AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Slicing CSP-OZ Specifications for Verification

by

Ingo Brückner Heike Wehrheim

AVACS Technical Report No. 7
November 2005
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Martin Fränzle, Ernst-Rüdiger Olderog,
Andreas Podelski, Reinhard Wilhelm
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Author(s) contact: Heike Wehrheim (`hw@avacs.org`).

Slicing CSP-OZ Specifications for Verification*

Ingo Brückner[1] and Heike Wehrheim[2]

[1]Universität Oldenburg, Department Informatik, 26111 Oldenburg, Germany
ingo.brueckner@informatik.uni-oldenburg.de
[2]Universität Paderborn, Institut für Informatik 33098 Paderborn, Germany
wehrheim@uni-paderborn.de

Abstract. Model checking specifications with complex data and behaviour descriptions often fails due to the large state space to be processed. In this paper we propose a technique for *reducing* such specifications (with respect to certain properties under interest) before verification. The method is an adaption of the *slicing technique* from program analysis to the area of integrated formal notations and temporal logic properties. It solely operates on the syntactic structure of the specification which is usually significantly smaller than its state space. We show how to build a reduced specification via the construction of a so called program dependence graph, and prove correctness of the technique with respect to a projection relationship between full and reduced specification. The reduction thus preserves all properties formulated in temporal logics which are invariant under stuttering, as for instance LTL_{-X}.

1 Introduction

Modelling complex systems usually involves the description of different views. In the UML this is facilitated by providing designers with a large number of different diagram types for modelling various aspects of systems. In the area of formal modelling notations *integrated formal methods* allow for a convenient specification of different views. Integrated formalisms combine different existing notations while still giving a semantics to the combination and thus preserving the formal rigour in a design. Models of complex systems in integrated specification formalisms usually contain views describing state-based aspects plus views describing the dynamic behaviour. A number of such integrations have been proposed in recent years [6, 26, 18, 23, 21, 14, 20, 7, 12]. They often combine state-based notations like Z or B with process algebras like CCS or CSP.

In this paper, we will be concerned with *verifying* specifications written in an integrated notation. Applications of model checking techniques often fail for such specifications due to the large amount of data (coming from the state-based side)

* This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See `www.avacs.org` for more information. A shorter version of this paper, [1], was presented at ICFEM2005.

combined with the large number of interleavings of parallel components (coming from the process algebra side). Consequently, the development of techniques for avoiding the state explosion problem is even more compelling for integrated formalisms. Here, we propose a method for reducing the specification (and as a consequence its state space) by removing all those parts which are irrelevant for the validity of a particular property under interest. The technique for determining relevant (or irrelevant) parts is an adaption of the *slicing* technique from program analysis to formal specifications. Slicing has originally been introduced by Weiser [25] to reduce programs for debugging. It basically involves the construction of a *program dependence graph* which precisely reflects the dependencies in a program. On this graph it is possible to determine the parts of a program which might affect the value of a variable at a certain program point (the slicing criterion). The irrelevant parts can then be sliced away (for an overview of slicing techniques see [22]). A similar principle is applied in hardware verification (under the name *cone of influence reduction* [13, 5]), where the parts of a circuit model are calculated which might affect a certain property. In software verification slicing has for instance been applied to Java [8], PROMELA (the input language of the SPIN model checker [15]), and SAL [24]. Being a static analysis technique slicing just operates on the *syntactic* level of the program, and a reduction of this can substantially facilitate the following model checking.

This work builds on previous ideas for slicing Object-Z specifications [2]. Here, we present a slicing technique for an *integrated* specification language. The formalism, called CSP-OZ [6], is a combination of the process algebra CSP [9] with the state-based formalism Object-Z [19]. For this notation we show how to construct graphs reflecting the mutual dependencies in a specification, in particular between the Object-Z and the CSP part. The slicing criteria are temporal logic formulae over atomic propositions (speaking about the state of the Object-Z part) *and* events (speaking about occurrence of operations of the CSP part). Instead of looking at one particular logic, we take a more general approach. We show that our reduction preserves properties formulated in any (linear-time) logic which is invariant under *stuttering*, i.e. which cannot distinguish between runs of a system which are equivalent up to some stuttering steps (defined by a set of irrelevant atomic propositions and events). This is obtained by proving that the runs of the reduced specification are *projections* of the runs of the full specification, projection being a particular form of stuttering. A logic fulfilling the requirements is for instance LTL$_{-X}$ (linear time temporal logic without Next operator) or the state/event based interval logic proposed in [2].

The paper is structured as follows. The next section introduces CSP-OZ by means of a small example and moreover defines a Kripke structure semantics for CSP-OZ. In section 3 we present the dependence graph construction and the slicing algorithm, both illustrated on the running example. The slicing algorithm will be proven correct with respect to preservation of the property under interest in section 4. The last section concludes and shortly discusses related work.

2 CSP-OZ Specifications: An Example

For illustrating our approach we use a CSP-OZ specification of an air condition system. It can operate in two modes, either heating or cooling. Initially the air condition is off. When it is switched on (*workswitch*), it starts to run. While running, the air condition either heats or cools the room and simultaneously allows the user to switch the mode (*modeswitch*), refill fuel (*refill*) or switch it off again. Cooling or heating is modelled by a consumption of one unit of fuel (*consume*) and an emission of hot or cold air (*dtemp*). For the specification we first define the mode of operating and a type for the fuel.

$$Mode ::= heat \mid cool \qquad Fuel == 0..100$$

AC

$\mathtt{chan}\ workswitch, consume, off \qquad \mathtt{chan}\ modeswitch : [m? : Mode]$
$\mathtt{chan}\ refill : [f? : Fuel] \qquad \mathtt{chan}\ dtemp : [t! : Mode]$
$\mathtt{chan}\ level : [f! : Fuel]$

$$\begin{array}{ll}
\mathtt{main} = workswitch \rightarrow On & Operate = modeswitch \rightarrow Operate \\
On = (Operate \mid\mid\mid Work) \fatsemi main & \qquad \Box\ refill \rightarrow Operate \\
Work = consume \rightarrow dtemp \rightarrow level \rightarrow Work & \qquad \Box\ workswitch \rightarrow SKIP \\
\qquad \Box\ off \rightarrow SKIP &
\end{array}$$

$work : \mathbb{B}$
$mode : Mode;\ fuel : Fuel$

$\mathtt{Init}$
$\neg work$
$mode = heat$

$\mathtt{effect\_}workswitch$
$\Delta(work)$
$work' = \neg work$

$\mathtt{enable\_}consume$
$work \land fuel > 5$

$\mathtt{effect\_}consume$
$\Delta(fuel)$
$fuel' = fuel - 1$

$\mathtt{effect\_}modeswitch$
$\Delta(mode);\ m? : Mode$
$mode' = m?$

$\mathtt{effect\_}dtemp$
$t! : Mode$
$t! = mode$

$\mathtt{effect\_}level$
$f! : Fuel$
$f! = fuel$

$\mathtt{enable\_}off$
$\neg work$

$\mathtt{enable\_}refill$
$fuel < 100$

$\mathtt{effect\_}refill$
$\Delta(fuel);\ f? : Fuel$
$fuel' = min(fuel + f?, 100)$

The first part of the class defines its interface towards the environment. The next part specifies its dynamic behaviour, i.e. the allowed ordering of method execution. It is defined via a set of CSP process equations. The operators appearing here are prefixing → (sequencing), sequential composition $\fatsemi$, interleaving $\mid\mid\mid$ (parallel composition with no synchronisation) and external choice □. The third part of a CSP-OZ class describes the attributes of the class and the methods. For every method we might have an *enabling* schema fixing a guard for the method execution (enabling schemas equivalent to *true* are left out) and an effect schema describing the effect of a method upon execution. For instance, for method *consume* the enabling schema tells us that the air condition has to be on and a minimal amount of fuel is necessary for *consume* to take place, and that upon execution one unit of fuel is consumed. The method *level* on the other hand is always enabled, it just displays the current level of fuel.

The semantics of such specifications is defined in terms of labelled Kripke structures. In contrast to ordinary Kripke structures, transitions are labelled with events. This allows us to also use temporal logics for property specification which talk about execution of events.

Definition 1. *Let AP be a non-empty set of atomic propositions, E an alphabet of events (consisting of method names plus values of parameters).*

An (event-)labelled Kripke structure $K = (S, S_0, \rightarrow, L)$ *over AP and E consists of a finite set of states S, a set of initial states $S_0 \subseteq S$, a transition relation $\rightarrow \subseteq S \times E \times S$ and a labelling function $L : S \rightarrow 2^{AP}$.*

For our example atomic propositions might for instance be $mode = cool$ or $fuel > 5$. The Kripke structure for a CSP-OZ class is derived in two steps: first, we separately compute the semantics of the CSP and the Object-Z part. In a second step, we combine the Kripke structure of the components by parallel composition. In the following we assume a global set of atomic propositions AP and events E which are built over method names $m \in M$, i.e. an event e has the form $m.i.o$ where m is the name of a method and i and o are (potential) values for input and output parameters. The transition relation for the CSP part is computed via the operational semantics of CSP [17].

Definition 2. *The* Kripke structure semantics *of the CSP part* `main` *of a CSP-OZ class is the labelled Kripke structure $K^{CSP} = (\mathcal{L}^{CSP}, \{\mathtt{main}\}, \rightarrow^{CSP}, L^{CSP})$ with $\mathcal{L}^{CSP}$ being the set of all CSP terms, $\rightarrow^{CSP}$ the transition relation derived via the operational semantics of CSP and $L^{CSP}(P) = AP$ for all $P \in \mathcal{L}^{CSP}$.*

In the states of the Kripke structure for the CSP part all atomic propositions hold since the CSP part makes no restrictions on values of attributes of the class.

Definition 3. *The* Kripke structure semantics *of the Object-Z part $C = (State, Init, (\mathtt{enable\_}m)_{m \in M}, (\mathtt{effect\_}m)_{m \in M})$ of a CSP-OZ class is the labelled Kripke structure $K^{OZ} = (State, Init, \rightarrow^{OZ}, L^{OZ})$ with the transition relation $\rightarrow^{OZ} = \{(s, m.i.o, s') \mid \mathtt{enable\_}m(s, i) \wedge \mathtt{effect\_}m(s, i, o, s')\}$, and the labelling function $L^{OZ}(s) = \{p \in AP \mid s \models p\}$.*

The states of the Kripke structure are simply the set of bindings of the state schema. These two Kripke structures are then combined via parallel composition. In the following we assume the alphabet of the CSP part and the set of methods in the Object-Z part to be equal, thus synchronisation takes places on all methods. Only one event remains which is executed by the CSP part alone, the invisible event τ which might arise out of internal choices in CSP processes.

Definition 4. *The* parallel composition *of two labelled Kripke structures* [1] $K_i = (S_i, S_{0,i}, \rightarrow_i, L_i)$*,* $i \in \{1,2\}$ *over the same set of atomic propositions* AP *and events* E*,* $K_1 \mid\mid K_2$*, is the Kripke structure* $K = (S, S_0, \rightarrow, L)$ *with*

- $S = S_1 \times S_2$, $S_0 = S_{0,1} \times S_{0,2}$,
- $\rightarrow = \left\{ ((s_1, s_2), e, (s_1', s_2')) \;\middle|\; \begin{array}{l} (s_1 \xrightarrow{e}_1 s_1' \wedge s_2 \xrightarrow{e}_2 s_2') \\ \vee\ (s_1 \xrightarrow{\tau}_1 s_1' \wedge s_2' = s_2) \vee (s_2 \xrightarrow{\tau}_2 s_2' \wedge s_1' = s_1) \end{array} \right\}$
- $L(s) = L(s_1) \cap L(s_2)$*, where* $s = (s_1, s_2)$.

For describing properties of CSP-OZ classes we can now use any temporal logic which can be interpreted on labelled Kripke structures. For the purpose of this paper we assume the logic to be a *linear-time* logic, i.e. which is interpreted on the *paths* without considering the branching structure. We furthermore only consider paths which are fair [5] with respect to a set of events.

Definition 5. *Let* $K = (S, S_0, \longrightarrow, L)$ *be a Kripke structure. An infinite sequence of events and states* $s_0 e_1 s_2 e_3 s_4 \ldots$ *is a* path *of the Kripke structure iff* $s_0 \in S_0$ *and* $(s_i, e_{i+1}, s_{i+2}) \in \rightarrow$ *holds for all* $i \geq 0, i$ *even.*

A path is fair *with respect to a set of events* $E' \subseteq E$ *(or* E'-fair*) iff* $inf(\pi) \cap E' \neq \varnothing$ *where* $inf(\pi) = \{e \in E \mid \exists$ *infinitely many* $i \in \mathbb{N} : e_i = e\}$.

Here, we will not introduce one particular logic, but instead only assume that our logic is invariant under projection, i.e. that it cannot distinguish paths where one is a projection of the other onto some set of atomic propositions and events of interest. A precise definition of projection is given in section 4. A temporal logic fulfilling this requirement is for instance the next-less part of LTL or the state-event interval logic presented in [2]. For our example we use the former logic. One property of interest for our air condition specification could for instance be whether the amount of fuel is always greater than 5 when the air condition is on (which in fact is not true): $\varphi := \Box(\mathit{work} \Rightarrow \mathit{fuel} > 5)$.

The main purpose of the technique proposed in this paper is to determine now which part of the specification actually has to be considered when checking for the property, i.e. whether it is possible to check the property on a reduced specification S^{red} such that the following holds (where $S \models \varphi$ stands for "the formula φ holds on the Kripke structure of the specification S"):

$$S \models \varphi \text{ iff } S^{red} \models \varphi$$

As we will see it is possible to omit both some of the attributes and some of the methods of the air condition for checking our property.

[1] Note that our definition is symmetric in general, while for the special case of parallel composition of CSP and Object-Z Kripke structures we assume only the CSP side to have τ transitions.

3 Slicing

Slicing means reducing a program or specification such that the reduced program/specification only contains those parts of the full specification which can influence a certain property under interest called the slicing criterion.

In order to determine these influences, slicing needs precise information about dependencies between different parts of a program/specification. Such dependencies are represented in a *program (or system) dependence graph*[2]. This section explains the construction of program dependence graphs for CSP-OZ classes and their slicing with respect to some temporal logic formula φ.

Control flow graph. In preparation for the construction of the program dependence graph we first construct the specification's control flow graph (CFG) which represents the execution order of the specification's schemas according to the specification's CSP processes. Starting with the $start.main$ node, its nodes ($n \in N$) and edges ($\longrightarrow \subseteq N \times N$) are derived from the syntactical elements of the specification's CSP part, based on an inductive definition for each CSP operator. Nodes either correspond to schemas of the Object-Z part (like $\texttt{enable\_}m$) or to operators in the CSP part (like nodes *interleave* and *uninterleave* for operator ||| or nodes *extchoice* and *unextchoice* for operator □). We refrain from giving a precise definition here. The result of this inductive definition for the first two process definitions in our AC example specification can be seen in fig. 1.

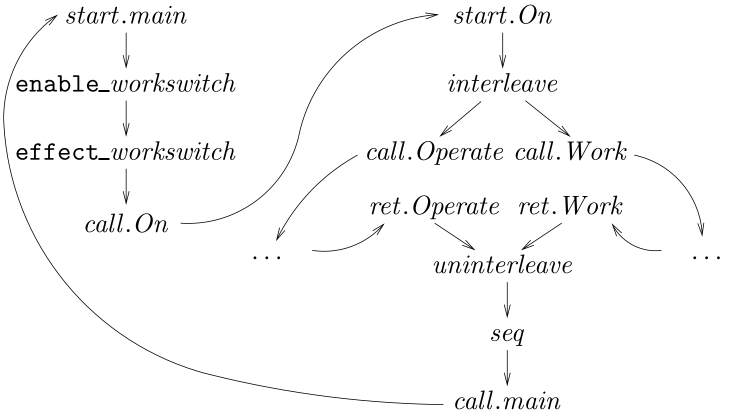


Fig. 1. Part of the control flow graph for the AC specification

Note, that we assume each syntactical CSP element and each associated CFG node to have a unique name. This can, for example, be achieved by extending their names by an index that represents the position of their textual occurrence inside the specification. For sake of clarity we omit these indices here.

[2] We stick to the word program, although we treat specifications.

Program dependence graph. The program dependence graph (PDG) represents data and control dependencies between nodes of the CFG. Thus both graphs have the same set of nodes ($n \in N$), but not the same set of edges. An edge connects two nodes in the PDG if control or data dependencies exist between these nodes according to the definitions given below. Before we continue with the construction of the PDG we first introduce some abbreviations. When reasoning about paths inside the CFG, we let $path_{CFG}(n, n')$ denote the set of sequences of CFG nodes that are visited when walking along CFG edges from node n to node n'. When we refer to the sets of variables appearing inside schemas associated to PDG nodes, we let $mod(n)$ denote the set of variables appearing in primed form (those modified by the method of the node), $ref(n)$ denote the set of variables appearing in unprimed form (those referenced by the method of the node), and $vars(n) = mod(n) \cup ref(n)$ denote the set of all variables inside the schema.

The further construction of the PDG starts with the introduction of *control dependence edges* ($\rightarrowtail \subseteq N \times N$). The idea behind these edges is to represent the fact that an edge's source node controls whether the target node will be executed. In particular, a node cannot be control dependent on itself. We distinguish the following types of control dependence edges:

- Control dependence due to *nontrivial precondition* exists between an `enable` node and its `effect` node iff the `enable` schema is non-empty (i.e. not equivalent to true).
- Control dependence due to *external (resp. internal) choice* exists between an *extch* (resp. *intch*) node and its immediate CFG successors.

Additionally, some further control dependence edges are introduced in order to achieve a well-formed graph:

- *Call* and *termination* edges exist between a *call* (resp. *term*) node and its associated *start* (resp. *ret*) node.
- *Start* and *return* edges exist between a *start* (resp. *ret*) node and its immediate CFG successor.

Finally, all previously defined (direct) control dependence edges are extended to CFG successor nodes as long as they do not bypass existing control dependence edges. The idea of this definition is to integrate indirectly dependent nodes (that would otherwise be isolated) into the PDG.

- *Indirect control dependence* edges exist between two nodes n and n' iff $\exists\, \pi \in path_{CFG}(n, n') : \forall\, m, m' \in \operatorname{ran} \pi : m \rightarrowtail m' \Rightarrow m = n$

The idea of *data dependence edges* ($\rightsquigarrow \subseteq N \times N$) is to represent the influence that one node might have on a different node by modifying some variable that the second node references. Therefore, the source node represents always an `effect` schema, while the target node may also represent an `enable` schema. We distinguish the following types of data dependence edges:

- *Direct data dependence* exists between two nodes n and n' iff there is a CFG path between both nodes without any further modification of the relevant variable: $\exists\, v \in (\mathit{mod}(n) \cap \mathit{ref}(n'))\,, \exists\, \pi \in \mathit{path}_{\mathit{CFG}}(n, n')$:

$$\forall\, m \in \operatorname{ran} \pi \colon v \in \mathit{mod}(m) \Rightarrow (m = n \vee m = n')$$

- *Interference data dependence* exists between two nodes n and n' iff both nodes are located in different CFG branches attached to the same interleaving operator: $\mathit{mod}(n) \cap \mathit{ref}(n') \neq \varnothing \wedge \exists\, m = \mathit{interleave}$:

$$\exists\, \pi \in \mathit{path}_{\mathit{CFG}}(m, n) \wedge \exists\, \pi' \in \mathit{path}_{\mathit{CFG}}(m, n') \colon \operatorname{ran}\ \pi \cap \operatorname{ran} \pi' = \{m\}$$

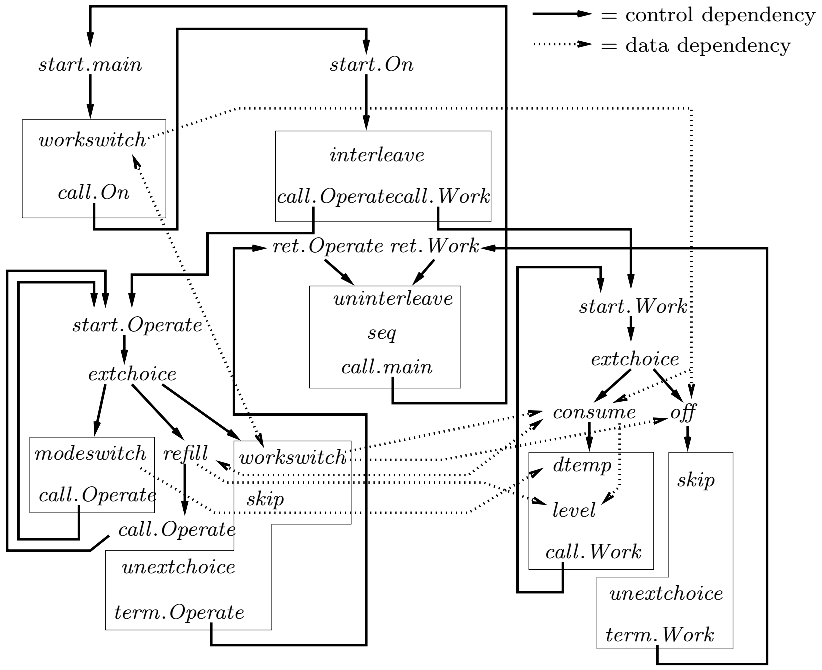


Fig. 2. Program dependence graph for the *AC* specification

The resulting program dependence graph for the *AC* specification is depicted in fig. 2. Note, that two aspects of the PDG have been slightly modified in order to achieve a more concise graphical representation without changing the outcome of the slicing algorithm for the given example.

1. The separate nodes for `enable` and `effect` schemas have been combined into one single node for each event.
2. Instead of explicitly drawing all control dependence edges originating from one node to different target nodes, this set of edges is represented by a single edge between the first node and a box around the set of target nodes.

Backward slice. For our purpose, slicing is used to determine that part of the specification that is directly or indirectly relevant for the property to be verified. Computation of this slice starts from the set of events E_φ and the set of variables V_φ that appear directly in the given formula φ. Based on this slicing criterion $(E_\varphi,\ V_\varphi)$ we can determine the set of PDG nodes with direct influence on the property under interest:

$$N_\varphi = \{n \mid mod(n) \cap V_\varphi \neq \varnothing\} \ \cup\ \{n \mid \exists\, e \in E_\varphi \colon n = \texttt{enable\_}e\}$$

From this initial set of nodes we compute the backward slice by a reachability analysis of the PDG. The resulting set contains all nodes that lead via an arbitrary number of control or data dependence edges to one of the nodes that already are in N_φ. Additional to all nodes from N_φ, the backward slice contains therefore also all PDG nodes with indirect influence on the given property, i.e. it is the set of all relevant nodes for the specification slice:

$$N' = \{n' \in N \mid \exists\, n \in N_\varphi \colon n'\ (\rightarrowtail \cup \rightsquigarrow)^*\ n\}$$

Thus relevant events are those associated to nodes from N'

$$E' = \{e \mid \exists\, n \in N' \colon n = \texttt{enable\_}e_i \vee n = \texttt{effect\_}e_i\}$$

and relevant variables are those associated to nodes from N': $V' = \bigcup_{n \in N'} vars(n)$.

Reduced specification. Having determined the sets E' and V' which might influence the property (formula) under interest the slice of a specification can next be determined. In contrast to the original specification it contains

- only channels from E',
- only CSP process definitions that are projections (as defined in sect. 4, def. 8) of CSP process definitions from the original specification onto E',
- inside the state schema only variables from V',
- inside the Init schema only predicates restricting variables from V', and
- only Object-Z schemas associated with events from E'.

When slicing the class AC with respect to the formula $\varphi : \Box(work \Rightarrow fuel > 5)$, i.e. $N_\varphi = \{\!|\ workswitch, consume, refill\ |\!\}^3$, the result is the following:

$$N' = N \setminus \{\texttt{effect\_}modeswitch, \texttt{effect\_}dtemp, \texttt{effect\_}level\}$$
$$E' = E \setminus \{\!|\ modeswitch, dtemp, level\ |\!\}, \qquad V' = V \setminus \{mode\}$$

This leads to the following specification slice:

AC

$\texttt{chan}\ workswitch, consume, off \qquad\qquad \texttt{chan}\ refill : [f? : Fuel]$

[3] Let $\{\!|\ M\ |\!\}$ denote the set of events over the set of methods M.

$$\begin{aligned} \mathtt{main} &= \mathit{workswitch} \rightarrow \mathit{On} \\ \mathit{On} &= (\mathit{Operate} \mid\mid\mid \mathit{Work}) \fatsemi \mathit{main} \\ \mathit{Work} &= \mathit{consume} \rightarrow \mathit{Work} \\ &\quad \Box\ \mathit{off} \rightarrow \mathrm{SKIP} \end{aligned}$$

$$\begin{aligned} \mathit{Operate} &= \mathit{Operate} \\ &\quad \Box\ \mathit{refill} \rightarrow \mathit{Operate} \\ &\quad \Box\ \mathit{workswitch} \rightarrow \mathrm{SKIP} \end{aligned}$$

$\mathit{work} : \mathbb{B};\ \mathit{fuel} : \mathit{Fuel}$

`Init`

$\neg \mathit{work}$

`effect`_$\mathit{workswitch}$

$\Delta(\mathit{work})$

$\mathit{work}' = \neg \mathit{work}$

`enable`_$\mathit{consume}$

$\mathit{work} \wedge \mathit{fuel} > 5$

`effect`_$\mathit{consume}$

$\Delta(\mathit{fuel})$

$\mathit{fuel}' = \mathit{fuel} - 1$

`enable`_off

$\neg \mathit{work}$

`enable`_refill

$\mathit{fuel} < 100$

`effect`_refill

$\Delta(\mathit{fuel});\ f? : \mathit{Fuel}$

$\mathit{fuel}' = \mathit{min}(\mathit{fuel} + f?, 100)$

The reductions achieved by applying our slicing algorithm to this example are:

1. Event *modeswitch* has been removed together with variable *mode*, which is sensible, since the air condition's mode (heating or cooling) does not have any influence on the slicing criterion (property $\Box(\mathit{work} \Rightarrow \mathit{fuel} > 5)$).
2. Events *dtemp* and *level* have been removed, which is also sensible, since neither modelling the effect on the environment (*dtemp*) nor communicating the current amount of fuel (*level*) influences the given property.

To summarise, the specification's state space has not only been reduced with respect to its control flow space (events *dtemp*, *modeswitch* and *level*) but also with respect to its data state space (variable *mode*).

Note, that neither the original nor the sliced AC specification satisfies the given property, so the verification result will be negative in both cases. Nevertheless, this is exactly what we wanted to achieve: A specification slice must satisfy a slicing criterion if and only if the original specification does so.

In the next section we will show that our slicing algorithm guarantees this outcome for any specification and any slicing criterion (formulated in a linear-time stuttering invariant logic).

4 Correctness

In this section we show correctness of the slicing algorithm, i.e. we show that the Kripke structure of the reduced specification is a projection of that of the full specification. As a consequence, the property (and slicing criterion) φ then holds on the full specification if and only if it holds on the reduced specification.

We start with the definition of a notion of projection that is used in the actual correctness proof.

Projection of event-labelled Kripke structures. The task of slicing is to compute a reduced specification which satisfies a certain property if and only if the full specification satisfies it. For proving this we will show that the reduced specification is a *projection* of the full specification onto some relevant subset of the atomic propositions and events, i.e. they only differ on atomic propositions and events that the formula does not mention.

The projection relation is first defined on paths and then lifted to Kripke structures. Intuitively, when computing the projection of a given path onto a set of atomic propositions and a set of events, one divides the path into blocks such that all states inside a block are projection-equivalent (i.e. they coincide on the given set of atomic propositions) and all events inside a block are irrelevant events (i.e. events not from the given set of events) except for the last event in the block which is a relevant event (i.e. an event from the given set of events). The projection of the original path contains then any path such that for each of the blocks of the original path all states and irrelevant events are mapped onto one single state of the new path, while the relevant event remains in the new path as illustrated in the following sketch of a projection of a path:

	Block 0	Block 1	Block 2	Block 3
$\pi \;=$	$s_0\ e_0\ s_1\ e_1$	$s_2\ e_2$	$s_3\ e_3\ s_4\ e_4$	$\dots$
$Pr(\pi) \;\ni$	$r_0\ e_1$	$r_1\ e_2$	$r_2\ e_4$	$\dots$

Definition 6. *Let $\pi = s_0 e_0 s_1 e_1 s_2 e_2 s_3 \dots$ be an E'-fair path over a set of atomic propositions AP and a set of events $E \supseteq E'$. The* projection of π onto a set of atomic propositions AP' and a set of events E' *$(Pr_{AP',E'}(\pi))$ contains any E'-fair path $\rho = r_0 f_0 r_1 f_1 r_2 f_2 r_3 \dots$ such that there is a sequence of indices $0 = i_0 < i_1 < i_2 < \dots$ (that divides π into blocks) with*

- $\forall\, k \geq 0$: $L(s_{i_k}) \cap AP' = L(s_{i_k+1}) \cap AP' = \cdots = L(s_{i_{k+1}-1}) \cap AP' = L(r_k) \cap AP'$ *(relevant atomic propositions do not change within a block and are the same in the correspondent state of ρ),*
- $\forall\, l \in \mathbb{N}, \forall\, k : i_l \leq k < i_{l+1} - 1 : e_k \in E \setminus E'$ *(no relevant events occur inside a block),*
- $\forall\, l \geq 1 : e_{i_l - 1} = f_{l-1} \in E'$ *(transitions between blocks are labelled with the same relevant event as the correspondent transition of ρ).*

For comparing the Kripke structures we restrict the definition to fair paths since we are only considering satisfaction of formulae on fair paths.

Definition 7. *Let $K_i = (S_i, S_{0,i}, \rightarrow_i, L_i)$, $i \in \{1, 2\}$, be labelled Kripke structures over a set of atomic propositions AP and a set of events E, $AP' \subseteq AP$ a subset of the atomic propositions and $E' \subseteq E$ a subset of the events.*

K_2 is in the projection *of K_1 onto AP' and E' ($K_2 \in Pr_{AP',E'}(K_1)$) iff the following holds:*

1. *For each E'-fair path π in K_1 there exists an E'-fair path π' in K_2 such that $\pi' \in Pr_{AP',E'}(\pi)$,*
2. *and vice versa, for each E'-fair path π' in K_2 there exists an E'-fair path π in K_1 such that $\pi' \in Pr_{AP',E'}(\pi)$.*

Given a temporal logic which is interpreted on paths (i.e. a linear time logic) and which is invariant under projections, such a projection relationship between two Kripke structures then guarantees that formulae which only mention propositions from AP' and events from E' hold for either both or none of the Kripke structures. The proof for formulae of state/event interval logic (SE-IL) can be found in [3]. Note that projection is a particular form of stuttering.

In the following we will show how such a projection relationship can be proven between full and sliced specification. For this we now first have to give a precise definition of the residual CSP processes which remain after slicing with respect to some set of events E'.

Definition 8. *Let P be the right side of a process definition from the CSP part of a specification and E be the set of communications that appear in the specification. The projection of P w.r.t. a set of communications $E' \subseteq E$ is inductively defined:*

1. $skip|_{E'} := skip$ *and* $stop|_{E'} := stop$
2. $(e \rightarrow P)|_{E'} := \begin{cases} P|_{E'} & \text{if } e \notin E' \\ e \rightarrow P|_{E'} & \text{else} \end{cases}$
3. $(P \circ Q)|_{E'} := P|_{E'} \circ Q|_{E'}$ *with* $\circ \in \{;\ , |||, \sqcap, \square\}$

The projection of the complete CSP part w.r.t. a set of communications $E' \subseteq E$ is defined by applying the above definition to the right side of each process definition.

Correctness proof. Now we start the actual correctness proof with several lemmas showing the relationships between CSP processes and events and variables which remain in the specification.

In the proofs we use the notation of the last section, i.e. let N', E', V' denote the nodes, events, variables which remain in the specification or PDG after slicing and $\overline{V}$, $\overline{AP}$ are the variables and atomic propositions, respectively, on which the full and reduced specification should agree.

Our first lemma considers the case of a single CSP transition: Either this transition is labelled with a relevant event $e \in E'$ or with an irrelevant event $e \notin E'$. In the former case it is easy to see that the associated projection also can perform this event e, while in the latter case some further considerations lead to the conclusion that the associated projection will finally perform the same relevant event as the original process.

Lemma 1 (Transitions of CSP process projections). *Let P and Q be two CSP processes with $P \xrightarrow{e} Q$ and E' a set of relevant events. The projections of P and Q with respect to E' are related as follows:*

1. *If e is a relevant event, the projection of P can perform this event which leads to the projection of Q:*

   $$e \in E'\colon P|_{E'} \xrightarrow{e} Q|_{E'}$$

2. *If e is no relevant event, then the projection of P can mimic any behaviour of the projection of Q:*

   $$e \notin E'\colon \forall f, \forall R\colon \left(\begin{array}{l} Q|_{E'} \xrightarrow{f} R|_{E'} \\ \Rightarrow P|_{E'} \xrightarrow{f} R|_{E'} \end{array}\right)$$

Proof: We show both cases by induction over the structure of P. Since we know that P can perform event e, we only have to consider a limited set of CSP constructs.

1. e is a relevant event:
   (a) $P \equiv e \longrightarrow Q$. $\checkmark$
   (b) $P \equiv P_1 \Box P_2$ with $P_i \xrightarrow{e} Q$ for $i \in \{1,2\}$.
   Then $P|_{E'} \equiv P_1|_{E'} \Box P_2|_{E'}$ and, since $P_i \xrightarrow{e} Q$, the induction assumption leads us to $P|_{E'} \xrightarrow{e} Q|_{E'}$. $\checkmark$
   (c) $P \equiv P_1 \mid\mid\mid P_2$ with $P_i \xrightarrow{e} P_i'$ for $i \in \{1,2\}$ and $Q \equiv \begin{cases} P_1' \mid\mid\mid P_2 & \text{if } i = 1 \\ P_1 \mid\mid\mid P_2' & \text{else} \end{cases}$
   Then $P|_{E'} \equiv P_1|_{E'} \mid\mid\mid P_2|_{E'}$ and, since $P_i \xrightarrow{e} P_i'$, we have $P|_{E'} \xrightarrow{e} Q|_{E'}$ with $Q|_{E'} \equiv \begin{cases} P_1'|_{E'} \mid\mid\mid P_2|_{E'} & \text{if } i = 1 \\ P_1|_{E'} \mid\mid\mid P_2'|_{E'} & \text{else} \end{cases}$ $\checkmark$.
2. e is no relevant event:
   (a) $P \equiv e \longrightarrow Q$.
   Then $P|_{E'} \equiv Q|_{E'}$. $\checkmark$
   (b) $P \equiv P_1 \Box P_2$ with $P_i \xrightarrow{e} Q$ for $i \in \{1,2\}$.
   Then $P|_{E'} \equiv P_1|_{E'} \Box P_2|_{E'}$ and, since $P_i \xrightarrow{e} Q$, it follows from the induction assumption that $\forall f\colon Q|_{E'} \xrightarrow{f} R|_{E'} \quad \Rightarrow \quad P_i|_{E'} \xrightarrow{f} R|_{E'}$ and thus the further implication $P|_{E'} \xrightarrow{f} R|_{E'}$. $\checkmark$
   (c) $P \equiv P_1 \mid\mid\mid P_2$ with $P_i \xrightarrow{e} P_i'$ for $i \in \{1,2\}$ and $Q \equiv \begin{cases} P_1' \mid\mid\mid P_2 & \text{if } i = 1 \\ P_1 \mid\mid\mid P_2' & \text{else} \end{cases}$
   From the induction assumption it follows that $\forall f\colon P_i'|_{E'} \xrightarrow{f} R|_{E'} \quad \Rightarrow \quad P_i|_{E'} \xrightarrow{f} R|_{E'}$ and, since the other component of P remains unchanged during the transition from P to Q, it is obvious that $\forall f\colon Q|_{E'} \xrightarrow{f} R|_{E'} \quad \Rightarrow \quad P|_{E'} \xrightarrow{f} R|_{E'}$. $\checkmark$.

The next lemma extends the previous one from single transitions to transition sequences associated to complete projection blocks. It states that the projection of each residual CSP process associated to a state inside a projection block as defined in definition 6 can mimic the behaviour of the residual CSP process associated to the last state of the projection block, i.e. the relevant event at the end of the block is enabled at any previous step inside the block when computing the CSP projection.

Lemma 2 (Transitivity of the previous lemma). *Let $P_j, \ldots, P_{j+k+1}$ be CSP processes, E' a set of relevant events, $e_{j+1}, \ldots, e_{j+k-2}$ irrelevant events ($\notin E'$), and e_{j+k} a relevant event ($\in E'$), such that $P_j \xrightarrow{e_{j+1}} P_{j+2} \xrightarrow{e_{j+3}} \ldots \xrightarrow{e_{j+k-2}} P_{j+k-1} \xrightarrow{e_{j+k}} P_{j+k+1}$ is a valid transition sequence.*
Then the following holds:

$$P \xrightarrow{e_{j+k}} P_{j+k+1}|_{E'} \text{ with } P \in \{P_j|_{E'}, \ldots, P_{j+k-1}|_{E'}\}$$

Note, that $P_j|_{E'} = \ldots = P_{j+k-1}|_{E'}$ does not necessarily hold.
Proof: To prove this we apply the two clauses of lemma 1 backwards, starting with the last step of the transition sequence:

1. For $P \equiv P_{j+k-1}|_{E'}$ this is obvious due to clause 1 of lemma 1.
2. For the projections of the remaining processes $P \equiv P_{j+k-3}|_{E'}, \ldots, P \equiv P_j|_{E'}$ we can repeatedly apply clause 2 of lemma 1 to the respective previous case.

Next, we bridge the gap between transition sequences that we can observe for CSP processes and paths that are present in the associated control flow graph.

Lemma 3 (CSP transition sequences and control flow graph paths). *Let C be a class specification, CFG the control flow graph of C, K^{CSP} the labelled Kripke structure associated to the CSP part of C, and $P \xrightarrow{e} Q \xrightarrow{f} R$ a transition sequence of K^{CSP}. Then the two nodes* `enable_`e *and* `enable_`f *of CFG are related in either one of the following ways:*

1. *There exists a path in CFG which leads from* `enable_`e *to* `enable_`f*:*

$$path_{CFG}(\texttt{enable\_}e, \texttt{enable\_}f) \neq \varnothing$$

2. *There exists a node interleave^i in CFG which has* `enable_`e *and* `enable_`f *as successors in different branches:*

$$\begin{aligned}&\exists\, n = interleave^i \in N_{CFG}:\\ &\exists\, \pi_e \in path_{CFG}(n, \texttt{enable\_}e) \wedge\\ &\exists\, \pi_f \in path_{CFG}(n, \texttt{enable\_}f) \wedge\\ &\pi_e \cap \pi_f = \{n\}\end{aligned}$$

Proof: We show both cases by considering the structure of P, resp. Q. Since we know that P can perform event e and Q can perform event f, we again only have to consider a limited set of CSP constructs for the structure of P and Q. For the structure of P we can distinguish the following cases:

1. $P \equiv e \longrightarrow Q$.
   Structure of Q:
   (a) $Q \equiv f \longrightarrow R$. ✓ (Path exists)
   (b) $Q \equiv Q_1 \,\Box\, Q_2$ with $Q_i \xrightarrow{f} R$ for $i \in \{1, 2\}$. ✓ (Path exists)
   (c) $Q \equiv Q_1 \sqcap Q_2$: analog. ✓ (Path exists)

(d) $Q \equiv Q_1 \mid\mid\mid Q_2$ with $Q_i \xrightarrow{f} Q_i'$ for $i \in \{1,2\}$ and $R \equiv \begin{cases} Q_1' \mid\mid\mid Q_2 \text{ if } i = 1 \\ Q_1 \mid\mid\mid Q_2' \text{ else} \end{cases}$
✓ (Path exists)

(e) $Q \equiv X$ with X as in one of the previous cases for Q: analog. ✓ (Path exists)

2. $P \equiv P_1 \,\Box\, P_2$ with $P_i \xrightarrow{e} Q$ for $i \in \{1,2\}$.
Structure of Q: as in the previous case for P. ✓ (Path exists)
3. $P \equiv P_1 \sqcap P_2$ with $P_i \xrightarrow{e} Q$ for $i \in \{1,2\}$.
Structure of Q: as in the previous cases for P. ✓ (Path exists)
4. $P \equiv P_1 \mid\mid\mid P_2$ with $P_i \xrightarrow{e} P_i'$ for $i \in \{1,2\}$ and $Q \equiv \begin{cases} P_1' \mid\mid\mid P_2 \text{ if } i = 1 \\ P_1 \mid\mid\mid P_2' \text{ else} \end{cases}$
Structure of Q:
(a) Either as in the previous cases for P. ✓ (Path exists)
(b) Or the other branch of the interleaving operator comes into play such that we have e and f in different branches of the interleaving operator of P. ✓ (Different Branches)
5. $P \equiv X$ with X as in one of the previous cases.
Structure of Q: see the respective previous case. ✓ (Either path exists or different branches.)

Our last lemma states that the set of irrelevant events appearing inside a projection block does not have any influence on the relevant variables (resp. atomic propositions) associated to the states inside the block.

Lemma 4 (No influence of irrelevant events on relevant variables). *Let C be a class specification with an associated labelled Kripke structure K, let*

$$(s_j, P_j) \xrightarrow{e_{j+1}} (s_{j+2}, P_{j+2}) \xrightarrow{e_{j+3}} \ldots \xrightarrow{e_{j+k-2}} (s_{j+k-1}, P_{j+k-1}) \xrightarrow{e_{j+k}} (s_{j+k+1}, P_{j+k+1})$$

be a transition sequence that is part of a path of K. Let furthermore be E' be the set of relevant events computed by the slicing algorithm with respect to some formula φ (with an associated set of variables V_φ), and $e_{j+1}, \ldots, e_{j+k-2} \notin E'$, and $e_{j+k} \in E'$. Then the following holds:

$$s_j|_{\overline{V}} = \ldots = s_{j+k-1}|_{\overline{V}} \quad \textit{with } \overline{V} = V_\varphi \cup \bigcup_{e \in \{e_i \in E' \mid i \geq j\}} \mathit{ref}(e)$$

Proof: Suppose, the equality does not hold. Then there is some irrelevant event e_l $(j+1 \leq l \leq j+k-2)$ that modifies some variable $v \in \overline{V}$. One of the following cases apply:

- $v \in V_\varphi$: According to the definition of the slice this leads to $e_l \in E'$, which is a contradiction.
- $v \notin V_\varphi$: According to the definition of $\overline{V}$, there is a subsequent relevant event $e_i \in E'$ $(i \geq j+k)$ that refers to v. According to lemma 3 the control flow graph nodes associated to e_l and e_i are related in either one of the following ways:

1. There is a path in the control flow graph that connects both nodes directly.
2. Both nodes are located in separate branches of an interleaving operator node.

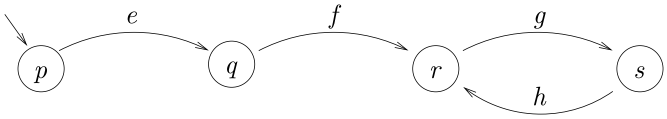
Both cases imply the existence of a data dependence between e_l and e_i. Therefore, according to the construction of the slice, $e_i \in E'$ implies that also $e_l \in E'$, which is a contradiction.

Now we come to our main theorem that states the existence of a projection relationship between the Kripke structures associated to the original and to the sliced specification.

Theorem 1. *Let C^{full} be a class specification and C^{red} the class obtained when slicing C^{full} wrt. a formula φ, associated with sets of events E_φ and variables V_φ. Let E' and AP' be the set of events and atomic propositions, respectively, which the slicing algorithm delivers as those of interest (in particular $E_\varphi \subseteq E'$ and $V_\varphi \subseteq V'$). Let furthermore K^{full} (resp. K^{red}) be the corresponding labelled Kripke structure. Then the following holds:*

$$K^{red} \in Pr_{AP_\varphi, E'}(K^{full})$$

Note that we overapproximate the set of events, while we underapproximate the set of atomic propositions. The reason for the latter can be seen in the following example Kripke structure:



Suppose, the slicing criterion does not refer to any variable, but only to events f and h. Further suppose, there is some variable $v \in ref(f)$ with $v \in mod(e)$ and $v \in mod(g)$, but $mod(g) \cap ref(h) = \varnothing$. Then the slicing algorithm delivers event e and variable v as part of the slice, but not event g, since its modification of variable v is not referenced by any subsequent relevant event. Therefore, the states inside projection blocks do not need to agree on all variables that the slicing algorithm delivers as those of interest, but rather only on the variables from V_φ.

For the events on the other hand we can not choose the same approach: If we only considered events from E_φ we might neglect events that modify variables from V_φ such that the states inside projection blocks would not agree on the modified variables. Therefore we have to consider all events that the slicing algorithm delivers as part of the slice.

Proof:

According to the definition of the projection relationship we need to consider two cases: (1) We have to show that for any E'-fair path of K^{full} we can construct an E'-fair path of K^{red} and (2) vice versa. For both directions we define a set of variables $\overline{V}_i$ that contains all variables associated to the slicing criterion and for each position i of the respective path all variables that are referenced by relevant events $e_i \in E'$ at position i or beyond:

$$\overline{V}_i = V_\varphi \cup \bigcup_{e \in \{e_j \in E' \mid j \geq i\}} ref(e)$$

1. Let $\pi = s_0 e_1 s_2 e_3 \ldots$ be an E'-fair path of K^{full} with $s_i = (s_i^{OZ}, P_i)$. We construct a sequence $\rho' = t_0 f_1 t_2 f_3 \ldots$ with $t_i = (t_i^{OZ}, Q_i)$

   $$\begin{aligned} &t_i^{OZ} : s_i^{OZ}|_{\overline{V}_i} \\ &Q_i : P_i|_{E'} \\ &f_i : \begin{cases} e_i & \text{if } e_i \in E' \\ nop & \text{else} \end{cases} \end{aligned}$$

   Out of ρ' we construct a sequence ρ by eliminating all subsequences of the form $nop\ t_i$. We have to show that ρ is an E'-fair path of K^{red}.

   (a) Fairness: Since π is fair, it contains infinitely many events from E' and thus also ρ does so.

   (b) Path: This is shown by induction over the corresponding paths.
   Induction base: According to the construction of ρ we have $t_0^{OZ}|_{\overline{V}_0} = s_0^{OZ}|_{\overline{V}_0}$ and $Q_0 = P_0|_{E'}$.
   Since the *Init* schema of C^{red} contains the same or fewer predicates than the *Init* schema of C^{full} there are fewer restrictions on values of variables, such that

   $$Init^{full}(s_0^{OZ}) \Rightarrow Init^{red}(s_0|_{\overline{V}_0})$$

   According to the construction of the slice $P_0|_{E'}$ represents the *main* process of the CSP part of C^{red}, thus it is obvious that Q_0 is the CSP part of the initial state for any path of K^{red}.
   Induction step: Suppose ρ is a path of K^{red} up to $t_i = (t_i^{OZ}, Q_i)$. We now have to show that the next transition $t_i \xrightarrow{f_{i+1}} t_{i+2}$ exists in K^{red}. This transition is either a synchronous transition or a CSP-asynchronous transition.
   Synchronous transition: We have to show that (i) transition $t_i^{OZ} \xrightarrow{f_{i+1}} t_{i+2}^{OZ}$ exists in $K^{OZ,red}$ and (ii) transition $Q_i \xrightarrow{f_{i+1}} Q_{i+2}$ exists in $K^{CSP,red}$.

   i. Object-Z part: From the construction of ρ we know that in $K^{OZ,full}$ a transition sequence

   $$s_j^{OZ} \xrightarrow{e_{j+1}} s_{j+2}^{OZ} \xrightarrow{e_{j+3}} \quad \ldots \quad \xrightarrow{e_{j+k-2}} s_{j+k-1}^{OZ} \xrightarrow{e_{j+k}} s_{j+k+1}^{OZ}$$

   exists with $\{e_{j+1}, e_{j+3}, \ldots, e_{j+k-2}\} \cap E' = \varnothing$
   and $e_{j+k} = f_{i+1} \in E'$
   such that lemma 4 leads to $s_j^{OZ}|_{\overline{V}_j} = \ldots = s_{j+k-1}^{OZ}|_{\overline{V}_j} = t_i^{OZ}|_{\overline{V}_j}$.

To show that transition $t_i^{OZ} \xrightarrow{f_{i+1}} t_{i+2}^{OZ}$ exists with $t_{i+2}^{OZ}|_{\overline{V}_j} = s_{j+k+1}|_{\overline{V}_j}$, we first have to show that f_{i+1} is enabled in t_i^{OZ} and, second, that it leads to such a state t_{i+2}^{OZ}.

To show the first claim we suppose the contrary: Let f_{i+1} be not enabled in t_i^{OZ}. Then there must be some predicate in f_{i+1} that is not satisfied in t_i^{OZ}. Since we know that $f_{i+1} = e_{j+k}$ is enabled in s_{j+k-1}^{OZ} and s_{j+k-1}^{OZ} and t_i^{OZ} coincide on all variables from $\overline{V}_j$, the unsatisfied predicate must contain a variable not in $\overline{V}_j$. This, however, is a contradiction to the construction of $\overline{V}_j$.

To show the second claim we exploit the knowledge that e_{j+k} leads from s_{j+k-1}^{OZ} to s_{j+k+1}^{OZ}. We suppose that f_{i+1} leads from t_i^{OZ} to some $\hat{t}_{i+2}^{OZ}$ with $\hat{t}_{i+2}^{OZ}|_{\overline{V}_j} \neq s_{j+k+1}^{OZ}|_{\overline{V}_j}$. This means there is a set $D \subseteq \overline{V}_j$ of variables whose valuation in $\hat{t}_{i+2}^{OZ}$ differs from that in s_{j+k+1}^{OZ}. Since $e_{j+k} = f_{i+1}$, this must be due to some predicates in f_{i+1} that allow multiple valuations for the variables in D. Instead of $\hat{t}_{i+2}^{OZ}$ we can therefore safely choose a state t_{i+2}^{OZ} with $t_{i+2}^{OZ}|_{\overline{V}_j \setminus D} = \hat{t}_{i+2}^{OZ}|_{\overline{V}_j \setminus D}$ and $t_{i+2}^{OZ}|_D = s_{j+k+1}^{OZ}|_D$ and hence also $t_{i+2}^{OZ}|_{\overline{V}_j} = s_{j+k+1}^{OZ}|_{\overline{V}_j}$.

ii. CSP part: From the construction of ρ we know that in $K^{CSP,full}$ a transition sequence

$$P_j \xrightarrow{e_{j+1}} P_{j+2} \xrightarrow{e_{j+3}} \quad \dots \quad \xrightarrow{e_{j+k-2}} P_{j+k-1} \xrightarrow{e_{j+k}} P_{j+k+1}$$

exists with $\{e_{j+1}, e_{j+3}, \dots, e_{j+k-2}\} \cap E' = \varnothing$
and $e_{j+k} = f_{i+1} \in E'$.

From lemma 2 we therefore know that $Q_i \equiv P_j|_{E'} \xrightarrow{e_{j+k} \equiv f_{i+1}} P_{j+k+1}|_{E'} \equiv Q_{i+2}$ is possible.

CSP-asynchronous transitions: Here we only need to consider the CSP part of the transition, since the Object-Z part remains unchanged. Consequently, the same arguments as for the CSP part in the synchronous case can be applied.

2. Let $\rho = t_0 f_1 t_2 f_3 \dots$ be an E'-fair path of K^{red} with $t_i = (t_i^{OZ}, Q_i)$. We inductively construct a path

$$\pi = s_0\, e_0^1\, s_0^2\, e_0^3 \dots s_0^{n_0}\, e_1\, s_2\, e_2^1\, s_2^2\, e_2^3 \dots s_2^{n_2}\, e_3\, s_4\, e_4^1\, s_4^2 \dots$$

of K^{full} with $s_i = (s_i^{OZ}, P_i)$ and $s_i^j = (s_i^{OZ,j}, P_i^j)$ and $e_i = f_i \in E'$ and $e_i^j \notin E'$.

Induction base: s_0

– OZ part: We know that $Init^{red}(t_0^{OZ})$ holds. $Init^{full}$ has at least all predicates that $Init^{red}$ has, possibly plus some additional predicates that restrict some further variables which are not present in C^{red}. Therefore, we can choose an s_0^{OZ} that coincides in all variables from C^{red} with t_0^{OZ} and is modified in the remaining variables in order to satisfy the additional predicates.

- CSP part: Out of the main process $main = P$ of C^{full} the slicing algorithm computes a reduced main process $main = P|_{E'}$ of C^{red}. Since $P|_{E'}$ exists, we know that an associated P exists which can be constructed by applying the CSP process projection rules backwards.

Induction step: Assume we have constructed the sequence up to some state $s_i = (s_i^{OZ}, P_i)$ with $s_i^{OZ}|_{\overline{V}_i} = t_i^{OZ}|_{\overline{V}_i}$ and $P_i|_{E'} = Q_i$.
From $e_{i+1} = f_{i+1}$ we can derive that $\texttt{enable\_}f_{i+1} = \texttt{enable\_}e_{i+1}$ and therefore we know that e_{i+1} is enabled in s_i^{OZ} and its execution leads to some s_{i+2}^{OZ}. Furthermore we know that $\texttt{effect\_}e_i$ has all predicates that $\texttt{effect\_}f_i =$ has, and therefore we have $s_{i+2}^{OZ}|_{\overline{V}_i} = t_{i+2}^{OZ}|_{\overline{V}_i}$.
Nevertheless, e_{i+1} might not yet be enabled in P_i, but some intermediate $e_i^j \notin E'$ might be necessary to reach a P_i^j such that e_{i+1} is enabled in P_i^j and leads to P_{i+2} with $P_{i+2}|_{E'} = Q_{i+2}$. We now have to show that these intermediate e_i^j are possible, that they do not change s_i^{OZ} on $\overline{V}_i$ s.t. e_{i+1} is enabled in any $s_i^{OZ,j}$ and that they lead to some P_i^j with $P_i^j \xrightarrow{e_{i+1}} P_{i+2}$ and $P_{i+2}|_{E'} = Q_{i+2}$.
We show this inductively by considering the structure of P_i from which the slicing algorithm computed Q_i.

(a) $P_i \equiv e \rightarrow P$:
   - $e \in E'$: $Q_i \equiv e \rightarrow P|_{E'}$ (no intermediate steps are necessary)
   - $e \notin E'$: $Q_i \equiv P|_{E'}$
     In this case, intermediate steps are necessary to get from (s_i^{OZ}, P_i) to (s_{i+2}^{OZ}, P_{i+2}). We now have to show that
      i. these steps are possible, i.e. enabled in (s_i^{OZ}, P_i). Suppose, one of the intermediate steps e_i^j is not enabled in s_i^{OZ}. This would be due to an unsatisfied predicate in its `enable` schema. According to lemma 3 (first case) we would therefore have either a control dependence between e_i^j and e_{i+1} that leads us (according to the construction of the slice) to $e_i^j \in E'$ which is a contradiction. The other possibility according to lemma 3 (second case) is that e_i^j and e_{i+1} are located in different branches of the same interleaving node. In this case we do not need to make a transition in this blocked branch of the interleaving operator but can safely proceed with a transition on the other branch.
      ii. these steps do not change variables in $\overline{V}_i$. This is a direct consequence of lemma 4.
      iii. these steps lead to some $(s_i^{OZ,j}, P_i^j)$ with $P_i^j \xrightarrow{e_{i+1}} P_{i+2}$ and $s_i^{OZ,j} \xrightarrow{e_{i+1}} s_{i+2}^{OZ}$.
         - $s_i^{OZ,j}$: Since e_{i+1} is already enabled in s_i^{OZ} and none of the e_i^j changes any $v \in V'$, e_{i+1} is still enabled in $s_i^{OZ,j}$.
         - P_i^j: Since e_i^j are the steps that are removed from P_i in order to get $Q_i \equiv P_i|_{E'}$ and e_{i+1} is enabled in Q_i, the execution of e_i^j will lead to some P_i^j such that e_{i+1} is enabled as well.

(b) $P_i \equiv P_{i,1} \,\Box\, P_{i,2}$:

- Either $\exists j \in \{1,2\}\colon P_{i,j} \xrightarrow{e} P'_{i,j}$ with $e \in E'$. Then $e_i + 1 \equiv e$ and $Q_{i+2} \equiv P'_{i,j}$. ✓
- Otherwise $\exists j \in \{1,2\}\colon P_{i,j} \xrightarrow{e} P'_{i,j}$ with $e \notin E'$. Then e is one of the intermediate events and we have start another case analysis for the intermediate process $P_{i,j}$.

(c) $P_i \equiv P_{i,1} \sqcap P_{i,2}$: Analog to the previous case. ✓

(d) $P_i \equiv P_{i,1} \mid\mid\mid P_{i,2}$:

- Either $\exists j \in \{1,2\}\colon P_{i,j} \xrightarrow{e} P'_{i,j}$ with $e \in E'$. Then $e_i + 1 \equiv e$ and $Q_{i+2} \equiv \begin{cases} P'_{i,1} \mid\mid\mid P_{i,2} \text{ if } j = 1 \\ P_{i,1} \mid\mid\mid P'_{i,2} \text{ else} \end{cases}$ ✓
- Otherwise $\exists j \in \{1,2\}\colon P_{i,j} \xrightarrow{e} P'_{i,j}$ with $e \notin E'$. Then e is one of the intermediate events and we have to start another case analysis for the intermediate process $\begin{cases} P'_{i,1} \mid\mid\mid P_{i,2} \text{ if } j = 1 \\ P_{i,1} \mid\mid\mid P'_{i,2} \text{ else} \end{cases}$

5 Conclusion

In this paper we have proposed a slicing algorithm for an integrated formal method covering state-based as well as behaviour-oriented aspects. We have shown correctness of the algorithm with respect to a projection relationship between the paths of the full and the reduced specification (starting from some set of relevant variables and events). Thus the reduction preserves formulae (speaking about these relevant variables and events) of any linear-time temporal logic which is invariant under projection. Slicing can thus help to reduce the specification before verification. Since the program dependence graph is in size linear in the syntactic representation of the specification (and thus usually much smaller than the state space), slicing can also be carried out in cases when model checking is too complex. Furthermore, the program dependence graph only has to be constructed once for every specification, only backward reachability has to be computed for every formula. Our slicing technique acts as a preparatory step in the verification of CSP-OZ specifications; the following model checking step is carried out by a constraint-based abstraction refinement model checker as recently proposed by Hoenicke and Maier [10].

In the future we plan to extend this technique to a third modelling dimension, namely timing requirements as covered by the formalism CSP-OZ-DC [11] (an extension of CSP-OZ with Duration Calculus). Furthermore, in order to complete the process of slicing and model checking, a non-trivial problem still remains to be solved: How can we relate a counterexample obtained for a reduced specification to a corresponding one for the original specification?

Related work. There are two strands of research which touch upon our work. The first is on slicing of formal specifications, which has mainly been done for Z specifications [16, 4, 27]. These works, however, do not consider verification, i.e. slicing is not carried out with respect to temporal logic properties of the specification. The second area of related work concerns slicing used for reducing programs before verification, as for instance done in [8] for Java (preserving LTL$_{-X}$ properties) and in [24] for SAL programs (preserving CTL$^*_{-X}$ properties). Slicing for integrated specification techniques has so far not been considered.

References

1. I. Brückner and H. Wehrheim. Slicing an Integrated Formal Method for Verification. In *ICFEM 2005: Seventh International Conference on Formal Engineering Methods*, volume 3785 of *LNCS*, pages 360–374. Springer, 2005.
2. I. Brückner and H. Wehrheim. Slicing Object-Z Specifications for Verification. In *ZB 2005: Formal Specification and Development in Z and B*, volume 3455 of LNCS, pages 414–433. Springer-Verlag, 2005.
3. I. Brückner and H. Wehrheim. Slicing Object-Z Specifications for Verification. Technical Report 3, SFB/TR 14 AVACS, http://www.avacs.org/, 2005.
4. D. Chang and D. Richardson. Static and Dynamic Specification Slicing. In *ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 138–153. ACM, 1994.
5. E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.
6. C. Fischer. CSP-OZ: A Combination of Object-Z and CSP. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-Based Distributed Systems (FMOODS '97)*, volume 2, pages 423–438. Chapman & Hall, 1997.
7. W. Grieskamp, M. Heisel, and H. Dörr. Specifying Embedded Systems with Statecharts and Z: An Agenda for Cyclic Software Components. In Egidio Astesiano, editor, *FASE '98*, volume 1382 of *LNCS*, pages 88–106. Springer, 1998.
8. J. Hatcliff, M. Dwyer, and H. Zheng. Slicing Software for Model Construction. *Higher-order and Symbolic Computation*, 13(4):315–353, 2000.
9. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.
10. J. Hoenicke and P. Maier. Model-checking specifications integrating processes, data and time. In *FM 2005*, volume 3582 of *LNCS*, pages 465–480. Springer, 2005.
11. J. Hoenicke and E.-R. Olderog. CSP-OZ-DC: A Combination of Specification Techniques for Processes, Data and Time. *Nordic Journal of Computing*, 9(4):301–334, 2002.
12. ISO/IEC. *Enhancements to LOTOS (E-LOTOS) – International Standard 15437:2001.* ISO/IEC – Information technology, 2001.
13. R.P. Kurshan. *Computer-Aided Verification of Coordinating Processes.* Princeton University Press, 1994.
14. B. Mahony and J.S. Dong. Timed communicating Object-Z. *IEE Transactions on Software Engineering*, 26(2):150–177, 2000.
15. L. Millett and T. Teitelbaum. Issues in Slicing PROMELA and its Applications to Model Checking, Protocol Understanding, and Simulation. *Software Tools for Technology Transfer*, 2(4):343–349, 2000.
16. T. Oda and K. Araki. Specification Slicing in Formal Methods of Software Development. In *Proceedings of the Seventeenth Annual International Computer Software & Applications Conference*, pages 313–319. IEEE Computer Society Press, 1993.
17. A.W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall Series in Computer Science, 1998.
18. G. Smith. A Semantic Integration of Object-Z and CSP for the Specification of Concurrent Systems. In J. Fitzgerald, C. B. Jones, and P. Lucas, editors, *FME'97: Industrial Application and Strengthened Foundations of Formal Methods*, volume

1313 of *Lecture Notes in Computer Science*, pages 62–81. Springer-Verlag, September 1997.

19. G. Smith. *The Object-Z Specification Language*. Kluwer Academic Publisher, 2000.
20. G. Smith and J. Derrick. Specification, Refinement and Verification of Concurrent Systems – An Integration of Object-Z and CSP. *Formal Methods in System Design*, 18(3):249 – 284, 2001.
21. K. Taguchi and K. Araki. Specifying Concurrent Systems by Z + CCS. In *International Symposium on Future Software Technology (ISFST)*, pages 101–108, 1997.
22. F. Tip. A Survey of Program Slicing Techniques. *Journal of Programming Languages*, 3(3):121–189, 1995.
23. H. Treharne and S.A. Schneider. Communicating B Machines. In *ZB2002: International Conference of Z and B Users*, volume 2272 of *LNCS*. Springer, 2002.
24. N. Shankar V. Ganesh, H. Saidi. Slicing SAL. Technical report, SRI International, http://theory.stanford.edu/, 1999.
25. M. Weiser. Programmers use slices when debugging. *Communications of the ACM*, 25(7):446–452, 1982.
26. J.C.P. Woodcock and A.L.C. Cavalcanti. The Semantics of Circus. In *ZB 2002: Formal Specification and Development in Z and B*, volume 2272 of LNCS, pages 184–203. Springer-Verlag, 2002.
27. Fangjun Wu and Tong Yi. Slicing Z Specifications. *SIGPLAN Not.*, 39(8):39–48, 2004.