Improving Coordinated SMT-Based System Synthesis by Utilizing Domain-Specific Heuristics

B. Andres[1], A. Biewer[2], J. Romero[1], C. Haubelt[3], T. Schaub[1,4] *

[1]University of Potsdam [2]Robert Bosch GmbH [3]University of Rostock [4]INRIA Rennes

Abstract. In hard real-time systems, where system complexity meets stringent timing constraints, the task of system-level synthesis has become more and more challenging. As a remedy, we introduce an SMT-based system synthesis approach where the Boolean solver determines a static binding of computational tasks to computing resources and a routing of messages over the interconnection network while the theory solver computes a global time-triggered schedule based on the Boolean solver's solution. The binding and routing is stated as an optimization problem in order to refine the solution found by the Boolean solver such that the theory solver is more likely to find a feasible schedule within a reasonable amount of time. In this paper, we enhance this approach by applying domain-specific heuristics to the optimization problem. Our experiments show that by utilizing domain knowledge we can increase the number of solved instances significantly.

1 Introduction

Embedded systems surround us in our daily life. Often, we interact or rely on them without noticing. Embedded systems are typically small application-specific computing systems that are part of a larger technical context. A computer program executed on a processor of an embedded system usually controls connected mechanical or electric components. For example, in a car, an embedded system manages the engine of the car while another one corrects over- and under-steering, e.g., when a driver goes into a bend too quickly.

Today, designing a new embedded system that serves a predefined purpose is becoming more and more challenging. With increasing system complexity due to regulatory requirements, customer demands, and migration to integrated architectures on massively parallel hardware, the task of system-level synthesis has become increasingly more challenging. During synthesis, the spatial binding of computational tasks to processing elements (PEs), the multi-hop routing of messages, and the scheduling of tasks and messages on shared resources has to be decided. Computational tasks can be viewed as small computer programs that realize control functions where messages between tasks are used to exchange information. In safety critical control application, e.g., the Electronic Stability Control (ESP) from the automotive domain, the correct functionality of an embedded system does not only depend on the correct result of a computation but also on the time, i.e. the schedule, when the result is available. Such systems are called (embedded) hard real-time systems.

* Affiliated with Simon Fraser University, Canada, and IIIS Griffith University, Australia.

With increasing complexity and interdependent decisions, system design demands for compact design space representations and highly efficient automatic decision engines, resulting in automatic system synthesis approaches. Previously presented approaches might fail due to the sheer system complexity in context of guaranteeing the timeliness of hard real-time systems. System synthesis based on satisfiability modulo theories (SMT) has been shown to solve this problem by splitting the work among a Boolean solver and a theory solver [1].

In the work at hand, we employed an answer set programming (ASP) solver to decide the static binding of computational tasks to the PEs of a given hardware architecture. Furthermore, the ASP solver determines for each message a static routing on the hardware architecture's network. Based on these decisions, a linear arithmetic solver ($\mathcal{T}$-solver) then computes global time-triggered schedules for all computational tasks as well as messages. Here, the $\mathcal{T}$-solver's solution guarantees the timing constraints of the system by construction. However, most often the time-triggered scheduling becomes too complex and the $\mathcal{T}$-solver cannot decide within a reasonable amount of time whether a binding and routing (B&R) provided by the ASP solver is schedulable or not. As a remedy, in [2] we introduced a *coordinated SMT-based system synthesis* approach that shows a significant better scalability compared to previous SMT-based synthesis approaches. We will present our proposed approach in detail in Section 3.

In [3] we presented a declarative framework for domain-specific heuristics in answer set solving. The approach allows modifying the heuristic of the solver directly from the ASP encoding. In this paper we show that the coordinated SMT-based system synthesis approach is considerably improved by applying domain-specific heuristics to the ASP solver `clasp`.

This paper is structured as follows. In Section 2 we provide a formal problem definition of the system synthesis problem for hard real-time systems. Furthermore, a formal system model is introduced. Section 3 explains our approach to solve the system synthesis problem in detail. In Section 4 we provide an introduction of the concepts enabling domain-specific heuristics in `clasp` along with the domain-specific heuristics we specified. The experimental results reported in Section 5 show the effectiveness of our heuristics. Section 6 concludes this paper.

2 Problem Formulation and System Model

In this section we introduce a formal system model that will be used in the remainder of this paper. Furthermore, on basis of our system model, we provide a problem formulation of the synthesis problem for hard real-time systems.

In the paper at hand the platform or hardware architecture is modeled as a directed graph $g_P = (\mathbf{R}, \mathbf{E_R})$, the platform graph. Vertices $\mathbf{R}$ represent all possibly shared resources of a platform whereas the directed edges $\mathbf{E_R} \subseteq \mathbf{R} \times \mathbf{R}$ model interconnections between resources (cf. Figure 1). The shared resources $\mathbf{R}$ can further be partitioned into tiles $\mathbf{R_t}$ and routers $\mathbf{R_r}$, with $\mathbf{R_t} \cup \mathbf{R_r} = \mathbf{R}$ and $\mathbf{R_t} \cap \mathbf{R_r} = \emptyset$. A tile $t \in \mathbf{R_t}$ implements exactly one processing element (PE) combined with local memory and is able to execute computational tasks. A router $r \in \mathbf{R_r}$ is able to transfer messages from one hardware resource to another. A mesh of interconnected routers form a so-called network-on-

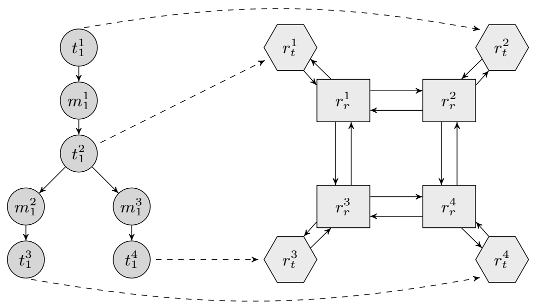

Fig. 1. An instance of our system synthesis model consisting of an application graph (left), a platform graph (right), and bindings of computational tasks to tiles.

chip (NoC). NoCs are a preferred communication infrastructure as they scale well with a growing number of tiles that have to be interconnected in massive-parallel hardware architectures. For the sake of clarity, the remainder of this paper assumes a homogeneous hardware architecture, i.e., tiles possess the same processing power and routers forward messages within the same delay.

The applications that have to be executed on a hardware architecture are modeled by a set of applications $\mathbf{A}$. Each application $A^i \in \mathbf{A}$ is specified by the tuple

$$A^i = (g_A^i, P^i, D^i).$$

An application in hard real-time systems often controls a mechanical system and therefore requests to be executed periodically with the period P^i. In order to guarantee stability of a control algorithm, all computations and communication of an application have to be completed within the constrained relative deadline $D^i \leq P^i$.

Furthermore, each application $A^i \in \mathbf{A}$ is modeled as a directed acyclic graph $g_A^i = (\mathbf{T^i}, \mathbf{E_A^i})$, the application graph (cf. Figure 1). The set of nodes $\mathbf{T^i} = \mathbf{T_t^i} \cup \mathbf{T_m^i}$ is the union of the set of computational tasks of an application $\mathbf{T_t^i}$ and the set of messages of an application $\mathbf{T_m^i}$. The directed edges $\mathbf{E_A^i} \subseteq (\mathbf{T_t^i} \times \mathbf{T_m^i}) \cup (\mathbf{T_m^i} \times \mathbf{T_t^i})$ of the graph g_A^i specify data dependencies between computational tasks and messages or vice versa.

With each computational task $t \in \mathbf{T_t^i}$ a worst-case execution time (WCET) C_t is associated that holds for all PEs. The workload W_t generated by a task $t \in \mathbf{T_t^i}$ on a PE of tile is computed by $W_t = C_t / P_i$. Concerning messages $m \in \mathbf{T_m^i}$, the remainder of the paper assumes that a message equals an atomic entity that is transferred on the resources of the hardware architecture. With this definition, the worst-case transfer time of a message $m \in \mathbf{T_m^i}$ on a resource $r \in \mathbf{R_r}$ is defined by $C_{(m,r)}$.

Problem Formulation In the design of time-triggered real-time systems, system synthesis is the task of computing a valid implementation $\mathbf{I} = (\mathbf{B}, \mathbf{R_m}, \mathbf{S})$ consisting of a binding $\mathbf{B} \subseteq \left(\bigcup_{\mathbf{A^i} \in \mathbf{A}} \mathbf{T_t^i}\right) \times \mathbf{R_t}$, a routing of each message on a tree of resources $\mathbf{R_m} \subseteq \mathbf{R_r}$ and a feasible global time-triggered schedule $\mathbf{S}$. Here, the binding $\mathbf{B}$ contains for each computational task $t \in \mathbf{T_t^i}$ of an application $A^i \in \mathbf{A}$ exactly one tile it is executed on. A routing $\mathbf{R_m}$ defines a set of connected resources of the hardware architecture that are utilized during the transfer of a message. The time-triggered schedule

$$\mathbf{S} = \{\mathbf{s}_{(\lambda,\mathbf{r})} | \lambda \in (\mathbf{T^i_t} \cup \mathbf{T^i_m}), \mathbf{A^i} \in \mathbf{A}, \mathbf{r} \in \mathbf{R_m} \vee (\lambda, \mathbf{r}) \in \mathbf{B}\}$$

contains the start times $\mathbf{s}_{(\lambda,\mathbf{r})}$, generally in clock cycles, for each computational task that starts its execution on a tile and the start time of the transfer for each message on the resources on which the message is routed.

Example Figure 1 presents an example of the system synthesis problem as described above. The platform graph $g_P = (\mathbf{R}, \mathbf{E_R})$ defines a regular 2x2 mesh with four tiles and four routers. All tiles are exclusively connected to exactly one router, while all routers are additionally connected to a set of two other routers. The example consists of only one application A_1 with four computational tasks and three messages. For the sake of clarity, we choose $P_1 = 10, D_1 = 9$, $C_{t_1^1} = C_{t_1^3} = C_{t_1^4} = 1$, $C_{t_1^2} = 2$ and $\forall m \in \mathbf{T^1_m}$, $\forall r \in \mathbf{R_r} : C_{(m,r)} = 1$. One possible implementation $\mathbf{I} = (\mathbf{B}, \mathbf{R_m}, \mathbf{S})$ is then given by the following sets:

$$\begin{aligned}
\mathbf{B} &= \{(t_1^1, r_t^2), (t_1^2, r_t^1), (t_1^3, r_t^4), (t_1^4, r_t^3)\} \\
\mathbf{R_{m_1^1}} &= \{r_r^2, r_r^1\} \\
\mathbf{R_{m_1^2}} &= \{r_r^1, r_r^3, r_r^4\} \\
\mathbf{R_{m_1^3}} &= \{r_r^1, r_r^3\} \\
\mathbf{S} &= \{\ \mathbf{s}_{(t_1^1,r_t^2)} = 0,\ \mathbf{s}_{(m_1^1,r_r^2)} = 1,\ \mathbf{s}_{(m_1^1,r_r^1)} = 2,\ \mathbf{s}_{(t_1^2,r_t^1)} = 3, \\
&\quad\ \mathbf{s}_{(m_1^2,r_r^1)} = 5,\ \mathbf{s}_{(m_1^2,r_r^3)} = 6,\ \mathbf{s}_{(m_1^2,r_r^4)} = 7,\ \mathbf{s}_{(t_1^3,r_t^4)} = 8, \\
&\quad\ \mathbf{s}_{(m_1^3,r_r^1)} = 6,\ \mathbf{s}_{(m_1^3,r_r^3)} = 7,\ \mathbf{s}_{(t_1^4,r_t^3)} = 8\ \}
\end{aligned}$$

3 SMT-based System Synthesis

In the following section we present our SMT-based approach to solve the system synthesis problem introduced in the previous section. SMT-based system synthesis approaches, e.g., [1, 4, 5], gained a lot of attention in domains with complex system specifications and stringent timing constraints such as the automotive domain.

In SMT-based system synthesis, the work is split between a Boolean solver and a theory solver ($\mathcal{T}$-solver) [1] (cf. Figure 2). Here, the Boolean solver computes the static bindings of computational tasks to the PEs of a platform graph and the multi-hop routes of messages on a tree of resource of the platform. In contrast to related work [1, 4, 5], our approach implements the answer set solver `clasp` instead of a SAT solver, since answer set programming (ASP) has been shown to scale better in the determination of multi-hop routes for messages in the densely connected hardware architectures investigated in this paper [6].

3.1 Binding and Routing Using Answer Set Programming (ASP)

The ASP facts based on the formal model of the platform graph $g_P = (\mathbf{R_t} \cup \mathbf{R_r}, \mathbf{E_R})$ and the applications $A^i = (g_A^i = (\mathbf{T^i_t} \cup \mathbf{T^i_m}, \mathbf{E^i_A}), P^i, D^i) \in \mathbf{A}$ (cf. Section 2) for a system instance are defined as follows:

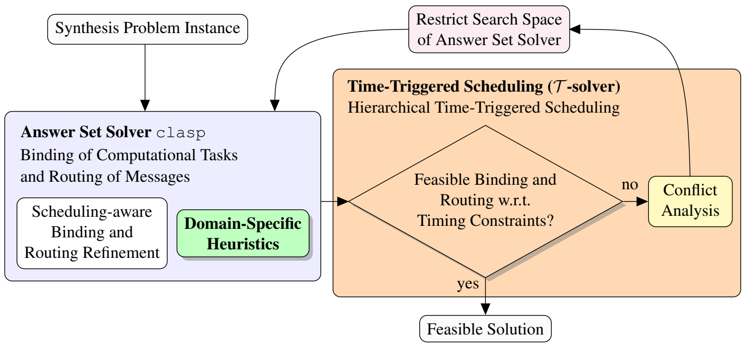
Synthesis Problem Instance
Restrict Search Space of Answer Set Solver
Time-Triggered Scheduling ($\mathcal{T}$-solver)
Hierarchical Time-Triggered Scheduling
Answer Set Solver clasp
Binding of Computational Tasks and Routing of Messages
Scheduling-aware Binding and Routing Refinement
Domain-Specific Heuristics
Feasible Binding and Routing w.r.t. Timing Constraints?
no
Conflict Analysis
yes
Feasible Solution

Fig. 2. Overview of the coordinated SMT-based synthesis approach introduced in this paper. We propose to utilize domain-specific heuristics in order to improve previous work [2].

$$\begin{aligned}
\{task(t, W_t).\mid t \in \mathbf{T^i_t}, A^i \in \mathbf{A}, W_t = \lfloor 1000 \cdot C_t/P_i \rfloor\} &\cup \\
\{send(t, m).\mid (t, m) \in \mathbf{E^i_A}, t \in \mathbf{T^i_t}, m \in \mathbf{T^i_m}, A^i \in \mathbf{A}\} &\cup \\
\{receive(t, m).\mid (m, t) \in \mathbf{E^i_A}, t \in \mathbf{T^i_t}, m \in \mathbf{T^i_m}, A^i \in \mathbf{A}\} &\cup \\
\{tile(t).\mid t \in \mathbf{R_t}\} &\cup \\
\{router(r).\mid r \in \mathbf{R_r}\} &\cup \\
\{edge(r, \tilde{r}).\mid (r, \tilde{r}) \in \mathbf{E_R}\}&.
\end{aligned} \tag{1}$$

The ASP encoding of the binding and routing problem is depicted in Figure 3. The rule in Line 1 specifies that every computational task provided in an instance must be mapped to exactly one tile (PE). Observe that the mapping of computational tasks t to a tile $r \in \mathbf{R_t}$ is represented by atoms `bind(`t`,`r`)` in an answer set. This provides the basis for specifying the routing of messages. The integrity constraint in Line 2 ensures that the workload of every tile does not exceed its maximal utilization. The routing is carried out by (recursively) constructing non-branching acyclic routes from resources of communication targets back to the resource of a sending task, where the routing stops. Line 4 (resp. 5) identifies the tile the sending (resp. receiving) task is bound to. The choice rule in Line 6 connects each encountered target resource to exactly one predecessor, with the only exception that the target resource is not the resource the sender is bound to. Each connected resource is then identified as new target resource in Line 7. Finally, the integrity constraint in Line 8 requires that each resource with a sending task must be a target of the message.

3.2 Time-Triggered Scheduling

Based on the binding and routing decisions by the answer set solver, a time-triggered scheduling problem in linear arithmetic is formulated and solved by the $\mathcal{T}$-solver. The workload is splitted since large numbers are involved in time-triggered scheduling that do not scale well in Boolean solvers. The $\mathcal{T}$-solver computes the start times for each computational task on a PE and each message on the resources of the NoC such that all timing constraints are fulfilled. All constraints in the scheduling formulation are compositions of terms that are formulated in quantifier-free integer difference logic

```
1 {bind(T,R):tile(R)} 1 :- task(T,_).
:- tile(R), 1001 #sum{U,T:bind(T,R),task(T,U)}.

root(C,R) :- send(T,C), bind(T,R).
sink(C,R) :- receive(T,C), bind(T,R).
1 {reached(C,R,S):edge(R,S)} 1 :- sink(C,S), not root(C,S).
sink(C,R) :- sink(C,S), reached(C,R,S).
:- root(C,R), not sink(C,R).
```

Fig. 3. ASP encoding of the binding and routing problem.

(QF_IDL), i.e., $\mathbf{s} - \tilde{\mathbf{s}} \leq k$, with $\mathbf{s}, \tilde{\mathbf{s}} \in \mathbb{N}$ being a start time variable of a computational task or message and a constant $k \in \mathbb{N}$. Constraints in the scheduling problem ensure for example that one resource is utilized at most by one computational task or message at the same time. Furthermore, constraints ensure the integrity of data flows between computational tasks and/or messages. Due to the limited space of the paper a detailed description of the scheduling problem and its formulation is omitted, but can be found in [7]. Our formulation of the scheduling problem is an adapted version of the one presented in [5].

3.3 Coupling of the Answer Set Solver and $\mathcal{T}$-solver

If the $\mathcal{T}$-solver is able to derive a time-triggered schedule based on the ASP solvers decision, the synthesis finishes with a valid solution. In contrast, if the $\mathcal{T}$-solver proves that a feasible schedule based on the binding and routing does not exist, a conflict analysis is started. Related work has shown that deriving a minimal reason (unsatisfiable core or irreducible inconsistent subset) why a schedule cannot be found can significantly increase scalability of SMT-based system synthesis [4].

In order to decrease the time for the $\mathcal{T}$-solver to prove that a binding and routing is not schedulable and to speed up a subsequent conflict analysis, we implemented a hierarchical scheduling scheme that has been established in previous work [5]. The $\mathcal{T}$-solver decides feasibility on subproblems before a schedule is derived for the complete system. Here, deciding feasibility, as well as the subsequent conflict analysis, of smaller subproblems tends to be much faster. Similar to [2], we apply the following hierarchical scheduling scheme:

1. Schedule the computational tasks on each tile independently (TS).
2. Schedule the computational tasks on each tile including incoming and outgoing messages from the tile independently (CS).
3. Schedule clusters of independent applications independently (AS).

If all scheduling problems in a hierarchical stage can be solved, the $\mathcal{T}$-solver starts to solve problems of the subsequent stage. In (AS), clusters of independent applications are specified such that no hardware resource is shared between two clusters. However, depending on the binding and routing, (AS) may be equivalent to the problem of scheduling the complete system.

Our approach implements a modified deletion filter [5] and forward filter [8] to compute a minimal reason why a schedule cannot be found. The deletion filter is applied on infeasible problems of (TS) whereas the forward filter is applied in conflict analysis in stages (CS) and (AS). Based on the first minimal reason from the conflict analysis in a hierarchical stage, the search space of the answer set solver is pruned via integrity constraints. As a result, further binding and routing solutions computed by the answer set solver do not lead to the same infeasibility that has already been observed. Note that the answer set solver is just halted while the $\mathcal{T}$-solver analyses a binding and routing. We do not restart the answer set solver repeatedly.

If a conflict has been found in scheduling hierarchy (TS), the deduced minimal reason only contains computational tasks. The constraint added to the context of the answer set solver ensures that the same set of conflicting computational tasks will not be bound to any PE again. In contrast, the result of a conflict analysis in stage (CS) and (AS) is a set of computational tasks and messages. Without breaking symmetries, the search space of the answer set solver is pruned by the concrete binding and routing of the conflicting computational tasks and messages. Once the search space of the answer set solver has been pruned, a new binding and routing is computed and subsequently analysed in the $\mathcal{T}$-solver. This iterative process stops once a feasible time-triggered schedule has been found or if the answer set solver returns that no further binding and routing can be derived. In the latter case the SMT-based system synthesis approach proved that no feasible solution to the synthesis problem exists for the provided problem specification.

3.4 Coordinated SMT-Based System Synthesis

Up to this point, we introduced our SMT-based system synthesis approach that is in general applicable to solve the synthesis problem presented in Section 2. However, a major drawback of the synthesis approach described so far lies in the often very time consuming scheduling in the $\mathcal{T}$-solver. In extreme cases, deciding schedulability even of small subproblems can take up hours. This is especially the case if the workload of a PE is close to the maximum utilization of 100%. As a remedy, in [2] we showed that the scalability of SMT-based synthesis can be improved considerably if the answer set solver and the $\mathcal{T}$-solver are coordinated. The basic idea of the *coordinated SMT-based synthesis* is to let the answer set solver compute bindings and routings where schedulability can be decided within a reasonable time by the $\mathcal{T}$-solver. This is realized by assigning a constant time budget to the answer set solver exclusively to refine (optimize) an initial binding and routing solution. With this *scheduling-aware binding and routing refinement* (cf. Figure 2) the $\mathcal{T}$-solver is expected to decide schedulability within a reasonable amount of time. Despite the additional time budget for the refinement, the coordinated SMT-based synthesis has been shown to scale significantly better than SMT-based approaches without coordination [2].

In the coordinated SMT-based synthesis approach of this paper, the scheduling-aware binding and routing refinement is realized by using a lexicographical optimization in `clasp`. With the highest priority, the load balancing of the PEs of the platform is optimized. Figure 4 depicts the encoding that realizes a simplified load balancing strategy. The basic idea of the strategy is to allow the solver to successively reduce the

```
1 maxu(1000).
2 {maxu(MU-slice)} :- maxu(MU), omu(Mean), MU-slice>Mean.
3 :- tile(R), maxu(MU), MU+1 #sum {U,T:bind(T,R),task(T,U)}.
4 #maximize{1@2,MU:maxu(MU)}.

6 #minimize{1@1,C:reached(C,R,S)}.
```

Fig. 4. Encoding of the scheduling-aware binding and routing refinement realized as lexicographical optimization of the maximal utilization of all tiles and the number of routed messages.

maximal utilization of PEs and then maximize the number of reductions. Beginning with a maximal utilization (`maxu`) of 1000 (Line 1) the choice rule in Line 2 allows to generate an additional `maxu` by reducing an existing one by a predefined amount `slice`, as long as the new `maxu` is greater then the optimal mean utilization `omu(Mean)`. The integrity rule in Line 3 enforces all maximal utilizations and Line 4 maximizes the number of generated `maxu`. Note that we used this simplified strategy instead of a true minmax optimization due to performance reasons. With the second highest priority in the lexicographical optimization, we minimize the total number of routed messages in a system as shown in Line 6. While the coordinated SMT-based synthesis approach is part of the current state-of-the-art in symbolic system synthesis, the paper at hand aims on improving the coordinated approach by utilizing domain-specific heuristics in the answer set solver (cf. green box in Figure 2). The following section describes in detail our formulated heuristics and the implementation details that enable the usage of domain-specific heuristics in `clasp`.

4 Heuristics

ASP provides a rich modelling language together with highly performant yet general-purpose solving techniques. Often these general-purpose solving capacities can be boosted by domain-specific heuristics. For this reason, we introduced a general declarative framework for incorporating domain-specific heuristics into ASP solving [3]. The rich modelling language is used to specify heuristic information, which is exploited by a dedicated `Domain` heuristic in `clasp` when it comes to non-deterministically assigning a truth value to an atom. Although this bears the risk of search degradation [9], it has already indicated great prospects by boosting optimization and planning in ASP [3]. In this framework, heuristic information is represented within a logic program by means of the dedicated predicate `_heuristic`. For expressing different types of heuristic information, the following basic modifiers are available: `sign`, `level`, `init` and `factor`. Here we explain the first two, that will be applied in our experiments, and refer the reader to [3] for further details. Modifier `sign` allows for controlling the truth value assigned to variables subject to a choice within the solver. With `1` representing true and `-1` false, repectively. For example, given the program

```
{a}.     _heuristic(a,sign,1).
```

atom `a` is chosen with positive sign and the answer set {`_heuristic(a,sign,1)`,`a`} is produced, while replacing `1` by `-1` we obtain {`_heuristic(a,sign,-1)`}. Mod-

```
% H1 - bindings first, then routing
_heuristic(bind(T,R),       level,2) :- task(T,_), tile(R).
_heuristic(reached(C,R,S),level,1) :- edge(R,S), send(_,C).
% H2 - discourage routing
_heuristic(reached(C,R,S),sign,-1) :- edge(R,S), send(_,C).
% H3 - bind sender and receiver together
_heuristic(bind(T',R),sign,1) :-
    bind(T,R), send(T,C), receive(T',C).
% H4 - Clustering computational tasks
_heuristic(bind(T,R),true,A+2) :-
    bind(T',R), send(T',M), receive(T,M), belongs(A,T).
_heuristic(bind(T,R),true,A+1) :-
    bind(T',R'), send(T',M), receive(T,M),
    neighbor(R',R), belongs(A,T).
_heuristic(bind(T,R),level,A)  :-
    belongs(A,T), task(T,_), tile(R).
```

Fig. 5. Domain-specific heuristics used in our system synthesis approach.

ifier `level` establishes a ranking among atoms such that unassigned atoms of highest rank are chosen first, with the default rank of 0. Extending the previous program with

```
{b}.       _heuristic(b,sign, 1).
:- a, b. _heuristic(a,level,1).
```

the solver chooses first `a` (because its level is `1`) with a positive sign, and returns the answer set where `a` is true and `b` is false. Adding the fact `_heuristic(b,level,2)` atom `b` is chosen first with positive sign, and we obtain the answer set with `b` true and `a` false. Further extending the program with the fact `_heuristic(a,level,3)` yields once more the solution with `a` true and `b` false. This illustrates the fact that when an atom gets two values for the same modifier (in the example, `1` and `3` for the `level` of `a`), the one with higher absolute value takes precedence. The modifier `true` (`false`) is defined as the combination of a positive (negative) `sign` and a `level`. For instance, in the last example we could use `_heuristic(a,true,3)` instead of `_heuristic(a,sign,1)` and `_heuristic(a,level,3)`. Note that domain heuristics are dynamic, in the sense that they depend on the changing partial assignment of the solver. As an example, consider the following program:

```
1  _heuristic(a,true, 1).   {a;b}.
2  _heuristic(b,sign, 1) :-     a.
3  _heuristic(b,sign,-1) :- not a.
```

The solver starts setting `a` to true, then the rule in Line 2 is fired, `b` is selected with a positive sign and the answer set with `a` and `b` is produced. On the other hand, replacing modifier `true` by `false` we obtain instead `a` false and `b` false. This ability to represent dynamic heuristics is crucial to boost the performance of our system.

In order to refine the binding and routing provided by `clasp` to the $\mathcal{T}$-solver, we defined a number of different domain-specific heuristics. Figure 5 presents the heuristics that where most successful in increasing the number of solved instances and reducing the overall synthesis time. Except when one heuristic is overriden by another (e.g., H1 and H2 by H4) it makes sense to combine the different heuristics. In fact, our experiments in Section 5 show that this is highly advantageous.

The first heuristic H1 (Line 2-3) places a higher level on binding before routing, since the routing of messages is highly dependent on the binding of their corresponding tasks. Analysis of the answer sets provided by `clasp` shows that many messages are routed over the whole platform even though shorter paths exist. This is one of the major causes for failure during scheduling, since the $\mathcal{T}$-solver has to consider all resources a message is routed over. To reduce unnecessary detours, the heuristic H2 in Line 5 gives a negative sign to all `reached/3` atoms.

Another technique to reduce the number of routed messages is to bind tasks exchanging a message with each other onto the same tile. This is stated by the heuristic H3 (Line 7), provided the sender is already bound to a tile.

The idea of H3 is extended in H4 (Line 10-16) where a task should be bound preferably onto the same tile as its corresponding communication partner or, to a lesser degree, to a neighboring tile. Note that only the rules for binding a receiving task to its sender are shown. The heuristics for encoding that a sending task should be bound to its receiver are analogous to the ones presented. Additionally, H4 tries to bind all task of one application before binding any other task. This allows to cluster tasks of one application onto the same tile before its full utilization is reached. The newly used fact `neighbor/2` identifies two neighboring tiles, while `belongs(A,T)` identifies tasks of the same application `A` and provides an ordering of applications. The identifier `A` is chosen in such a way that there are no conflicts with the offset of the heuristics in Lines 10 and 12. Note that both heuristics H3 and H4 are dynamic.

5 Experiments

In the following section we present the experimental results which quantify the improvement of our domain-specific heuristics from the previous section in coordinated SMT-based system synthesis introduced in Section 3.

All experiments were performed on a dual-processor Linux workstation containing two quad-core 2.4GHz Intel Xeon E5620 and 24GB RAM. All benchmark runs utilized only one CPU core and were carried out sequentially. We report average values over three independent runs per instance, to reduce the non-deterministic factors during the synthesis, e.g., through interrupting the process.

As answer set solver we implemented the python module `gringo`, containing `clasp` (version 3.1.0) with command-line switch `--configuration=auto`. In our initial tests unsatisfiable core based strategies (command-line switch `--opt-strategy= usc,4`) revealed to be highly efficient for this problem class and is taken as reference. Due to the algorithmic approach of unsatisfiable core based optimization, domain-specific heuristics are not working with unsatisfiable core based optimization strategies. We report results for the domain-specific heuristics with branch and bound based optimization (command-line switch `--opt-strategy=bb,2`). Both optimization strategies were selected as the best strategies by comparison of the results on test instances.

strategy / heuristic	solved instances [of 60]			successful couplings		solving time		
	completely	partially	unsolved	synthesis		B&R	Scheduling	Total
usc optimization	4	32	24	35%	118	157.46	83.74	241.20
b&b optimization	0	20	40	13%	199	280.03	154.76	434.79
structural heuristic	34	8	18	62%	16	241.91	7.05	248.96
H1	12	30	18	44%	115	169.37	89.77	259.14
H1 + H2	17	28	15	53%	102	150.86	51.14	202.00
H1 + H2 + H3	14	38	8	59%	99	145.67	116.73	262.39
H4	34	25	1	79%	40	54.75	56.16	110.90
H2 + H4	35	19	6	75%	47	70.68	27.78	98.46
H4 + structural heuristics	38	17	5	79%	36	48.00	43.48	91.48

Table 1. Experimental results of our benchmarks. First two lines are optimization only, while the lower ones include domain-specific heuristics and optimization (with –opt-strategy=bb,2).

As $\mathcal{T}$-solver we implemented `yices` (version 2.3.0, [10]) with command-line arguments `--logic=QF_IDL` and `--arith-solver=floyd-warshall`. Additionally, the `Z3` theorem prover (version 4.3.2, [11]) in logic `QF_IDL` was used in the forward filter in conflict analysis.

In our experiments, the overall time limit for a complete system synthesis run was set to $900s$. After this time, a still running benchmark was interrupted and documented as unsolved. The time for `clingo` to refine an initial binding and routing solution (optimization time) was set to $1s$. Note that in our tests more optimization time decreased the number of successful synthesised systems. We conjecture that the additional time spend on optimization reduces the time in the SMT-based synthesis approach to learn "just enough" infeasible binding and routing solutions within the overall time limit. However, the chosen optimization time is still sufficient to refine solutions such that the $\mathcal{T}$-solver can decide feasibility within a reasonable amount of time.

Concerning the instances of our experiments, the platform graphs $g_R = (\mathbf{R}, \mathbf{E_R})$ of all the system synthesis models were set to a regular 5x5-mesh composed of 25 tiles and 25 routers. The characteristics of an application $A_i = (g^i_A, P_i, D_i)$ in the system's application set were generated similar to the ones reported in [2], representing relevant problem instances in the field of system synthesis. As a difference, the total system utilization in our instances is 70% whereas the reported instances in [2] utilized only up to 40%. In different tests we found that a system's utilization of nearly 70% results in significant harder instances for the coordinated synthesis approach compared to utilization around approximately 60%. Overall, we generated 60 different system instances with the average number of applications per instance being 88 ± 2, a total number of computational tasks of 391 ± 5 and a total number of messages of 303 ± 5.

Table 1 presents the selected results of our experiments. The first column presents the strategy and / or heuristic used in the experiment run. "Structural heuristics" references the commandline switch `--dom-mod=5,8`, automatically applying an heuristic with the false modifier to all atoms involved in a minimization statement. Note that all maximization statements are reduced to minimization before solving. The combination of multiple heuristics is depicted by "+". The next three columns depict the number of instances that where solved completely, partially or not at all, meaning that all three, some, or none of the three independent runs per instance were successful. "Successful synthesis" shows the amount of test runs that finished successfully in percent of overall 180 runs, "couplings" present the average number of couplings needed to solve one instance run with the respective approach. Note that both the resulting numbers are rounded up to integer values. The table concludes with the average time needed to solve the binding and routing ("B&R"), the scheduling and the complete system synthesis problem ("total") in seconds without instances that exceeded the time limit of 900 seconds. Note that "B&R" includes the time for the scheduling-aware binding and routing refinement (optimization time of 1s per answer set solver call).

Although only a few (most interesting) heuristic combinations are shown, we conducted tests with over 70 different strategy / heuristic combinations in total. Many of these combinations yielded average or worse results. Test runs without optimization, i.e., without coordination of the answer set solver and the $\mathcal{T}$-solver, are omitted from the table. None of these synthesis runs were successful (even with the application of heuristics). Extensive data of our tests accompanied by the full encoding can be found in the Labs section of [13].

Comparing the result of the different heuristics reveals that dynamic heuristics are useful in practice and that the combination of different heuristics is most efficient. Note that the heuristic H4 is a stronger reformulation of H1 and H3.

While the structural-based heuristic solves as many instances as the best domain-specific heuristic H4 completely, a large number of instances were not solved at all. The structural-based heuristic also has a huge unbalance of workload between `clasp` and `yices`, with the pure B&R solving time without optimization over one order of magnitude larger than in the other approaches. The reason for this is that the structural-based heuristic is very aggressive in finding an optimal solution, with the first solution being very close to it. While this is very effective in terms of couplings needed, the overall runtime is worse than in the domain-specific heuristics. We suspect that structural-based heuristic does not scale well when the number of necessary routings increases. The combination of H4 and the structural heuristic is controversial. While it solves 4 more instances completely, the same number of instances were unsolved.

Our experiments show that the application of domain-specific heuristics yields a significant increase of successful synthesis runs with almost twice as many compared to the best strategy relying only on the scheduling-aware binding and routing refinement (using a `clasp`'s unsatisfiable core based optimization strategy). At the same time, both the average number of couplings and the total synthesis time is reduced. Furthermore, and more importantly, the number of completely solved instances were increased by a factor of 8.

6 Conclusion

In this paper we presented an SMT-based synthesis approach for hard real-time system synthesis. The approach at hand utilizes the answer set solver `clasp` for generating a binding and routing and the linear arithmetic solver `yices` ($\mathcal{T}$-solver) for time-triggered scheduling based on `clasp`'s decisions. We discussed that a scheduling-aware refinement of the solution provided by `clasp` is needed such that the $\mathcal{T}$-solver is able to solve the scheduling problem within a reasonable amount of time. Based on the optimization-based refinement of [2] we apply domain-specific heuristics, utilizing a novel technique [3] for specifying heuristics for `clasp`. A number of efficient domain-specific heuristics were proposed and benchmarked against optimization-only approaches, as well as structural heuristics provided by `clasp`. The results of the benchmark show that our domain-specific heuristics had a significant positive impact on both instances solved as well as on the runtime for solved instances. It is expected that the approach of utilizing domain-specific heuristics for the refinement of the solutions provided by the Boolean solver can be applied to other applications that combine Boolean with theory solving as well.

Future work includes the search for additional heuristics / approaches toward abolishing the need for optimization, currently consuming approximately half of the total synthesis time. We would also like to explore the possible advantages of a tighter integration of the Boolean and the theory solver as described in [8, 12]. To the best of our knowledge none support domain-specific heuristics as utilized in `clasp` (3.1.0) yet.

Acknowledgments. This work was partly funded by DFG (550/9).

References

1. Reimann, F., Glaß, M., Haubelt, C., Eberl, M., Teich, J.: Improving platform-based system synthesis by satisfiability modulo theories solving. In Proceedings of CODES+ISSS. (2010) 135–144
2. Biewer, A., Andres, B., Gladigau, J., Schaub, T., Haubelt, C.: A symbolic system synthesis approach for hard real-time systems based on coordinated SMT-solving. In Proceedings of DATE. (2015) 357–362
3. Gebser, M., Kaufmann, B., Otero, R., Romero, J., Schaub, T., Wanko, P.: Domain-specific heuristics in answer set programming. In Proceedings of AAAI. Press (2013) 350–356
4. Reimann, F., Lukasiewycz, M., Glaß, M., Haubelt, C., Teich, J.: Symbolic system synthesis in the presence of stringent real-time constraints. In Proceedings of DAC. (2011) 393–398
5. Lukasiewycz, M., Chakraborty, S.: Concurrent architecture and schedule optimization of time-triggered automotive systems. In Proceedings of CODES+ISSS. (2012) 383–392
6. Andres, B., Gebser, M., Glaß, M., Haubelt, C., Reimann, F., Schaub, T.: Symbolic system synthesis using answer set programming. In Proceedings of LPNMR. Springer (2013) 79–91
7. Biewer, A., Munk, P., Gladigau, J., Haubelt, C.: On the influence of hardware design options on schedule synthesis in time-triggered real-time systems. In Proceedings of MBMV. (2015) 105–114
8. Ostrowski, M., Schaub, T.: ASP modulo CSP: The clingcon system. Theory and Practice of Logic Programming **12**(4-5). (2012) 485–503
9. Järvisalo, M., Junttila, T., Niemelä, I.: Unrestricted vs Restricted Cut in a Tableau Method for Boolean Circuits. Annals of Mathematics and Artificial Intelligence **44**(4). (2005) 373–399
10. Dutertre, B.: Yices 2.2. In Proceedings of CAV. Springer (2014) 737–744
11. Moura, L., Bjørner, N.: Z3: An efficient smt solver. In Proceedings of TACAS. Springer (2008) 337–340
12. Janhunen, T., Liu, G., Niemelä, I.: Tight integration of non-ground answer set programming and satisfiability modulo theories. In Proceedings of GTTV. (2011) 1–13
13. Potassco website. http://potassco.sourceforge.net