Chapter 3

SAT-based Verification

Model checking [CGP99, BK08] is the problem of automatically verifying (proving or disproving) whether a system conforms to its specification, given as a set of properties/constraints. Model checking usually consists of enumerating all reachable configurations of the system, and then checking whether the properties hold in these. Yet, the infinite state-space of real-time systems leads to severe limitations in scalability, even in very well-established model checkers like Uppaal [upp]. Especially for the verification of safety properties of real-time systems [CBRZ01, BCC$^+$03], Bounded Model Checking (BMC, [BCCZ99]) has turned out to be amongst the most promising approaches. Safety properties declare what should not happen—or equivalently, what should always happen—and are typically expressed as reachability properties. Safety properties can be disproved with a finite counterexample, i.e., a finite run, where the last configuration contains a contradiction to the property. The principle of BMC for safety properties is to examine prefix fragments of the transition system, and successively increase the exploration bound until it reaches (a computable indicator of) the diameter of the system—in which case the system has been proven safe—or an unsafe run has been discovered [ACKS02].

In this chapter, for each of the system models defined in the previous chapter (i.e., TA, TCA, TNA), we present an encoding in propositional logic, plus linear arithmetic on the rational numbers, which is tailored towards BMC. A *satisfiability check* on the resulting formula (SAT solving) using SMT solvers[1] like FOCI [FOC] or MathSAT [mat], is then used to find possible runs of the system.

The main idea is that for any system model $\mathfrak{S}$, with $\mathfrak{S}\in\{\mathfrak{A}, \mathfrak{T}, \mathfrak{N}\}$ (cf. Definitions 2.2.1, 2.3.1 and 2.4.2), we define a formula $\varphi(\mathfrak{S})$. This formula encodes the transition characteristics of $\mathfrak{S}$, that means, the possibilities to evolve to the next step $\mathtt{t{+}1}$, based on the configuration in the current step $\mathtt{t}$. For BMC, we unfold

[1] Satisfiability Modulo Theory (SMT) problems combine propositional satisfiability with an underlying theory, for example the theory of linear arithmetic over the real numbers. Atoms in an SMT problem can consist of propositional variables or theory atoms, and are combined with the Boolean connectives.

the formula $\varphi(\mathfrak{S})$ k times, i.e., we instantiate the "abstract" indices $\mathtt{t}$ and $\mathtt{t+1}$ for all steps 1 up to bound k, which yields a variant $\varphi(\mathfrak{S})_k$. Intuitively, a satisfying interpretation of the formula $\varphi(\mathfrak{S})_k$ corresponds to a run of the associated LTS $\mathfrak{S}_{\mathfrak{S}}$, i.e., to one possible behaviour for the first k steps. Consequently, the set of all possible valuations of $\varphi(\mathfrak{S})_k$ corresponds to the complete possible behaviour of $\mathfrak{S}$ for the first k steps.

The rest of this Chapter is organised as follows: in Section 3.1, we define the formula representation. As in the previous Chapter, we start with a general part (Section 3.1.1), and then discuss in detail the formula representation of the different system models and their products (Sections 3.1.2 to 3.1.4). In Section 3.2, we present the unfolding for BMC, and discuss some issues related to BMC. We discuss other possibilities for encoding, and motivate our design decisions in Section 3.3, and conclude the Chapter in Section 3.4.

3.1 Formula Representation

The possible behaviour (i.e., which transition can be taken) of a real-time system $\mathfrak{S}$ depends on the current system configuration (location, clock valuation, data variables, events, ports, memory cells) and changes over time. This time-dependent behaviour needs to be reflected by the formula $\varphi(\mathfrak{S})_k$. Therefore, we "parametrise" the variables representing the constituents of $\mathfrak{S}$ by the step $\mathtt{t}$ they are evaluated in. This is called *localisation*: the localisation ψ_t of a formula ψ is obtained by adding index t to all variable symbols occurring in ψ. Thus, if ψ is of vocabulary x, s, p, then ψ_t is of vocabulary $\mathtt{x_t}, \mathtt{s_t}, \mathtt{p_t}$.

In the next section, we present the representation for constituents common to more than one real-time system: clocks, locations, events, ports, and data variables/memory cells. In Sections 3.1.2, 3.1.3 and 3.1.4, we introduce the specific transition characteristics of TA, TCA and TNA, respectively.

3.1.1 Preliminaries

We now show how to represent constituents common to more than one real-time system.

3.1.1.1 Clocks, Clock Constraints

Let $\mathcal{X}$ be the set of clocks in $\mathfrak{S}$. For the representation of clocks, we first introduce a fresh clock z, called *absolute time reference*, which is not used in any clock constraint and which is never updated, thus, the value of z increases constantly. This clock is used to measure the absolute amount of time that has passed since the beginning of computation: for any step $\mathtt{t}$, the rational variable $\mathtt{z_t}$ (called *representation of* z) represents the value of z in step $\mathtt{t}$, i.e., the absolute amount of time which has passed from the beginning of computation up to step $\mathtt{t}$. For every clock $x \in \mathcal{X}$, the rational variable $\mathtt{x_t}$ (*clock reference (of clock* x*)*) is used to compute the value of clock x in step t, which is given by the difference $\mathtt{z_t} - \mathtt{x_t}$. Thus, for clock constraints $cc = x \sim n$ and

$cc'{=}x{-}y{\sim}n$ (cf. Definition 2.1.2), the formulas $\mathtt{cc_t}{=}\mathtt{z_t}{-}\mathtt{x_t}{\sim}n$ and $\mathtt{cc'_t}{=}\mathtt{y_t}{-}\mathtt{x_t}{\sim}n$,[2] (called *representation of cc and cc'*, respectively) evaluate to `true` iff cc and cc' hold in step `t`. The representation of other clock constraints is straightforward, by using conjunctions of the above constraints.

The underlying idea of clock references is that the variable $\mathtt{x_t}$ will keep its value as long as clock x is not updated in $\mathfrak{S}$. When clock x is updated, there are two possibilities (cf. Definition 2.1.5): either x is updated to a natural number $n{\in}\mathbb{N}$, or x is updated to the value of another clock x'. In the former case, the value of $\mathtt{x_t}$ is set to $\mathtt{z_t}{-}n$, in the latter case, it is set to the value of $\mathtt{x'_t}$. In both cases, the difference $\mathtt{z_t}{-}\mathtt{x_t}$ yields the correct value of x. This temporal difference representation significantly improves the SAT solving performance, due to the decreased number of arithmetic operations, see Section 3.3 for a more detailed discussion.

We illustrate the idea describe above in Figure 3.1. Above the line representing the value of z, we denote the updates, as found on transitions of $\mathfrak{S}$. Below the time line (x-axis), we denote the formulas which are used to set $\mathtt{x_t}$ to the correct value.

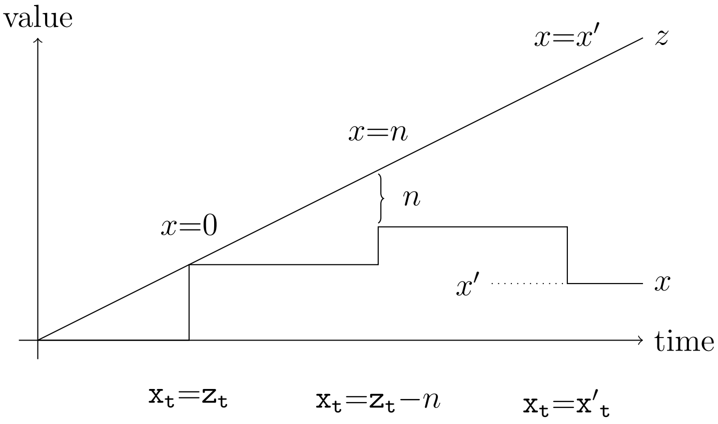
value
$x{=}x'$
z
$x{=}n$
n
$x{=}0$
x'
x
time
$\mathtt{x_t}{=}\mathtt{z_t}$
$\mathtt{x_t}{=}\mathtt{z_t}{-}n$
$\mathtt{x_t}{=}\mathtt{x'_t}$

Figure 3.1: Representation of Clock Values, Concept

By definition of the allowed updates for a clock x, the value of $\mathtt{x_t}$ is always smaller than the value of $\mathtt{z_t}$. The value of $\mathtt{x_t}$ can become negative in case of an update $\lambda(x){=}n$, with $n{>}\mathtt{z_t}$ (or, obviously, in case x is updated to the value of another clock x' where $\mathtt{x'_t}$ is already negative).

3.1.1.2 Locations

Let S be the set of locations of $\mathfrak{S}$. We use a linear Boolean encoding for locations: for every location $s{\in}S$, the Boolean variable $\mathtt{s_t}$ (called *representation of s*) represents whether $\mathfrak{S}$ is in location s in step `t`. Please refer to Section 3.3 for a discussion of other possible encodings.

3.1.1.3 Data Values

To represent the (possibly infinite, countable) set of data values $\mathcal{D}ata$ for TCA and TNA, we use an injective mapping $\Delta{:}\mathcal{D}ata{\to}\mathbb{Z}$, which maps each element $\mathbb{d}_i{\in}\mathcal{D}ata$,

[2] Actually, the formula is $((\mathtt{z_t}{-}\mathtt{x_t}){-}(\mathtt{z_t}{-}\mathtt{y_t})){\sim}n$, but this simplifies to $\mathtt{y_t}{-}\mathtt{x_t}{\sim}n$.

$\mathbb{d}_i{\neq}\bot$, to an integer number $\mathtt{n^i}$ (called *representation of* $\mathbb{d}_i$). If a total order $\leqslant$ exists on $\mathcal{D}ata$ (cf. Definition 2.1.7), we require Δ to preserve this total order, i.e., for all $\mathbb{d}_i, \mathbb{d}_j{\in}\mathcal{D}ata$ such that $\mathbb{d}_i{\neq}\mathbb{d}_j$, if $\mathbb{d}_i \leqslant \mathbb{d}_j$, then $\Delta(\mathbb{d}_i){=}\mathtt{n^i}{<}\mathtt{n^j}{=}\Delta(\mathbb{d}_j)$. For the *representation of* $\bot$, we introduce an integer constant, denoted by $\mathtt{n}^{\bot}$, without assigning a specific value to it; see Remark 3.1.5 for further explanations. For explanatory purposes, we treat $\mathtt{n}^{\bot}$ similarly to $\mathtt{n^i}$, i.e., as if it was an element of $\mathbb{Z}$.

3.1.1.4 Events, Ports, Data Variables

To represent the set of events Σ of a TA $\mathfrak{A}$, we use a linear Boolean encoding: for every event $\mathfrak{a}{\in}\Sigma$, the Boolean variable $\alpha_\mathtt{t}$ (called *representation of* $\mathfrak{a}$) represents whether $\mathfrak{A}$ executes a transition in step $\mathtt{t}$ that is labelled with $\mathfrak{a}$. Please refer to Section 3.3 for a discussion on other possible encodings of events.

The basic idea for ports (in TCA or TNA) is the same as for events: for every port $p{\in}\mathcal{P}$, the Boolean variable $\mathtt{p_t}$ (called *port activity variable of* p) represents whether port p is active in step $\mathtt{t}$. In addition, to encode which data value is transmitted over an active port, for every port $p{\in}\mathcal{P}$, we introduce an integer variable $\mathtt{Dp_t}$ (called *port data variable*), which represents the data value pending on p in step $\mathtt{t}$. If p is inactive in step $\mathtt{t}$, $\mathtt{Dp_t}$ evaluates $\mathtt{n}^{\bot}$; see Remark 3.1.5 for further explanations.

For data variables, we use a similar set of variables, though with a slightly different meaning: for every data variable $d{\in}\mathcal{D}$, we introduce a Boolean variable $\mathtt{d_t}$, and an integer variable $\mathtt{Dd_t}$. The variable $\mathtt{d_t}$ (called *data fullness variable*) is used to indicate whether d is full in step $\mathtt{t}$. As for ports, the variable $\mathtt{Dd_t}$ (called *data content variable*) represents which data value (according to mapping Δ) is contained in d in step $\mathtt{t}$, in case d is not empty, $\mathtt{Dd_t}$ evaluates to $\mathtt{n}^{\bot}$ if d is empty in step $\mathtt{t}$.

Though the encoding using two variables per port/data variable might seem unnecessary, we need this for efficient abstraction. Please refer to Section 4.1 for more details.

3.1.1.5 Data Constraints

For a data constraint $dc{=}(\mathbf{D}{\simeq}\mathbf{D}')$, $\simeq\in\{=,\leqslant\}$ (cf. Definition 2.1.7), the formula $\mathtt{dc_t}$ (the *representation of* dc) evaluates to $\mathtt{true}$ iff dc holds in step $\mathtt{t}$. The constraint $\mathtt{dc_t}$ is defined by replacing ports and data values occurring in dc with their corresponding representations; i.e., replace $p{\in}\mathcal{P}|_{dc}$ by $\mathtt{Dp_t}$, and $\mathbb{d}_i{\in}\mathcal{D}ata|_{dc}$ by $\Delta(\mathbb{d}_i){=}\mathtt{n}^i$. For data variables $d{\in}\mathcal{D}|_{dc}$, we need to take into account whether they are used by the source or the target location of the transition to which dc belongs (i.e., whether they occur as $s.d$ or as $t.d$ in dc, cf. Remarks 2.3.2 and 2.4.3), or only in dc itself. In the latter two cases, we replace $d{\in}\mathcal{D}$ by $\mathtt{Dd_t}$, in the first case, we replace d by $\mathtt{Dd_{t\text{-}1}}$. This corresponds to the fact that d can be used by the source location only *before* the execution of the transition, i.e., at step $\mathtt{t}{-}\mathtt{1}$. The representation of other data constraints is straightforward, using conjunctions and negations of the aforementioned.

3.1.2 Timed Automata

The representation of the transition relation of TA has to model both action and delay transitions, cf. Section 2.2. It constrains the possible valuations of variables representing the automaton configuration at subsequent step $\mathtt{t}{+}\mathtt{1}$ depending on those at $\mathtt{t}$. Recalling the representation of clocks, locations and events from the previous section, the representation of TA is defined as follows.

Definition 3.1.1 (TA Representation). Let $\mathfrak{A}$ be a TA, with initial location $\bar{s}$,[3] let $e{=}(s, \mathfrak{a}, cc, \lambda, s')$ be a transition in $\mathfrak{A}$. The *formula representation of the transition relation of* $\mathfrak{A}$, denoted $\varphi(\mathfrak{A})$, is defined in (3.7) of Table 3.2.

$$\varphi^{init}(\mathfrak{A}) = \bar{\mathtt{s}}_0 \wedge \bigwedge_{s\in S, s\neq\bar{s}} \neg\mathtt{s}_0 \wedge \mathtt{I}(\bar{\mathtt{s}})_{\mathtt{0}} \wedge \bigwedge_{\mathfrak{a}\in\Sigma} \neg\alpha_0 \wedge (\mathtt{z}_0{=}0) \wedge \bigwedge_{x\in\mathcal{X}} (\mathtt{x}_0{=}0) \tag{3.1}$$

$$\begin{aligned}\varphi^{action}(e) = {} & \mathtt{s}_{\mathtt{t}} \wedge \alpha_{\mathtt{t+1}} \wedge \mathtt{cc}_{\mathtt{t}} \wedge (\mathtt{z}_{\mathtt{t}}{=}\mathtt{z}_{\mathtt{t+1}}) \wedge \bigwedge_{\lambda(x)=id} (\mathtt{x}_{\mathtt{t+1}}{=}\mathtt{x}_{\mathtt{t}}) \wedge \\ & \bigwedge_{\lambda(x)=x'} (\mathtt{x}_{\mathtt{t+1}}{=}\mathtt{x'}_{\mathtt{t+1}}) \wedge \bigwedge_{\lambda(x)=n} (\mathtt{x}_{\mathtt{t+1}}{=}\mathtt{z}_{\mathtt{t+1}}{-}n) \wedge \mathtt{s'}_{\mathtt{t+1}} \wedge \mathtt{I}(\mathtt{s'})_{\mathtt{t+1}}\end{aligned} \tag{3.2}$$

$$\varphi^{delay}(s) = \mathtt{s}_{\mathtt{t}} \wedge \bigwedge_{\mathfrak{a}\in\Sigma} \neg\alpha_{\mathtt{t+1}} \wedge (\mathtt{z}_{\mathtt{t}}{\leq}\mathtt{z}_{\mathtt{t+1}}) \wedge \bigwedge_{x\in\mathcal{X}} (\mathtt{x}_{\mathtt{t}}{=}\mathtt{x}_{\mathtt{t+1}}) \wedge \mathtt{s}_{\mathtt{t+1}} \wedge \mathtt{I}(\mathtt{s})_{\mathtt{t+1}} \tag{3.3}$$

$$\varphi^{trans}(\mathfrak{A}) = \bigvee_{e\in E} \varphi^{action}(e) \vee \bigvee_{s\in S} \varphi^{delay}(s) \tag{3.4}$$

$$\varphi^{location}(\mathfrak{A}) = \bigvee_{s\in S} (\mathtt{s}_{\mathtt{t+1}} \wedge \bigwedge_{s'\in S, s'\neq s} \neg\mathtt{s}'_{\mathtt{t+1}}) \tag{3.5}$$

$$\varphi^{mutex}(\mathfrak{A}) = \bigvee_{\mathfrak{a}\in\Sigma} (\alpha_{\mathtt{t+1}} \wedge \bigwedge_{\mathfrak{a}'\in\Sigma, \mathfrak{a}'\neq\mathfrak{a}} \neg\alpha'_{\mathtt{t+1}}) \vee \bigwedge_{\mathfrak{a}\in\Sigma} (\neg\alpha_{\mathtt{t+1}}) \tag{3.6}$$

$$\varphi(\mathfrak{A}) = \varphi^{init}(\mathfrak{A}) \wedge \varphi^{trans}(\mathfrak{A}) \wedge \varphi^{location}(\mathfrak{A}) \wedge \varphi^{mutex}(\mathfrak{A}) \tag{3.7}$$

Table 3.2: Transition Relation Representation of TA

The idea of these formulas is as follows: the TA starts in its initial location (3.1), the invariant of which has to hold (by $\mathtt{I}(\mathtt{s})_{\mathtt{t}}$, we denote the localisation of the invariant $I(s)$ of some location s, cf. Section 3.1), no action is enabled, and all clocks start with value 0. Before executing an action transition $e{=}(s, \mathfrak{a}, cc, \lambda, s'){\in}E$ of step $\mathtt{t}{+}\mathtt{1}$ in (3.2), the automaton is in location s (at step $\mathtt{t}$), and the transition guard $\mathtt{cc}_{\mathtt{t}}$ is satisfied. On occurrence of event $\alpha_{\mathtt{t+1}}$, the transition fires. The value of the absolute time reference does not change (action transitions are instantaneous), other clocks are updated according to their value under update map λ (they either keep their value, are set to $\mathtt{z}_{\mathtt{t}}{-}n$ if $\lambda(x){=}n$, cf. Section 3.1.1.1, or get the value of another clock reference $\mathtt{x'}_{\mathtt{t}}$). After the execution (at step $\mathtt{t}{+}\mathtt{1}$), the automaton is in location s', the invariant of which has to hold. For a delay transition (3.3), the automaton remains in location s, the value of the absolute time reference increases, all clock references keep their value (there is no update, cf. (2.2)), no event $\mathfrak{a}{\in}\Sigma$ must occur, and the invariant has to hold after the time delay. Due to convexity, the invariant of the target location in both action and delay transitions only needs to be checked at the end of the transition/delay (that means at step $\mathtt{t}{+}\mathtt{1}$), as it inductively holds at

[3] To avoid confusion with localisation indices, we denote the initial location as $\bar{s}$ rather than s_0, so its representation is $\bar{\mathtt{s}}_0$ rather than the odd-locking $(\mathtt{s}_0)_{\mathtt{0}}$.

the beginning (3.1). The disjunction of these formulas expresses (nondeterministic) transition choice (3.4). In any step, the current location and event are unique (mutual exclusion of location (3.5) and event variables (3.6)), to prevent $\varphi(\mathfrak{A})$ from following multiple transitions simultaneously.

Example 3.1.2 (TA Representation). Consider again the intelligent light switch presented in Figure 2.1 (Example 2.2.2). Let $\mathfrak{A}$ be the name of the automaton, let e_1, e_2, e_3, e_4 refer to the transitions from *off* to *light*, *light* to *off*, *light* to *bright*, and *bright* to *off*, respectively. The representation of $\mathfrak{A}$ is shown in Table 3.3. We omit constraints equal to `true`, like for example clock guards or empty conjunctions.

$$\begin{aligned}
\varphi^{init}(\mathfrak{A}) &= \texttt{off}_0\wedge\neg\texttt{light}_0\wedge\neg\texttt{bright}_0\wedge\neg\texttt{press}_0\wedge\neg\tau_0\wedge(\texttt{z}_0{=}0)\wedge(\texttt{x}_0{=}0)\\
\varphi^{action}(e_1) &= \texttt{off}_\texttt{t}\wedge\texttt{press}_{\texttt{t+1}}\wedge(\texttt{z}_\texttt{t}{=}\texttt{z}_{\texttt{t+1}})\wedge(\texttt{x}_{\texttt{t+1}}{=}\texttt{z}_{\texttt{t+1}})\wedge\texttt{light}_{\texttt{t+1}}\\
\varphi^{action}(e_2) &= \texttt{light}_\texttt{t}\wedge\texttt{press}_{\texttt{t+1}}\wedge(\texttt{z}_\texttt{t}{=}\texttt{z}_{\texttt{t+1}})\wedge(\texttt{z}_\texttt{t}{-}\texttt{x}_\texttt{t}{>}3)\wedge(\texttt{x}_{\texttt{t+1}}{=}\texttt{x}_\texttt{t})\wedge\texttt{off}_{\texttt{t+1}}\\
\varphi^{action}(e_3) &= \texttt{light}_\texttt{t}\wedge\texttt{press}_{\texttt{t+1}}\wedge(\texttt{z}_\texttt{t}{=}\texttt{z}_{\texttt{t+1}})\wedge(\texttt{z}_\texttt{t}{-}\texttt{x}_\texttt{t}{\leq}3)\wedge(\texttt{x}_{\texttt{t+1}}{=}\texttt{x}_\texttt{t})\wedge\texttt{bright}_{\texttt{t+1}}\\
\varphi^{action}(e_4) &= \texttt{bright}_\texttt{t}\wedge\texttt{press}_{\texttt{t+1}}\wedge(\texttt{z}_\texttt{t}{=}\texttt{z}_{\texttt{t+1}})\wedge(\texttt{x}_{\texttt{t+1}}{=}\texttt{x}_\texttt{t})\wedge\texttt{off}_{\texttt{t+1}}\\
\varphi^{delay}(\mathit{off}) &= \texttt{off}_\texttt{t}\wedge\neg\texttt{press}_{\texttt{t+1}}\wedge\neg\tau_{\texttt{t+1}}\wedge(\texttt{z}_\texttt{t}{\leq}\texttt{z}_{\texttt{t+1}})\wedge(\texttt{x}_\texttt{t}{=}\texttt{x}_{\texttt{t+1}})\wedge\texttt{off}_{\texttt{t+1}}\\
\varphi^{delay}(\mathit{light}) &= \texttt{light}_\texttt{t}\wedge\neg\texttt{press}_{\texttt{t+1}}\wedge\neg\tau_{\texttt{t+1}}\wedge(\texttt{z}_\texttt{t}{\leq}\texttt{z}_{\texttt{t+1}})\wedge(\texttt{x}_\texttt{t}{=}\texttt{x}_{\texttt{t+1}})\wedge\texttt{light}_{\texttt{t+1}}\\
\varphi^{delay}(\mathit{bright}) &= \texttt{bright}_\texttt{t}\wedge\neg\texttt{press}_{\texttt{t+1}}\wedge\neg\tau_{\texttt{t+1}}\wedge(\texttt{z}_\texttt{t}{\leq}\texttt{z}_{\texttt{t+1}})\wedge(\texttt{x}_\texttt{t}{=}\texttt{x}_{\texttt{t+1}})\wedge\texttt{bright}_{\texttt{t+1}}\\
\varphi^{trans}(\mathfrak{A}) &= \ \varphi^{action}(e_1)\vee\varphi^{action}(e_2)\vee\varphi^{action}(e_3)\vee\varphi^{action}(e_4)\vee\\
&\varphi^{delay}(\mathit{off})\vee\varphi^{delay}(\mathit{bright})\vee\varphi^{delay}(\mathit{light})\\
\varphi^{location}(\mathfrak{A}) &= \ (\texttt{off}_{\texttt{t+1}}\wedge\neg\texttt{light}_{\texttt{t+1}}\wedge\neg\texttt{bright}_{\texttt{t+1}})\vee\\
&(\texttt{light}_{\texttt{t+1}}\wedge\neg\texttt{off}_{\texttt{t+1}}\wedge\neg\texttt{bright}_{\texttt{t+1}})\vee\\
&(\texttt{bright}_{\texttt{t+1}}\wedge\neg\texttt{off}_{\texttt{t+1}}\wedge\neg\texttt{light}_{\texttt{t+1}})\\
\varphi^{mutex}(\mathfrak{A}) &= (\texttt{press}_{\texttt{t+1}}\wedge\neg\tau_{\texttt{t+1}})\vee(\tau_{\texttt{t+1}}\wedge\neg\texttt{press}_{\texttt{t+1}})\vee(\neg\texttt{press}_{\texttt{t+1}}\wedge\neg\tau_{\texttt{t+1}})\\
\varphi(\mathfrak{A}) &= \varphi^{init}(\mathfrak{A})\wedge\varphi^{trans}\wedge\varphi^{location}(\mathfrak{A})\wedge\varphi^{mutex}(\mathfrak{A})
\end{aligned}$$

Table 3.3: Transition Relation Representation of TA: Example

The product of TA, as defined in Definition 2.2.8, in the worst case is exponential in the size of the underlying TA. We now present a representation of the product which is *linear* in the size of the underlying TA. The basic idea is to retain the representations of the individual automata, and define the product as their juxtaposition.

Definition 3.1.3 (TA Product Representation). Let $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be TA, with $\mathcal{X}_1{\cap}\mathcal{X}_2{=}\emptyset$ and $S_1{\cap}S_2{=}\emptyset$, let $\varphi(\mathfrak{A}_1)$ and $\varphi(\mathfrak{A}_2)$ be the respective representations, as defined in Definition 3.1.1, with (3.3) replaced by (3.3'). The *formula representation* $\varphi(\mathfrak{A}_1{\bowtie}\mathfrak{A}_2)$ *of the product* $\mathfrak{A}_1{\bowtie}\mathfrak{A}_2$ is defined in (3.9).

$$\varphi^{delay}(s) = \mathtt{s}_\mathtt{t}\wedge \bigwedge_{\mathfrak{a}\in\Sigma_v} \neg\alpha_{\mathtt{t+1}}\wedge(\mathtt{z}_\mathtt{t}\leq\mathtt{z}_{\mathtt{t+1}})\wedge \bigwedge_{x\in\mathcal{X}} (\mathtt{x}_\mathtt{t}=\mathtt{x}_{\mathtt{t+1}})\wedge\mathtt{s}_{\mathtt{t+1}}\wedge\mathtt{I(s)}_{\mathtt{t+1}} \tag{3.3'}$$

$$\varphi^{mutex}(\mathfrak{A}_1\bowtie\mathfrak{A}_2) = \bigvee_{\mathfrak{a}\in\Sigma_1\setminus\Sigma_2} (\alpha_{\mathtt{t+1}}\wedge \bigwedge_{\mathfrak{a}'\in\Sigma_2\setminus\Sigma_1} \neg\alpha'_{\mathtt{t+1}})\vee \bigvee_{\mathfrak{a}\in\Sigma_2\setminus\Sigma_1} (\alpha_{\mathtt{t+1}}\wedge \bigwedge_{\mathfrak{a}'\in\Sigma_1\setminus\Sigma_2} \neg\alpha'_{\mathtt{t+1}}) \tag{3.8}$$

$$\varphi(\mathfrak{A}_1\bowtie\mathfrak{A}_2) = \varphi(\mathfrak{A}_1)\wedge\varphi(\mathfrak{A}_2)\wedge\varphi^{mutex}(\mathfrak{A}_1\bowtie\mathfrak{A}_2) \tag{3.9}$$

The product representation (3.9) faithfully models the intended behaviour of the product of TA, as defined in Definition 2.2.8: to ensure that the event occurring in step $\mathtt{t}$ is unique within the system, we add the constraint on mutual exclusion between events that are local to one of the TA (3.8), shared events are already dealt with in (3.6).

3.1.3 Timed Constraint Automata

The main ideas of the representation of the transition relation of TCA are similar to the representation of TA, as defined in the previous section. In particular, the modelling of locations and clocks is identical. Yet, the representation needs to take care of the special behaviour of TCA, namely, that every visible transition is preceded by a positive time delay, whereas invisible transitions may be instantaneous. Conceptually, on execution of a transition, the delay is represented by evolving from step $\mathtt{t}$ to step $\mathtt{t+1}$, while the (instantaneous) location change takes place at $\mathtt{t+1}$.

To correctly represent these delayed transitions (cf. (2.7)) in the associated LTS $\mathfrak{S}_\mathfrak{T}$ (cf. Definition 2.3.5), we need a second type of clock constraints. Clock constraints in (2.7) are evaluated under two different valuations: the invariant $I(s')$ of the target location s' is evaluated under $(\nu{+}t)[\lambda]$, that means *after* the time delay $(+t)$ and *after* the execution (λ) of the transition. In contrast, the invariant $I(s)$ of the source location s and the clock guard cc of the transition are evaluated under $(\nu{+}t)$, that means *after* the passage of time, but *before* the execution of the transition. To access the clock values at this particular point in time "in the middle" of the execution step, we define the *inter-step representation* $\mathtt{cc}_{\mathtt{t}\Delta}$ *of a clock constraint cc* . For $cc = (x{\sim}n)$, the inter-step representation is given by $\mathtt{cc}_{\mathtt{t}\Delta} = \mathtt{z}_{\mathtt{t+1}}{-}\mathtt{x}_\mathtt{t}{\sim}n$. Note that for a clock constraint $cc' = (x{-}y{\sim}n)$, the inter-step representation is equivalent to the representation of cc' as defined in Section 3.1.1.1, since this representation does not contain the absolute time reference $\mathtt{z}$ anymore, and delaying does not change the difference of x and y.

We are now ready to define the formula representation of TCA.

Definition 3.1.4 (TCA Representation). Let $\mathfrak{T}$ be a TCA, with initial location $\bar{s}$ (as before, we denote the initial location as $\bar{s}$ rather than s_0), let $e{=}(s, P, dc, cc, \lambda, s')$ and $e'{=}(s, \emptyset, dc, cc, \lambda, s')$ be a visible and invisible transition in $\mathfrak{T}$, respectively. The *formula representation of the transition relation of* $\mathfrak{T}$, denoted $\varphi(\mathfrak{T})$,[4] is defined in (3.16) in Table 3.4.

[4] Without confusion, we use the same formula identifiers for all real-time systems. For example, we use φ^{init} to denote the initial constraints for TA (3.1), TCA (3.10), and (in the next section) TNA (3.20).

$$\begin{aligned}
\varphi^{init}(\mathfrak{T}) =\ & \bar{\mathtt{s}}_0\wedge \bigwedge_{s\in S,s\neq\bar{s}} \neg\mathtt{s}_0\wedge\mathtt{I}(\bar{\mathtt{s}})_{\mathtt{0}}\wedge \bigwedge_{p\in\mathcal{P}}(\neg\mathtt{p}_0\wedge(\mathtt{Dp}_{\mathtt{0}}=\mathtt{n}^{\perp}))\wedge \\
& \bigwedge_{d\in\mathcal{D}}(\neg\mathtt{d}_0\wedge(\mathtt{Dd}_0=\mathtt{n}^{\perp}))\wedge(\mathtt{z}_0=0)\wedge\bigwedge_{x\in\mathcal{X}}(\mathtt{x}_0=0) && (3.10)\\
\varphi^{visible}(e) =\ & \mathtt{s}_{\mathtt{t}}\wedge\mathtt{I}(\mathtt{s})_{\mathtt{t\Delta}}\wedge\bigwedge_{p\in P}\mathtt{p}_{\mathtt{t+1}}\wedge\bigwedge_{p\notin P}\neg\mathtt{p}_{\mathtt{t+1}}\wedge\bigwedge_{d\notin\#(s')}\neg\mathtt{d}_{\mathtt{t+1}}\wedge\mathtt{dc}_{\mathtt{t+1}}\wedge && (3.11)\\
& \mathtt{cc}_{\mathtt{t\Delta}}\wedge(\mathtt{z}_{\mathtt{t}}<\mathtt{z}_{\mathtt{t+1}})\wedge\bigwedge_{\lambda(x)=id}(\mathtt{x}_{\mathtt{t+1}}=\mathtt{x}_{\mathtt{t}})\wedge\bigwedge_{\lambda(x)=x'}(\mathtt{x}_{\mathtt{t+1}}=\mathtt{x'}_{\mathtt{t+1}})\wedge \\
& \bigwedge_{\lambda(x)=n}(\mathtt{x}_{\mathtt{t+1}}=\mathtt{z}_{\mathtt{t+1}}-n)\wedge\mathtt{s'}_{\mathtt{t+1}}\wedge\mathtt{I}(\mathtt{s'})_{\mathtt{t+1}} \\
\varphi^{invisible}(e') =\ & \mathtt{s}_{\mathtt{t}}\wedge\mathtt{I}(\mathtt{s})_{\mathtt{t\Delta}}\wedge\bigwedge_{p\in\mathcal{P}}\neg\mathtt{p}_{\mathtt{t+1}}\wedge\bigwedge_{d\notin\#(s')}\neg\mathtt{d}_{\mathtt{t+1}}\wedge\mathtt{dc}_{\mathtt{t+1}}\wedge\mathtt{cc}_{\mathtt{t\Delta}}\wedge && (3.12)\\
& (\mathtt{z}_{\mathtt{t}}\leq\mathtt{z}_{\mathtt{t+1}})\wedge\bigwedge_{\lambda(x)=id}(\mathtt{x}_{\mathtt{t+1}}=\mathtt{x}_{\mathtt{t}})\wedge\bigwedge_{\lambda(x)=x'}(\mathtt{x}_{\mathtt{t+1}}=\mathtt{x'}_{\mathtt{t+1}})\wedge \\
& \bigwedge_{\lambda(x)=n}(\mathtt{x}_{\mathtt{t+1}}=\mathtt{z}_{\mathtt{t+1}}-n)\wedge\mathtt{s'}_{\mathtt{t+1}}\wedge\mathtt{I}(\mathtt{s'})_{\mathtt{t+1}} \\
\varphi^{trans}(\mathfrak{T}) =\ & \bigvee_{e\in E,P\neq\emptyset}\varphi^{visible}(e)\vee\bigvee_{e'\in E,P=\emptyset}\varphi^{invisible}(e') && (3.13)\\
\varphi^{location}(\mathfrak{T}) =\ & \bigvee_{s\in S}(\mathtt{s}_{\mathtt{t+1}}\wedge\bigwedge_{s'\in S,s'\neq s}\neg\mathtt{s}'_{\mathtt{t+1}}) && (3.14)\\
\varphi^{mutex}(\mathfrak{T}) =\ & \bigwedge_{p\in\mathcal{P}}(\neg\mathtt{p}_{\mathtt{t+1}}\vee\neg(\mathtt{Dp}_{\mathtt{t+1}}=\mathtt{n}^{\perp}))\wedge(\mathtt{p}_{\mathtt{t+1}}\vee(\mathtt{Dp}_{\mathtt{t+1}}=\mathtt{n}^{\perp}))\wedge && (3.15)\\
& \bigwedge_{d\in\mathcal{D}}(\neg\mathtt{d}_{\mathtt{t+1}}\vee\neg(\mathtt{Dd}_{\mathtt{t+1}}=\mathtt{n}^{\perp}))\wedge(\mathtt{d}_{\mathtt{t+1}}\vee(\mathtt{Dd}_{\mathtt{t+1}}=\mathtt{n}^{\perp})) \\
\varphi(\mathfrak{T}) =\ & \varphi^{init}(\mathfrak{T})\wedge\varphi^{trans}(\mathfrak{T})\wedge\varphi^{location}(\mathfrak{T})\wedge\varphi^{mutex}(\mathfrak{T}) && (3.16)
\end{aligned}$$

Table 3.4: Transition Relation Representation of TCA

The automaton starts in its initial location $\bar{s}$ (3.10) in step $\mathtt{0}$, the invariant of which has to be satisfied, data must not flow through any port, all memory cells are empty, and all clocks are set to zero. Before executing a visible transition (3.11) in step $\mathtt{t}$, $\mathfrak{T}$ is in location s. After the elapse of a positive amount of time ($\mathtt{z}_{\mathtt{t-1}}<\mathtt{z}_{\mathtt{t}}$), after which the invariant $\mathtt{I}(\mathtt{s})_{\mathtt{t\Delta}}$ of s and the clock guard $\mathtt{cc}_{\mathtt{t\Delta}}$ of the transition hold, $\mathfrak{T}$ switches to location s', the invariant of which has to hold. All clocks are updated according to their value under λ, data flows through all ports p contained in the port set P, while the other ports are inactive, and the data constraint $\mathtt{dc}_{\mathtt{t}}$ is satisfied. Memory cells which are not used by the target location s' are empty after execution of the transition, i.e. they get the value $\perp$. As for TA (cf. explanations after Definition 3.1.1), convexity allows to check the invariant at the end of the time delay only, as it inductively holds at the beginning (3.10). The execution of an invisible transition (3.12) is similar, except that the amount of time elapsed may be zero, and data must not flow through any port. The disjunction of all visible and invisible transitions expresses nondeterministic transition choice (3.13). In any step, the current location is unique (3.14), the special value "no data" may only be pending at inactive ports, and may only be "contained" in a memory cell if that memory cells is indicated to be empty (3.15).

Remark 3.1.5 (Representation of $\bot$). As explained in Section 3.1.1, we leave the value of $\mathtt{n}^{\bot}$ initially unspecified. During Bounded Model Checking (see Section 3.2 for details), if a satisfying assignment for $\varphi(\mathfrak{T})$ exists, the solver will find and assign to $\bot$ a integer value such that the constraints in Table 3.4 are satisfied.

The actual value which is assigned to $\bot$ is not important. By construction, the constraints in Table 3.4 ensure that the value is different from all $\mathtt{n}^{\mathtt{i}}$ used in the constraints, i.e., from (the representation of) all data values pending at any active port or contained in any memory cell. Since we allow $\bot$ to be used in data constraints only in combination with equality (but not with $\leqslant$, cf. Definition 2.1.7), this uniqueness ensures that a data constraint ($\mathtt{Dp}_{\mathtt{t}}{=}\bot$) is satisfied iff port p is inactive in step $\mathtt{t}$, and a data constraint ($\mathtt{Dd}_{\mathtt{t}}{=}\bot$) is satisfied iff memory cell d is empty in step $\mathtt{t}$.

For finite data domains, we can make the following improvement to $\varphi(\mathfrak{T})$.

Remark 3.1.6 (Finite Data Domain). For finite domains, i.e., with $|\mathcal{D}ata\backslash\bot|{=}k$ for some $k{\in}\mathbb{N}$, we require that Δ maps the elements of $\mathcal{D}ata$ to subsequent integer numbers, with smallest element $\Delta(\bot){=}{-1}$, such that $\Delta(\mathcal{D}ata){=}\{-1, 0, \ldots, k{-}1\}{\subset}\mathbb{Z}$. Further, we add the constraint

$$\bigwedge_{p\in\mathcal{P}} (\mathtt{Dp}_{\mathtt{t+1}}{\le}k{-}1){\wedge}(\mathtt{Dp}_{\mathtt{t+1}}{\ge}{-1}){\wedge}\bigwedge_{d\in\mathcal{D}} (\mathtt{Dd}_{\mathtt{t+1}}{\le}k{-}1){\wedge}(\mathtt{Dd}_{\mathtt{t+1}}{\ge}{-1})$$

to φ^{mutex} (3.15). This speeds up verification, since the number of possible valuations for ports and memory cells is decreased.

Example 3.1.7 (TCA Representation). Consider again the 1-bounded FIFO buffer presented in Figure 2.5. Let $\mathfrak{T}$ be the name of the automaton, let e_1, e_2, e_3 refer to the transitions from *empty* to *full*, *full* to *empty* (visible), and *full* to *empty* (invisible), respectively. The representation of $\mathfrak{T}$ is shown in Table 3.5. Again, we omit constraints equal to `true`.

We now present a *linear* representation of products of TCA, which avoids the worst case exponential blow-up of the product definition in Definition 2.3.9. As for TA, the basic idea is to define the representation of the product as the conjunction of the individual representations. We require variables representing common ports to have the same name in both representations, such that constraints involving these ports are automatically satisfied simultaneously in both representations.

To correctly model transitions described by (2.11) in Definition 2.3.9, we first need to introduce explicit delay transitions: as explained after Definition 2.3.9, in case the transition described by (2.11) is preceded by a time delay, the other automaton actually performs a delay transition. The representation of a delay transition $\varphi^{delay}(s)$ in location s is defined in (3.17).

$$\begin{aligned}
\varphi^{init}(\mathfrak{T}) =\ & \mathtt{empty}_0\wedge\neg\mathtt{full}_0\wedge\neg\mathtt{p}_0\wedge(\mathtt{Dp}_0{=}\mathtt{n}^{\perp})\wedge\neg\mathtt{q}_0\wedge(\mathtt{Dq}_0{=}\perp)\wedge\\
& \neg\mathtt{m}_0\wedge(\mathtt{Dm}_0{=}\mathtt{n}^{\perp})\wedge(\mathtt{z}_0{=}0)\wedge(\mathtt{x}_0{=}0)\\
\varphi^{visible}(e_1) =\ & \mathtt{empty}_\mathtt{t}\wedge\mathtt{p}_\mathtt{t+1}\wedge\neg\mathtt{q}_\mathtt{t+1}\wedge(\mathtt{Dp}_\mathtt{t+1}{=}\mathtt{m}_\mathtt{t+1})\wedge(\mathtt{z}_\mathtt{t}{<}\mathtt{z}_\mathtt{t+1})\wedge\\
& (\mathtt{x}_\mathtt{t+1}{=}\mathtt{z}_\mathtt{t+1})\wedge\mathtt{full}_\mathtt{t+1}\wedge(\mathtt{z}_\mathtt{t+1}{-}\mathtt{x}_\mathtt{t+1}{\leq}3)\\
\varphi^{visible}(e_2) =\ & \mathtt{full}_\mathtt{t}\wedge\mathtt{q}_\mathtt{t+1}\wedge\neg\mathtt{p}_\mathtt{t+1}\wedge(\mathtt{Dm}_\mathtt{t+1}{=}\mathtt{n}^{\perp})\wedge(\mathtt{Dq}_\mathtt{t+1}{=}\mathtt{m}_\mathtt{t})\wedge(\mathtt{z}_\mathtt{t+1}{-}\mathtt{x}_\mathtt{t}{<}3)\wedge\\
& (\mathtt{z}_\mathtt{t}{<}\mathtt{z}_\mathtt{t+1})\wedge(\mathtt{x}_\mathtt{t+1}{=}\mathtt{x}_\mathtt{t})\wedge(\mathtt{z}_\mathtt{t+1}{-}\mathtt{x}_\mathtt{t}{\leq}3)\wedge\mathtt{empty}_\mathtt{t+1}\\
\varphi^{invisible}(e_3) =\ & \mathtt{full}_\mathtt{t}\wedge(\mathtt{z}_\mathtt{t+1}{-}\mathtt{x}_\mathtt{t}{\leq}3)\wedge\neg\mathtt{p}_\mathtt{t+1}\wedge\neg\mathtt{q}_\mathtt{t+1}\wedge(\mathtt{Dm}_\mathtt{t+1}{=}\mathtt{n}^{\perp})\wedge(\mathtt{z}_\mathtt{t+1}{-}\mathtt{x}_\mathtt{t}{=}3)\wedge\\
& (\mathtt{z}_\mathtt{t}{\leq}\mathtt{z}_\mathtt{t+1})\wedge(\mathtt{x}_\mathtt{t+1}{=}\mathtt{x}_\mathtt{t})\wedge\mathtt{empty}_\mathtt{t+1}\\
\varphi^{trans}(\mathfrak{T}) =\ & \varphi^{visible}(e_1)\vee\varphi^{visible}(e_2)\vee\varphi^{invisible}(e_3)\\
\varphi^{location}(\mathfrak{T}) =\ & (\mathtt{empty}_\mathtt{t+1}\wedge\neg\mathtt{full}_\mathtt{t+1})\vee(\mathtt{full}_\mathtt{t+1}\wedge\neg\mathtt{empty}_\mathtt{t+1})\\
\varphi^{mutex}(\mathfrak{T}) =\ & (\neg\mathtt{p}_\mathtt{t+1}\vee\neg(\mathtt{Dp}_\mathtt{t+1}{=}\mathtt{n}^{\perp}))\wedge(\mathtt{p}_\mathtt{t+1}\vee(\mathtt{Dp}_\mathtt{t+1}{=}\mathtt{n}^{\perp}))\wedge\\
& (\neg\mathtt{q}_\mathtt{t+1}\vee\neg(\mathtt{Dq}_\mathtt{t+1}{=}\mathtt{n}^{\perp}))\wedge(\mathtt{q}_\mathtt{t+1}\vee(\mathtt{Dq}_\mathtt{t+1}{=}\mathtt{n}^{\perp}))\wedge\\
& (\neg\mathtt{m}_\mathtt{t+1}\vee\neg(\mathtt{Dm}_\mathtt{t+1}{=}\mathtt{n}^{\perp}))\wedge(\mathtt{m}_\mathtt{t+1}\vee(\mathtt{Dm}_\mathtt{t+1}{=}\mathtt{n}^{\perp}))\\
\varphi(\mathfrak{T}) =\ & \varphi^{init}(\mathfrak{T})\wedge\varphi^{trans}(\mathfrak{T})\wedge\varphi^{location}(\mathfrak{T})\wedge\varphi^{mutex}(\mathfrak{T})
\end{aligned}$$

Table 3.5: Transition Relation Representation of TCA: Example

$$\begin{aligned}
\varphi^{delay}(s) =\ & \mathtt{s}_\mathtt{t}\wedge\bigwedge_{p\in\mathcal{P}}\neg\mathtt{p}_\mathtt{t+1}\wedge\bigwedge_{d\in\mathcal{D}}(\mathtt{Dd}_\mathtt{t+1}{=}\mathtt{Dd}_\mathtt{t})\wedge\bigwedge_{x\in\mathcal{X}}(\mathtt{x}_\mathtt{t+1}{=}\mathtt{x}_\mathtt{t})\wedge(\mathtt{z}_\mathtt{t}{\leq}\mathtt{z}_\mathtt{t+1})\wedge\\
& \mathtt{I(s)}_{\mathtt{t}\Delta}\wedge\mathtt{s}_\mathtt{t+1}\wedge\mathtt{I(s)}_\mathtt{t+1}
\end{aligned}\tag{3.17}$$

Note that these delay transitions are in accordance with Definition 2.3.1, since they correspond to the representation of invisible transitions (cf. (3.12)) of the form $(s,\emptyset,\bigwedge_{d\in\mathcal{D}}(s.d{=}t.d),\mathtt{true},id,s)$. Therefore, in particular, (3.17) permits zero-delays.

Definition 3.1.8 (TCA Product Representation). Let $\mathfrak{T}_1$, $\mathfrak{T}_2$ be TCA, with $\mathcal{X}_1\cap\mathcal{X}_2{=}\emptyset$ and $S_1\cap S_2{=}\emptyset$, let $\varphi(\mathfrak{T}_1)$ and $\varphi(\mathfrak{T}_2)$ be the respective representations, as defined in Definition 3.1.4, with (3.13) replaced by (3.13') for $i{=}1,2$. The *formula representation* $\varphi(\mathfrak{T}_1{\bowtie}\mathfrak{T}_2)$ *of the product* $\mathfrak{T}_1{\bowtie}\mathfrak{T}_2$ is defined in (3.18).

$$\varphi^{trans}(\mathfrak{T}_i) = \bigvee_{e\in E_i,P\neq\emptyset}\varphi^{visible}(e)\vee\bigvee_{e'\in E_i,P=\emptyset}\varphi^{invisible}(e')\vee\bigvee_{s\in S_i}\varphi^{delay}(s)\tag{3.13'}$$

$$\varphi(\mathfrak{T}_1{\bowtie}\mathfrak{T}_2) = \varphi(\mathfrak{T}_1)\wedge\varphi(\mathfrak{T}_2)\wedge\bigwedge_{s\in S_1}\varphi^{delay}(s)\wedge\bigwedge_{s\in S_2}\varphi^{delay}(s)\tag{3.18}$$

The product representation (3.18) faithfully models the intended behaviour, as defined in Definition 2.3.9, but is still *linear* in the size of the underlying TCA. Note that the existence of such a linear product is not immediately clear, but in fact is a result of our design decision of explicitly mentioning all ports—active and inactive—on each transition (cf. (3.11), (3.12) and (3.17)). This decision—though seeming unnecessary at first glance—together with the assumption that common ports have the same name in both TCA, ensures that transitions in different TCA may only be executed in parallel (i.e., synchronise) if they fulfil the conditions described in Definition 2.3.9. In this way, we do not need to list all possible synchronisations (which are allowed by (2.10) and (2.11)) explicitly, and in this way avoid the exponential blow-up.

The hiding operation (cf. Definition 2.3.12) removes all information about a set of ports O from a TCA $\mathfrak{T}$. Hiding a set of ports O in the formula representation $\varphi(\mathfrak{T})$ amounts to existential quantification over the corresponding variables, i.e., port activity and data variables of the ports in O. For a TCA $\mathfrak{T}$, with formula representation $\varphi(\mathfrak{T})$, and a port set $O{\subseteq}\mathcal{P}$, the formula representation $\varphi(\mathfrak{T}\backslash_O)$ of automaton $\mathfrak{T}\backslash_O$ corresponds to

$$\exists\, \mathtt{O}\, \varphi(\mathfrak{T}), \tag{3.19}$$

with $\mathtt{O}{=}\bigcup_{p\in O}\{\mathtt{p_t}, \mathtt{Dp_t}\}$

In Definition 2.3.12, an additional clock is introduced to ensure correct timed behaviour of invisible transitions in $\mathfrak{T}\backslash_O$ which originate from visible transitions in $\mathfrak{T}$. Here, we do not need to introduce an additional clock: the formula representation of a visible transition explicitly requires a positive amount of time to elapse ($(\mathtt{z_{t\text{-}1}}{<}\mathtt{z_t})$, cf. (3.11)). Since $\mathtt{O}$ does not contain clock variables, this constraint remains unchanged even in case the transition becomes invisible, and therefore, correct timed behaviour, as required by Definition 2.3.12, is guaranteed.

3.1.4 Timed Network Automata

The representation of TNA follows the same ideas as the representations of TA and TCA. For clocks, clock constraints, locations, memory cells (data variables) and data constraints, we use the concepts introduced in Section 3.1.1. Yet, for ports of TNA, we need to extend the encoding with an additional variable, to be able to identify—in case of no dataflow—where the reason to delay comes from (cf. Section 2.4.1).

As defined in Section 3.1.1.4, for every port $p{\in}\mathcal{P}$, we have a Boolean variable $\mathtt{p_t}$ (port activity variable), with the intended meaning that $\mathtt{p_t}$ evaluates to `true` iff data flows through port p in step $\mathtt{t}$, and an integer variable $\mathtt{Dp_t}$ (port data variable), which represents the data value pending at p in step $\mathtt{t}$. In addition, the Boolean variable $\mathtt{cp_t}$ (called *port colour variable*) denotes where the reason for delay comes from in case $\mathtt{p_t}$ evaluates to `false`. For a given colouring $\mathbb{c}$, the *representation of* p *under* $\mathbb{c}$ *in step* $\mathtt{t}$, denoted $\langle \mathtt{p}_{\mathbb{c}}\rangle_{\mathtt{t}}$, is given by $\neg\mathtt{p_t}\wedge\neg\mathtt{cp_t}$ iff $\mathbb{c}(p){=}$-?-, by $\neg\mathtt{p_t}\wedge\mathtt{cp_t}$ iff $\mathbb{c}(p){=}$-!-, and by $\mathtt{p_t}\wedge(\mathtt{cp_t}\vee\neg\mathtt{cp_t})$ iff $\mathbb{c}(p){=}$——. In the latter case, the representation simplifies to $\mathtt{_t}$.

The representation of a TNA $\mathfrak{N}$ is now given as follows.

Definition 3.1.9 (TNA Representation). Let $\mathfrak{N}$ be a TNA, with initial location $\bar{s}$ (as before, we denote the initial location as $\bar{s}$ rather than s_0), $f{=}(s, \mathbb{c}, dc, cc, \lambda, s')\in E$ a communication, and $d{=}(s, \mathbb{c}, dc, cc, id, s)\in E$ a delay. The *formula representation of the transition relation of* $\mathfrak{N}$, denoted $\varphi(\mathfrak{N})$, is defined in (3.26) in Table 3.6.

$$
\begin{aligned}
\varphi^{init}(\mathfrak{N}) =\; & \bar{\mathtt{s}}_0\wedge \bigwedge_{s\in S, s\neq\bar{s}} \neg\mathtt{s}_0\wedge\mathtt{I}(\bar{\mathtt{s}})_0\wedge \bigwedge_{p\in\mathcal{P}}(\neg\mathtt{p}_0\wedge(\mathtt{Dp}_0{=}\mathtt{n}^{\perp})\wedge\mathtt{cp}_0)\wedge && (3.20)\\
& \bigwedge_{d\in\mathcal{D}}(\neg\mathtt{d}_0\wedge(\mathtt{Dd}_0{=}\mathtt{n}^{\perp}))\wedge(\mathtt{z}_0{=}0)\wedge \bigwedge_{x\in\mathcal{X}}(\mathtt{x}_0{=}0) &&\\
\varphi^{commu}(f) =\; & \mathtt{s}_\mathtt{t}\wedge \bigwedge_{p\in\mathcal{P}}\langle \mathtt{p}_{\mathbb{c}}\rangle_{\mathtt{t+1}}\wedge \bigwedge_{d\notin\#(s')}\neg\mathtt{d}_{\mathtt{t+1}}\wedge\mathtt{dc}_{\mathtt{t+1}}\wedge\mathtt{cc}_\mathtt{t}\wedge(\mathtt{z}_{\mathtt{t+1}}{=}\mathtt{z}_\mathtt{t})\wedge && (3.21)\\
& \bigwedge_{\lambda(x)=id}(\mathtt{x}_{\mathtt{t+1}}{=}\mathtt{x}_\mathtt{t})\wedge \bigwedge_{\lambda(x)=x'}(\mathtt{x}_{\mathtt{t+1}}{=}\mathtt{x'}_{\mathtt{t+1}})\wedge &&\\
& \bigwedge_{\lambda(x)=n}(\mathtt{x}_{\mathtt{t+1}}{=}\mathtt{z}_{\mathtt{t+1}}{-}n)\wedge\mathtt{s'}_{\mathtt{t+1}}\wedge\mathtt{I}(\mathtt{s'})_{\mathtt{t+1}} &&\\
\varphi^{delay}(d) =\; & \mathtt{s}_\mathtt{t}\wedge \bigwedge_{p\in\mathcal{P}}\langle \mathtt{p}_{\mathbb{c}}\rangle_{\mathtt{t+1}}\wedge \bigwedge_{d\in\mathcal{D}}(\mathtt{Dd}_{\mathtt{t+1}}{=}\mathtt{Dd}_\mathtt{t})\wedge\mathtt{dc}_{\mathtt{t+1}}\wedge\mathtt{cc}_\mathtt{t}\wedge(\mathtt{z}_{\mathtt{t+1}}{\geq}\mathtt{z}_\mathtt{t})\wedge && (3.22)\\
& \bigwedge_{x\in\mathcal{X}}(\mathtt{x}_{\mathtt{t+1}}{=}\mathtt{x}_\mathtt{t})\wedge\mathtt{cc}_{\mathtt{t+1}}\wedge\mathtt{s}_{\mathtt{t+1}}\wedge\mathtt{I}(\mathtt{s})_{\mathtt{t+1}} &&\\
\varphi^{trans}(\mathfrak{N}) =\; & \bigvee_{f\text{ comm.}}\varphi^{commu}(f)\vee\bigvee_{d\text{ delay}}\varphi^{delay}(d) && (3.23)\\
\varphi^{location}(\mathfrak{N}) =\; & \bigvee_{s\in S}(\mathtt{s}_{\mathtt{t+1}}\wedge \bigwedge_{s'\in S, s'\neq s}\neg\mathtt{s}'_{\mathtt{t+1}}) && (3.24)\\
\varphi^{mutex}(\mathfrak{N}) =\; & \bigwedge_{p\in\mathcal{P}}(\neg\mathtt{p}_{\mathtt{t+1}}\vee\neg(\mathtt{Dp}_{\mathtt{t+1}}{=}\mathtt{n}^{\perp}))\wedge(\mathtt{p}_{\mathtt{t+1}}\vee(\mathtt{Dp}_{\mathtt{t+1}}{=}\mathtt{n}^{\perp}))\wedge && (3.25)\\
& \bigwedge_{d\in\mathcal{D}}(\neg\mathtt{d}_{\mathtt{t+1}}\vee\neg(\mathtt{Dd}_{\mathtt{t+1}}{=}\mathtt{n}^{\perp}))\wedge(\mathtt{d}_{\mathtt{t+1}}\vee(\mathtt{Dd}_{\mathtt{t+1}}{=}\mathtt{n}^{\perp})) &&\\
\varphi(\mathfrak{N}) =\; & \varphi^{init}(\mathfrak{N})\wedge\varphi^{trans}(\mathfrak{N})\wedge\varphi^{location}(\mathfrak{N})\wedge\varphi^{mutex}(\mathfrak{N}) && (3.26)
\end{aligned}
$$

Table 3.6: Transition Relation Representation of TNA

The TNA starts in its initial location, the invariant of which holds, all ports are inactive, all memory cells are empty, and all clocks are set to zero (3.20).[5] The representation of a communication (3.21) ensures that the TNA is in location s before firing, and the clock guard cc holds. On execution of the transition, data flows according to colouring $\mathbb{c}$, the data values satisfy the data guard dc, all clocks are updated according to their value under λ, while the value of the absolute time reference z does not change, and memory cells not used by the target location loose their contents. After firing, the TNA is in location s', the invariant of which holds. The representation of a delay (3.22) is similar, except that the value of the absolute time reference increases, while all other clocks keep their value, and all memory cells keep their values as well. In addition, clock guard cc still needs to be satisfied after the time delay. Again, convexity of clock constraints allows us to check the invariant at the end of the time delay only, as it inductively holds at the beginning (3.20). The disjunction of these formulas expresses (nondeterministic) transition choice (3.23). In any step, the current location is unique (3.24), the special value "no data" may only be pending at inactive ports, and may only be "contained" in a memory cell if

[5] Note that it is not necessary to specify initial values for the port colour variables $\mathtt{cp}_0$, since the constraint $\neg\mathtt{p}_0$ is sufficient to express inactivity of port p. Yet, adding a valuation reduces the number of unspecified variables, and in this way speeds up verification.

that memory cells is indicated to be empty (3.25).

The results of Remark 3.1.5 (representation of $\perp$) and Remark 3.1.6 (representation of finite data domains) directly carry over from TCA to TNA.

Example 3.1.10 (TNA Representation). Consider again the 1-bounded FIFO buffer presented in Figure 2.8. Let $\mathfrak{N}$ be the name of the TNA, let f_1, f_2, f_3 refer to the communications from *empty* to *full*, *full* to *empty* (with clock guard $x<3$), and from *full* to *empty* (with clock guard $x=3$), respectively, and let d_1, d_2 refer to the delays in *empty* and *full*, respectively. The representation of $\mathfrak{N}$ is shown in Table 3.7. We omit constraints equal to `true`.

$$\begin{aligned}
\varphi^{init}(\mathfrak{N}) =\ & \mathtt{empty}_0\wedge\neg\mathtt{full}_0\wedge\neg\mathtt{r}_0\wedge(\mathtt{Dr}_0=\mathtt{n}^{\perp})\wedge\mathtt{cr}_0\wedge\neg\mathtt{w}_0\wedge(\mathtt{Dw}_0=\mathtt{n}^{\perp})\wedge\mathtt{cw}_0\wedge\\
& (\neg\mathtt{m}_0\wedge(\mathtt{Dm}_0=\mathtt{n}^{\perp}))\wedge(\mathtt{z}_0=0)\wedge(\mathtt{x}_0=0)\\
\varphi^{commu}(f_1) =\ & \mathtt{empty}_\mathtt{t}\wedge\mathtt{r}_{\mathtt{t+1}}\wedge(\neg\mathtt{w}_{\mathtt{t+1}}\wedge\mathtt{cw}_{\mathtt{t+1}})\wedge(\mathtt{Dr}_{\mathtt{t+1}}=\mathtt{Dm}_{\mathtt{t+1}})\wedge(\mathtt{z}_{\mathtt{t+1}}=\mathtt{z}_\mathtt{t})\wedge\\
& (\mathtt{x}_{\mathtt{t+1}}=\mathtt{z}_{\mathtt{t+1}})\wedge\mathtt{full}_{\mathtt{t+1}}\wedge(\mathtt{z}_{\mathtt{t+1}}-\mathtt{x}_{\mathtt{t+1}}\leq 3)\\
\varphi^{commu}(f_2) =\ & \mathtt{full}_\mathtt{t}\wedge(\neg\mathtt{r}_{\mathtt{t+1}}\wedge\mathtt{cr}_{\mathtt{t+1}})\wedge\mathtt{w}_{\mathtt{t+1}}\wedge\neg\mathtt{m}_{\mathtt{t+1}}\wedge(\mathtt{Dw}_{\mathtt{t+1}}=\mathtt{Dm}_\mathtt{t})\wedge\\
& (\mathtt{z}_\mathtt{t}-\mathtt{x}_\mathtt{t}<3)\wedge(\mathtt{z}_{\mathtt{t+1}}=\mathtt{z}_\mathtt{t})\wedge(\mathtt{x}_{\mathtt{t+1}}=\mathtt{x}_\mathtt{t})\wedge\mathtt{empty}_{\mathtt{t+1}}\\
\varphi^{commu}(f_3) =\ & \mathtt{full}_\mathtt{t}\wedge(\neg\mathtt{r}_{\mathtt{t+1}}\wedge\mathtt{cr}_{\mathtt{t+1}})\wedge(\neg\mathtt{w}_{\mathtt{t+1}}\wedge\mathtt{cw}_{\mathtt{t+1}})\wedge\neg\mathtt{m}_{\mathtt{t+1}}\wedge(\mathtt{z}_\mathtt{t}-\mathtt{x}_\mathtt{t}=3)\wedge\\
& (\mathtt{z}_{\mathtt{t+1}}=\mathtt{z}_\mathtt{t})\wedge(\mathtt{x}_{\mathtt{t+1}}=\mathtt{x}_\mathtt{t})\wedge\mathtt{empty}_{\mathtt{t+1}}\\
\varphi^{delay}(d_1) =\ & \mathtt{empty}_\mathtt{t}\wedge(\neg\mathtt{r}_{\mathtt{t+1}}\wedge\neg\mathtt{cr}_{\mathtt{t+1}})\wedge(\neg\mathtt{w}_{\mathtt{t+1}}\wedge\mathtt{cw}_{\mathtt{t+1}})\wedge(\mathtt{Dm}_{\mathtt{t+1}}=\mathtt{Dm}_\mathtt{t})\wedge\\
& (\mathtt{z}_{\mathtt{t+1}}\geq\mathtt{z}_\mathtt{t})\wedge(\mathtt{x}_{\mathtt{t+1}}=\mathtt{x}_\mathtt{t})\wedge\mathtt{empty}_{\mathtt{t+1}}\\
\varphi^{delay}(d_2) =\ & \mathtt{full}_\mathtt{t}\wedge(\neg\mathtt{r}_{\mathtt{t+1}}\wedge\mathtt{cr}_{\mathtt{t+1}})\wedge(\neg\mathtt{w}_{\mathtt{t+1}}\wedge\neg\mathtt{cw}_{\mathtt{t+1}})\wedge(\mathtt{Dm}_{\mathtt{t+1}}=\mathtt{Dm}_\mathtt{t})\wedge(\mathtt{z}_\mathtt{t}-\mathtt{x}_\mathtt{t}\leq 3)\wedge\\
& (\mathtt{z}_{\mathtt{t+1}}\geq\mathtt{z}_\mathtt{t})\wedge(\mathtt{x}_{\mathtt{t+1}}=\mathtt{x}_\mathtt{t})\wedge(\mathtt{z}_{\mathtt{t+1}}-\mathtt{x}_{\mathtt{t+1}}\leq 3)\wedge\mathtt{full}_{\mathtt{t+1}}\wedge(\mathtt{z}_{\mathtt{t+1}}-\mathtt{x}_{\mathtt{t+1}}\leq 3)\\
\varphi^{trans}(\mathfrak{N}) =\ & \varphi^{commu}(f_1)\vee\varphi^{commu}(f_2)\vee\varphi^{commu}(f_3)\vee\varphi^{delay}(d_1)\vee\varphi^{delay}(d_2)\\
\varphi^{location}(\mathfrak{N}) =\ & (\mathtt{empty}_{\mathtt{t+1}}\wedge\neg\mathtt{full}_{\mathtt{t+1}})\vee(\mathtt{full}_{\mathtt{t+1}}\wedge\neg\mathtt{empty}_{\mathtt{t+1}})\\
\varphi^{mutex}(\mathfrak{N}) =\ & (\neg\mathtt{r}_{\mathtt{t+1}}\vee\neg(\mathtt{Dr}_{\mathtt{t+1}}=\mathtt{n}^{\perp}))\wedge(\mathtt{r}_{\mathtt{t+1}}\vee(\mathtt{Dr}_{\mathtt{t+1}}=\mathtt{n}^{\perp}))\wedge\\
& (\neg\mathtt{w}_{\mathtt{t+1}}\vee\neg(\mathtt{Dw}_{\mathtt{t+1}}=\mathtt{n}^{\perp}))\wedge(\mathtt{w}_{\mathtt{t+1}}\vee(\mathtt{Dw}_{\mathtt{t+1}}=\mathtt{n}^{\perp}))\wedge\\
& (\neg\mathtt{m}_{\mathtt{t+1}}\vee\neg(\mathtt{Dm}_{\mathtt{t+1}}=\mathtt{n}^{\perp}))\wedge(\mathtt{m}_{\mathtt{t+1}}\vee(\mathtt{Dm}_{\mathtt{t+1}}=\mathtt{n}^{\perp}))\wedge\\
\varphi(\mathfrak{N}) =\ & \varphi^{init}(\mathfrak{N})\wedge\varphi^{trans}(\mathfrak{N})\wedge\varphi^{location}(\mathfrak{N})\wedge\varphi^{mutex}(\mathfrak{N})
\end{aligned}$$

Table 3.7: Transition Relation Representation of TNA: Example

Though the flip rule (Remark 2.4.11) reduces the size of TNA, the size of a composed TNA is still exponential in the worst case. We now present a *linear size* representation of the composition of TNA. The basic idea is similar to the product definition of TA and TCA (cf. Definitions 3.1.3 and 3.1.8): we do not explicitly compute the composition, but instead retain the representations of the single TNA, and define the representation of the composition via conjunction. Unfortunately,[6]

[6] Actually, we do not consider this a disadvantage, since we gain a composition that is linear.

this does not allow us to explicitly remove the ports contained in a merge set from the representation, and replace them by the same data variable (cf. Definition 2.4.13). Instead, we need to add additional constraints to ensure that (1) the representation of the composition correctly models the dataflow behaviour of the resulting internal port (cf. Definition 2.4.9), and that (2) all ports in the merge set agree on the same data value (cf. Definition 2.4.13 and preceding explanations).

Definition 3.1.11 (Internal Port Representation). Let $\mathcal{Q}$ be a merge set, $\mathcal{Q}^r{\subseteq}\mathcal{Q}$ and $\mathcal{Q}^w{\subseteq}\mathcal{Q}$ the subsets of read respectively write ports in $\mathcal{Q}$. For $p{\in}Q$, let $\mathtt{p_t}$, $\mathtt{Dp_t}$ and $\mathtt{cp_t}$ be the port activity, port data and port colour variable, respectively, let d be a fresh data variable (i.e., not yet used elsewhere), with data fullness variable $\mathtt{d_t}$ and data content variable $\mathtt{Dd_t}$. The *representation* $\varphi^{int\_port}(\mathcal{Q})$ *of internal port* $p_{\prec\mathcal{Q}}$ is given in (3.27).

$$\begin{aligned}
\varphi^{valid\_col1}(\mathcal{Q}) &= \bigvee_{w\in\mathcal{Q}^w} \mathtt{w_{t+1}}{\rightarrow}\Big(\bigwedge_{r\in\mathcal{Q}^r}\mathtt{r_{t+1}}{\wedge}\bigwedge_{w,w'\in\mathcal{Q}^w,w\neq w'}\neg(\mathtt{w_{t+1}}{\wedge}\mathtt{w'_{t+1}})\Big)\\
\varphi^{valid\_col2}(\mathcal{Q}) &= \bigvee_{r\in\mathcal{Q}^r}\mathtt{r_{t+1}}{\rightarrow}\bigvee_{w\in\mathcal{Q}^w}\mathtt{w_{t+1}}\\
\varphi^{valid\_col3}(\mathcal{Q}) &= \bigwedge_{p\in\mathcal{Q}}\neg\mathtt{p_{t+1}}{\rightarrow}\Big(\bigwedge_{w\in\mathcal{Q}^w}\mathtt{cw_{t+1}}{\vee}\bigvee_{r\in\mathcal{Q}^r}\mathtt{cr_{t+1}}\Big)\\
\varphi^{data\_flow}(\mathcal{Q}) &= \bigwedge_{p\in\mathcal{Q}}\neg\mathtt{p_{t+1}}{\vee}(\mathtt{Dp_{t+1}}{=}\mathtt{Dd_{t+1}})\\
\varphi^{int\_port}(\mathcal{Q}) &= \varphi^{valid\_col1}(\mathcal{Q}){\wedge}\varphi^{valid\_col2}(\mathcal{Q}){\wedge}\varphi^{valid\_col3}(\mathcal{Q}){\wedge}\varphi^{data\_flow}(\mathcal{Q}) \qquad (3.27)
\end{aligned}$$

The first three constraints directly correspond to the three conditions in Definition 2.4.9. For example, $\varphi^{valid\_col3}(\mathcal{Q})$ describes the constraints in condition 3 in Definition 2.4.9: if there is no flow at all ($\bigwedge_{p\in\mathcal{Q}}\neg\mathtt{p_t}$), then either all write ports provide a reason for delay ($\bigwedge_{w\in\mathcal{Q}^w}\mathtt{cw_t}$), or at least one read port provides a reason for delay ($\bigvee_{r\in\mathcal{Q}^r}\mathtt{cr_t}$). These three constraints thus capture *whether* data flows through $p_{\prec\mathcal{Q}}$. The fourth constraint $\varphi^{data\text{-}flow}(\mathcal{Q})$—the conjuncts should be read as $\mathtt{p_t}{\rightarrow}(\mathtt{Dp_t}{=}\mathtt{Dd_t})$—expresses the fact that all active ports in the merge set (if $\mathtt{p_t}$ holds, p is active, cf. the beginning of Section 3.1.4) agree on the same data value, we use the data variable d as a placeholder for any possible data value. This constraint thus captures *which* data flows through $p_{\prec\mathcal{Q}}$.

Using this, the representation of TNA composition is defined as follows.

Definition 3.1.12 (TNA Composition Representation). Let $\mathcal{N}$ be a set of disjoint TNA, $\mathcal{Q}$ a set of disjoint merge sets over ports of TNA in $\mathcal{N}$. The *formula representation* $\varphi(\mathcal{N}{\bowtie_{\mathcal{Q}}})$ *of the composed TNA* $\mathcal{N}{\bowtie_{\mathcal{Q}}}$ is defined as

$$\varphi(\mathcal{N}{\bowtie_{\mathcal{Q}}}) = \bigwedge_{\mathfrak{N}\in\mathcal{N}}\varphi(\mathfrak{N}){\wedge}\bigwedge_{\mathcal{Q}'\in\mathcal{Q}}\varphi^{int\_port}\mathcal{Q}' \qquad (3.28)$$

To accommodate the fact that a port cannot be merged more than once (cf. beginning of Section 2.4.3), which now can be translated to "cannot be contained in more than one merge set", we hide the ports in a merge set, using existential quantification: the *reduction of* $\varphi(\mathcal{N}{\bowtie_{\mathcal{Q}}})$ *to the external interface* (i.e., the set of ports, cf. Definition 2.4.2) is defined as

$$\exists \bigcup_{\mathcal{Q}' \in \mathcal{Q}} \mathcal{Q}'(\varphi(\mathcal{N} \bowtie_{\mathcal{Q}}))$$

In this way, ports that are contained in any merge set in $\mathcal{Q}$ cannot be merged again when composing $\mathcal{N} \bowtie_{\mathcal{Q}}$ with another TNA.

3.2 Bounded Model Checking

In this section, we briefly recall the concepts of Bounded Model Checking (BMC), and show how they can be applied to the representations of real-time systems defined in Section 3.1.

Bounded Model Checking (BMC) [BCC$^+$03, BCCZ99, CBRZ01] has evolved from Symbolic Model Checking (SMC) [McM93], and can be seen as a subcategory of it. SMC techniques represent the system symbolically, and typically rely on binary decision diagrams (BDDs) [Bry86]. These BDD representations can handle hundreds of variables, but often blow up in space. In addition, the efficiency highly depends on the variable ordering in the BDD, yet, the problem of finding an efficient order is NP-hard, that means, there exists no efficient way of determining an efficient ordering a priori.

BMC was introduced “in an attempt to replace BDDs with SAT in SMC” [Bie09]. The key idea is to represent the system and the property to be checked symbolically (using propositional formulas), examine prefix fragments of the transition system for whether the property holds, and successively increase the exploration bound until it reaches (a computable indicator of) the diameter of the system—in which case the results are guaranteed to be complete, and the property holds—or an unsafe run violating the property has been discovered.

Although BMC is complete in theory once the diameter of the system is reached, it is often impractical to increase the exploration bound that far (see Section 3.2.4 for a more detailed discussion). Therefore, BMC techniques focus on falsification of (temporal) properties. Such properties can be disproved with a finite counterexample, i.e., a finite run, where at least one of the configurations contains a contradiction to the property. Reachability properties are well-suited to express safety properties of the form “a certain behaviour should not happen”, where the erroneous behaviour is defined by the possibility to reach a certain error location.

We first introduce some notations in Section 3.2.1, and then formalise these notions in Sections 3.2.2 and 3.2.3.

3.2.1 Notations

In the remainder of this section, we use $\mathfrak{S}$ to refer to any of the system models defined in Chapter 2: $\mathfrak{S} \in \{\mathfrak{A}, \mathfrak{T}, \mathfrak{N}\}$, cf. Definitions 2.2.1, 2.3.1 and 2.4.2. We use $\varphi(\mathfrak{S})$ to refer to the corresponding formula representation of $\mathfrak{S}$: $\varphi(\mathfrak{S}) \in \{\varphi(\mathfrak{A}), \varphi(\mathfrak{T}), \varphi(\mathfrak{N})\}$, for both simple (Definitions 3.1.1, 3.1.4 and 3.1.9) and composed (Definitions 3.1.3, 3.1.8 and 3.1.12) systems.

For a formula $\varphi(\mathfrak{S})$, we use $\mathit{Vars}(\varphi(\mathfrak{S}))$ to denote the *set of variables* of $\varphi(\mathfrak{S})$, we write $\mathit{Vars}(\varphi(\mathfrak{S}))|_{\mathbb{B}}$, $\mathit{Vars}(\varphi(\mathfrak{S}))|_{\mathbb{N}}$ and $\mathit{Vars}(\varphi(\mathfrak{S}))|_{\mathbb{Q}}$ to denote the subsets of $\mathit{Vars}(\varphi(\mathfrak{S}))$ of propositional, integer and rational variables, respectively. We use $\mathit{Atoms}(\varphi(\mathfrak{S}))$ to denote the *set of propositional atoms* of $\varphi(\mathfrak{S})$, and write $\mathit{Conts}(\varphi(\mathfrak{S}))$ ("contents") as an abbreviation for $\mathit{Atoms}(\varphi(\mathfrak{S}))\cup\mathit{Vars}(\varphi(\mathfrak{S}))$.

An *interpretation σ of (the variables in)* $\varphi(\mathfrak{S})$ is a mapping $\sigma\colon\mathit{Vars}(\varphi(\mathfrak{S}))\rightarrow(\mathbb{B}\cup\mathbb{N}\cup\mathbb{Q})$, assigning to each variable $v\in\mathit{Vars}(\varphi(\mathfrak{S}))$ a value from the appropriate range (i.e., $\sigma(v)\in\mathbb{B}$ iff $v\in\mathit{Vars}(\varphi(\mathfrak{S}))|_{\mathbb{B}}$, $\sigma(v)\in\mathbb{N}$ iff $v\in\mathit{Vars}(\varphi(\mathfrak{S}))|_{\mathbb{N}}$, and $\sigma(v)\in\mathbb{Q}$ iff $v\in\mathit{Vars}(\varphi(\mathfrak{S}))|_{\mathbb{Q}}$).

Interpretation σ is called *model of* $\varphi(\mathfrak{S})$, denoted $\sigma\models\varphi(\mathfrak{S})$, if it *satisfies* $\varphi(\mathfrak{S})$, i.e., $\varphi(\mathfrak{S})$ evaluates to `true` under valuation σ. $\varphi(\mathfrak{S})$ is called *satisfiable* if at least one model of $\varphi(\mathfrak{S})$ exists. We denote the set of all models of $\varphi(\mathfrak{S})$ by $\mathcal{V}(\varphi(\mathfrak{S}))$. If $\varphi(\mathfrak{S})$ evaluates to `true` under *all* valuations, then $\varphi(\mathfrak{S})$ is called *tautology*, denoted $\models\varphi(\mathfrak{S})$.

We lift the above notations in the straightforward way from $\varphi(\mathfrak{S})$ to all types of formulas.

3.2.2 Unfolding for BMC

The formula representation $\varphi(\mathfrak{S})$ of a real-time system $\mathfrak{S}$, as defined in the Section 3.1, describes the transition characteristics of $\mathfrak{S}$ in terms of "abstract" steps `t` and `t+1`. In order to describe the reachability problem of BMC for k steps, the formula representation $\varphi(\mathfrak{S})$ is *unfolded*, i.e., instantiated for all steps 1 up to bound k. The resulting formula $\varphi(\mathfrak{S})_k$ is called *k-unfolding*.

Definition 3.2.1 (k-unfolding). Let $\mathfrak{S}$ be a real-time system, with formula representation $\varphi(\mathfrak{S})$, let $k\in\mathbb{N}$, $k\geq 1$ (called *unfolding depth*). The *k-unfolding $\varphi(\mathfrak{S})_k$ of* $\varphi(\mathfrak{S})$ is defined as

$$\varphi(\mathfrak{S})_k = \varphi^{\mathit{init}}(\mathfrak{S})\wedge\bigwedge_{0\leq j\leq k-1}(\varphi^{\mathit{trans}}(\mathfrak{S})_{\mathtt{j/t}}\wedge\varphi^{\mathit{location}}(\mathfrak{S})_{\mathtt{j/t}}\wedge\varphi^{\mathit{mutex}}(\mathfrak{S})_{\mathtt{j/t}}), \tag{3.29}$$

where $\psi_{\mathtt{j/t}}$ denotes a variant of formula ψ with index `t` replaced by `j`.

Intuitively, a model σ of $\varphi(\mathfrak{S})_k$ corresponds to a run of $\mathfrak{S}_{\mathfrak{S}}$ of length k, i.e., to one possible behaviour of $\mathfrak{S}$ for the first k steps. Consequently, the set $\mathcal{V}(\varphi(\mathfrak{S})_k)$ of all models of $\varphi(\mathfrak{S})_k$ describes all possible behaviours of $\mathfrak{S}$ for the first k steps.

Notation 3.2.2 (Unfolding). For $j\in\mathbb{N}$, $j\geq 0$, we write $\varphi^{\mathit{trans}}(\mathfrak{S})_{(\mathtt{j})}$ to denote the transition constraints from step `j` to step `j+1`, that is, $\varphi^{\mathit{trans}}(\mathfrak{S})_{(\mathtt{j})}=\varphi^{\mathit{trans}}(\mathfrak{S})_{\mathtt{j/t}}$. Equivalently, we write $\varphi^{\mathit{location}}(\mathfrak{S})_{(\mathtt{j})}$ respectively $\varphi^{\mathit{mutex}}(\mathfrak{S})_{(\mathtt{j})}$ to denote the mutual exclusion constraints on locations respectively events or ports in step `j`, that is, $\varphi^{\mathit{location}}(\mathfrak{S})_{(\mathtt{j})}=\varphi^{\mathit{location}}(\mathfrak{S})_{(\mathtt{j-1})/\mathtt{t}}$, and $\varphi^{\mathit{mutex}}(\mathfrak{S})_{(\mathtt{j})}=\varphi^{\mathit{mutex}}(\mathfrak{S})_{(\mathtt{j-1})/\mathtt{t}}$. Intuitively, for a formula $\psi_{(\mathtt{j})}$, `j` denotes the (smallest) index which appears on variables in ψ.

3.2.3 BMC of Properties

BMC is best suited for checking safety properties of the form "a certain behaviour should not happen", expressed through reachability of error locations. If an error location s is (not) reachable within k steps, s is called *(not) k-step reachable.* The safety property ϕ saying *s is not k-step reachable* is expressed by the formula

$$\phi = \bigwedge_{0 \leq i \leq k} \neg \mathtt{s_i}, \tag{3.30}$$

where $\mathtt{s}$ is the representation of s. To express that s is not reachable after *exactly* k-steps (for example because we already know that s is not $(k-1)$-step reachable) we simply choose $\phi = \neg \mathtt{s_k}$.

Checking whether a system $\mathfrak{S}$ is safe with respect to s amounts to conjoining the k-unfolding (3.29) with the negated property $\neg\phi$ (as explained above, BMC focusses on falsification of properties):

$$\varphi(\mathfrak{S})_k \wedge \bigvee_{0 \leq i \leq k} \mathtt{s_i}. \tag{3.31}$$

In this way, a model of (3.31) corresponds to a run of $\mathfrak{S}$ that contains error location s, that means to a system behaviour violating the property ϕ. In this case, $\mathfrak{S}$ is unsafe with respect to s, that means, the property ϕ does not hold in the system $\mathfrak{S}$. If no such model exists, i.e., the formula (3.31) is unsatisfiable, $\mathfrak{S}$ is safe with respect to s within bound k, but nothing can be said about safety within bounds $>k$.

Lifting (3.30) to reason about configurations or even execution sequences is straightforward. For example, the property "if the location in the current step is s, then the location in the next step will be s'" can be represented as

$$(\mathtt{s_0} \wedge \mathtt{s'_1}) \vee (\mathtt{s_1} \wedge \mathtt{s'_2}) \vee \ldots (\mathtt{s_{k\text{-}1}} \wedge \mathtt{s'_k}). \tag{3.32}$$

Other properties can be represented using the encoding in [ACKS02], where the authors have defined a translation from LTL to propositional formulas for BMC. For example, (3.32) corresponds to the LTL property $s \rightarrow \bigcirc\, s'$. We skip the details of the encoding, as they are beyond the scope of this thesis. In general, the only restriction we impose on formulas ϕ used as properties is that they may only reason about variables contained in $\varphi(\mathfrak{S})_k$, i.e., we require $\mathit{Vars}(\phi) \subseteq \mathit{Vars}(\varphi(\mathfrak{S})_k)$.

3.2.4 Completeness of BMC

BMC is used to inspect runs of a certain length k. If an error location is reachable within k steps, a counterexample to the safety property has been found. Otherwise, the exploration bound is increased, and the reachability check is repeated. The question remains whether there exists a *completeness threshold*, i.e., some bound k', for which we can safely conclude that the error location is not reachable, even when further increasing the exploration bound.

A first candidate is the diameter of the associated LTS $\mathfrak{S}_\mathfrak{S}$, reduced to runs which start in the initial configuration q_0.[7] This is indeed a completeness threshold: as soon as the exploration bound k is equal to the diameter, the results of BMC are complete, since BMC considers *all* runs of length k, and thus every reachable configuration will be reached by at least one of the runs. Unfortunately, no efficient algorithms exist to compute the diameter of a timed system.

Another candidate is the *recurrence diameter* of the associated LTS $\mathfrak{S}_\mathfrak{S}$, cf. also [BCCZ99]: the recurrence diameter of $\mathfrak{S}_\mathfrak{S}$ is the length of the longest loop-free run (cf. Definitions 2.2.4, 2.3.5 and 2.4.6). This is a completeness threshold as well: by definition, any run with length greater than the recurrence diameter must contain a loop, and thus cannot contain new configurations which have not been visited before. Unfortunately, the recurrence diameter can be considerably larger than the real diameter. Consider for example a fully connected graph with n nodes: while the diameter is 1 (every node is reachable from every other node), the recurrence diameter is $n-1$ (longest loop-free path). Yet, the recurrence diameter is more practical than the (real) diameter, since the fact that a run is loop-free can easily be expressed in our framework.

Definition 3.2.3 (Representation of Loop-Free Runs). Let $\mathfrak{S}$ be a real-time system, with set of locations S, set of clocks $\mathcal{X}$, and set of data variables $\mathcal{D}$ (only for TCA and TNA). Let $\varphi(\mathfrak{S})_k$ be the k-unfolding, and $\sigma\in\mathcal{V}(\varphi(\mathfrak{S})_k)$ a model of $\varphi(\mathfrak{S})_k$. The model σ corresponds to a *loop-free run* if it satisfies the *loop-free condition* $\varphi_k^{\textit{lp-free}}(\mathfrak{S})$, i.e. (in addition to $\sigma\models\varphi(\mathfrak{S})_k$) $\sigma\models\varphi_k^{\textit{lp-free}}(\mathfrak{S})$, where $\varphi_k^{\textit{lp-free}}(\mathfrak{S})$ is defined as

$$\varphi_k^{\textit{lp-free}}(\mathfrak{S}) = \bigwedge_{0\leq i<j\leq k} \Big(\bigvee_{s\in S} \neg(\mathtt{s_i}{=}\mathtt{s_j}) \vee \bigvee_{x\in\mathcal{X}} \neg(\mathtt{x_i}{=}\mathtt{x_j}) \vee \bigvee_{d\in\mathcal{D}} \neg(\mathtt{Dd_i}{=}\mathtt{Dd_j}) \Big) \tag{3.33}$$

If $\mathfrak{S}$ is a TA, we omit the last disjunct.

For a run to be loop-free, all configurations need to be different. Configurations consist of (cf. Definitions 2.2.4, 2.3.5 and 2.4.6) the current location, the valuation of the clocks, and (only for TCA and TNA) the valuation of the data variables. Condition (3.33) expresses that for any two configurations (in steps i and j), at least one of these constituents is different.

We can use the loop-free condition in two ways: testing whether a model corresponds to a loop-free run, and calculating the recurrence diameter. The former is done by checking whether $\sigma\models\varphi_k^{\textit{lp-free}}(\mathfrak{S})$ for a model $\sigma\in\mathcal{V}(\varphi(\mathfrak{S})_k)$. To actually calculate the recurrence diameter, we inductively check (i.e., for increasing values of k) whether the formula $\varphi(\mathfrak{S})_k\wedge\varphi_k^{\textit{lp-free}}(\mathfrak{S})$ is satisfiable.[8] The recurrence diameter is the smallest k such that $\varphi(\mathfrak{S})_{k+1}\wedge\varphi_{k+1}^{\textit{lp-free}}(\mathfrak{S})$ is not satisfiable anymore.

[7] The diameter of a system—a well-known concept from graph theory—is a natural number, which corresponds to the longest shortest path between any two states. Here, the diameter is the longest shortest run from the initial configuration q_0 to any other configuration of $\mathfrak{S}_\mathfrak{S}$.

[8] Assuming that the subformula $\varphi(\mathfrak{S})_k$, when checked in isolation, is satisfiable for all values of k, i.e., we can always find a run of length k.

3.2.5 Correctness

Theorem 3.2.4 (Correctness). The formula representation $\varphi(\mathfrak{S})$ of a real-time system $\mathfrak{S}$ is correct, that means $\varphi(\mathfrak{S})$ exhibits the same behaviour as $\mathfrak{S}$.

The proof of Theorem 3.2.4 can be found in the Appendix, in Section A.1.

3.3 Discussion

In this section, for some of the constituents of real-time systems, we discuss other possible encodings, and motivate our design decisions.

3.3.1 Occurrence of Actions on Transitions of TA

The transition relation representation of TA models transitions from step $\mathtt{t}$ to step $\mathtt{t+1}$ (cf. (3.2) and (3.3)), with the action α occurring at step $\mathtt{t+1}$. The choice of whether to model the occurrence of α at step $\mathtt{t}$ or $\mathtt{t+1}$ is to a certain extend arbitrary, since conceptually, the event occurs *while* taking the transition (that means, *in between* steps $\mathtt{t}$ and $\mathtt{t+1}$). We believe both ways are intuitive by some means or other.

We decided to model the occurrence at step $\mathtt{t+1}$ since this is in accordance with the activity of ports in TCA and TNA, which also occurs at steps $\mathtt{t+1}$. This results in a uniform handling of all "transition labels", whether they are actions in TA or ports in TCA respectively TNA. We benefit from this point in the concretisation of abstract counterexamples, cf. Section 4.3.3.

3.3.2 Choice of Variable Types

When designing the formula representation for a real-time system, different aspects and requirements influence the design decisions, the two major ones being *universality* and *efficiency*. Universality in this context means that the resulting formulas should be platform independent and general enough such that they can be integrated seamlessly (or with only minor adaptations) into most (standard) frameworks. Efficiency is understood with respect to speed of verification.

To meet the first requirement, we have reduced the variable types used in the representation to rational, integer and Boolean variables. Rational variables are needed to represent real-time. Though clocks are real-valued (cf. Section 2.1.1), a rational encoding is sufficient here, since linear arithmetic using only integer constants (as is used in clock constraints, cf. Definition 2.1.2) is equisatisfiable for rational and real numbers. Integer variables are used for data variables and data values. We could have actually used rational variables for these as well, but this would have required additional constraints to restrict the admissible valuations of such variables, for example, to preserve the total order, cf. Definition 2.1.7. Boolean variables are used to represent the remaining constituents. Moreover, we reduce the number of arithmetic operations to addition, subtraction, equality and comparison.

To meet the second requirement, we use Boolean variables whenever possible. Propositional satisfiability (SAT solving) has been extensively studied in recent years (see for example [PBG05] for an overview); as a result, existing techniques are by now well-optimised and efficient, cf. for example [GN07, MMZ$^+$01]. SAT solvers internally work with formulas in conjunctive normal form (CNF).[9] We design our formulas in CNF whenever possible, to avoid unnecessary transformations in the SAT solver. Moreover, most of the CNF clauses are binary (two literals) or even unit (one literal) clauses. With respect to speed of verification, this is very efficient: the 2-SAT problem (i.e., with binary clauses only) is polynomial, and formulas with n unit clauses can be solved in $O(n)$. Formulas (3.5), (3.6), (3.14) and (3.24) could be expressed in CNF with binary clauses as well. Yet, they are not, but are rather tailored for abstraction already: after abstraction, the formulas in CNF would loose the information of mutual exclusion, and result in a too coarse (or even wrong) abstraction. Due to the disjunctive nature of transition choices, (3.4), (3.13) and (3.23) are not in CNF, but they could easily be transformed to short CNF (see e.g. [Häh93]) when introducing new symbols.

3.3.3 Temporal Difference Encoding of Clocks

For the representation of clock values, we use an approach similar to the one presented in [ACKS02]. We introduce a fresh clock z (the absolute time reference), to measure the absolute amount of time that has passed since the beginning of computation, cf. Section 3.1.1.1. The representation of the value of clock x in step $\mathtt{t}$ is given by the difference $\mathtt{z_t}{-}\mathtt{x_t}$ of the representation variables of z and x, this difference is also used in the representation of clock constraints. An update of clock x according to some update map λ is represented by setting $\mathtt{x_t}{=}\mathtt{z_t}{-}n$ iff $\lambda(x){=}n$, and by setting $\mathtt{x_t}{=}\mathtt{x'_t}$ iff $\lambda(x){=}x'$ for some clock x'. If $\lambda(x){=}id$, the value of $\mathtt{x_t}$ carries over to the next step: $\mathtt{x_t}{=}\mathtt{x_{t+1}}$.

This design decision might seem unintuitive at first glance. Yet, the major advantage of the approach becomes clear when considering the representation of delays. In our approach, on execution of a delay from step $\mathtt{t}$ to step $\mathtt{t{+}1}$, all clock variables keep their values, only the value of the absolute time reference changes:

$$\bigwedge_{x\in\mathcal{X}} (\mathtt{x_t}{=}\mathtt{x_{t+1}}){\wedge}(\mathtt{z_t}{\sim}\mathtt{z_{t+1}}),\ \text{with}\ \sim\,\in\{\leq,<\} \tag{*}$$

(cf. (3.3), (3.11), (3.12), (3.17) and (3.22)). In an encoding which does not use the absolute time reference, on the other hand, *all* clock variables of step $\mathtt{t{+}1}$ are different from those in step $\mathtt{t}$. Furthermore, the representation needs take care of the fact that all clock variables change by the same amount. The encoding of clock variables in a delay step might look like

$$\bigwedge_{x\in\mathcal{X}} (\mathtt{x_{t+1}}{-}\mathtt{x_t}{=}\mathtt{d_t}),\ \text{with}\ \mathtt{d_t}\ \text{a rational variable.} \tag{**}$$

Here, $\mathtt{d_t}$ is a variable representing the amount of time for which the system delays.

[9] A formula is in conjunctive normal form, if it is a conjunction of clauses, where a clause is a disjunction of literals, and a literal is a propositional atom or its negation.

With respect to speed of verification, (*) is more efficient than (**): in the former case, the values of all but one (namely $\mathtt{z_{t+1}}$) clock variable in step $\mathtt{t+1}$ are predetermined by the values in step $\mathtt{t}$. Thus, the satisfiability of the (clock constraint) formulas of step $\mathtt{t+1}$ depends on one variable only. All conflict clauses which the solver learns while trying to find a model of the formula reason about $\mathtt{z_{t+1}}$, which quickly restricts the number of possible valuations of $\mathtt{z_{t+1}}$, and in this way leads to fewer backtracking steps. In the latter case, the satisfiability is subject to a number of free variables. Since all variables of step $\mathtt{t+1}$ depend on the values of a different variable of step $\mathtt{t_t}$, each learnt conflict clause can only restrict the possible valuations of a single variable, which potentially leads to more backtracking steps.

A second advantage of the encoding using the absolute time reference is the fact that properties of the form "something happens after x time units" can be specified more easily, by using the difference $\mathtt{z_j}-\mathtt{z_i}$, for some $0 \leq i < j \leq k$, $i, j \in \mathbb{N}$.

3.3.4 Linear Boolean Encoding of Finite Sets

After having chosen to represent the set of locations of a real-time system $\mathfrak{S}$ using Boolean variables (cf. Section 3.1.1), there are still two options: using a *linear* Boolean encoding or a *logarithmic* Boolean encoding.

Let $S = \{s_0, \ldots, s_{n-1}\}$ be the set of locations. In the linear encoding which we use in Section 3.1.1, we introduce one Boolean variable $\mathtt{s}$ for each location s, with the intended meaning that the localisation $\mathtt{s_t}$ is true iff $\mathfrak{S}$ is in location s in step $\mathtt{t}$. Since $\mathfrak{S}$ can only be in one location at the same time, this encoding requires an additional mutual exclusion constraint, to ensure exactly one of the variables is true in every step (cf. (3.5), (3.14) and (3.24)).[10] This mutual exclusion constraint is quadratic in the number of locations.

For a logarithmic encoding, we would introduce a vector $[s]$ of $j = \lceil \log_2(n) \rceil$ Boolean variables, encoding a j-digit Boolean value. The intended meaning is that the localisation $[\mathtt{s}]_\mathtt{t}$ of $[s]$ encodes the value i, denoted by $[\mathtt{s^i}]_\mathtt{t}$, iff the system $\mathfrak{S}$ is in location s_i in step $\mathtt{t}$. No additional mutual exclusion constraint is needed. Depending on the representation of transitions (see below), we might however need an additional constraint to disallow superfluous valuations of $[s]$, since in case $2^{\lceil \log_2(n) \rceil} > n$, the number of possible valuations of $[s]$ exceeds the actual number of locations.[11]

While the logarithmic encoding is slightly more efficient, due to the decreased number of variables, we have nevertheless chosen for Boolean encoding. One reason is that the differences in speed of verification are small, the real bottleneck is generated by rational clock variables. Moreover, abstraction of locations would be more involved when using a logarithmic encoding, and there is no straightforward way anymore of defining an abstraction function purely based on the syntactic categories of variables (cf. Definition 4.1.5).

[10] Observe that these mutual exclusion constraints could easily be expressed in CNF. Yet, they are not, but are already tailored for abstraction, cf. Chapter 4. After abstraction, an equivalent formula in CNF would loose the information of mutual exclusion, and result in a too coarse abstraction.

[11] For example, for six locations, i.e., $|S| = 6$, $[s]$ consists of $\lceil \log_2(6) \rceil = 3$ Boolean variables, with which we could represent $2^3 = 8 > 6$ locations.

This argumentation holds in the same way for the set of events of TA, since every transition in a TA is labelled with a distinct event. In TCA and TNA, however, a *set of* ports can be active on each transition, so no mutual exclusion constraint is needed.

3.3.5 Encoding of Transitions

For the encoding of transitions, there are a number possibilities of how to define the logical structure of the representation. For explanatory purposes, let $e \in E$ denote a transition from location s to location s'. Let $\mathtt{s}_{\mathtt{t}}$ and $\mathtt{s'}_{\mathtt{t+1}}$ be the representations of source location s (in step $\mathtt{t}$) and target location s' (in step $\mathtt{t+1}$), let $\varphi_{\mathtt{t}}$ denote the remaining constraints of step $\mathtt{t}$ ("preconditions of e", e.g., the representation $\mathtt{I(s)}_{\mathtt{t}}$ of the invariant of s), and let $\varphi_{\mathtt{t+1}}$ denote the remaining constraints of step $\mathtt{t+1}$ ("postconditions of e"). Constraints involving variables of both $\mathtt{t}$ and $\mathtt{t+1}$ are also contained in $\varphi_{\mathtt{t+1}}$. The exact structure of $\varphi_{\mathtt{t}}$ and $\varphi_{\mathtt{t+1}}$ is irrelevant here, as we only use them to explain the conceptual idea of transition representation.

The two main conceptual alternatives one can think of for encoding the transition relation are

$$\bigwedge_{e \in E} (\mathtt{s}_{\mathtt{t}} \wedge \varphi_{\mathtt{t}} \rightarrow \mathtt{s'}_{\mathtt{t+1}} \wedge \varphi_{\mathtt{t+1}}) \text{ (or, equivalently, using } \leftarrow \text{) and} \tag{3.34}$$

$$\bigvee_{e \in E} (\mathtt{s}_{\mathtt{t}} \wedge \varphi_{\mathtt{t}} \wedge \mathtt{s'}_{\mathtt{t+1}} \wedge \varphi_{\mathtt{t+1}}) \tag{3.35}$$

Though (3.34) might seem more intuitive (it can be read as "if the preconditions are fulfilled, move to the next location"), and is in a way closer to CNF (after rewriting →, note that (3.35) is actually in DNF), our encoding is based (3.35), cf. Section 3.1. The reason is that (3.34) cannot handle nondeterminism. To illustrate this, consider the real-time system part in Figure 3.8. If $(x \leq 2)$, both clock guards are satisfied, i.e., both transitions are enabled, and the next location is chosen nondeterministically. The representation (omitting irrelevant constraints) of

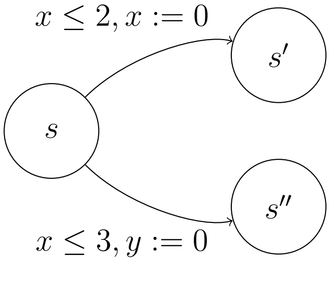

Figure 3.8: Representation of Transitions, Motivation

the two transitions in Figure 3.8, based on the two alternatives presented above, is given in (3.36) respectively (3.37).[12]

[12] Note that (3.37) does not entirely correspond to the representations defined in Section 3.1, but is used for illustration purposes only.

$$\begin{aligned}&(\mathtt{s_t}\wedge(\mathtt{z_t}-\mathtt{x_t}\leq 2)\rightarrow \mathtt{s'_t}\wedge(\mathtt{x_{t+1}}=\mathtt{z_{t+1}})\wedge(\mathtt{y_{t+1}}=\mathtt{y_t})\wedge(\mathtt{z_{t+1}}=\mathtt{z_t}))\wedge\\&(\mathtt{s_t}\wedge(\mathtt{z_t}-\mathtt{x_t}\leq 3)\rightarrow \mathtt{s''_t}\wedge(\mathtt{y_{t+1}}=\mathtt{z_{t+1}})\wedge(\mathtt{x_{t+1}}=\mathtt{x_t})\wedge(\mathtt{z_{t+1}}=\mathtt{z_t}))\end{aligned}\tag{3.36}$$

$$\begin{aligned}&(\mathtt{s_t}\wedge(\mathtt{z_t}-\mathtt{x_t}\leq 2)\wedge \mathtt{s'_t}\wedge(\mathtt{x_{t+1}}=\mathtt{z_{t+1}})\wedge(\mathtt{y_{t+1}}=\mathtt{y_t})\wedge(\mathtt{z_{t+1}}=\mathtt{z_t}))\vee\\&(\mathtt{s_t}\wedge(\mathtt{z_t}-\mathtt{x_t}\leq 3)\wedge \mathtt{s''_t}\wedge(\mathtt{y_{t+1}}=\mathtt{z_{t+1}})\wedge(\mathtt{x_{t+1}}=\mathtt{x_t})\wedge(\mathtt{z_{t+1}}=\mathtt{z_t}))\end{aligned}\tag{3.37}$$

The obvious problem in (3.36) is the fact that there exists no satisfying valuation in case $x \leq 2$: the left hand side of both implications is satisfied, but due to the mutual exclusion constraint in locations,[13] only one of the right hand sides can be satisfied; thus, no satisfying valuation exists.

A second problem with a representation according to (3.34) is the fact that logical implication allows the right hand side to be true, while the left hand side is false. For the transition relation, this means that it would be possible to find a satisfying valuation for the next step, even if there is no satisfying valuation for the current step.

3.4 Conclusion

In this Chapter, we have established the formal basis for extensive model checking and verification of the real-time systems presented in Chapter 2.

In Section 3.1, we have presented an encoding in propositional logic with linear rational arithmetic for each of the system models defined in Chapter 2. The representation of TA, as defined in Section 3.1.2, has essentially been presented in [KP07]. The representation of TCA, as defined in Section 3.1.3, is an extension of the work presented in [Kem11] (which in turn in based on [Kem09]). Since in Chapter 2, we have extended the formal model of TCA from [ABdBR07] (which was also used in [Kem11]) with memory cells, consequently this extension had to be included in the representation as well. Equivalently, we have extended the TNA model from [Kem10] with data values and memory cells in Section 2.4, and have extended the representation of TNA in Section 3.1.4 accordingly. For both TCA and TNA, we have sketched the difficulties that arise from memory cells being used in source and target location of the same transition in Section 3.1.1.5.

We have shown how to apply Bounded Model Checking to the representation of real-time systems in Section 3.2. The notion of unfolding (Section 3.2.2) and the concepts for representation of properties (Section 3.2.3) have been presented in previous work, and have been restated here for clarity and consistency. We have provided a discussion on completeness of BMC in Section 3.2.4, and a proof of correctness in Section 3.2.5. The latter in particular takes into account the new representation features regarding memory cells, data variables and data values.

Finally, in Section 3.3, we have provided an extensive discussion on other possibilities to represent real-time systems (using different encodings or variable types, for example), and we have motivated our design decisions.

[13] This problem would occur in the very same way for logarithmic encoding, since the current location is unique at any time.