Scheduling a Steel Plant with Timed Automata*

Ansgar Fehnker**
Computing Science Institute Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, the Netherlands
ansgar@cs.kun.nl

ABSTRACT: Scheduling in an environment with a large number of constraints is known to be a hard problem. We tackle this problem for a integrated steel plant in Ghent, Belgium, using UPPAAL, a model checker for networks of timed automata. We show how to translate schedulability to reachability, enabling us to use UPPAAL's model checking algorithms.
KEYWORDS AND PHRASES: Timed automata, Static Scheduling, Test automata, Reachability, Model checking, UPPAAL
AMS SUBJECT CLASSIFICATION:90B06, 90B35, 90B70, 90B90, 93B50
CR SUBJECT CLASSIFICATION:D.2.2., D.2.4, D.2.10, F.1.1, F.3.1, G.2.1, G.2.4, I.2.2

1 Introduction

This report is a result of the participation of CSI Nijmegen in the European Union Esprit long term research project *Verification of Hybrid Systems*. The Esprit program was set up to improve the take-up of modern information technologies in industry. The VHS project in particular is meant to stimulate research in the area of hybrid systems. These systems typically consists of digital components in a continuous environment. The correct behavior depends strongly on the interaction between the digital components, say the controller, and the controlled process.

Hybrid Systems are important in numerous application areas like avionics, consumer electronics and process control. The research in the VHS-project is focused mainly on industrial process control. One of the major objectives of the project is to analyze a number of case studies of the industrial partners. This includes explicitly the use of existing verification tools for timed and hybrid automata.

Case study 5 of the VHS project (CS5) is brought into the project by SIDMAR, a flat steel producer from Ghent, Belgium. It deals with the part of an integrated steel plant where molten pig iron coming from the blast furnace, is converted into steel of different qualities before it enters the hot rolling mill. The layout of the plant is presented in figure 1. We assume that the transformation of pig iron to steel consists of a number of atomic, uninterruptible operations. The number of treatments and the time needed for each of these depend on the steel quality. Ultimately we want to control the output of the plant, i.e. the quality of the steel leaving the system.

*This work was partially supported by the European Community Esprit-LTR Project 26270 VHS (Verification of Hybrid systems)
**Research supported by Netherlands Organization for Scientific Research (NWO) under contract SION 612-14-004

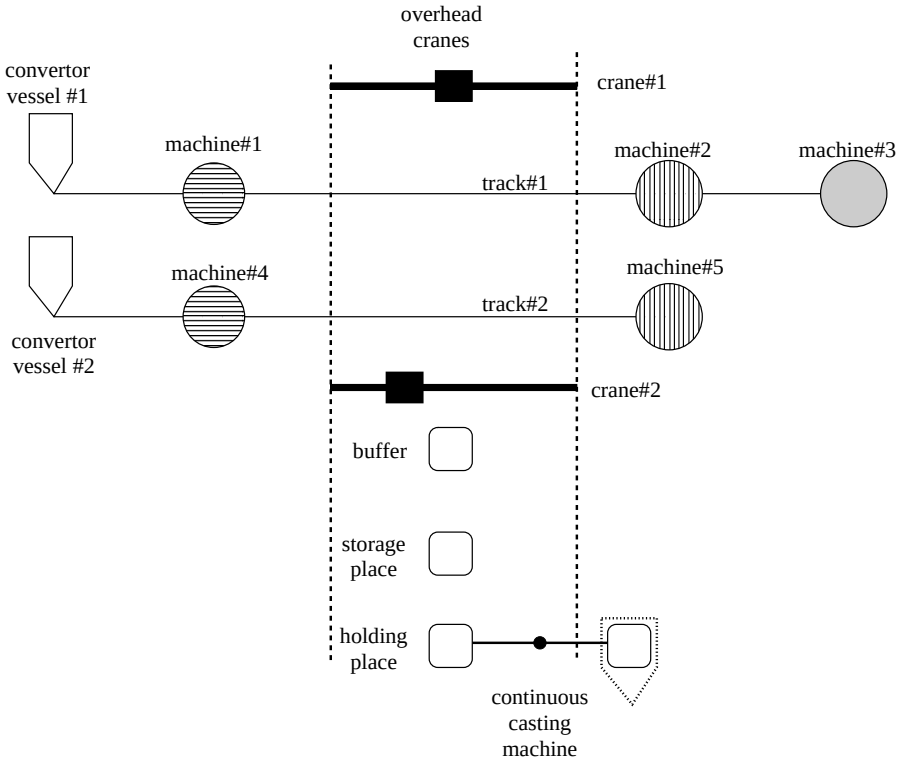

Figure 1: Layout of the plant

At first sight this problem looks similar to a job-shop scheduling problem. In job-shop scheduling theory one supposes that there are a number of jobs and machines. A job is usually defined as a sequence of operations which have to be executed in a given order [AvLLU94]. Each of these operations is performed by a particular machine for a given period of time. It is also assumed that each machine can perform only one operation at the same time. Sometimes it is also assumed that each job can perform a particular operation only once or that machines are not allowed to be idle in between two operation [Fre82]. The problem is to schedule the operations in a way that it minimizes the time it takes to complete all jobs. There are many efficient and fast local search algorithms like simulated annealing or tabu search or approaches like branch and bound algorithms to tackle different classes of the job shop problem.

In case study 5 none of the assumptions mentioned in the previous paragraph hold. Even worse: due to the topology of the plant there are operations on certain machines that prevent operations of other jobs on other machines. This machines are not accessible during that operation. There are also resources that move, jobs that can not wait indefinitely for a machine to become free and operations that do not have a fixed duration. In this context it can even be difficult to decide whether a feasible schedule exists.

Timed Automata (TA) have proven to be a useful formalism to model and verify real-time systems. Timed Automata, due to Alur and Dill [AD94], are finite state automata with clock variables. This formalism can be used to model real-time requirements of systems in a natural way. In recent years several tools for automatic

model checking based on timed automata became available, such as UPPAAL and KRONOS. Several case studies have proven that these tools can be used for realistic applications.

In this paper we give a model of the steel plant at SIDMAR, using UPPAAL's networks of timed automata. The model describes the behavior that satisfies the given constraints. We use this model to search for a behavior that does complete the jobs in a prescribed order. To do so we will use the UPPAAL model checker. To be sure that a controller can effectively realize this behavior we remove uncontrollable nondeterminism in the model. Doing this the model just describes a subset of the behavior of the plant.

Our model is based on two descriptions; a Petri net model by René Boel en Geert Stremersch [BS98] and a textual description from SIDMAR [SID98]. We will try to use the same terminology. In this paper a job will be called batch, the order of operations is defined by a recipe and most operations will be called treatments. In the next section a informal description of the plant is given, followed by a UPPAAL model in section 3. The last section discusses some results.

2 Plant description

2.1 Layout of the plant

The part of the plant we consider in this case study consists of two convertor vessels, five machines, one storage place, one buffering place, two normal tracks, a two overhead cranes and one continuous casting machine. Figure 1 gives an impression of the layout. Every load pig iron is transported in a steel ladle. The pig iron enters the system at one of the the convertor vessels, where it is poured portion-wise in steel ladles. Depending on the quality which has to be produced the pig iron undergoes different treatments, moving through the system. The steel leaves the system at the continuous casting machine, after which the empty ladle is transported to the storage place. The casting machine consists of two parts, a holding place and the casting machine itself, and works like a merry-go-round.

Ladles can move along the two normal tracks autonomously. Moving ladles from one track to another or to the buffer, storage place or holding place involves the overhead cranes. Whenever a ladle has to be moved for example from machine#3 to machine#4, an empty crane has to be available. Transport of the empty ladle from the casting machine to the storage place, also involves a crane.

2.2 Recipes

The steel quality depends, as mentioned before, on the the order of the different treatments. Machine#1 and machine#4 are identical and so are machine#2 and machine#5. Hence we have three types of machines that can perform different types of treatments. A treatment is defined by the time a ladle stays at a machine of a certain type. For each duration the recipe gives an upper and lower bound. Depending on the quality there is also an upper bound on the total amount that a load can stay in the system. Normally a recipe involves at most four treatments. All recipes start with a treatment on machine#1 or machine#4. Next, all recipes require

a treatment on machine#2 or machine#5. Some recipes also require a treatment at machine#3 and a final treatment at machine#2 or machine#5.

2.3 Timing constraints

Except for the timing constraints which are imposed by the recipes there are three other types of timing constraints. First, whenever a ladle is filled at the convertor vessel, it needs some time before the next load pig iron can be tapped at this vessel. Second, the cranes need some time to pick and drop a ladle, and to move from one position to another. Finally, a ladle has to wait for a certain amount of time in the holding place before it is allowed to enter the casting machine. After being casted the ladle waits for the same amount of time, before it is leaving the holding place. The duration of casting a ladle is controllable within known bounds. All other times are unspecified.

2.4 Other constraints

In a machine and between two machines there is at most one ladle. These positions can therefore serve as short term buffer. Since each position can hold at most one ladle, there is no possibility for ladles to pass each other. No ladle can move for example from the convertor# 1 to machine#2 if another ladle is at machine#1. The cranes cannot pass each other. This also implies that only one crane can reach the bottom section where the ladles enter the continuous casting machine. The buffer between the second track and the storage place can hold at most five ladles. It can be used to pass a ladle from one crane to the other.

A ladle can only leave the casting machine if there is already a filled ladle at the holding place (except for the case it is the last ladle). As soon as the pouring out of a steel ladle has been completed, this ladle is removed and casting of a new ladle starts. The casting machine works like a merry-go-round. Filling the continuous casting machine with a filled ladle happens synchronously with filling the holding place with the empty ladle. This guarantees that the casting machine is continuously in use.

2.5 Objective

The foregoing constraints define the safety requirements of the plant. To meet the economic constraints we want that the different steel qualities enter the continuous casting machine in a predefined order. To begin with we are just interested in a method that finds for a given order (of at most 30 batches) a detailed schedule that realizes this order. In a later stage we would like to have a method that gives an optimized schedule with a minimized production time. It should also be possible to change the model easily. Suppose a crane fails, while the plant is in use, a new schedule that takes this into account has to be computed.

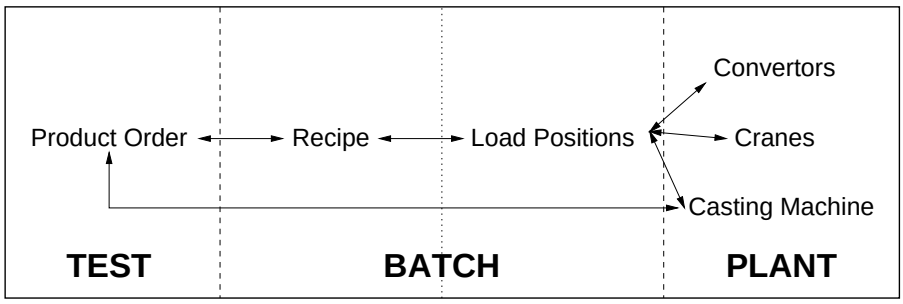

Figure 2: Synchronization of the TAs

3 The model

We will use UPPAAL's input language to model the SIDMAR steel plant. Systems in UPPAAL are modeled as networks of timed automata. The different components of the system are modeled as timed automata [AD94]. The automata are combined using UPPAAL's operation of parallel composition with binary (handshake) synchronization. Two transitions in different automata can synchronize, if they have the same label and one of them is suffixed with "!" and the other with "?". If there is more than one possible pair, the choice is made nondeterministically. Note however that in our model synchronization is symmetric, therefore no meaning is attached to "!" or "?".

Since in our model more than two components need to synchronize, we will use UPPAAL's concept of committed locations. Whenever a committed location is entered no delay is allowed and the next action transition must be an outgoing transition of that location. The names of committed locations are prefixed with `C:` or `c:`. To ensure mutual exclusion we use binary integer arrays. A process is only allowed to enter a certain location if a corresponding bit is zero. It then sets the bit to one and releases it as soon as it leaves the location. For a more detailed introduction to UPPAAL see [LPY97].

3.1 Overview

In the model we distinguish between batches and the plant. The state of a batch is defined by the position of the load in the plant and the position within the recipe and the time that has passed since the load entered the system. We have for each batch a single TA that keeps track of the load. We use two binary integer arrays, one array for each track, to ensure that at most one ladle can be at a position. This TA resembles the layout of the plant. Secondly, we have a linear (non branching) automaton for the recipe that determines the duration and order of the treatments. This automaton also takes care that the load cannot spend more time in the system than specified. We will give more details in the next sections.

In parallel with the TAs we have a test automaton. This is a linear automaton that reaches its final state[1] only, if the ladles enter the casting machine in the prescribed order. The recipes synchronize with the test automaton before entering the

[1] This is not a final state in the usual sense, but just a state called `final`

casting machine. The recipes synchronize with the positions of the ladle to ensure that a treatment is performed by the proper machine. Figure 2 shows which TAs synchronize with each other.

By introducing the test automaton we translate the question whether a feasible schedule exists into the question whether the state `final` is reachable. We use the diagnostic trace to obtain the schedule. Hence, we assume that all times are either deterministic or controllable. However the original description [BS98] says that the duration of a treatment is only known within certain bounds. Fortunately each machine can also function as short term buffer. In our model we take the upper bound as duration of the treatment. If the operation finishes earlier, we let the wait until the upper bound has expired. Doing this we remove a main source of nondetermism. The durations of the treatments in the obtained schedule are considered as combined times for treatment and waiting.

3.2 A small example

In this subsection we give a model of a simplified plant (figure 3) to illustrate the basic ideas of the complete model. We assume that three batches of steel of two qualities must be produced. There is only one track with two machines, a storage place and a simple casting machine. We assume that there are four positions `i0, i3, i2` and `i3` that can store a load of pig iron for some time. At the first two positions we have machines machine#1 en machine#2. Empty ladles can enter a storage place from the third position. The last position `i3` is connected to the casting machine. Transportation along the track is autonomous and untimed.

The processes `loadB1`, `loadB2` and `loadB3` keep track of the position of the ladle in the plant. We use a binary array `posI` of length 4 to ensure that there is at most one ladle at a particular position. A ladle moves e.g. from `i2` to `i3` only if `posI[3]==0`. When it takes the transition to `i3` the assignment `posI[3]:=1` is made. In this way mutual exclusion is build into the model.

We assume that we have two machines. These machines can function as buffer and also perform a certain type of treatment. As long as this position is used as buffer the load is at location `i0` or `i1`, respectively. The locations `machine1` and `machine2` corresponds to loads that undergo a treatment. When e.g. a treatment of type 2 on machine 2 starts, the transition `B1T2on` in the recipe automaton synchronizes with a transition in automaton `loadB1` from location `i2` to location `machine2`.

The recipes automata `recipeB1-B3` are divided into two parts. The transitions from location `r0` to `rend` and the invariants on locations `r1`,..,`rend` determine the duration of the treatments, and the total amount of time that a load may stay in the system. The transitions from location `rend` to `terminus` consider the casting of the ladle and its transport to the storage place.

Ladles can enter the casting machine at position `i3`. Initially the casting machine is empty. If there is a ladle at position `i3`, the casting machine `casting` can leave location empty and enter location full by transition `turn`. Recall that the casting machine works like a merry-go-round. As soon as a second ladle enters location `i3` the machine may turn again. The empty ladle leaves the casting machine via transition `nrut` and the full ladle enters it via `turn`. Note that both transitions are performed as one atomic step since location `busy` is committed. Analogously transition `turn` is

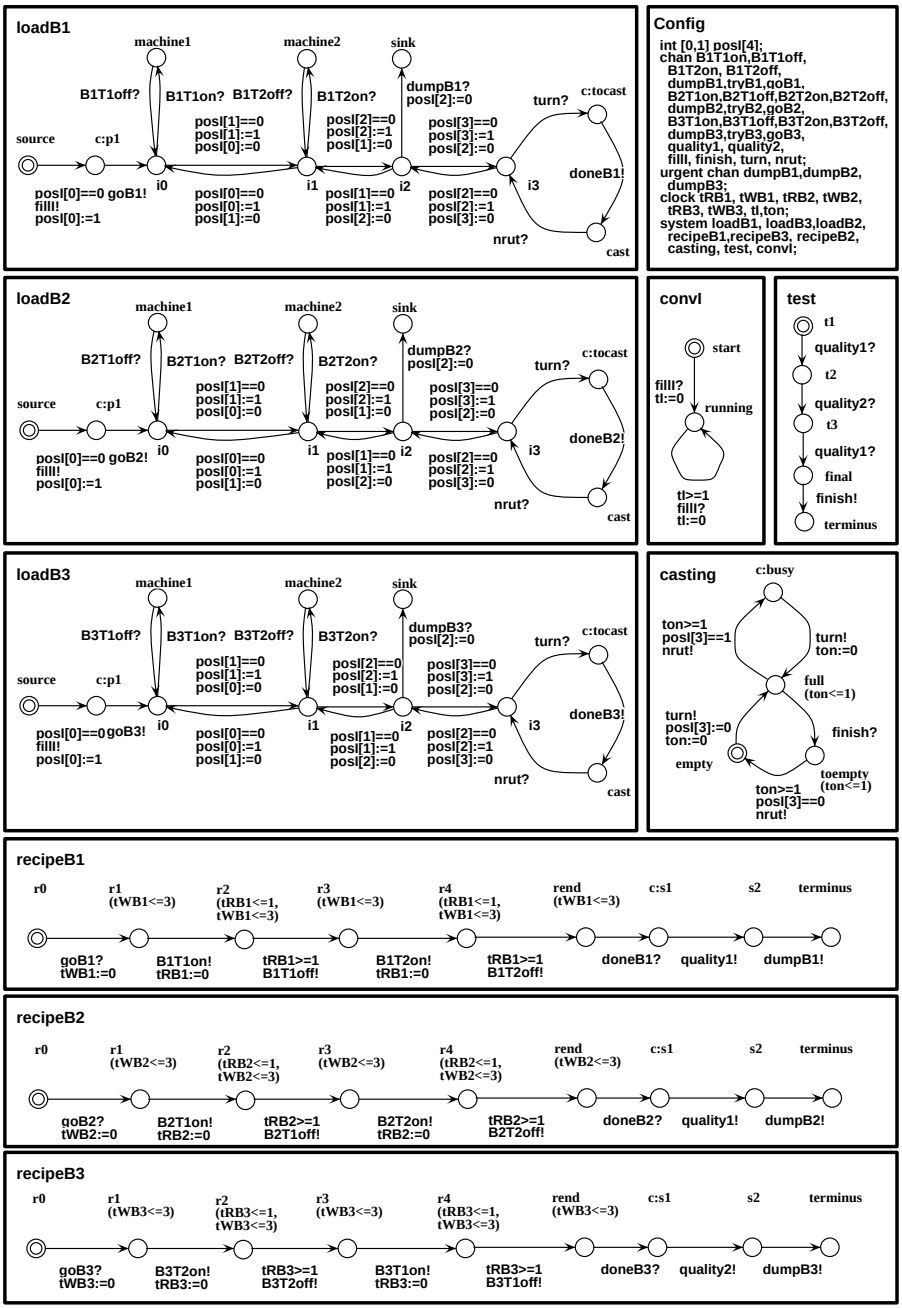

Figure 3: Model of a simplified plant

succeeded by a transition `doneB1-B3`, which in turn is succeeded by either transition `quality1` or `quality2`, without any intervening delay or interleaving. By this cascade of transitions, connected by committed locations, we guarantee that the ladle has completed its recipe and that its quality was scheduled when it enters the casting machine. Casting a load lasts exactly one time unit; this is enforced by the invariant `ton<=1` and the guard `ton>=1`.

When a ladle is empty it may not pass the storage place without being stored.

Therefore the transitions `dumpB1-B3` are declared to be urgent. They will be taken as soon as they are enabled. When the test automaton has reached the state `final`, we know for sure that the recipes were completed in the predefined order. Finally, the transition `finish` allows the casting machine to release the last ladle.

In this example we modeled three batches of pig iron. There are two different steel qualities. The first one requires 1 time unit at machine#1 and 1 time unit at machine#2. The recipe for the second quality asks for treatments at machine#2 and machine#1, both taking one time unit. Filled ladles are not allowed to stay for more than three time units in the system. This is guaranteed in the recipe automata by the invariants `tWB1<=3`, `tWB2<=3` and `tWB3<=3` respectively. The desired order is first one ladle of quality 1, then one ladle of quality 2 and finally one ladle of quality 1.

The model is not deadlock free. When for example a load has been in the system for exactly 4 time units, and then enters location `machine1` or `machine2` a deadlock may occur. But this is not a problem of poor modeling. As soon as we find a trace without time-locks that reaches the final state, we have found a schedule. Time- and deadlocks are part of the model and rule out infeasible schedules.

3.3 The model in pieces

The largest automata in the full model are the automata that represent the position of the load in the system (figure 5). The locations `i0` to `i5` represent positions on the first track. The second track is represented by locations `ii0` to `ii3`. Mutual exclusion is guaranteed by the use of two binary arrays `posI` and `posII`. The loads are moved by the cranes along locations `c0` to `c5`. To do this the loads synchronize with the automata for the cranes. Location c5 is connected to the casting machine. According to the specification the ladle waits before entering and after leaving the casting machine. Hence, we included transitions `waitin` and `waitout`.

Each of the cranes is modeled by one automaton (figure 7). Location `c0bot` to `c5bot` model positions in which the crane is empty. The locations `c0top` to `c5top` correspond to positions in which the the crane is in use. When a filled crane moves from one position to another it synchronizes with the corresponding ladle via a `move` label. Lifting and lowering a ladle takes time and the corresponding position in the load automata, `i2` or `ii2`, are occupied during that operation. The labels `incast` en `outcast` make the cranes synchronize with the casting machine. The assignment `park:=park+1` and the guard `park<5` ensure that at most five ladles are stored in the buffer.

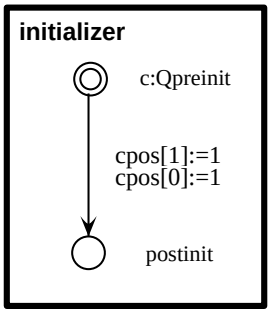

Figure 4: Automata that initializes `cpos`

The array `cpos` is used to ensure that the two cranes can not pass each other. Initially both cranes are in different positions. In UPPAAL it is however not possible to initialize binary arrays to values different from zero. Therefore we include the automaton `initializer` (figure 4).

The automaton `casting` models the continuous casting machine (figure 6). It is essentially the same as the automaton in the example, but it also takes care for the

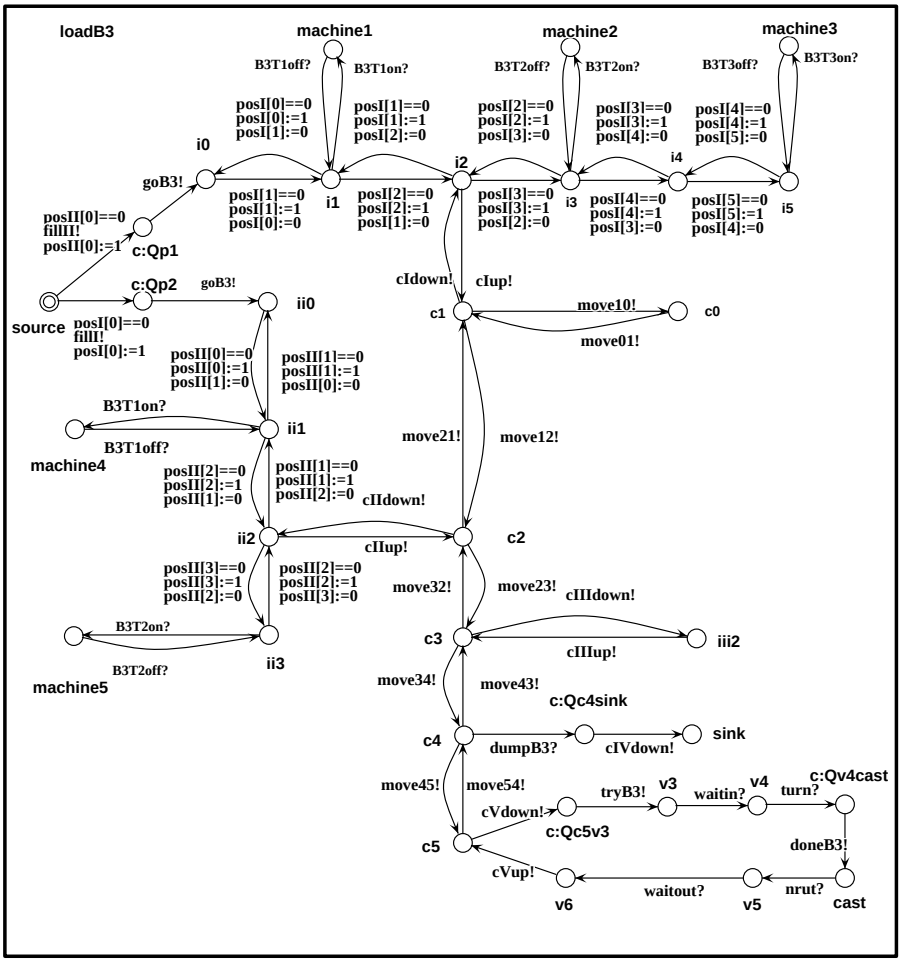

Figure 5: A timed automaton modeling a single load

waiting time before and after casting the ladle. Transition `incast` e.g. resets the clock, and when the waiting time expires transition `waitin` is taken. The casting machine synchronizes via `incast` and `outcast` with the crane to ensure that ladles enter only empty holding places. Casting a ladle takes 20 to 30 time units. This is modeled by the guard `ton>=20` and the invariant ton<=30.

The convertors, the test automaton and the recipes are the same as in the example. The recipes may of course depend on the quality of steel, and the test automaton on the given order. The durations of the treatments in the full model range between 10 and 30 time units. The upper bound on the total amount of time that a load of steel is allowed to stay in system is about 120 time units. Note that all durations in this model are just rough estimation of the actual durations in the real plant.

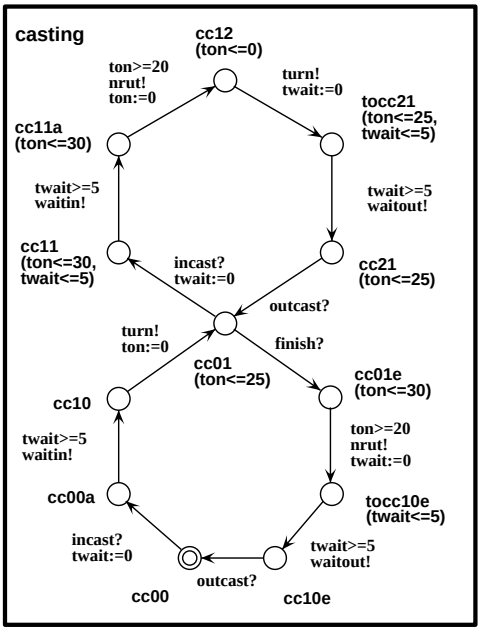

Figure 6: TA modeling the casting machine

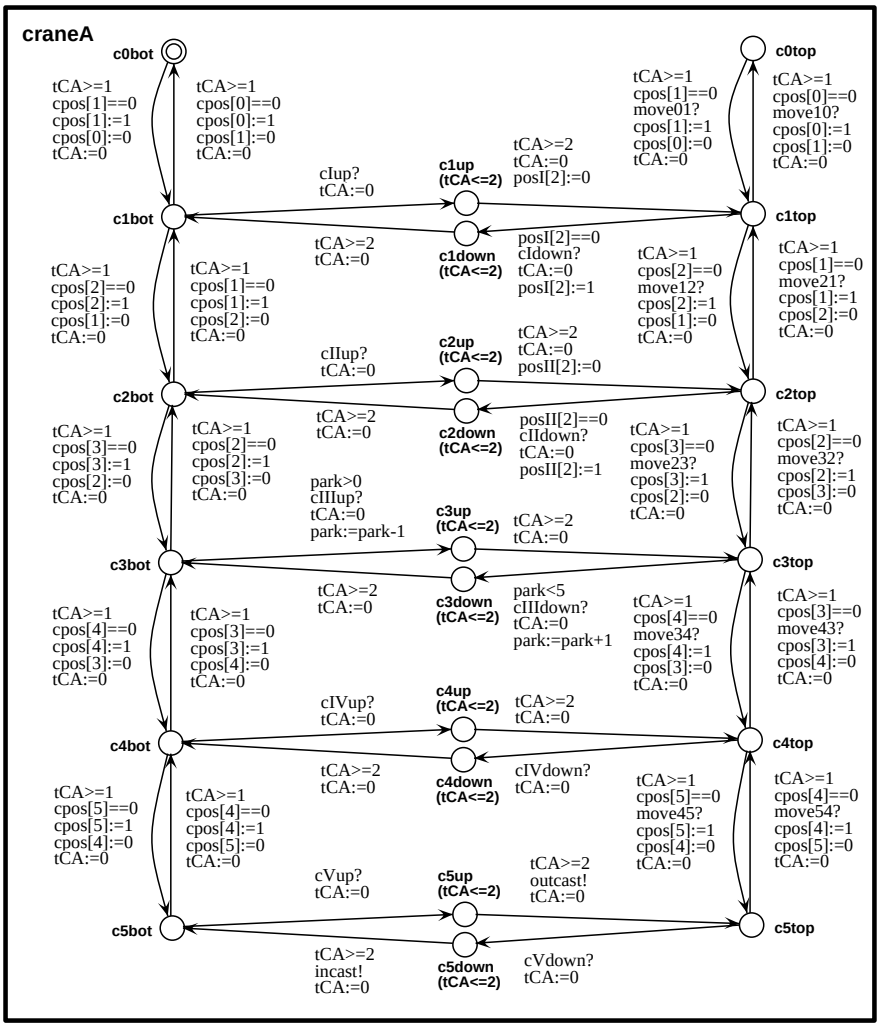

Figure 7: Except for their own clocks the cranes are modelled identical.

4 Results and Conclusion

The model described in the previous section was used to some experiments on a Sun Sparc 20 with 384Mb memory. We used the depth-first model checking algorithm from UPPAAL version 2.17. We computed the minimal makespan, i.e. found an optimal schedule with respect to total amount of time for the example in section 3.2. To do so we asked the UPPAAL to verify

```
E<> loadB1.sink and loadB2.sink and loadB3.sink and klok<=6
E<> loadB1.sink and loadB2.sink and loadB3.sink and klok<=5
```

The clock variable `klok` is used to define the upper bound on the makespan. We needed 6 minutes CPU time to compute that we need at least 6 time units to complete all recipes and store the ladles. Table 1 shows the result of the verification.

We were able to generate a schedule for a model with three batches and just one crane (figure 8) in less than one minute CPU time. We requested that first a load of quality 2 should be produced and then two loads of quality 1. Steel of the first quality needs a treatment of 30 time units on either machine#1 or machine#4 and than a treatment of 30 minutes at machine#2 or machine#5. The recipes for steel of quality start the same, and add a final treatment of 10 time units at machine#3.

We asked UPPAAL to verify `E<> test.final`. Figure 9 shows the automatically generated schedule. It starts with filling a load on the first track. This load then undergoes its treatment on the machines on this track (time units 0 to 70). Then the second load is tapped at the convertor on the second track. This load undergoes its treatment on the second track, while the first load is transported to the casting machine to wait, and the last load is tapped on the second track(time units 70 to 120). Then while the first load is casted, the second load transported is to the casting machine and the third load is treated on track 2 (time units 120 to 150). Finally load B3 is transported to the storing place, load B1 is casted and load B2 finishes it recipe and enters the casting machine. This schedule is of course not optimal, and modest variations to this model made verification even impossible.

In UPPAAL a forward reachability algorithms is implemented [LPY97],[YPD93]. We also did some premature experiments with backward analysis techniques which seem to be promising. The most restricting constraints in this case study is the requirement that the casting machine is continuous in use and that the ladles should enter the casting machine in a predefined order. The experiments yielded encouraging results; finding a schedule for a model with two cranes and up to seven batches was possible.

We want to stress that our approach does not intend to compete with local search algorithms for the job-shop scheduling problem. The advantage of our attempt is that we can use well known algorithms and a powerful formalism to model a scheduling problem. We can add topological and timing constraints easily without being forced to change the underlying algorithms. We also hope to profit from new developments in the field of model checking.

Acknolegements Thanks to the people at BRICS Aalborg and Thomas Hune for the fruitful discussions on my UPPAAL model, and to the people from the SYSTeMS Group Ghent en Manfred van Vlierberghe from SIDMAR for answering all my questions about the plant.

```
Property 1 (line 1) is satified
Showing example trace.
loadB1.source loadB3.source loadB2.source
Sync:  fillI
loadB1.source loadB3.source loadB2.p1
Sync:  goB2
loadB1.source loadB3.source loadB2.i0
Sync:  B2T1on
loadB1.source loadB3.source loadB2.machine1
delay(1)
loadB1.source loadB3.source loadB2.machine1
Sync:  B2T1off
loadB1.source loadB3.source loadB2.i0
loadB1.source loadB3.source loadB2.i1
loadB1.source loadB3.source loadB2.i2
loadB1.source loadB3.source loadB2.i3
Sync:  fillI
loadB1.source loadB3.p1 loadB2.i3
Sync:  goB3
loadB1.source loadB3.i0 loadB2.i3
loadB1.source loadB3.i0 loadB2.i2
loadB1.source loadB3.i1 loadB2.i2
Sync:  B3T2on
loadB1.source loadB3.machine2 loadB2.i2
loadB1.source loadB3.machine2 loadB2.i3
delay(1)
loadB1.source loadB3.machine2 loadB2.i3
Sync:  B3T2off
loadB1.source loadB3.i1 loadB2.i3
loadB1.source loadB3.i0 loadB2.i3
loadB1.source loadB3.i0 loadB2.i2
loadB1.source loadB3.i0 loadB2.i1
Sync:  B2T2on
loadB1.source loadB3.i0 loadB2.machine2
Sync:  B3T1on
loadB1.source loadB3.machine1 loadB2.machine2
delay(1)
loadB1.source loadB3.machine1 loadB2.machine2
Sync:  B2T2off
loadB1.source loadB3.machine1 loadB2.i1
loadB1.source loadB3.machine1 loadB2.i2
loadB1.source loadB3.machine1 loadB2.i3
Sync:  turn
loadB1.source loadB3.machine1 loadB2.tocast
Sync:  doneB2
loadB1.source loadB3.machine1 loadB2.cast
Sync:  quality1
loadB1.source loadB3.machine1 loadB2.cast
Sync:  B3T1off
loadB1.source loadB3.i0 loadB2.cast
loadB1.source loadB3.i1 loadB2.cast
loadB1.source loadB3.i2 loadB2.cast
loadB1.source loadB3.i3 loadB2.cast
Sync:  fillI
loadB1.p1 loadB3.i3 loadB2.cast
Sync:  goB1
loadB1.i0 loadB3.i3 loadB2.cast
loadB1.i0 loadB3.i2 loadB2.cast
loadB1.i0 loadB3.i1 loadB2.cast
Sync:  B1T1on
loadB1.machine1 loadB3.i1 loadB2.cast
delay(1)
loadB1.machine1 loadB3.i1 loadB2.cast
Sync:  B1T1off
loadB1.i0 loadB3.i1 loadB2.cast
loadB1.i0 loadB3.i2 loadB2.cast
loadB1.i0 loadB3.i3 loadB2.cast
Sync:  nrut
loadB1.i0 loadB3.i3 loadB2.i3
Sync:  turn
loadB1.i0 loadB3.tocast loadB2.i3
Sync:  doneB3
loadB1.i0 loadB3.cast loadB2.i3
Sync:  quality2
loadB1.i0 loadB3.cast loadB2.i3
loadB1.i0 loadB3.cast loadB2.i2
Urgent sync:  dumpB2
loadB1.i0 loadB3.cast loadB2.sink
loadB1.i1 loadB3.cast loadB2.sink
Sync:  B1T2on
loadB1.machine2 loadB3.cast loadB2.sink
delay(1)
loadB1.machine2 loadB3.cast loadB2.sink
Sync:  B1T2off
loadB1.i1 loadB3.cast loadB2.sink
loadB1.i2 loadB3.cast loadB2.sink
loadB1.i3 loadB3.cast loadB2.sink
Sync:  nrut
loadB1.i3 loadB3.i3 loadB2.sink
Sync:  turn
loadB1.tocast loadB3.i3 loadB2.sink
Sync:  doneB1
loadB1.cast loadB3.i3 loadB2.sink
Sync:  quality1
loadB1.cast loadB3.i3 loadB2.sink
Sync:  finish
loadB1.cast loadB3.i3 loadB2.sink
loadB1.cast loadB3.i2 loadB2.sink
Urgent sync:  dumpB3
loadB1.cast loadB3.sink loadB2.sink
delay(1)
loadB1.cast loadB3.sink loadB2.sink
Sync:  nrut
loadB1.i3 loadB3.sink loadB2.sink
loadB1.i2 loadB3.sink loadB2.sink
Urgent sync:  dumpB1
loadB1.sink loadB3.sink loadB2.sink

Property 2 (line 2) is NOT satisfied.
```

Table 1: UPPAAL verified that `E<> loadB1.sink and loadB2.sink and loadB3.sink and klok<=6` does hold but `E<> loadB1.sink and loadB2.sink and loadB3.sink and klok<=5` not. We ommitted the clock valuations and part of the state information

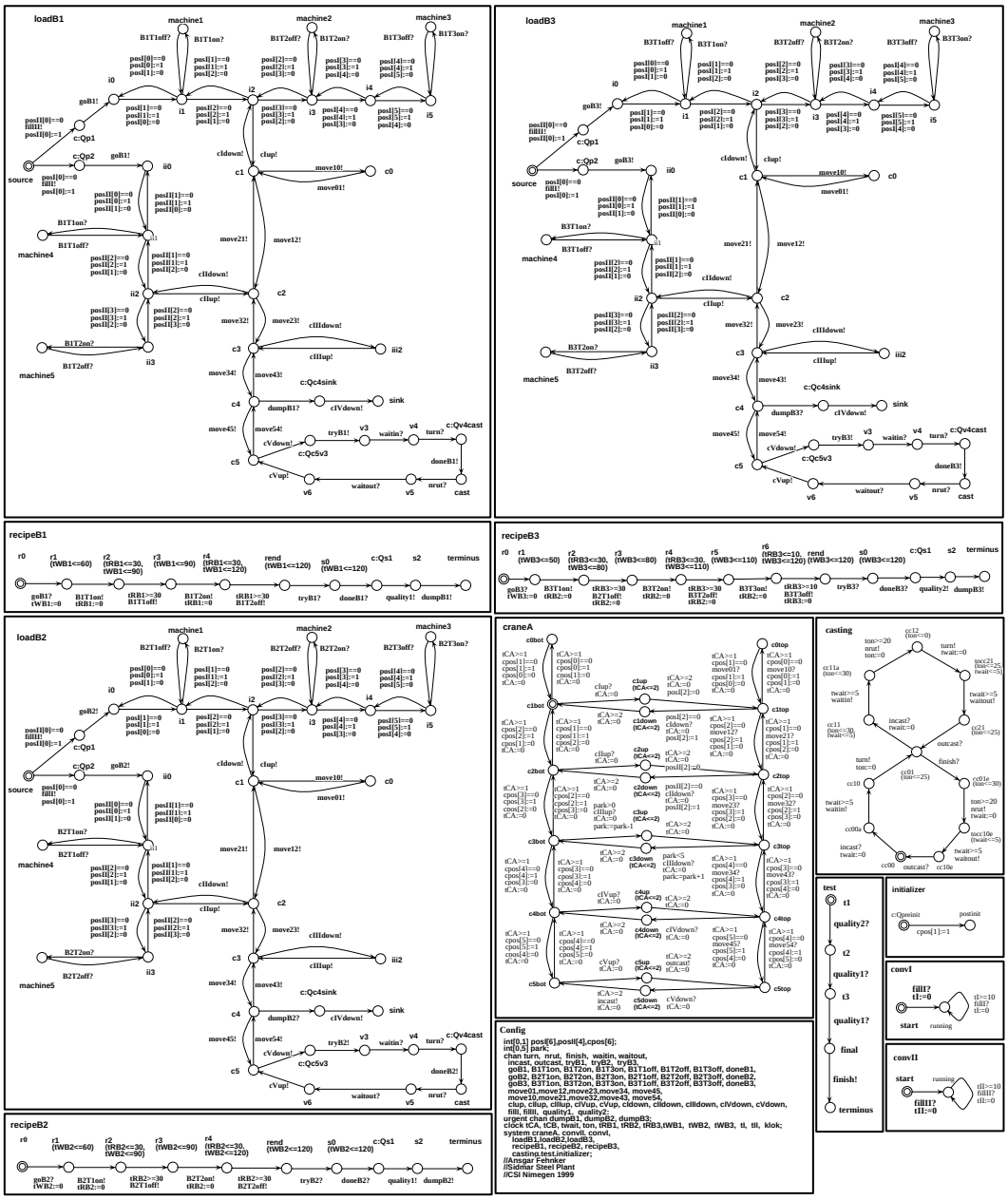

Figure 8: Model of a plant with three loads and one crane

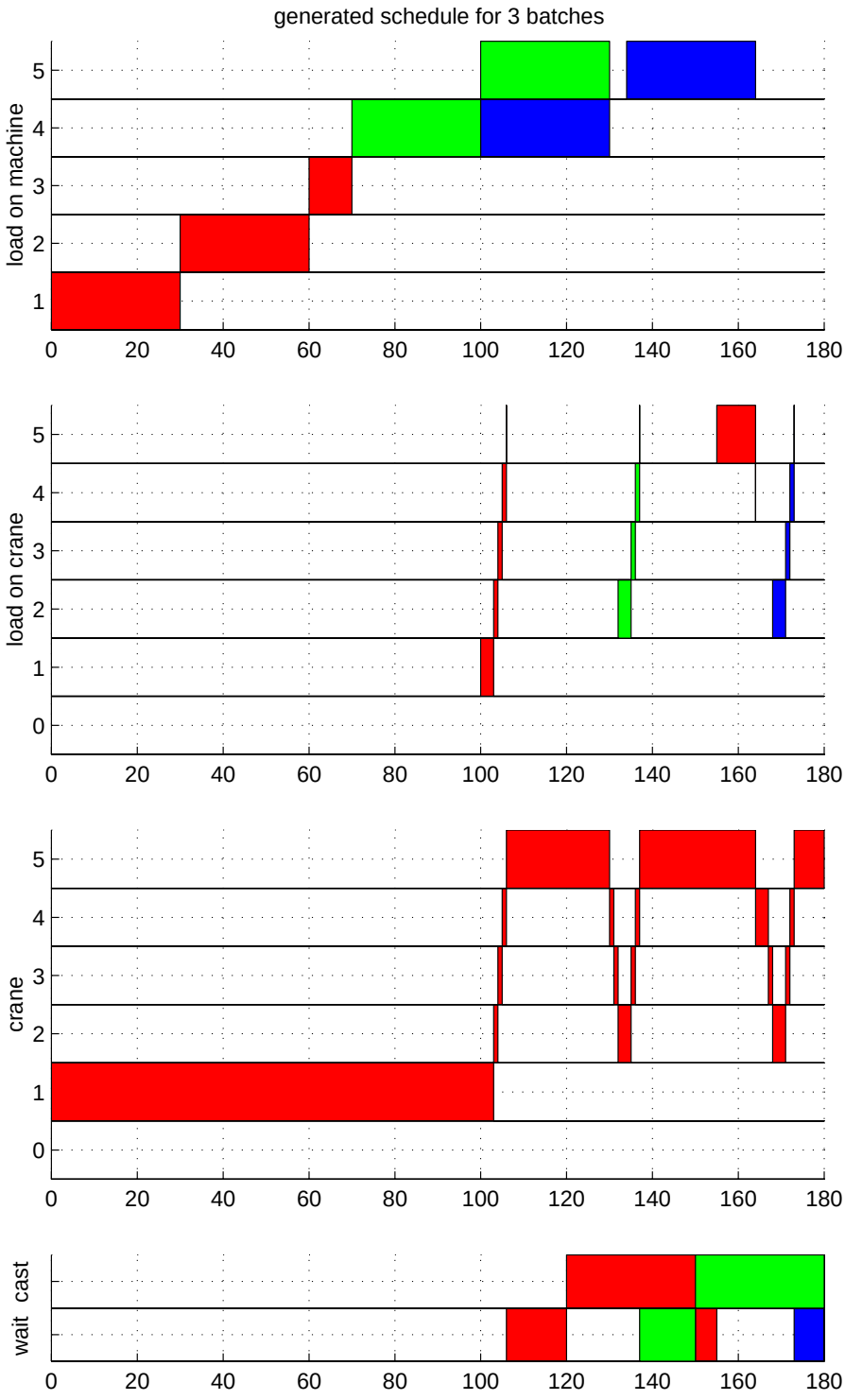

Figure 9: Schedule obtained from an automatically generated trace.

References

[AD94] R. Alur and D.L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AvLLU94] E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra, and N.L.J. Ulder. A Computational Study of Local Search Algorithms for Job-Shop Scheduling. *OSRA Journal on Computing*, 6(2):118–125, Spring 1994.

[BS98] René Boel and Geert Stremersch. Report for VHS: Timed Petri Net Model of Steel Plant at SIDMAR. Technical report, SYSTeMS Group,University Ghent, 1998.

[Fre82] Simon French. *Sequencing and Scheduling: An Introduction to the Mathematicas of the Job-Shop*. Mathematics and its Application. Ellis Horwood, Chichester, England, 1982.

[LPY97] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1(1+2):134–152, 1997.

[SID98] N.V. SIDMAR. Planung und Synchronisation der Anlagen im Stahlwerk SIDMAR. Distributed among VHS-participants, October 1998.

[YPD93] Wang Yi, Paul Pettersson, and Mats Daniels. Automatic Verification of Real-time Communicating Systems by Constraint Solving. In *Proc. pf 5th Int Conf. on CAV*, LNCS 697, pages 210–224, 1993.