to overcome previous limitations.

This work extends our previous research in several directions:

– We shift our attention towards the *production phase*, where robots receive orders dynamically and cooperate to deliver finished products within fixed deadlines. With similar methods as used in (Leofante et al, 2017), we encode the underlying planning and scheduling problem as a Boolean combination of linear constraints over the reals, and compute optimal controllers by using OMT solvers such as `Z3` (Bjørner et al, 2015), `SMT-RAT` (Corzilius et al, 2015) and `OptiMathsat` (Sebastiani and Trentin, 2015a).
– We present an *integrated* system that unites OMT synthesis with online execution and monitoring based on `CLIPS`. We describe how our architecture manages the integration between controller synthesis and online execution, the data structures involved and solutions achieved to obtain seamless integration.
– We detail the recent achievements obtained regarding the *execution* of intrinsically concurrent controllers. We extend previous work by adding to our system synchronization mechanisms to manage multiple robots.
– We extend our comparison with heuristic approaches by considering both *anytime* and *oneshot* planning. This allows us to check whether any improvement can be expected if a planner can use additional time to further optimize the first-found feasible solution.
– We show how our architecture can be extended to provide *explanations* about the decision-making process underlying our synthesis procedure in a human-readable fashion. We elucidate specific features of OMT that have the potential to facilitate such explanations, and provide illustrative results along this direction.

After presenting some theoretical preliminaries in Section 2, we describe the salient features of a game in the RoboCup Logistics League in Section 3. In Section 4 we provide a general formalization of our approach together with insights on the integration of our approach into an on-line execution and monitoring system based on `CLIPS`. Our solution and simulation-based experimental evaluations for the exploration and production phases are presented in Sections 5 and 6 respectively. Section 7 introduces the problem of generating human-readable explanations for controllers generated with OMT, together with some preliminary results. Finally, we draw some general conclusions and discuss future directions of research in Sections 8 and 9.

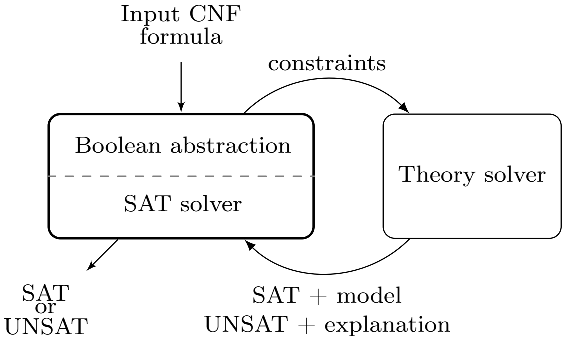


Fig. 1: The SMT solving framework.

2 Preliminaries

2.1 Mixed-integer arithmetic

Problems considered in this work are encoded as *mixed-integer arithmetic* formulas. Syntactically, arithmetic *terms* are constant symbols, variables, and sums, differences or products of terms. Arithmetic *constraints* compare two arithmetic terms using $<$, $\leq$, $=$, $\geq$ or $>$. Quantifier-free arithmetic *formulas* use conjunction $\wedge$ and negation $\neg$ (and further syntactic sugar like disjunction $\vee$ or implication $\implies$) to combine theory constraints. A formula in *conjunctive normal form* (*CNF*) is a conjunction of disjunctions of theory constraints or negated theory constraints (see Eq. (1) for a simple example formula in CNF). An arithmetic formula is called *linear* if it does not contain any multiplication, and *non-linear* otherwise. Semantically, each variable is interpreted over its domain – either the reals $\mathbb{R}$ or the integers $\mathbb{Z}$ – by an *assignment*, assigning to each variable a value from its domain; we use the standard semantics to evaluate formulas.

2.2 Satisfiability modulo theories

Satisfiability Modulo Theories (*SMT*) solving aims at deciding the satisfiability of (usually quantifier-free) first-order logic formulas over some theories like, *e.g.*, the theories of lists, arrays, bit vectors, real or (mixed-)integer arithmetic. To decide the satisfiability of an input formula φ in CNF, SMT solvers proceed as depicted in Fig. 1. Typically, a *Boolean abstraction* $abs(\varphi)$ of φ

is built first by replacing each constraint by a fresh Boolean variable (proposition), *e.g.*,

$$\begin{array}{rcccccccccl} \varphi & = x \geq y & \wedge & (& y > 0 & \vee & x > 0 &) & \wedge & y \leq 0 \\ & & & & & \downarrow & & & & \\ \mathit{abs}(\varphi) & = \quad A & \wedge & (& B & \vee & C &) & \wedge & \neg B \end{array} \tag{1}$$

where $x, y \in \mathbb{R}$, and $A, B, C \in \mathbb{B} = \{0, 1\}$.

A *Boolean satisfiability* (*SAT*) *solver* searches for a satisfying assignment S for $\mathit{abs}(\varphi)$, *e.g.*, $S(A) = 1$, $S(B) = 0$, $S(C) = 1$ for the above example. If no such assignment exists then the input formula φ is unsatisfiable. Otherwise, the consistency of the assignment in the underlying theory is checked by a *theory solver*. In our example, we check whether the set $\{x \geq y,\ y \leq 0,\ x > 0\}$ of linear inequalities is feasible, which is the case. If the constraints are theory-consistent then a satisfying solution (*model*) is found for φ. Otherwise, the theory solver returns a theory lemma φ_E giving an *explanation* for the conflict, *e.g.*, the negated conjunction of some inconsistent input constraints. The explanation is used to refine the Boolean abstraction $\mathit{abs}(\varphi)$ to $\mathit{abs}(\varphi) \wedge \mathit{abs}(\varphi_E)$. These steps are iteratively executed until either a theory-consistent Boolean assignment is found, or no more Boolean satisfying assignments exist.

2.3 Optimization modulo theories

Optimization Modulo Theories (*OMT*) – see for example (Nieuwenhuis and Oliveras, 2006; Cimatti et al, 2010) and (Bjørner et al, 2015; Corzilius et al, 2015; Sebastiani and Trentin, 2015a,b) for related solvers – extends SMT solving with optimization procedures to find a variable assignment that defines an optimal value for an objective function f (or a combination of multiple objective functions) under all models of a formula φ. As noted in (Sebastiani and Tomasi, 2015), most OMT solvers implement a *linear-search* scheme, which can be summarized as follows. Let φ_S be the conjunction of all theory constraints that are true under a satisfying assignment S and the negation of those that are false under S. A local optimum μ for f can be computed under the side condition φ_S, and φ is updated as

$$\varphi := \varphi \wedge (f \bowtie \mu) \wedge \neg\varphi_S \quad , \quad \bowtie \in \{<, >\}$$

This forces the solver to find a new assignment under which the value of the objective function improves, while discarding all previously found assignments. Repeating this procedure until the formula becomes unsatisfiable will lead to an assignment optimizing f under all models of φ.

2.4 Controller synthesis and SMT

SMT solvers are nowadays embedded as core engines in a wide range of technologies – see *e.g.* (Ábrahám and Kremer, 2016) for some examples. In the area of robotics several interesting applications of SMT can be found. For instance, (Nedunuri et al, 2014) and (Wang et al, 2016) use SMT solving to generate task and motion plans starting from a static roadmap, employing plan outlines to guide the synthesis process. The authors of (Dantam et al, 2016) perform task and motion planning leveraging incremental solving in `Z3` to update constraints about motion feasibility. The work presented in (Saha et al, 2014) defines a motion planning framework where SMT solving is used to combine motion primitives so that they satisfy some linear temporal logic (LTL) requirements. In (Cashmore et al, 2016) the authors present a framework to translate planning languages into SMT encodings.

In contrast to the above works, (*i*) we do not use prior knowledge (*e.g.*, motion primitives, plan outlines) to seed the search performed by the solver and (*ii*) we exploit OMT solving to synthesize controllers that are not only *feasible* but also *optimal*.

2.5 Online execution

The *online execution* of plans is a very intricate problem. When controllers are executed on concrete systems, several modeling assumptions may be challenged, jeopardizing their real applicability. Examples of such challenges may be slack during execution, or uncertainty, *e.g.*, in travel times due to other agents in the environment. In the multi-robot case, issues of synchronization and mutual exclusion may be relevant.

Several execution systems have been proposed in the past. Most executives mentioned in (Verma et al, 2005b) are associated with a specific modeling language. For example, the Universal Executive (Verma et al, 2006) is a general processor for the PLEXIL (Verma et al, 2005a) language. It allows to describe the execution flow as a number of hierarchically structured nodes consisting of a set of conditions when to execute and a body that describes what to execute. The Universal Executive then ties these descriptions with interfaces to actual actors. While PLEXIL is more of a control language, Procedural Reasoning Systems (Ingrand et al, 1996) lean more towards a knowledge-based representation with an explicit fact base, a notion of goals to achieve or maintain, and activation conditions for procedures. An advantage here is a less constrained execution flow, however, this gain in expressiveness may

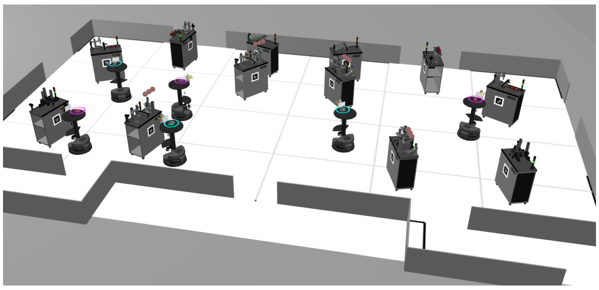


Fig. 2: RCLL factory environment as seen in the simulator (Zwilling et al, 2014).

easily come with unintended execution orders without the required caution.

A more recent system integrating planning and execution is `ROSPlan` (Cashmore et al, 2015). It provides a general framework for execution where the individual components can be exchanged (with a varying degree of effort). One of the available dispatchers uses a representation of the plan as an Esterel (Berry and Gonthier, 1992) program. There, a plan is described as a set of modules interconnected with signals and receiver slots. However, at this point the translation and execution is opaque and no influence can be exerted on the formulation of the program. There is currently only a limited form of concurrency available. A slightly different approach that has been compared to Esterel is `RMPL` (Ingham et al, 2001). Instead of a signal flow, it models the flow more as an evolution of states. Both provide primitives for sequential or parallel execution of code blocks, and conditionals.

An earlier system to provide an extensible planning system based on the *Planning Domain Definition Language* (*PDDL*) (McDermott et al, 1998) was `TFD/M` with semantic attachments (Dornhege et al, 2009). However, the executive was a C++ program which had to be augmented each time for the respective available actions and did not provide a flexible specification language. A more unified approach was recently taken through integrating continual planning in Golog (Hofmann et al, 2016). The overall domain model and execution specifics are encoded in Golog. For planning (sub-)problems the model is translated into PDDL and a planner is called. The specification contains assertions to deal with incomplete knowledge and improve planning efficiency. However, the modeling in Golog can be somewhat tedious and it is often deeply intertwined with its Prolog implementation.

In this work, we propose a new formulation of the execution as a rule-based system. With the experience of modeling the decision making, multi-robot coordination, and task execution for the RoboCup Logistics League (Niemueller et al, 2016b), we intend to generalize the framework to be applicable with various planning, reasoning, and decision making components. This decoupling between synthesis and execution comes at the cost of having to link two separate models in a consistent way. However, it provides a great flexibility for the executive to choose the appropriate planning system and to add domain-specific interpretations of the plan easily.

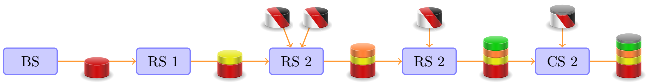


Fig. 3: Example of order configuration for the competition (Niemueller et al, 2015; RCLL Technical Committee, 2017).

3 The RoboCup Logistics League

The example domain chosen for evaluating our approach is based on the *Planning and Execution Competition for Logistics Robots in Simulation*[2] (Niemueller et al, 2016a), which provides a simulated Smart Factory environment shown in Fig. 2. There, two teams of three autonomous robot agents each compete to handle the logistics of materials through several dynamic stages to produce final goods according to a dynamic order schedule known only at run-time. Each game sees two teams competing against each other during two phases, the *exploration* and the *production* phase.

In the exploration phase, robots must roam the environment and determine where the team's own machines are positioned. For this, the playing field is divided into 24 virtual zones, of which 12 belong to each of the two teams (operating at the same time in the environment increasing execution duration uncertainty considerably). However, only 6 of these zones will contain machines. Therefore, the task is to efficiently assign the three robots to the 12 zones and identify the zones which contain a machine.

In the production phase instead, robots have to handle the logistics of materials through several (dynamic) stages to produce final goods to fulfill orders. Products to be assembled have different complexities and usually require a base, 0 to 4 rings to me mounted on top of it, and a cap as a finishing touch. To increase complexity, orders not only fix the components to be used, but also specify colors to be used, and in what order. Bases

[2] `http://www.robocup-logistics.org/sim-comp`

are available in three different colors, four colors are admissible for rings and two for caps.

Several machines that perform different processing steps are scattered around the factory shop floor. Each of them completes a particular production step such as providing bases, mounting colored rings or caps. Based on such differences, it is possible to distinguish four types of machines:

- Base Station (BS): acts as dispenser of base elements. There is one single BS per team.
- Cap Station (CS): mounts a cap as the final step in production on an intermediate product. CS have a slide to store at most one cap piece at a time. At the beginning of the game this slide is empty and has to be filled as follows. A base element with a cap must be taken from a shelf in the game arena and fed to the machine; the cap is then unmounted and buffered in the slide. The cap can then be mounted on the next intermediate product taken to the machine. There are two CS per team.
- Ring Station (RS): mounts one colored ring (of a specific color) onto an intermediate product. Some ring colors require additional tokens to be "unlocked": robots will have to feed a RS with a specified number of bases before the required color can be mounted. There are two RS per team.
- Delivery Station (DS): accepts finished products. A DS contains three conveyor belts, robots have to prepare the proper one as per specific order. There is one DS per team.

The challenge for autonomous robots is then to transport intermediate products between processing machines and optimize a multistage production cycle of different product variants until delivery of final products. A sample production trace is shown in Fig. 3.

Orders that denote the products which must be assembled with these operations are posted at run-time by an automated referee box (RefBox) broadcasting information via Wi-Fi and therefore require quick planning and scheduling. Orders come with a delivery time window introducing a temporal component into the problem.

4 System Overview

The system we describe in this work unites the power of Optimization Modulo Theories with the flexibility of an on-line executive, providing optimal controllers for high-level task planning, and runtime feedback on their feasibility. The proposed architecture is depicted in Figure 4. The `CLIPS` executive controls the overall execution and, when needed, triggers the OMT module to synthesize a controller. To start synthesizing, the world model, with all relevant information, must be encoded in a way accessible to the OMT solver. We have opted for Google Protocol Buffers (`protobuf`) to handle communications to and from the OMT solving module. Once a controller is computed, `CLIPS` retrieves it and distributes it to the robots for execution. Robots then execute their respective partial plans by invoking the appropriate basic behaviors through the behavioral and functional components of the Fawkes[3] software framework (for instance, BE in Fig. 4 represents the Lua-based Behavior Engine (Niemueller et al, 2009) that provides the basic skills to execute the controllers). Only through this framework does the reasoning system interact with the simulation.

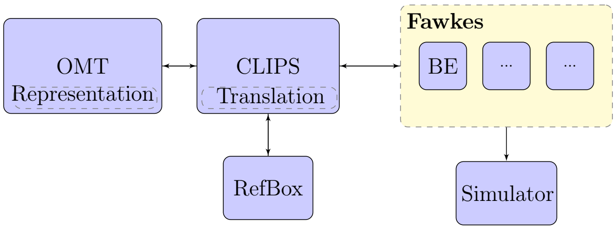


Fig. 4: The overall architecture.

Several challenges can arise during execution, as original modeling assumptions might not hold in the real system due to, *e.g.*, action failure, plan failure due to ignorance or change in a dynamic environment. If this happens, controllers might become inconsistent and lead to undesired behaviors. In our framework, we rely on the interplay between the controller synthesis module and the on-line executive to tackle this problem. Once controllers have been synthesized, `CLIPS` automatically starts the appropriate tasks. Updates on execution (*e.g.*, if a certain task is currently in progress, task failures) are always distributed in the world model, therefore the executive is constantly informed about execution progress. When inconsistencies with the model are detected, the executive can ask for a new controller, and our encoding allows to compute this starting from any arbitrary initial world state.

In the following, we describe the main components of our system and show how they operate together in our pipeline.

4.1 Optimal controllers with OMT

The approach used is based on symbolic reachability techniques used to solve the SMT-based bounded model-

[3] Fawkes is a component-based software framework for robotic real-time applications. URL: www.fawkesrobotics.org

checking problem, which we extended to OMT – see (Biere et al, 1999) for the original formulation.

States are described using an ordered set of real-valued *variables* $x = \{x_1, \ldots, x_n\}$; we also use the vector notation $x = (x_1, \ldots, x_n)$ and write x' and x_i for $(x'_1, \ldots, x'_n)$ and $(x_{1,i}, \ldots, x_{n,i})$ respectively. There is a special variable $A \in x$ which encodes the *action* to be executed next, and a variable $rew \in x$ which encodes the *reward* achieved when executing action A in the current state. A *state* $s{=}(v_1, \ldots, v_n) \in \mathbb{R}^n$ specifies a real value $v_i \in \mathbb{R}$ for each variable $x_i \in x$.

The RCLL system is represented symbolically by mixed-integer arithmetic formulas defining the *initial states* $I(x)$, the *transition relation* $T(x, x')$ (where x describes the state before the transition and x' the state after it) and a set of *final* states $F(x)$. *Executions* (*paths*) of length p are sequences $s_0, \ldots, s_p$ of states such that $I(s_0)$ and $T(s_i, s_{i+1})$ hold for all $i = 1, \ldots, p-1$. Thus, paths are solutions for the formula:

$$I(s_0) \wedge \bigwedge_{0 \leq i < p} T(s_i, s_{i+1}) \tag{2}$$

The total reward rew_{tot} associated to such a path is specified by:

$$rew_{tot} = \sum_{0 \leq i < p} rew_i \tag{3}$$

The *optimal bounded synthesis problem*, defined by a tuple $OBSR{=}(I, T, F, rew_{tot}, p)$, poses the problem to find a path of length p that reaches a target state and achieves thereby the highest possible reward, *i.e.*, to maximize rew_{tot} under the side condition:

$$\begin{aligned} & I(x_0) \wedge \left(\bigwedge_{0 \leq i < p} T(x_i, x_{i+1}) \right) \wedge \left(\bigvee_{0 \leq i \leq p} F(x_i) \right) \wedge \\ & rew_{tot} = \sum_{0 \leq i < p} rew_i \end{aligned} \tag{4}$$

4.2 CLIPS rules engine

CLIPS (Wygant, 1989) is a rule-based production system using forward chaining inference based on the Rete algorithm (Forgy, 1982). CLIPS consists of three building blocks (Giarratano, 2007): a *fact base*, a *knowledge base* and an *inference engine*.

The fact base can be seen as a global memory where data is stored in the form of *facts*, high-level statements that encode pieces of information about the world state. The knowledge base instead, is used to represent knowledge. More specifically, CLIPS provides heuristic and procedural paradigms for representing knowledge in the form of *rules* and *functions* respectively.

```
1  (defrule production-call-clips-omt
2  (phase EXPLORATION)
3  (team-color ?team-color&CYAN|MAGENTA)
4  (state IDLE)
5  (not (plan-requested))
6  (test (eq ?*ROBOT-NAME* "R-1"))
7  =>
8  (bind ?p
9  (omt-create-data
10 (omt-create-robots ?team-color)
11 (omt-create-machines ?team-color)
12 (omt-create-orders ?team-color)
13 )
14 )
15 (omt-request "explore-zones" ?p)
16 (assert (plan-requested))
17 )
```

Listing 1: CLIPS rule to trigger synthesis.

Rules specify heuristics to decide which actions to perform in what situations. An example of a CLIPS rule is shown in Listing 1. Formally, rules are composed of an *antecedent* and a *consequent*. The antecedent is defined as a set of conditions expressed over facts (ll. 2–6), while the consequent consists of a set of *actions* to be performed (ll. 8–16) when the rule is applicable. Actions in CLIPS are represented by functions (ll. 8–14, omt-create-* calls are functions), pieces of executable code which can return values or perform side-effects (*e.g.*, interact with the low-level control layer for robots).

The inference engine is the mechanism that CLIPS provides to control the overall execution of rules. At system initialization, the inference engine is instructed to begin execution of applicable rules. To determine whether a rule is applicable, the inference engine checks for each rule in the knowledge base whether their antecedent is met by the facts initially asserted in the fact base.

If all conditions specified in the antecedent of a rule are satisfied then the rule is activated and added to the execution agenda. If more than one rule is applicable, the inference engine uses a *conflict resolution* strategy to select which rule should have its actions executed. The actions of the selected rule are executed (which may affect the list of applicable rules) and then the inference engine selects another rule and executes its actions. This process continues until no applicable rules remain.

4.3 Communication infrastructure

For controller synthesis, the world model, with all relevant information, must be encoded in a way accessible to the solver. In this work, we have used Google Protocol Buffers[4] (`protobuf`) to encode the world state when synthesis is triggered, as well as the resulting controller. Protocol buffers define a language-independent mechanism for serializing structured data. To use them, one needs to specify the structure of the data to be serialized (*i.e.*, specify the data type). Once this is done, the protocol buffer compiler needs to be run to automatically generate data access classes in the language of interests – C++ in our case. Protobuf buffers provide a convenient transport, exchange, and storage representation that is easy to create and read. They also have powerful introspection capabilities which are particularly useful for generic access from reasoning systems. For example, the `CLIPS`-based access requires only the message definition files and not any pre-generated code. We use the exploration problem as a working example to show the interaction between the solving module and the `CLIPS` agent. The rule to trigger the synthesis process is shown in Listing 1. Once the game is started (ll. 2–4), the first robot (l. 6) will create the data structure with the relevant information and request a plan from the OMT solver (l. 15).

The OMT side notifies the executive once a solution is ready for retrieval. An excerpt of the message specifications for plan representation is shown in Listing 2. First, a list of actor (robot) specific plans is defined in ll. 1–3, where the keyword `repeated` specifies that the field may be repeated multiple times. Each plan (ll. 4–11) requires the actor for the plan to be defined (`required` keyword) and either a sequential or a temporal plan (`oneof` keyword). In this example, we show how a message for sequential plans is defined (ll. 12–14). A sequential plan simply consists of a series of actions (ll. 15–18), each of which is defined by a name and parameterized by a number of key-value pairs (ll. 19–22). Listing 3 shows a concrete example of a plan for two robots – `"R-1"` and `"R-2"` – with two `"move"` action commands.

```
 1 message ActorGroupPlan {
 2 repeated ActorSpecificPlan plans = 1;
 3 }
 4 message ActorSpecificPlan {
 5 required string  actor_name = 1;
 6
 7 oneof plan {
 8 SequentialPlan sequential_plan = 2;
 9 TemporalPlan   temporal_plan = 3;
10 }
11 }
12 message SequentialPlan {
13 repeated PlanAction actions = 1;
14 }
15 message PlanAction {
16 required string name = 1;
17 repeated PlanActionParameter params = 2;
18 }
19 message PlanActionParameter {
20 required string key = 1;
21 required string value = 2;
22 }
```

Listing 2: Plan data type specfication in protobuf. Each field requires a numerical tag, that identifies the field in the binary encoding.

4.4 Execution and monitoring

Once a plan has been retrieved, it must be translated into a native `CLIPS` representation. Each action specified by the OMT module (see Listing 3) is added to the fact base by means of facts which identify tasks and steps to be executed on the `CLIPS` side. Rules are defined to process such tasks and steps, defining the actions to be executed. Listing 4 shows an example of such translation for Listing 3, ll. 1–19. First, a task fact is added (ll. 1–2) to specify robot actor and steps to be executed. Step facts are specified in ll. 2–8, where more details about the low-level robot actions needed are added.

In our system, we rely on the communication infrastructure used to share world model updates among the robots to. This encapsulates fact base updates in protobuf messages and broadcasts them to the other robots. A (dynamically elected) master generates a consistent view and distributes it to the robots. On each robot, the `CLIPS` executive has rules that automatically start tasks when applicable. Basic behaviors in our framework are provided by a Lua-based Behavior Engine, but could in principle be provided by other sources. A step in a task is executed by triggering the execution of an asynchronous durative procedure. Then, information about the execution of the state is read and asserted in the fact base. Updates on task execution (*e.g.*, whether a task is currently in progress) are distributed in the world model, making sure that the on-line executive is informed about the status of execution.

During execution, the modeling assumptions may be challenged and, in general, actions may fail or produce an unexpected result. For instance, an object might be misplaced, or slack during execution could make a plan invalid, for example if a specified deadline cannot be met. As explained above, steps of a task are triggered non-blocking, *i.e.*, rule evaluation continues normally. This can be used to implement execution monitoring, where rules can be defined to identify situations where a step should be skipped or a task be aborted.

[4] `https://developers.google.com/protocol-buffers/`

```
plans[0]:ActorSpecificPlan {
actor_name: "R-1"
sequential_plan:SequentialPlan {
actions[0]:PlanAction {
name: "move"
params[0]:PlanActionParameter {
key: "to"
value: "C-BS-I"
}
}
actions[1]:PlanAction {
name: "move"
params[0]:PlanActionParameter {
key: "to"
value: "C-DS-I"
}
}
}
}
plans[1]:ActorSpecificPlan {
actor_name: "R-2"
sequential_plan:SequentialPlan {
actions[0]:PlanAction {
name: "move"
params[0]:PlanActionParameter {
key: "to"
value: "C-CS1-I"
}
}
actions[1]:PlanAction {
name: "move"
params[0]:PlanActionParameter {
key: "to"
value: "C-RS2-I"
}
}
}
}
```

Listing 3: Plan represented through the messages from Listing 2 (shown in augmented JavaScript Object Notation).

```
(task (task-id 1910) (robot "R-1") (name
    explore)
(state proposed) (steps 1911 1912))
(step (id 1911) (name drive-to) (state
    inactive)
(machine C-BS) (side INPUT)
(sync-id (next-sync-id)))
(step (id 1912) (name drive-to) (state
    inactive)
(machine C-DS) (side INPUT)
(sync-id (next-sync-id)))
```

Listing 4: Task representation in CLIPS.

5 Exploration Phase

In this section we show how to construct controllers for the exploration phase of a game in RCLL. Although exploration does not play a major role in determining the outcome of a competition, we decided to start with this phase because of the easy formulation of the underlying problem. As explained in Section 3, in the exploration phase the robots must roam the environment and determine where the team's own machines are positioned. Each team is assigned 12 virtual zones to explore, out of which only 6 contain machines. However, even though the problem formulation looks simple, computing an optimal solution (in terms of fastest execution) proved to be challenging: optimal exploration is a variant of the multiple traveling salesman problem, which is known to be NP-hard. As we learned, the combinatorial nature of this problem poses a great challenge to the OMT solver: naive encodings fail to cope with the complexity of the domain.

This section, based on the work presented in (Leofante et al, 2017), shows the chronological development of our synthesis approach. We examine how different design choices can affect the solving process and draw general observations which we then applied when encoding the production problem of Section 6.

The experimental analysis presented here has been carried out using the Z3 solver[5] (Bjørner et al, 2015). Though most of the encodings we present in the following generate linear arithmetic problems, due to the Boolean stucture of these formulas we could not use any linear programming tools. We considered also the OMT solvers SMT-RAT (Corzilius et al, 2015) and OptiMathsat (Sebastiani and Trentin, 2015a). The latter specializes in optimization for real arithmetic problems, whereas SMT-RAT is tuned for the satisfiability check of non-linear real arithmetic formulas. However, the nature of our problems rather requires combinatorial optimization at the Boolean level and therefore the strengths of these two solvers could not be exploited to their fullest. Z3 was the tool which could solve all the instances proposed, therefore it was chosen as best candidate for our empirical analysis.

First encoding (A). We encode the high-level task to explore Z zones by 3 robots as shown in Fig. 5.

Robots start from a depot, modeled by some *fictitious zones* $-3, -2, -1$. Each robot $i \in \{1, 2, 3\}$ starts at zone $-i$, moves over to the zones $-i+1, \ldots, 0$, and explores, from the *start zone* 0, at most Z of the zones $1, \ldots, Z$. The distance between two zones i and j is denoted by $D(i, j)$. Here we assume the distance that a robot needs to travel to reach the start zone to be 0, but it could be also set to any positive value (see Fig. 6).

The movements of robot i are encoded by a sequence $pos_{i,-i}, \ldots, pos_{i,Z}$ of zones it should visit, with $pos_{i,j} \in$

[5] Running on a machine running Ubuntu Mate 16.4, Intel Core i7 CPU at 2.10GHz and 8GB of RAM

Z	A		B		C		D		E		F		Optimum
	Time	Conf	Time	Conf	Time	Conf	Time	Conf	Time	Conf	Time	Conf	
6	0.40	4841	0.25	3206	0.18	2525	0.17	2069	0.29	3416	0.16	1103	10.9
8	2.07	14400	1.91	15248	1.16	9237	1.62	14355	5.32	30302	1.23	3876	11.4
10	80.06	225518	59.71	184685	26.71	91648	21.72	89785	TO		8.97	27811	12.1
12	286.70	486988	255.55	449485	81.64	198249	54.17	161134	TO		36.21	101308	12.6

Table 1: Running times (sec) and #conflicts for encodings A-F (Z: number of zones to be visited, TO: 5min).

$$\varphi_{depot} := \begin{cases} pos_{1,-1} = -1 \wedge pos_{1,0} = 0 \wedge pos_{2,-2} = -2 \wedge pos_{2,-1} = -1 \wedge pos_{2,0} = 0 \wedge \\ pos_{3,-3} = -3 \wedge pos_{3,-2} = -2 \wedge pos_{3,-1} = -1 \wedge pos_{3,0} = 0 \end{cases} \tag{5}$$

$$\varphi_{move} := \begin{cases} \bigwedge_{i=1}^{3} d_{i,0} = 0 \wedge \\ \bigwedge_{j=1}^{Z} \left[\left(\bigvee_{k=0}^{Z} \bigvee_{\substack{l=1 \\ l \neq k}}^{Z} \left(pos_{i,j-1}{=}k \wedge pos_{i,j}{=}l \wedge d_{i,j}{=}d_{i,j-1}{+}D(k,l) \right) \right) \vee \left(pos_{i,j}{=}-4 \wedge d_{i,Z}{=}d_{i,j-1} \right) \right] \end{cases} \tag{6}$$

$$\varphi_{each} := \begin{cases} \bigwedge_{k=1}^{Z} \left[\bigvee_{i=1}^{3} \bigvee_{j=1}^{Z} \left(pos_{i,j} = k \wedge \bigwedge_{u=1}^{3} \bigwedge_{\substack{v=1 \\ (u,v) \neq (i,j)}}^{Z} pos_{u,v} \neq k \right) \right] \end{cases} \tag{7}$$

$$\varphi_{max} := \begin{cases} \bigwedge_{i=1}^{3} \left[m_i \Leftrightarrow \left(\bigwedge_{\substack{l=1 \\ l<i}}^{3} d_{l,Z} < d_{i,Z} \wedge \bigwedge_{\substack{l=1 \\ i<l}}^{3} d_{l,Z} = d_{i,Z} \right) \right] \end{cases} \tag{8}$$

Fig. 5: SMT encoding A for the exploration phase.

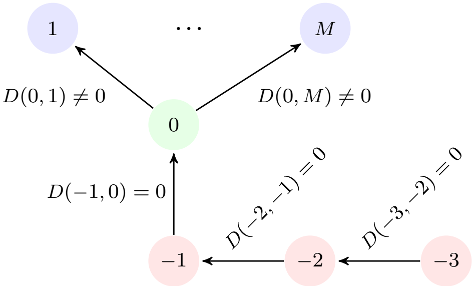


Fig. 6: Initial robot configuration.

$\mathbb{Z}$. The variables $pos_{i,-i}, \ldots, pos_{i,0}$ in φ_{depot} in Eq. (5) represent the movements from the depot to the start zone.

For $j > 0$, if the value of $pos_{i,j}$ is between 1 and Z then it encodes the jth zone visited by robot i. Otherwise, $pos_{i,j} = -4$ encodes that the robot stopped moving and stays at $pos_{i,j-1}$ for the rest of the exploration (*i.e.*, the plan does not require robot i to explore any more zones). The total distance traveled by robot i to visit zones until step j is stored in $d_{i,j} \in \mathbb{R}$. These facts are encoded by φ_{move} in Eq. (6) for each robot $i \in \{1, 2, 3\}$ by $d_{i,0} = 0$ and for each $j \in \{1, \ldots, Z\}$, which ensures that, at each step j, either the robot moves and its travel distance is incremented accordingly, or the robot stops moving. Notice that in this second case, we can immediately determine the total travel distance for the robot at the last step in the plan and, furthermore, the above constraints imply that once robot i stops moving ($pos_{i,j}{=}-4$) it will not move in the future ($pos_{i,j'}{=}-4$ and $d_{i,j'} = d_{i,j'-1}$ for all $j \le j' \le Z$).

For each zone $k \in \{1, \ldots, Z\}$ we enforce that it is visited exactly once by requiring φ_{each} in Eq. (7).

Finally φ_{max} in Eq. (8) uses for each robot $i \in \{1,2,3\}$ a Boolean variable m_i to encode whether the robot has the smallest index under all robots with maximal total travel distances at the end of their plans (note that there is exactly one robot with this property).

Our optimization objective is to minimize the largest total travel distance:

$$\text{minimize} \quad \textstyle\sum_{i=1}^{3} m_i \cdot d_{i,Z} \qquad (9)$$
$$\textit{subject to } \varphi_{depot} \wedge \varphi_{move} \wedge \varphi_{each} \wedge \varphi_{max}$$

Results. We consider four benchmarks with 6, 8, 10 and 12 zones to be visited. Encoding A allowed us to compute optimal plans, but it does not scale with the number of zones to be visited. The solving time 286.7 seconds listed in Table 1 for the optimal objective 12.6 for a benchmark with $Z = 12$ zones claims a large part of the overall duration of the exploration phase.

Tackling loosely connected constraints (B). By analyzing solver statistics we noticed that the number of theory conflicts was quite large, and theory conflicts typically appeared at relatively high decision levels, *i.e.*, at late stages of the Boolean search in the SAT solver. One reason for this is that during optimization, violations of upper bounds on the total travel distances can be recognized by the theory solver only if all the zones that a robot should visit are already decided. In other words, the constraints defining the total travel distance of a robot build a loosely connected chain in their variable-dependency graph. Furthermore, explanations of the theory conflicts blamed the whole plan of a robot, instead of restricting it to prefixes that already lead to violation. As a result, the propositional search tree could not be efficiently pruned. To alleviate this problem, we added to the encoding A the following formula, which is implied by the monotone increment of the partial travel distances by further zone visits:

$$\bigwedge_{i=1}^{3} \bigwedge_{j=1}^{Z} d_{i,j} \leq d_{i,Z} \qquad (10)$$

Results. As Table 1 shows, adding the above constraints led to a slight improvement, but the solving time of 255.55 seconds for 12 zones is still too long for our application.

Symmetry breaking (C). Although the robots start from different zones, all move to the start location 0 at cost 0 before exploration. Thus, given a schedule for the three robots, a renaming of the robots gives another schedule with the same maximal travel distance. These symmetries result in the solver covering unnecessarily redundant search space, significantly increasing solving time. However, breaking these symmetries by modifying the encoding and without modifying the solver-internal algorithms is hard. A tiny part of these symmetries, however, can be broken by imposing on top of encoding B that a single, heuristically determined zone k (*e.g.*, the closest or furthest to zone 0) should be visited by a fixed robot i:

$$\bigvee_{j=1}^{Z} pos_{i,j} = k \qquad (11)$$

Results. This at first sight rather weak symmetry-breaking formula proved to be beneficial, resulting in a greatly reduced number of conflicts as well as solving time (81.64 seconds for $Z = 12$ zones, see Table 1). However, this encoding just fixes the robot that should visit a given single zone, thus the computational effort for Z zones reduces only to a value comparable to the previous effort (using encoding B) for $Z - 1$.

Explicit scheduler choice (D). In order to make the domain over which the variables $pos_{i,j}$ range more explicit, we added to encoding C the following requirement:

$$\bigwedge_{i=1}^{3} \bigwedge_{j=1}^{Z} (pos_{i,j} = -4 \vee \bigvee_{k=1}^{Z} pos_{i,j} = k) \qquad (12)$$

Results. This addition led to some performance gain. With a solving time of 54.17 seconds for 12 zones, our approach could be successfully integrated in the RCLL planning framework.

Partial bit-blasting (E). To reduce the number and size of theory checks, we also experimented with partial bit-blasting: the theory constraints $pos_{i,j}{=}k$ in encoding C were replaced by Boolean propositions $pos_{i,j,k}{\in}\mathbb{B}$, which are true iff robot i visits zone k at step j. For each $i{\in}\{1,2,3\}$ and $j{\in}\{-3,\ldots,Z\}$ we ensure that there is exactly one $k{\in}\{-4,\ldots,Z\}$ for which $pos_{i,j,k}$ is true by bit-blasting for the Z+5 possible values (using fresh propositions $p_{i,j,k} \in \mathbb{B}$):

$$\begin{array}{lll} pos_{i,j,0} \iff (\neg p_{i,j,\lceil \log{(Z+5)} \rceil} \wedge \ldots \wedge \neg p_{i,j,0}) & & \\ pos_{i,j,1} \iff (\neg p_{i,j,\lceil \log{(Z+5)} \rceil} \wedge \ldots \wedge \ \ p_{i,j,1}) & \ldots & (13) \end{array}$$

Results. As shown in Table 1, partial bit-blasting did not introduce any improvement. On the contrary, an optimal solution for 12 zones could not be computed within 5 minutes. We made several other attempts to improve the running times by modifying encoding D, but they did not bring any major improvement.

Explicit decisions (F). Even though encoding D could be integrated in the RCLL framework, we investigated ways to further reduce the solving times.

To this purpose, we developed a new encoding shown in Fig. 7, in which we made some decisions explicit by means of additional variables.

In particular, for each $k \in \{1, \ldots, Z\}$ we introduced an integer variable m_k to encode which robot visits zone k, and an integer variable $n_{i,k}$ for each $i \in \{1, 2, 3\}$ to count how many of the zones $1, \ldots, k$ robot i has to visit. The meaning of these variables are encoded in φ_{visits} in Eq. (14).

We keep the position variables $pos_{i,j}$ to store which zone is visited in step j of robot i, but their domain is slightly modified: knowing the number $n_{i,Z}$ of visits for each robot, the fictitious location $pos_{i,j} = -4$ is not needed anymore. Instead, we will simply disregard all $pos_{i,j}$ assigned for $j > n_{i,Z}$.

We also keep the variables $d_{i,j}$, but with a different meaning: $d_{i,j}$ stores the distance traveled by robot i from its $(j-1)$th position $pos_{i,j-1}$ to its jth position $pos_{i,j}$. We add the constraints Eq. (15) for defining the positions up to the start zone and additionally the constraints in Eq. (16). Note that we replaced $d_{i,j} = D(k, l)$ with a weak inequality constraint. As we discuss later in this section, this was possible as the minimization of travel distances will anyways enforce the equality, but solving inequalities seems to be easier for `Z3`.

A new variable d_i for $i \in \{1, 2, 3\}$ is used to store the total travel distance for each robot in φ_{tot} in Eq. (17), which ensures that, if robot i has to visit k zones ($n_{i,Z} = k$) then its total travel distance d_i is (at least equal to) the sum of the distances traveled from $pos_{i,0}$ to $pos_{i,k}$. If robot i does not move at all (*i.e.*, $n_{i,Z} = 0$) then d_i will be (at least) zero.

The formula φ_{all} in Eq. (18) makes sure that each robot visits all zones it has been assigned to by means of variables m_k: if robot i is assigned to zone k then this zone will be visited at some step j (within the upper bound on the number of zones to be visited $n_{i,Z}$).

Z	F		F$_1$		F$_2$	
	Time	#solved	Time	#solved	Time	#solved
12	54.78	66/100	57.02	66/100	66.84	46/100

Table 2: Average solving time (sec) and #instances solved for encodings F, F$_1$ and F$_2$ on 100 benchmarks (TO: 2min).

Furthermore, in φ_{bounds} in Eq. (19) we introduce bounds on integer variables so that the solver can represent integers as bit-vectors and internally perform bit-blasting.

Finally, we replace the nonlinear objective function specified in Eq. (9) by a linear one: since all robots start from the start zone, we exploit symmetry and require an order on the total travel distances in Eq. (20).

We can now minimize the total distance for the first robot d_1 under the side condition that the conjunction of all formulas in Fig. 7 holds.

Results. Table 1 shows a considerable improvement by encoding F over previous solutions for the selected benchmarks. In order to obtain statistically significant results, we also tested encoding F on 100 most recurring instances of the RCLL problem with 12 zones (see Table 2). Especially the replacement of a non-linear objective function with a linear one allowed us to reduce the complexity of the optimization problem at hand.

To analyze the potential sources of improvement, we made additional experiments with two variants of encoding F: in encoding F$_1$ we removed the bounds for integer variables as specified in Eq. (19), and in encoding F$_2$ we replaced the inequalities in Eq. (16) and (17) with equalities (while the constraints from Eq. (19) are kept in F$_2$). Table 2 and Figure 8 show results for the previously used 100 benchmarks. While working with unbounded integers in encoding F$_1$ does not seem to significantly affect the solving times, the solving time for the encoding F$_2$ with equalities is almost always higher, and a fewer number of instances could be solved within the timeout.

6 Production Phase

Building on the results obtained for the exploration phase, we moved on to consider the second phase of the RCLL competition, *i.e.*, production. This part of the game poses challenges to the OMT solver that are different in nature with respect to the ones met before. One the one hand, production tasks are more con-

$$\varphi_{visits} := \left\{ \quad \bigwedge_{i=1}^{3} \left[n_{i,0} = 0 \wedge \bigwedge_{k=1}^{Z} ((m_k = i \wedge n_{i,k} = n_{i,k-1} + 1) \vee (m_k \neq i \wedge n_{i,k} = n_{i,k-1})) \right] \right. \tag{14}$$

$$\varphi_{depot} := \left\{ \begin{array}{l} pos_{1,-1} = -1 \wedge pos_{1,0} = 0 \wedge pos_{2,-2} = -2 \wedge pos_{2,-1} = -1 \wedge pos_{2,0} = 0 \wedge \\ pos_{3,-3} = -3 \wedge pos_{3,-2} = -2 \wedge pos_{3,-1} = -1 \wedge pos_{3,0} = 0 \end{array} \right. \tag{15}$$

$$\varphi_{dist} := \left\{ \quad \bigwedge_{i=1}^{3} \bigwedge_{j=1}^{Z} \left[\bigvee_{k=0}^{Z} \bigvee_{\substack{l=1 \\ l \neq k}}^{Z} \left(pos_{i,j-1}{=}k \wedge pos_{i,j}{=}l \wedge d_{i,j} \geq D(k,l) \right) \right] \right. \tag{16}$$

$$\varphi_{tot} := \left\{ \quad \bigwedge_{i=1}^{3} (n_{i,Z} = 0 \wedge d_i \geq 0) \vee \bigvee_{k=1}^{Z} (n_{i,Z} = k \wedge d_i \geq \sum_{j=1}^{k} d_{i,l}) \right. \tag{17}$$

$$\varphi_{all} := \left\{ \quad \bigwedge_{i=1}^{3} \bigwedge_{k=1}^{Z} m_k = i \implies \bigvee_{j=1}^{Z} n_{i,Z} \geq j \wedge pos_{i,j} = k \right. \tag{18}$$

$$\varphi_{bounds} := \left\{ \quad \bigwedge_{i=1}^{3} \bigwedge_{k=1}^{Z} \bigwedge_{j=1}^{Z} 1 \leq m_k \leq 3 \qquad 0 \leq n_{i,k} \leq Z \qquad 1 \leq pos_{i,j} \leq Z \right. \tag{19}$$

$$\varphi_{symm} := \left\{ \quad d_1 \geq d_2 \wedge d_2 \geq d_3 \right. \tag{20}$$

Fig. 7: SMT encoding F for the exploration phase.

strained and therefore present less symmetries than exploration. On the other hand, they require more sophisticate robot-robot and robot-environment interactions, which affect both plan synthesis and execution.

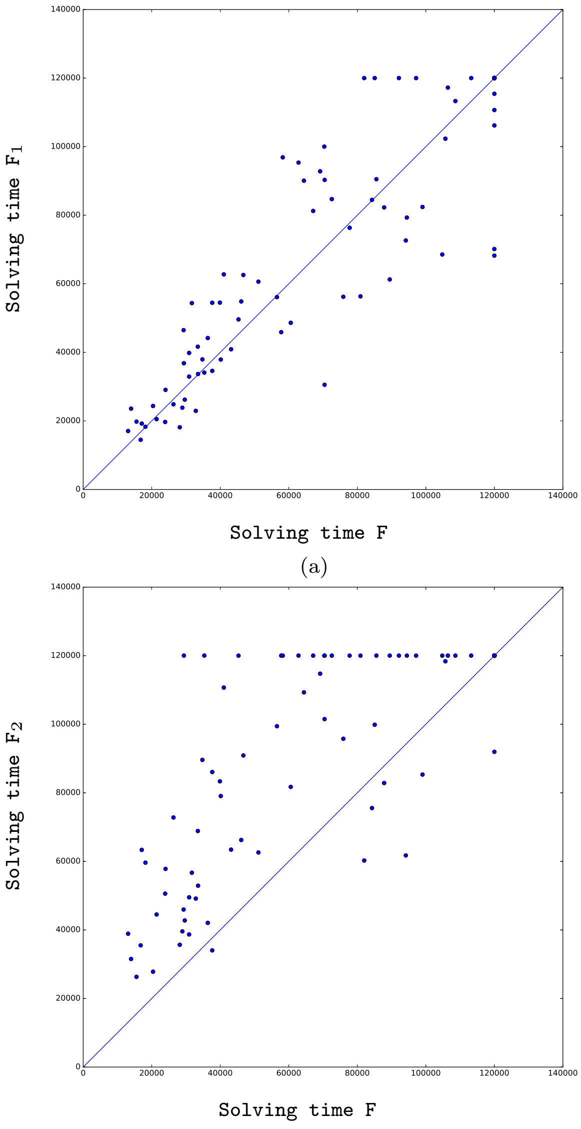
The methodology presented here has been fully integrated in the system presented in Section 4 and tested in the context of the Planning and Execution Competition for Logistics Robots in Simulation.

6.1 Building a formal model for production processes

Given an RCLL configuration, our goal is to find a bounded sequence of robot actions that maximizes the total reward achieved for delivering ordered products. Due to the complexity of the RCLL domain, several challenges arise during the logical encoding of this optimization problem. The formal model needs to account for concurrent robot motions, production processes and machine states, order schedules, deadlines and rewards.

We assume that decisions on actions are made sequentially for one robot at a time; the transitions in T will model those decisions and their effects by updating the states of all components of the model accordingly. Continuous variables are used to keep track of time – *e.g.*, when a robot starts an action or a machine completes a production step – and are used to ensure that decisions made locally during each step are time-consistent at a global level.

Let M represent the total number of machines in the arena, R the number of robots used and p the planning horizon (number of robot actions) considered. The first step towards defining a formal model for robot motions and machine processes is to identify a set of variables that encode all the relevant properties of the system's state. To be able to refer to the jth action and its effects, we attach an index from the domain $\{1, \ldots, p\}$ to the variables. Furthermore, since actions have preconditions and effects, for each step we encode explicitly the state of the system *before* and *after* an action is performed; we do so by appending A and B respectively to



(a)

(b)

Fig. 8: Comparison of solving times (msec) for encodings F, F$_1$ and F$_2$ ($Z = 12$, TO: 2min).

the variable names. Thus, if x is a variable describing the state of a component then xA_j and xB_j encode the component state before and after the jth action.

Actions. Each action has a unique integer identifier. For $j \in \{1, \ldots, p\}$ we use

- A_j to store the identifier of the action performed in step j,
- t_j is the time when the execution of the action of step j starts and
- rd_j is the time needed to complete the action of step j.

Robots. The identity and state of the robot executing the action of step $j \in \{1, \ldots, p\}$ will be described using the following variables:

- R_j stores the integer identifier of the robot executing step j,
- $holdA_j$ and $holdB_j$ tell whether the robot is holding something before respectively after the action at step j and
- pos_j specifies the position where the robot needs to be to execute the action assigned at step j.

Machines. The identity and state of the machine used in step $j \in \{1, \ldots, p\}$ is encoded by the following integer-valued variables:

- M_j tells what machine is involved in the action performed at step j,
- md_j specifies the action duration,
- $state1A_j$ and $state1B_j$ encode whether the machine used in step j is prepared before resp. after the step,
- $state2A_j$ and $state2B_j$ encode whether a CS used at step j is loaded with a cap or not and
- $state3A_j$ and $state3B_j$ encode whether the slide of a CS used in step j is full or not.

Initial state. We introduce dedicated variables to describe the initial state before the first step. Though the game always starts in a fixed initial state, such variables give us the flexibility to synthesize plans on-the-fly during the game. We define for all $i \in \{1, ..., R\}$ and $k \in \{1, \ldots, M\}$:

- $initPos_i$ and $initHold_i$ to encode initial conditions for robot i and
- $initState1_k$, $initState2_k$ and $initState3_k$ to store the initial state for machine k.

Rewards. To define the objective function to be optimized as specified in Eq. (3), we use for each $j \in \{1, \ldots, p\}$

- a real-valued variable rew_j to store the reward achieved for executing step j.

Using the above variables, we define the encoding of plans as shown in Fig. 9. In the following, *products* are encoded by integer values – *e.g.*, "no product" is represented by 0, black base by 1, etc. We start with defining sub-formulas to encode the initial system state, the preconditions and effects of the possible actions and the rewards that can be achieved.

Initialization. For the initial state of the game we define the formula φ_{init} in Eq. (21), meaning that robots start from the depot and do not hold objects, while machines are not prepared nor loaded for production.

Making initial states consistent. Formula φ_{start} in Eq. (22) ensures that the above initial values are propagated to

$$\varphi_{init} := \left\{ \bigwedge_{i=1}^{R} \mathit{initHold}_i = 0 \land \mathit{initPos}_i = 0 \land \bigwedge_{k=1}^{M} \mathit{initState1}_k = 0 \;\land\; \mathit{initState2}_k = 0 \land \mathit{initState3}_k = 0 \right. \tag{21}$$

$$\varphi_{start} := \left\{ \begin{array}{l} \bigwedge_{i=1}^{R} \bigwedge_{j=1}^{p} (R_j = i \land \bigwedge_{j'=1}^{j-1} \neg(R_{j'} = i) \implies (\mathit{holdA}_j = \mathit{initHold}_i) \land \\ \bigvee_{u=0}^{M} \bigvee_{v=1}^{M} \mathit{initPos}_i = u \land \mathit{pos}_j = v \land t_j \geq \mathit{Dist}(u,v)) \land \\ \bigwedge_{k=1}^{M} \bigwedge_{j=1}^{p} (M_j = k \land \bigwedge_{j'=1}^{j-1} \neg(M_{j'} = k) \implies (\mathit{state1A}_j = \mathit{initState1}_k \\ \land \;\; \mathit{state2A}_j = \mathit{initState2}_k \land \mathit{state3A}_j = \mathit{initState3}_k)) \end{array} \right. \tag{22}$$

$$\varphi_{id} := \left\{ \begin{array}{l} \bigwedge_{j=1}^{p} \bigwedge_{j'=j+1}^{p} (R_{j'} = R_j \land \bigwedge_{j''=j+1}^{j'-1} \neg(R_{j''} = R_j) \implies (\mathit{holdA}_{j'} = \mathit{holdB}_j) \land \\ \bigvee_{u=0}^{M} \bigvee_{v=1}^{M} \mathit{pos}_j = u \land \mathit{pos}_{j'} = v \land t_{j'} \geq t_j + \mathit{rd}_j + \mathit{Dist}(u,v)) \land \\ \bigwedge_{j=1}^{p} \bigwedge_{j'=j+1}^{p} (M_{j'} = M_j \land \bigwedge_{j''=j+1}^{j'-1} \neg(M_{j''} = M_j) \implies (\mathit{state1A}_{j'} = \mathit{state1B}_j \\ \land \;\; \mathit{state2A}_{j'} = \mathit{state2B}_j \land \mathit{state3A}_{j'} = \mathit{state3B}_j \land t_{j'} \geq t_j + \mathit{md}_j)) \end{array} \right. \tag{23}$$

$$\varphi_a := \left\{ \;\; A_j = \mathit{id} \implies (\mathit{preconditions} \land \mathit{effects}) \right. \tag{24}$$

$$\varphi_{rew} := \left\{ \;\; \mathit{rew}_j = \mathit{dl} - t_j - \mathit{md}_j \right. \tag{25}$$

Fig. 9: SMT encoding for the transition system underlying the RCLL domain.

the initial states for robots and machines. If robot i is active at step j and it has never been active before, then j is its first step and it must start in the robot's initial state. Moreover, for each step, the robot timer is incremented by *at least* the travel time, which is encoded using constants $\mathit{Dist}(u, v)$ for the travel time between the machines u and v. Similar requirements are imposed on the machines.

Making successor states inductively consistent. The formula φ_{id} in Eq. (23) ensures that when a robot or machine is not involved in an action then the action does not change the robot's resp. machine's state. The formula states that if robot i is active at step j' and it has not been active since step $j < j'$, then we ensure that its *hold* state is propagated to j' (we say that effects of previous step j are equal to the preconditions at j'). The robot moves to the location required by the action assigned at j'. The robot timer will be incremented by *at least* travel time plus action duration. A similar interpretation holds for the machines.

Action rules. Eq. (24) defines φ_a that specifies the preconditions and effects of action a. The formula means that when an action a – encoded by its integer identifier – is selected, the appropriate preconditions are checked and effects are propagated. For instance, the rule for the delivery action will have the following definition:

$$\begin{array}{l} A_j = 11 \implies (M_j = 2 \land \mathit{state1A}_j = 8 \land \mathit{state1B}_j = 0 \land \\ \mathit{state2B}_j = \mathit{state2A}_j \land \mathit{state3B}_j = \mathit{state3A}_j \land \mathit{md}_j{=}15 \land \\ \mathit{pos}_j = 2 \land \mathit{holdA}_j = 3 \land \mathit{holdB}_j = 0 \land \mathit{rd}_j = 10) \end{array}$$

Production steps:

ID	Action
1	Retrieve base with cap from shelf at CS
2	Prepare CS to retrieve cap
3	Feed base into CS
8	Discard cap-less base
7	Prepare BS to provide black base
6	Retrieve base from BS
4	Prepare CS to mount cap
5	Feed black base to CS
9	Retrieve black base with cap from CS
10	Prepare DS for slide specified in order
11	Deliver to DS

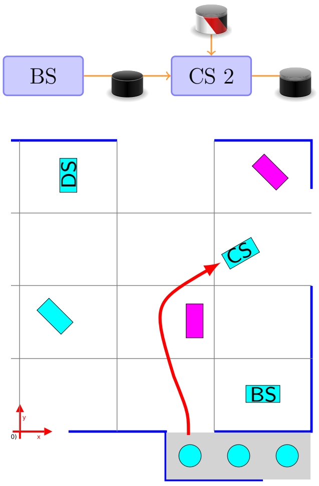


Fig. 10: Production steps necessary to produce a C_0 product (left) as decomposed (top right). The first step for the robot is to move to feed a cap in to the Cap station (right). (Niemueller et al, 2015; RCLL Technical Committee, 2017).

Reward scheme. Finally, we need to specify a reward scheme for actions. As already mentioned, by means of rewards we can drive the synthesis towards optimal controllers. We chose to assign positive rewards to the delivery action only, while all other actions bring no rewards. The reward is defined in Eq. (25) by the formula φ_{rew} where dl is the deadline for delivering a specific product and $t_j + md_j$ indicates the instant when the appropriate station completes the delivery process. Such reward strategy yields to controllers that minimize the makespan of the plan executed by robots.

Controllers. Controller synthesis can now be encoded by the problem to maximize rew_p under the side condition $\varphi_{init} \wedge \varphi_{start} \wedge \varphi_{id} \wedge (\bigwedge_{a \in A} \varphi_a) \wedge \varphi_{rew}$, where A is the set of all actions needed to produce the requested product.

6.2 Experimental evaluation

To evaluate controllers synthesized by our system, we consider the production process shown in Fig. 10. We generated 100 problems, determined by a unique machine placement and order set each. This allows for qualitatively validating plan generation and determining costs of plans generated. We vary the complexity through the number of robots participating in the task. We limited our experiments to a single product of the lowest complexity C_0 (cf. Fig. 10).

We compare our solutions with domains encoded using the well-known Planning Domain Description Language (PDDL2.1) (Fox and Long, 2003). We consider both, temporal domains with durative actions (`T`) and the same domains without (`NT`). We run planners and solvers[6] on the benchmark files we generated, and we validate results generated by our approach using the simulator developed for the Planning Competition for Logistics Robots in Simulation[7] shown in Fig 2.

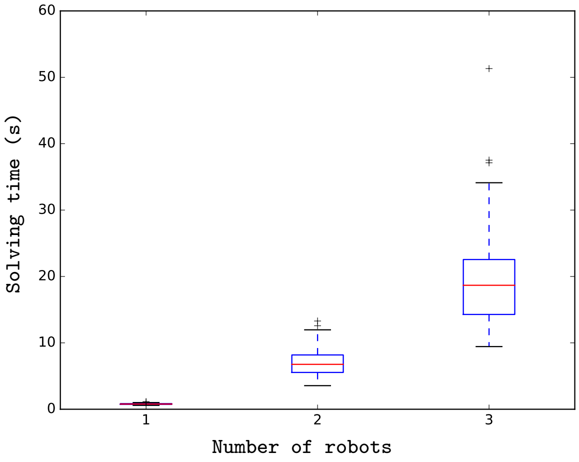


Fig. 11: Boxplots for solving times using `Z3` for 1, 2, 3-robot teams. Red lines represent median values of $0.78s$, $6.76s$ and $18.67s$ respectively.

	One robot					Two robots					Three robots				
	`OMT`	`POPF/NT`		`POPF/T`		`OMT`	`POPF/NT`		`POPF/T`		`OMT`	`POPF/NT`		`POPF/T`	
	`oneshot`	`anytime`	`oneshot`	`anytime`	`oneshot`	`oneshot`	`anytime`	`oneshot`	`anytime`	`oneshot`	`oneshot`	`anytime`	`oneshot`	`anytime`	`oneshot`
# of instances solved	**100**	**100**	**100**	**100**	**100**	**100**	0	0	19	19	**100**	0	0	9	9
solving time average (s)	**0.79**	5.09	0.87	20.73	11.02	**7.06**	–	–	25.66	17.68	**19.45**	–	–	34.25	31.30
plan makespan average (s)	**64.1**	186.99	99.22	67.49	76.06	**51.98**	–	–	60.09	62.10	**51.85**	–	–	54.65	57.56

Table 3: Comparison of `OMT` and `POPF` for temporal (`T`) and non-temporal (`NT`) domains using anytime and oneshot planning.

6.2.1 Evaluation of OMT solvers

Again we compared performances of `Z3`, `SMT-RAT` and `OptiMathsat` on this benchmark. A timeout for solving is set to 60 seconds, which is the time teams can afford spending in planning during an RCLL game without compromising their chances to win. Solving for the domain considered proved to be challenging, however controllers could be successfully synthesized with our approach. `Z3` was the only tool which could solve all the instances proposed, therefore it was chosen for our analysis.

We investigated how the solving time varies with the number of robots used. As Fig. 11 shows, the solving time increases with the size of the team of robots used, moving from an average solving time of $0.79s$ with only one robot, to $19.45s$ for three robots. A natural explanation for this could be that having more robots increases the size of the search space and introduces symmetries which make the solving process harder. In any case, the times obtained are well within the suggested desirable limits for the RCLL competition.

6.2.2 Off-line comparison with other approaches

In the off-line comparison, we consider the `POPF` (Coles et al, 2011) planner and `SMTPlan` (Cashmore et al, 2016), a tool that compiles PDDL domains into SMT encodings and solves them by calling `Z3` internally. We choose the former as it comes readily integrated with `ROSPlan`, a framework for task planning and execution used in the validation (cf. Section 6.2.3). `SMTPlan`, instead, was selected because it represents an interesting solution building a bridge between AI planning and SMT solving. Both tools are evaluated on non-temporal (`NT`) and temporal (`T`) domains.

Table 3 shows the results of this comparison, carried out using a timeout of 60 seconds, a typical time still acceptable in the RCLL. A total of 100 different domain instances were run for each approach for 1, 2 and 3 robots respectively, resulting in a total of 900 runs. For `POPF anytime` we report the time needed for the planner to compute an improved solution, although the tool still ran for the whole 60 seconds allocated. `SMTPlan` is not listed, as it timed out for all the instances considered. We conjecture that this may be due to the way PDDL domains are compiled to SMT, resulting in unnecessarily redundant encodings that are difficult to solve. We can observe in Table 3 that only `OMT` could solve all benchmarks within the given timeout. While `POPF` could always compute solutions for domains where only one robot was used, it failed to do so when the number of robots increased. Furthermore, our approach is able to solve the synthesis problem in less time, when the comparison is possible, and produces solutions with average makespans that are always smaller than other approaches.[8] Furthermore, giving `POPF` additional time to optimize on the first feasible solution (`anytime`) did not seem to lead to major improvements compared to the `oneshot` evaluation. We should mention that all models (OMT and PDDL-based) use approximate values to represent action durations. In particular we assume for the navigation actions that the robot moves at $1\,{}^{\mathrm{m}}\!/_{\mathrm{sec}}$, *i.e.*, using distance as time. While this is unrealistic for actual execution, the values remain comparable among the approaches.

6.2.3 Validation of results

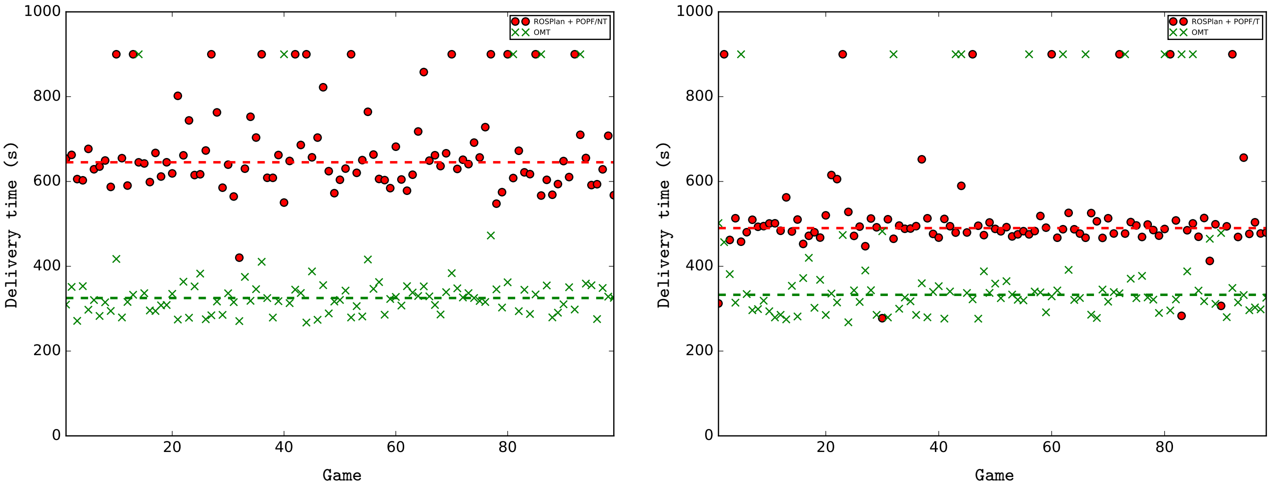
Controllers generated with our approach were validated in the Planning Competition for Logistics Robots in Simulation using the framework described in Section 4.

We tested the robustness of our solutions under realistic competition settings by having two teams of robots competing against each other, one being controlled with

[6] We use a machine running Debian 9, Intel Core 2 Quad CPU Q9450 at 2.66 GHz.

[7] Available at `https://www.fawkesrobotics.org/projects/rcll-sim/`

[8] Makespan for non-temporal `POPF` with single robot is computed as follows. We read the sequence of actions contained in the plan and assign to each the same duration specified in the temporal models used by other approaches.



(a) `OMT` vs `POPF/NT`

(b) `OMT` vs `POPF/T`

Fig. 12: Game statistics for a single robot, `OMT` playing against `ROSPlan` combined with `POPF` using non-temporal (left) and temporal (right) reasoning (20 seconds timeout).

our approach. If we had tested using one team of robots only (that is, our team), we would have reduced the uncertainty present in the game due to the strategies adopted by the opponent. To control the other team, we considered two approaches: *(i)* a PDDL-based approach that embeds `POPF` into `ROSPlan`, a framework for task planning and execution and, *(ii)* a purely rule-based approach based on `CLIPS` (Niemueller et al, 2013), currently used by the RCLL world champions. It must be noted that the execution engine currently used in our framework supports concurrent execution of actions on multiple robots, `ROSPlan` does not.

We therefore decided start our experimental campaign with controllers synthesized for single robots, and have them compete with `ROSPlan` using a single robot. Fig 12 shows statistics for 100 simulations, where our approach competed with `ROSPlan` combined with non-temporal (left) and temporal (right) reasoning. We plot delivery times for both approaches and for each game.

Confirming our off-line results, our controllers were able to control the robot to deliver the order requested. However, for some simulations, controllers computed by `OMT` or `ROSPlan` failed to be executed – we set the corresponding delivery time to 900 seconds. For what concerns our approach, we suggest this may be due to the fact that we assume all machines in the shop-floor are correctly working, however sometimes machines are out of order for a limited time to simulate real world failures. Since we do not capture this uncertainty in our logical encoding, it may happen that the assumptions about the world state made during synthesis become inconsistent during execution. During the first batch of games (Fig. 12, left) we can observe that our controllers failed 5 times, while the opponent failed 12 times. In all other cases we could deliver products successfully within the deadline of the game (15 minutes). Comparing delivery times between the two approaches would not be fair in this case, as `ROSPlan` did not perform any temporal reasoning during these games. We therefore proceeded with a comparison with `ROSPlan` combined with temporal reasoning (Fig. 12, right). There, our approach failed 11 times, while `ROSPlan` failed 7. However, we can observe that when successful, our team had a median delivery time of $\sim 332\,\text{s}$, against $\sim 490\,\text{s}$ of the other team. Such simulations reflect the results we obtained during our off-line evaluation, where our approach could compute plans with the smallest makespans.

We then proceeded with the evaluation of controllers synthesized for multiple robots. Synthesizing global controllers for multi-robot teams could, in principle, increase the chances of failure due to,*e.g.*, synchronization issues. To test the robustness of our controllers, we ran 100 games where `ROSPlan` (again, single robot) competed against our approach, where multiple robots were used.

Figure 13 shows results obtained after 100 games. Interestingly our controllers proved to be as robust as sequential controllers computed for a single robot. Indeed, our approach failed 9 times while `ROSPlan` failed 12. Given that our approach employed multiple robots, median delivery times for our team are always lower than the opponent's.

Finally, we compared the performances of our controllers with the rule-based approach used by the RCLL

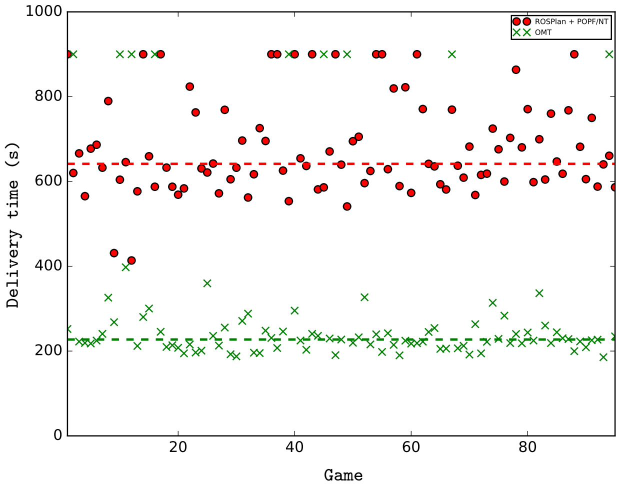


Fig. 13: Game statistics for OMT controllers (multi-robot) playing against ROSPlan combined with POPF using non-temporal reasoning (single robot) (timeout 20 seconds).

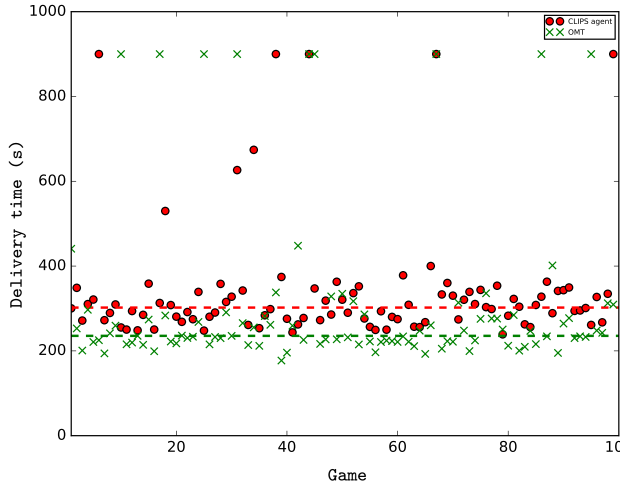


Fig. 14: Game statistics for OMT (multi-robot, timeout 20 seconds) playing against the rule-based approach presented in (Niemueller et al, 2013) (multi-robot).

world champions. This approach employs the full team of robot, allowing a fair comparison between solutions for multi-robot systems. Figure 14 shows the results obtained after 100 games. Results obtained show that, when successful in delivering, our approach guarantees a shorter delivery time, with a median delivery time of $\sim 235\,\mathrm{s}$ against $\sim 302\,\mathrm{s}$ of the rule-based approach. On the other hand, the rule-based agent proved to be more robust, failing only 4 times against 9 times of our approach.

7 Explaining Controllers

The problem of generating explanations for decisions taken by autonomous robotic systems is a very pressing one. The effectiveness of these systems is limited by their current inability to explain their decisions and actions in a human-readable way. Several initiatives have been launched recently to tackle this problem. For instance, DARPA started the *Explainable AI program*[9] with the aim to develop new machine-learning techniques that will produce more explainable models that could be translated into understandable and useful explanation dialogues for the end user. In the same spirit, *Explainable Planning* is proposed in (Fox et al, 2017), where the authors consider the opportunities that arise in AI planning to form a familiar and common basis for communication with the users.

In this section we discuss how OMT-based synthesis implemented in our system could be leveraged to generate explanations for the controller synthesis process. While we acknowledge the existence of a gap between the way OMT solving proceeds and human problem-solving, here we wish to show that OMT solvers exploit techniques that have the potential to ease explaining and facilitate understanding of the underlying decision process.

In particular, we discuss explanations that can be used to understand *(i.)* why a certain controller should be preferred, or *(ii.)* why no controllers could be produced for a given scenario. The discussion that follows is intended to provide initial ideas for achieving the objective of providing effective explanations in OMT synthesis. Examples discussed are specific to the RCLL domain, however we expect that our results can provide a basis for general synthesis of explanations supporting OMT-based decision making.

Explaining why a controller should be chosen. The first question we wish to consider is explaining why a solution computed by the solver should be preferred over different ones. To the best of our knowledge, there exist no planner able to optimize for a metric different than minimizing plan makespan. Therefore, while answering such a question could prove challenging, if not impossible, in other AI-based solutions, OMT could provide useful answers.

As introduced in Section 2, OMT differs from SMT solving in that it produces solutions that are not only *feasible*, but also *optimal*. The key point here is that OMT allows to specify different metrics to measure the quality of a controller. Modern OMT solvers support

[9] https://www.darpa.mil/attachments/DARPA-BAA-16-53.pdf

combinations (lexicographic, pareto, box) of objective functions that can be specified by the user. In this framework, a valid explanation could be to point out the differences in the metrics and show the different effects they have, *e.g.*, in terms of the obtained final reward.

Example. Let us consider a simple example based on the production process of Fig. 10. Let us assume a controller has been requested by the user and the reward scheme of Eq. (25) – which yields plans with minimum makespan – has been used. A sample plan as produced by one of our controllers might have the structure depicted in black in Fig. 15, where three robots are used.

Now suppose we want to know whether a better makespan could be achieved using less robots. One simple way to check this could be to extend our optimization problem by including an additional objective, *e.g.*,

$$\mathit{maximize} \sum_{j=1}^{p} R_j$$

which implicitly forces the solver to select robots whose integer ids have higher value – *e.g.*, robot 3 will be preferred over robot 1. The result is shown in red in Fig. 15. As we can see, it is sufficient to ask the same robot – robot 2 in this case – to perform actions 7 and 6 to obtain a plan that has the same makespan as the original one, but uses only two robots. So in this case, by pointing out the differences in the metrics used to drive the synthesis procedure, one could produce a reason as to explain to the end-user why the second solution should be preferred over the first one.

Explaining why a controller can not be synthesized. This question arises when the solver fails to synthesize a controller for the problem at hand. AI planners are typically not very effective at proving unsolvability of a plan. In contrast, OMT-based approaches are well positioned to address this challenge. Our system frames controller synthesis as a bounded model-checking problem, therefore if the solver states that the desired objective can not be met within a given deadline (and/or within a planning horizon) then this is a proof that no controllers can be produced to accomplish the task.

Besides proving the non-existence of a controller, modern OMT solvers also allow to extract *unsatisfiable cores* that additionally provide a reason for unsatisfiability. Formally, given an unsatisfiable input formula $\varphi = \wedge_{i=1}^{n} \varphi_i$, an unsatisfiable core of φ is an unsatisfiable formula $\psi = \wedge_{i \in I} \varphi_i$ for some $I \subseteq \{1, \ldots, n\}$. With other words, an unsatisfiable core of φ is an unsatisfiable formula ψ which is either φ itself or $\varphi = \psi \wedge \psi'$ for some ψ'.

Though smaller unsat cores typically provide more compact information, *minimal* unsat cores (*i.e.*, unsat cores $\wedge_{i \in I} \varphi_i$ for which $\wedge_{i \in I'} \varphi_i$ is satisfiable for all $I' \subset I$) are computationally hard to compute. Therefore, most solvers aim at generating small explanations but they seldomly guarantee minimality. Since for practical problems unsat cores might be too large to be analyzed by humans, SMT/OMT solvers that follow the SMT-LIB standard[10] require that the user specifies a label for each of the conjunctive subformulas (also called *assertions*) of interest, and only the labeled formulas in the unsat core are listed as output (*i.e.*, the provided explanation together with the unlabeled assertions form an unsat core).

Example. To illustrate how unsat cores can be used to explain unsolvability, consider the following example. The RCLL rules impose that machines can be out of service for a given time at any point in the game. To capture this information, we extend the encoding of machine states of Section 6 by introducing the integer-valued variables $state0A_j$ and $state0B_j$. Such variables encode whether the machine used in step j is fully functional before and after action A_j respectively. If a machine goes down, then all the actions involving that machine can not be performed any more, making it impossible to complete the production of pieces requiring the broken machine. To model this, we extend each action rule (Eq. (24), Section 6) with the additional precondition that the machine required at step j must be working. For instance, the rule for the delivery action will become:

$$\begin{aligned}
&A_j = 11 \implies \\
&(M_j = 2 \wedge \mathit{state0A}_j = 1 \wedge \mathit{state0B}_j = 1 \wedge \\
&\mathit{state1A}_j = 8 \wedge \mathit{state1B}_j = 0 \wedge \mathit{state2B}_j = \mathit{state2A}_j \wedge \\
&\mathit{state3B}_j = \mathit{state3A}_j \wedge \mathit{md}_j = 15 \wedge \\
&\mathit{pos}_j = 2 \wedge \mathit{holdA}_j = 3 \wedge \mathit{holdB}_j = 0 \wedge \mathit{rd}_j = 10)
\end{aligned}$$

Furthermore, we label each constraint in order to enable unsat core generation. Let us now assume our synthesis procedure is triggered under the condition that the *delivery station* DS is broken. This means that the actions involving DS won't be realizable, as a precondition for them to be performed is that the machine has to be operational, *i.e.*, $\mathit{state0A}_j = 1$.

Listing 5 shows the unsat core produced by Z3 when we impose that the delivery station breaks at step 10, *i.e.*, $\mathit{state0A}_{10} = 0$. The unsat core produced here shows that the delivery station could not be prepared for delivery, therefore making delivery impossible.

[10] https://smtlib.github.io/jSMTLIB/SMTLIBTutorial.pdf

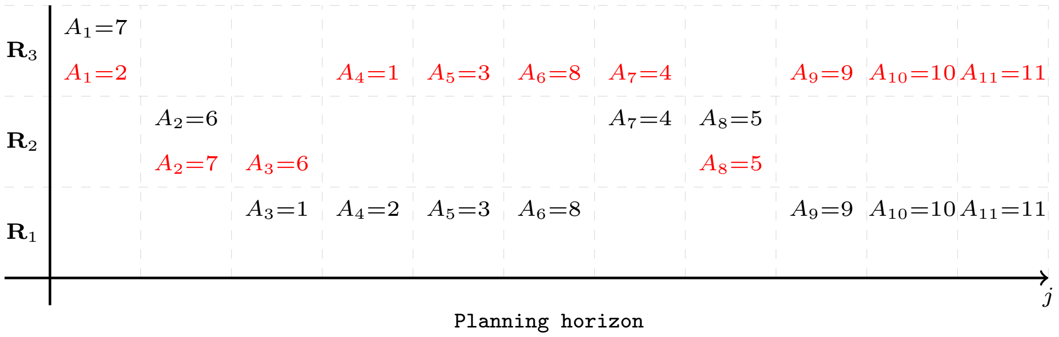


Fig. 15: Example of plans with minimum makespan for the production of a C_0 piece, using two (red) and three (black) robots. The makespan of both plans is 58.52s . Action are encoded by integer ids as in Fig. 10.

```
Start solving...
No solution found
Time: 0.209315061569
Unsat core:
prepare_DS_for_slide_specified_order_10
```

Listing 5: Unsat core generated when DS is down.

8 Challenges, Observations and Recommendations

As a result of the efforts put into solving the problem presented in this work, we gained interesting insights on the problem of synthesizing controllers for robotics using OMT. We detail in the following some observations and ideas that could be useful for other researchers considering similar problems and applications.

Domain-specific knowledge. Incorporating domain-specific knowledge in the encoding leads to considerable speedup in the solving process. This could be done, *e.g.*, by explicitly encoding partial orders on actions, where one specifies causal/temporal relations between production steps (and actions).

Building efficient models. To reduce the number and size of theory checks, we also experimented with partial bit-blasting. We directly encoded some states of the system by means of bit-vector variables instead of integers. We did so to help the solver to detect inconsistencies at the propositional level without the need to call more expensive theory checks. However, such *hand-crafted* encoding turned out to be more error prone and less efficient than relying on the bit-blasting some solvers perform internally – when bounds on integer-valued variables are given. Furthermore, when working with complex domains as the RCLL, dealing with integer quantities instead of Boolean ones reduces the modeling effort.

Solvers and optimization. There exist efficient solvers for different types of optimization problems like combinatorial optimization or integer programming. However, there seems to be room for improvements on problems where the objective function is an arithmetic function but the search is over a finite set of objects, *i.e.*, where the problem seems to involve optimization in the arithmetic domain but at its core it is a purely combinatorial optimization problem. For our application, the controller generation problem could be specified as a Boolean combination of *equalities* between arithmetic terms, *i.e.*, only the combinatorial optimization plays a role. However, the solvers do not recognize this fact and invoke also arithmetic optimization. For the latter, equalities seem to be more problematic, therefore we partially replaced them by inequalities and forced equalities by the objective function. This is an example where knowledge about the internal solving mechanisms is needed to achieve better encodings.

Planning and OMT. Encoding domains using PDDL is easier for non-expert users, as the language provides a more general and intuitive way to describe planning problems. For this reason, we believe it would be interesting to study how the two communities, AI planning and SMT, could benefit from each other. Empirical evidence shows that general-purpose planners achieve impressive performances when it comes to fast exploration of large state-spaces. Starting from this observation, it would be interesting to investigate whether planning heuristics can be imported as tactics into OMT solvers to speed up the search.

Explaining OMT solutions. We have introduced the problem of computing understandable explanations for controllers generated with OMT. We showed the potential of OMT solving techniques to ease explaining, and provided initial results in this direction. We believe that these initial ideas open up several interesting research directions in automated reasoning to provide effective explanations. To do so, a full formalization of what represents a good explanation is required.

Further technical challenges. Several problems of technical nature are to be faced when integrating OMT-based solutions in planning and execution systems. Not all solvers provide anytime solving in the context of optimization, making it difficult to implement online synthesis strategies. Furthermore, current software architectures in robotics do not offer easily usable interfaces for the integration of OMT solvers. Given the recent advancements in SMT and OMT, solvers are now able to deal with rich and complex models. Therefore, having interfaces with software libraries for robotics such as the Robot Operating System (ROS) (Quigley et al, 2009) would ease the process to challenge solvers with concrete problems from that field.

9 Conclusions

In this work we presented an integrated system for generating, executing and monitoring optimal-by-construction controllers for multi-robot systems. By combining the power of Optimization Modulo Theories with the flexibility of an on-line executive, we showed how to synthesize optimal controllers for high-level task planning, and provide runtime feedback on their feasibility. We also discussed how our system can be extended to communicate in a human-readable fashion the decision-making process underlying our synthesis procedure.

This work could be extended in several directions. First of all, we would like to investigate how OMT could be used to implement reactive control with fixed-step lookahead. To do so, we will improve the on-line capabilities of our approach by increasing the amount of information exchanged between our OMT module and the execution and monitoring system. The problem of providing effective explanations will be further investigated, starting from a sound formalization of what constitutes a valid explanation. Finally, we would like to study how our module could be integrated into a goal reasoning framework, where solutions computed by the OMT solver could be used to make informed choices on what goals to pursue in the future.

References

Ábrahám E, Kremer G (2016) Satisfiability checking: Theory and applications. In: Proc. of SEFM'16, pp 9–23

Bensalem S, Havelund K, Orlandini A (2014) Verification and validation meet planning and scheduling. STTT 16(1):1–12

Berry G, Gonthier G (1992) The Esterel synchronous programming language: Design, semantics, implementation. Sci Comput Program 19(2):87–152

Biere A, Cimatti A, Clarke EM, Zhu Y (1999) Symbolic model checking without BDDs. In: In Proc. of TACAS'99, pp 193–207

Bjørner N, Phan A, Fleckenstein L (2015) νz - An optimizing SMT solver. In: Proc. of TACAS'15, pp 194–199

Cashmore M, Fox M, Long D, Magazzeni D, Ridder B, Carrera A, Palomeras N, Hurtós N, Carreras M (2015) Rosplan: Planning in the robot operating system. In: Proc. of ICAPS'15, pp 333–341

Cashmore M, Fox M, Long D, Magazzeni D (2016) A compilation of the full PDDL+ language into SMT. In: Proc. of ICAPS'16, pp 79–87

Cimatti A, Franzén A, Griggio A, Sebastiani R, Stenico C (2010) Satisfiability modulo the theory of costs: Foundations and applications. In: Proc. of TACAS'10, pp 99–113

Coles A, Coles AJ, Clark A, Gilmore S (2011) Cost-sensitive concurrent planning under duration uncertainty for service-level agreements. In: Proc. of ICAPS'11, pp 34–41

Corzilius F, Kremer G, Junges S, Schupp S, Ábrahám E (2015) SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In: Proc. of SAT'15, pp 360–368

Dantam NT, Kingston ZK, Chaudhuri S, Kavraki LE (2016) Incremental task and motion planning: A constraint-based approach. In: Proc. of RSS'16

Dornhege C, Eyerich P, Keller T, Trüg S, Brenner M, Nebel B (2009) Semantic attachments for domain-independent planning systems. In: Proc. of ICAPS'09, pp 114–121

Forgy CL (1982) Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence 19(1):17–37

Fox M, Long D (2003) PDDL2.1: An extension to PDDL for expressing temporal planning domains. J Artif Intell Res (JAIR) 20:61–124

Fox M, Long D, Magazzeni D (2017) Explainable planning. CoRR abs/1709.10256, URL `http://arxiv.org/abs/1709.10256`, `1709.10256`

Giarratano JC (2007) CLIPS Reference Manuals. `http://clipsrules.sf.net/OnlineDocs.html`

Hofmann T, Niemueller T, Claßen J, Lakemeyer G (2016) Continual planning in Golog. In: Proc. of AAAI'16, pp 3346–3353

Ingham M, Ragno R, Williams B (2001) A reactive model-based programming language for robotic space explorers. In: Proc. of i-SAIRAS'01

Ingrand FF, Chatila R, Alami R, Robert F (1996) PRS: A high level supervision and control language for autonomous mobile robots. In: Proc. of ICRA'96, pp 43–49

Leofante F, Ábrahám E, Niemueller T, Lakemeyer G, Tacchella A (2017) On the synthesis of guaranteed-quality plans for robot fleets in logistics scenarios via optimization modulo theories. In: Proc of IRI'17, pp 403–410

McDermott D, Ghallab M, Howe A, Knoblock C, Ram A, Veloso M, Weld D, Wilkins D (1998) PDDL – The Planning Domain Definition Language. Tech. rep., AIPS-98 Planning Competition Committee

Nedunuri S, Prabhu S, Moll M, Chaudhuri S, Kavraki LE (2014) SMT-based synthesis of integrated task and motion plans from plan outlines. In: Proc. of ICRA'14, pp 655–662

Niemueller T, Ferrein A, Lakemeyer G (2009) A Lua-based behavior engine for controlling the humanoid robot Nao. In: RoboCup Symposium 2009

Niemueller T, Lakemeyer G, Ferrein A (2013) Incremental task-level reasoning in a competitive factory automation scenario. In: Proc. of AAAI'13 Spring Symposium

Niemueller T, Lakemeyer G, Ferrein A (2015) The RoboCup Logistics League as a benchmark for planning in robotics. In: Proc. of PlanRob@ICAPS'15

Niemueller T, Karpas E, Vaquero T, Timmons E (2016a) Planning competition for logistics robots in simulation. In: Proc. of PlanRob@ICAPS'16

Niemueller T, Neumann T, Henke C, Schönitz S, Reuter S, Ferrein A, Jeschke S, Lakemeyer G (2016b) Improvements for a robust production in the RoboCup Logistics League 2016. In: Proc. of RoboCup'16, pp 589–600

Nieuwenhuis R, Oliveras A (2006) On SAT modulo theories and optimization problems. In: Proc. of SAT'06, pp 156–169

Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng AY (2009) ROS: An open-source robot operating system. In: ICRA Workshop on Open Source Software, vol 3, p 5

RCLL Technical Committee (2017) RoboCup Logistics League – Rules and regulations 2017

Saha I, Ramaithitima R, Kumar V, Pappas GJ, Seshia SA (2014) Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In: Proc. of IROS'14, pp 1525–1532

Sebastiani R, Tomasi S (2015) Optimization modulo theories with linear rational costs. ACM Trans Comput Log 16(2):12:1–12:43

Sebastiani R, Trentin P (2015a) OptiMathSAT: A tool for optimization modulo theories. In: Proc. of CAV'15, pp 447–454

Sebastiani R, Trentin P (2015b) Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions. In: Proc. of TACAS'15, pp 335–349

Verma V, Estlin T, Jónsson A, Pasareanu C, Simmons R, Tso K (2005a) Plan execution interchange language (PLEXIL) for executable plans and command sequences. In: Proc. of i-SAIRAS'05

Verma V, Jónsson A, Simmons R, Estlin T, Levinson R (2005b) Survey of command execution systems for NASA spacecraft and robots. In: Plan Execution: A Reality Check, Workshop at ICAPS'05

Verma V, Jónsson A, Pasareanu C, Iatauro M (2006) Universal executive and PLEXIL: Engine and language for robust spacecraft control and operations. In: American Institute of Aeronautics and Astronautics Space

Wang Y, Dantam NT, Chaudhuri S, Kavraki LE (2016) Task and motion policy synthesis as liveness games. In: Proc. of ICAPS'16, p 536

Wygant RM (1989) CLIPS: A powerful development and delivery expert system tool. Computers & Industrial Engineering 17(1–4)

Zwilling F, Niemueller T, Lakemeyer G (2014) Simulation for the RoboCup Logistics League with real-world environment agency and multi-level abstraction. In: Robot Soccer World Cup, Springer, pp 220–232

Biographies

Francesco Leofante is a Ph.D. student under the joint supervision of Erika Ábrahám at RWTH Aachen University (Germany) and Armando Tacchella at the University of Genoa (Italy). Francesco received his M.Sc. Degree in Advanced Robotics from the University of Genoa in 2014 and one in Robotique et Informatique Appliquée from Ecole Centrale Nantes (France) in 2015. His research focuses on the application of automated reasoning techniques to model and verify controllers for autonomous robotic systems.

Erika Ábrahám graduated in computer science at the Christian-Albrechts-University Kiel (Germany), and received her Ph.D. from the University of Leiden (The Netherlands) for her work on the development and application of deductive proof systems for concurrent programs. Then she moved to the Albert-Ludwigs-University Freiburg (Germany), where she started to work on formal methods for hybrid and probabilistic systems. Currently she is Full Professor at RWTH Aachen University (Germany) and leads the research group Theory of Hybrid Systems. Her research focuses on SMT solving for real and integer arithmetic, and formal methods for probabilistic and hybrid systems.

Tim Niemueller received the M.Sc. (diploma) degree in computer science from the RWTH Aachen University (Germany) in 2010, and is currently working towards the Ph.D. degree at the Knowledge-Based Systems Group at the same university. He has worked at the Personal Robotics Lab of the Carnegie Mellon University with Siddhartha Srinivasa and as a freelancer for SRI International supervised by Robert C. Bolles. He is member of the RoboCup Executive Committee and team leader of the Carologistics and AllemaniACs RoboCup teams. His research interests are cognitive robotics in the areas of task planning, reasoning, and execution monitoring, world modeling and memory persistence, and robust system integration for personal and industrial autonomous mobile robots.

Gerhard Lakemeyer received his Ph.D. from the University of Toronto in 1990. After six years at the University of Bonn he joined the Department of Computer Science at RWTH Aachen University, where he is Full Professor and heads the Knowledge-Based Systems Group. He is also a Full Professor (status-only) at the University of Toronto, Canada, and a Professorial Fellow at the University of New South Wales, Australia. His research interests include knowledge representation and cognitive robotics. He has published more than 150 scientific papers and has served on numerous program committees, including IJCAI, AAAI, ECAI, and KR. He is a Fellow of the European Association for Artificial Intelligence (EurAI), current President of EurAI and an Associate Editor of Artificial Intelligence and Computational Intelligence. He is also a member of the Editorial Board of the Journal of Applied Logic, and was a member of the Editorial and Advisory Board of the Journal of Artificial Intelligence Research.

Armando Tacchella graduated in computer science and engineering in 1997 at the University of Genoa (Italy), and received his Ph.D. in electrical engineering and computer science in 2001 from the same university. He is Associate Professor in Information Processing Systems at the Polytechnic School of the University of Genoa (Italy), where he teaches Design and Analysis of Algorithms, AI, Formal Verification, and Machine Learning courses for students in computer science and engineering. His research interests focus on modeling and verification of intelligent systems, with an emphasis on adaptive cyber-physical systems. He published more than fifty papers in international journals and conferences in the fields of AI, Computer Aided Verification and Reasoning, and Knowledge Representation.

IEEE FMi paper extension for publication in ISF

To the anonymous reviewers,

This letter is meant to highlight the changes that were made to the original paper when preparing this extended version.

The original paper submitted to the 5th IEEE International Workshop on Formal Methods Integration was extended with the following content:

- Section 2 includes now related work in plan execution, needed to better situate our contribution in the field;
- The RoboCup Logistics League (RCLL) is now presented more in detail in Section 3. The original paper only dealt with exploration in RCLL, while this version presents production as well. For this reason more details on the RCLL domain are given.
- Section 4 presents the integrated system which constitutes one of the elements of novelty of the extended version. There we detail how control synthesis via Optimization Modulo Theories (OMT) is integrated with an online execution and monitoring agent based on CLIPS. We present the formalization of our synthesis approach, then present the execution and monitoring agent based on CLIPS. We then show how the two components are integrated, discussing the data structures used to guarantee seamless integration;
- Section 6 presents recent results obtained for the production problem in RCLL. This section extends the original paper with a new encoding to handle production of products in the RCLL, and with a thorough experimental evaluation comparing our approach with other AI-based approaches;
- Section 7 represents a novel addition to the original paper. There, the problem of generating explanations for controller synthesis is presented. We then show how salient features of our system can be used to solve the above mentioned problem. Illustrative examples and preliminary results demonstrating how approach are presented;
- Section 8 is also new, and is meant to collect together observations and useful recommendations for fellow researchers who might be interested in venturing in this field (controller synthesis with OMT, controller execution). There we discuss the insights we gained while working on encoding the RCLL into OMT, and present some standing challenges.