

Compositional Hoare-style Reasoning about Hybrid CSP in the Duration Calculus

Dimitar Guelev*, Shuling Wang and Naijun Zhan†

November 17, 2018

Abstract

Deductive methods for the verification of hybrid systems vary on the format of statements in correctness proofs. Building on the example of Hoare triple-based reasoning, we have investigated several such methods for systems described in Hybrid CSP, each based on a different assertion language, notation for time, and notation for proofs, and each having its pros and cons with respect to expressive power, compositionality and practical convenience. In this paper we propose a new approach based on weakly monotonic time as the semantics for interleaving, the Duration Calculus (DC) with infinite intervals and general fixpoints as the logic language, and a new meaning for Hoare-like triples which unifies assertions and temporal conditions. We include a proof system for reasoning about the properties of systems written in the new form of triples that is complete relative to validity in DC.

1 Introduction

Hybrid systems exhibit combinations of discrete and continuous evolution, the typical example being a continuous plant with discrete control. A number of abstract models and requirement specification languages have been proposed for the verification of hybrid systems, the commonest model being *hybrid automata* [3, 24, 20]. Hybrid CSP (HCSP) [19, 39] is a process algebra which extends CSP by constructs for continuous evolution described in terms of ordinary differential equations, with domain boundary- and communication-triggered interruptions. The mechanism of synchronization is message passing. Because of its compositionality, HCSP can be used to handle complex and open systems. Here follows an example of a simple generic HCSP description of a continuously evolving plant with discrete control:

$$\begin{aligned} &(\mathbf{while} \top \mathbf{do} \langle F(\dot{x}, x, u) = 0 \rangle \sqsupseteq \mathbf{sensor}!x \rightarrow \mathbf{actuator}?u) \parallel \\ &(\mathbf{while} \top \mathbf{do} (\mathbf{wait} \ d; \mathbf{sensor}?s; \mathbf{actuator}!C(s))) \end{aligned}$$

The plant evolves according to some continuous law F that depends on a control parameter u . The controller samples the state of the plant and updates the control parameter once every d time units.

In this paper we propose a Hoare-style proof system for reasoning about hybrid systems which are modelled in HCSP. The features of HCSP which are handled by the logic include communication, timing constraints, interrupts and continuous evolution governed by differential equations. Our proof system is based on the Duration Calculus (DC, [5, 4]), which is a first-order real-time temporal logic and therefore enables the verification of HCSP systems for temporal properties. DC is an interval-based temporal logic. The form of the satisfaction relation in DC is $I, \sigma \models \varphi$, where φ is a temporal formula, I is an interpretation of the respective vocabulary over time, and σ is a *reference interval* of real time, unlike *point-based* TLs, where a reference time point is used. The advantages of intervals stem from the possibility to accommodate a complete execution of a process and have reference to termination time points of processes as well as the

*Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, e-mail: gelevdp@math.bas.bg

†State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, e-mail: {wangsl,znj}@ios.ac.cn.

starting points. Pioneering work on interval-based reasoning includes Allen’s interval algebra and Halpern and Shoham’s logic [2, 15]. ITLs have been studied in depth with respect to the various models of time by a number of authors, cf. e.g. [9]. Since an interval can be described as the pair of its endpoints, interval-based logics are also viewed as *two-dimensional* modal logics [35, 36]. Interval Temporal Logic (ITL) was first proposed and developed by Moszkowski [26, 27, 7] for discrete time, as a reasoning tool for digital circuits. DC can be viewed as a theory in *real time* ITL. We use the infinite interval variant of DC which was proposed in [6], which allows intervals whose right end is ∞ for modelling non-terminating behaviour. We include an operator for Kleene star in order to model iterative behaviour, to facilitate the handling of liveness properties. Axioms and proof rules about infinite time and Kleene star in DC can be found in [13, 11].

Hoare-style proof systems are about proving *triples* of the form $\{P\} \text{code} \{Q\}$, which stand for the *partial correctness* property $P(x) \wedge \text{code}(x, x') \rightarrow Q(x)$. The meaning of triples generalizes to the setting of reactive systems in various ways, the common feature of them all being that P and/or Q are *temporal* properties. In our system **code** is a HCSP term, P and Q are written in DC. The intended meaning is

Given an infinite run which satisfies P at some initial subinterval, **code**
causes it to satisfy also Q at the initial subinterval representing the
execution of **code**. (1)

The initial subinterval which is supposed to satisfy P , can as well be a degenerate (0-length) one. Then P boils down to an assertion on the initial state. This interval can also be to the entire infinite run in question. In this case P can describe conditions provided by the environment throughout runs. Q is supposed to hold at an interval which exactly matches that of the execution of **code**. In case **code** does not terminate, this would be the whole infinite run too. Using our DC semantics $\llbracket \cdot \rrbracket$ for HCSP terms, the validity of $\{P\} \text{code} \{Q\}$ is defined as the validity of

$$P \frown \top \Rightarrow \neg(\llbracket \text{code} \rrbracket \wedge \neg(Q \frown \top))$$

at infinite intervals, which is equivalent to (1).

We exploit the form of triples to obtain a *compositional* proof system, with each rule corresponding to a basic HCSP construct. This forces proofs to follow the structure of the given HCSP term. Triples in this form facilitate assume-guarantee reasoning too. For instance,

$$\frac{\{A\} \text{code}_1 \{B\} \quad \{B\} \text{code}_2 \{C\}}{\{A\} \text{code}_1 \parallel \text{code}_2 \{((B \frown \top) \wedge C) \vee (B \wedge (C \frown \top))\}}$$

where \parallel denotes parallel composition, is an admissible rule in our proof system, despite not being among the basic rules. A detailed study of assume-guarantee reasoning about discrete-time reactive systems in terms of triples of a similar form with point-based temporal logic conditions can be found in [1].

The main result about our proof system in the paper is its completeness relative to validity in DC.

Structure of the paper After brief preliminaries on HCSP, we propose a weakly-monotonic time semantics for it in terms of DC formulas and prove its equivalence to an appropriate operational semantics. Next we give our proof system and use the DC-based semantics to demonstrate its relative completeness. Finally we summarize a generalization of the approach where arbitrary fixpoints can be used instead of HCSP’s tail recursion and the use of \parallel and the respective rather involved proof rule can be eliminated. That turns out to require both the general fixpoint operator of DC [31] and the right-neighbourhood modality (cf. [40]) to handle the meaning of P s in the presence of properly recursive calls. We conclude by discussing related work and make some remarks.

2 Preliminaries

2.1 Syntax and informal semantics of Hybrid CSP

Process terms have the syntax

$P, Q ::=$	skip	do nothing;
	$x_1, \dots, x_n := e_1, \dots, e_n$	simultaneous assignment;
	wait d await b	fixed time delay; wait until b becomes true;
	$ch?x$ $ch!e$ IO	input and output; communication-guarded choice;
	$\langle F(\dot{x}, x) = 0 \wedge b \rangle$	x evolves according to F as long as b holds;
	$\langle F(\dot{x}, x) = 0 \wedge b \rangle \geq IO$	evolve by F until $\neg b$ or IO becomes ready;
		terminate, if $\neg b$ is reached first;
		otherwise execute IO ;
	$P; Q$ $P \parallel Q$	sequential composition; parallel composition
	if b then P else Q $P \sqcup Q$	conditional; internal non-deterministic choice;
	$\mu X.G$	recursion.

In the above BNF, IO has the following form:

$$ch_1?x_1 \rightarrow P_1 \parallel \dots \parallel ch_k?x_k \rightarrow P_k \parallel ch_{k+1}!e_{k+1} \rightarrow P_{k+1} \parallel \dots \parallel ch_n!e_n \rightarrow P_n \quad (2)$$

for some arbitrary $k, n, x_1, \dots, x_k, e_{k+1}, \dots, e_n$ and some distinct ch_1, \dots, ch_n . IO engages in one of the indicated communications as soon as a partner process becomes ready, and then proceeds as the respective P_i . In $\mu X.G$, G has the syntax

$$G ::= H \mid \vec{x} := \vec{e}; P \mid \langle F(\dot{x}, x) = 0 \wedge b \rangle; P \mid \text{if } b \text{ then } G \text{ else } G \\ \mid G \sqcup G \mid G; P \mid \mu Y.G \mid H \parallel H$$

where H stands for arbitrary X -free terms, $Y \neq X$ and P can be any process term. This restricts X of $\mu X.G$ to be *guarded* in G and rules out occurrences of X of $\mu X.G$ in the scope of \parallel in G . The communication primitives $ch?x$ and $ch!e$ are not mentioned in the syntax for G as they are treated as derived in this paper. They can be assigned the role of guards which $\vec{x} := \vec{e}$ has in (2.1). Obviously X is guarded in the P_1, \dots, P_n of IO as in (2) too. Below we focus on the commonest instance of μ

$$\text{while } b \text{ do } P \equiv \mu X. \text{if } b \text{ then } (P; X) \text{ else skip} \quad (3)$$

CSP's Kleene star $P^* \equiv \mu X.(\text{skip} \sqcup (P; X))$, which stands for some unspecified number of successive executions of P , is handled similarly. We explain how our setting ports to general fixpoints, and some technical benefits from that, other than the obvious gain in expressive power, in a dedicated section.

2.2 The Duration Calculus

We use DC with infinite intervals as in [6, 13] and Kleene star. The reader is referred to [4] for a comprehensive introduction. Here we summarize only the main notions for the sake of self-containedness. DC is a classical first-order predicate modal logic with one normal binary modality called *chop* and written \frown . The time domain is $\mathbb{R}^\infty = \mathbb{R} \cup \{\infty\}$. Satisfaction has the form $I, \sigma \models \varphi$ where I is an interpretation of the non-logical symbols, $\sigma \in \mathbb{I}(\mathbb{R}^\infty)$, $\mathbb{I}(\mathbb{R}^\infty) = \{[t_1, t_2]; t_1 \in \mathbb{R}, t_2 \in \mathbb{R}^\infty, t_1 \leq t_2\}$. *Flexible* non-logical symbols depend on the reference intervals for their meaning. *Chop* is defined by the clause

$$I, \sigma \models (\varphi \frown \psi) \quad \text{iff} \quad \begin{array}{l} \text{either there exists a } t \in \sigma \setminus \{\infty\} \text{ such that } I, [\min \sigma, t] \models \varphi \\ \text{and } I, [t, \max \sigma] \models \psi, \text{ or } \max \sigma = \infty \text{ and } I, \sigma \models \varphi. \end{array}$$

Along with the usual first-order non-logical symbols, DC features boolean valued *state variables*, which form boolean combinations called *state expressions*. The value $I_t(S)$ of a state expression S is supposed to change between 0 and 1 only finitely many times in every bounded interval of time. *Duration terms* $\int S$ take a state expression S as the operand and evaluate according to the clause

$$I_\sigma(\int S) = \int_{\min \sigma}^{\max \sigma} I(S)(t) dt.$$

ℓ is used for $\int(\mathbf{0} \Rightarrow \mathbf{0})$ and always evaluates to the length of the reference interval, and $\lceil S \rceil$ stands for $\ell \neq 0 \wedge \int S = \ell$ to denote that S holds almost everywhere and the interval is non-degenerate. $\lceil S \rceil^0$ denotes just $\int S = \ell$.

In Section 6 we use the converse neighbourhood modality \Diamond_I^c and the least fixpoint operator μ . The former appears in *Neighbourhood Logic* and the corresponding system of DC, and is defined by the clause

$$I, \sigma \models \Diamond_I^c \varphi \text{ iff } I, [\min \sigma, t] \models \varphi \text{ for some } t \in \mathbb{R} \cup \{\infty\}, t \geq \min \sigma.$$

Formulas $\mu X.\varphi$, where X is a dedicated type of variable, are well formed only if φ has no negative occurrences of X . To define the meaning of $\mu X.\varphi$, φ is regarded as the defining formula of a operator of type $\mathcal{P}(\mathbb{I}(\mathbb{R}^\infty)) \rightarrow \mathcal{P}(\mathbb{I}(\mathbb{R}^\infty))$, $A \mapsto \{\sigma \in \mathbb{I}(\mathbb{R}^\infty) : I_X^A, \sigma \models \varphi\}$, and $I, \sigma \models \mu X.\varphi$ iff σ appears in the least fixpoint of this operator, which happens to be monotonic by virtue of the syntactic condition on φ . Kleene star is defined in terms of μ by the clause

$$\models \varphi^* \Leftrightarrow \mu X.(\ell = 0 \vee \varphi \frown X).$$

3 Operational semantics of HCSP

Ownership of variables We write $\text{Var}(P)$ for the set of the program variables which occur in P . Expressions of the form \dot{x} in continuous evolution process terms are, syntactically, just program variables, and are restricted not to appear in arithmetical expressions e outside the $F(\dot{x}, x)$ of continuous evolution terms, or on the left hand side of $:=$. As it becomes clear below, the dependency between x and \dot{x} as functions of time is spelled out as part of the semantics of continuous evolution. We write $\text{Var}_{:=}(P)$ for all the variables in $\text{Var}(P)$ which occur on the left hand side of $:=$, in $ch?$ statements, and the x -es or \dot{x} -es in any of the forms of continuous evolution within P . Parallel composition $P \parallel Q$ is well-formed only if $\text{Var}_{:=}(P) \cap \text{Var}_{:=}(Q) = \emptyset$.

Modelling input and output We treat $ch?x$ and $ch!e$ as derived constructs as they can be defined in terms of dedicated shared variables $ch?$, $ch!$ and ch after [28]:¹

$$\begin{aligned} ch!e &\Leftrightarrow ch := e; ch! := \top; \mathbf{await} \ ch?; \mathbf{await} \ \neg ch?; ch! := \perp \\ ch?x &\Leftrightarrow ch? := \top; \mathbf{await} \ ch!; x := ch; ch? := \perp; \mathbf{await} \ \neg ch! \end{aligned}$$

We assume that $ch!, ch \in \text{Var}_{:=}(ch!e)$ and $ch? \in \text{Var}_{:=}(ch?x)$. Communication-guarded external choice IO can be defined similarly. We omit the definition as it is lengthy and otherwise un insightful. The other derived constructs are defined as follows:

$$\begin{aligned} \langle F(\dot{x}, x) = 0 \wedge b \rangle \triangleright_d Q &\Leftrightarrow t := 0; \langle F(\dot{x}, x) = 0 \wedge t = 1 \wedge b \wedge t \leq d \rangle; \\ &\quad \mathbf{if} \ \neg(\neg b) \ \mathbf{then} \ Q \ \mathbf{else} \ \mathbf{skip} \\ \mathbf{wait} \ d &\Leftrightarrow \langle 0 = 0 \wedge \top \rangle \triangleright_d \mathbf{skip} \\ \langle F(\dot{x}, x) = 0 \wedge b \rangle \triangleright IO &\Leftrightarrow \langle F(\dot{x}, x) = 0 \wedge b \wedge \bigwedge_{i \in I} \neg ch_i^* \rangle; \mathbf{if} \ \bigwedge_{i \in I} \neg ch_i^* \ \mathbf{then} \ \mathbf{skip} \ \mathbf{else} \ IO \end{aligned}$$

Here ch_i^* stands for $ch_i?$, resp. $ch_i!$, depending on whether the respective action in IO is input or output. To account of the impossibility to mechanically (and computationally) tell apart $x < c$ from $x \leq c$ about time-dependent quantities x , in $\neg(\neg b)$ we use \overline{a} for a condition that defines the topological closure of $\{\mathbf{x} : a(\mathbf{x})\}$. It is assumed that \overline{b} admits a syntactical definition. E.g., $\overline{x < c}$ is $x \leq c$ for x being a continuous function of time.

¹Hoare style proof rules for a system with $ch?x$ and $ch!e$ appearing as primitive constructs were proposed by Zhou Chaochen *et al.* in [12]. That work features a different type of triples and follows the convention that process variables are not observable across threads thus ruling out a shared-variable emulation of communication.

Reduction of HCSP process terms Next we define a *reduction relation* $P \xrightarrow{A,V} Q$ where V is a set of process variables and

$$A : \sigma \rightarrow (V' \cup \{r, n\} \rightarrow \mathbb{R}^\infty \cup \{0, 1\}), \quad (4)$$

where V' is a set of process variables, r and n are boolean variables outside V' and $\sigma \in \mathbb{I}$. In $\mathbb{R}^\infty \cup \{0, 1\}$ we emphasize the additional use of $0, 1 \in \mathbb{R}$ as truth values. We consider $P \xrightarrow{A,V} Q$ only for V such that $\text{Var}_{:=}(P) \subseteq V \subseteq V'$. For HCSP terms P in the scope of a Q which on its turn is an operand of a \parallel , with no other \parallel s between this one and P , the semantics of P must specify the behaviour of all the variables from $\text{Var}_{:=}(Q)$, which are *controlled* by the *enveloping thread* Q of P . $V' \setminus V$ is meant to include of the variables which are not controlled by the enveloping thread of P but still may be accessed in it. In the sequel we write $\text{dom}A$ for σ and $\text{Var}(A)$ for V' from (4).

If $V \subseteq \text{Var}(A)$, then $A|_V$ stands for the restriction of A to the variables from V . I.e., given A as in (4),

$$A|_V : \sigma \rightarrow (V \cup \{r, n\} \rightarrow \mathbb{R}^\infty \cup \{0, 1\}).$$

Given an arithmetic or boolean expression e such that $V(e) \subseteq V(A)$, we write $A_t(e)$ for the value of e under A at time $t \in \text{dom}A$. Given A and B such that $\max \text{dom}A = \min \text{dom}B$, $\text{Var}(A) = \text{Var}(B)$ and $A_{\max \text{dom}A}(x) = B_{\min \text{dom}B}(x)$ for all $x \in \text{Var}(A) \cup \{r, n\}$, $A; B$ is determined by the conditions $\text{dom}A; B = \text{dom}A \cup \text{dom}B$, $(A; B)_t(x) = A_t(x)$ for $t \in \text{dom}A$ and $(A; B)_t(x) = B_t(x)$ for $t \in \text{dom}B$ for all $x \in \text{Var}(A) \cup \{r, n\}$. A complete and possibly infinite behaviour of P can be defined as $A_1; A_2; \dots$ where $P_{i-1} \xrightarrow{A_i, V} P_i$, and $P_0 = P$.

The auxiliary variables r and n To handle the causal ordering of computation steps without having to account of the *negligibly* small time delays they contribute, we allow stretches of time in which continuous evolution is 'frozen', and which are meant to just keep apart time points with different successive variable values that have been obtained by computation. Intervals of negligible time are marked by the boolean variable r . P (or any of its descendant processes) claims exclusive control over the process variables during such intervals, thus achieving atomicity of assignment. Time used for computation steps by *any* process which runs in parallel with P or P itself is marked by n . Hence $A_t(r) \leq A_t(n)$, $t \in \text{dom}A$ always holds in the mappings (4). As it becomes clear below, each operand P_i of a $P_1 \parallel P_2$ has its own r , and no two such variables evaluate to 1 at the same time, which facilitates encoding the meaning of \parallel by conjunction. In processes with loops and no other recursive calls, the r s can be enumerated globally. More involved form of recursive calls require the r s to be quantified away.

This approach is known as the *true synchrony hypothesis*. It was introduced to the setting of DC in [29] and was developed in [10, 11] where properties φ of the overall behaviour of a process in terms of the relevant continuously evolving quantities are written $(\varphi/\neg N)$, the *projection of φ onto state $\neg N$* , which holds iff φ holds at the interval obtained by gluing the $\neg N$ -parts of the reference one. The approach is alternative to the use of *super-dense chop* [16].

3.1 The reduction rules

To abbreviate conditions on A in the rules which generate the valid instances of $P \xrightarrow{A,V} Q$ below, given an $X \subseteq \text{Var}(A)$ and a boolean or arithmetical expression e , we put:

$$\begin{aligned} \text{const}(X, A) &\quad \Rightarrow \bigwedge_{x \in X} (\forall t \in \text{dom}A) (A_t(x) = A_{\min \text{dom}A}(x)) \\ \text{const}^\circ(X, A) &\quad \Rightarrow \bigwedge_{x \in X} (\forall t \in \text{dom}A \setminus \{\max \text{dom}A\}) (A_t(x) = A_{\min \text{dom}A}(x)) \\ \text{const}^\circ(e, A, a) &\quad \Rightarrow (\forall t \in \text{dom}A \setminus \{\max \text{dom}A\}) (A_t(e) = a) \end{aligned}$$

In these abbreviations $^\circ$ indicates that no restriction is imposed at $\max \text{dom}A$.

Reduction of process terms of the primitive types The valid transitions which are specific to primitive process terms are given by the following rules:

$$\begin{array}{c}
\text{const}(V \setminus \{x_1, \dots, x_n\}, A) \\
\text{const}^\circ(\{x_1, \dots, x_n\}, A) \\
\text{const}^\circ(r \wedge n, A, 1) \\
A_{\max \text{dom} A}(x_i) = A_{\min \text{dom} A}(e_i), \ i = 1, \dots, n \\
\max \text{dom} A < \infty \\
\hline
\text{skip} \xrightarrow{A, V} \checkmark
\end{array}
\quad
\begin{array}{c}
\hline
x_1, \dots, x_n := e_1, \dots, e_n \xrightarrow{A, V} \checkmark
\end{array}$$

$$\begin{array}{c}
\text{const}^\circ(F(\dot{x}, x), A, 0) \\
\text{const}^\circ(A_t(\dot{x}) - \frac{d}{dt}A_t(x), A, 0) \\
\text{const}^\circ(\neg r \wedge \neg n \wedge b, A, 1) \\
\text{const}(V \setminus \{\dot{x}, x\}, A) \\
\hline
\langle F(\dot{x}, x) = 0 \wedge b \rangle \xrightarrow{A, V} \langle F(\dot{x}, x) = 0 \wedge b \rangle
\end{array}
\quad
\begin{array}{c}
A_{\min \text{dom} A}(b) = 0 \\
\max \text{dom} A = \min \text{dom} A \\
\hline
\langle F(\dot{x}, x) = 0 \wedge b \rangle \xrightarrow{A, V} \checkmark
\end{array}$$

The rule about simultaneous assignment states that assignment takes an interval of negligible time, with its enveloping thread claiming exclusive control over the process variables. All variables except the ones which are assigned are kept constant throughout.

The first rule about $\langle F(\dot{x}, x) = 0 \wedge b \rangle$ describes 'ordinary' (non-negligible) periods in which continuous evolution takes place within the boundary condition b . The second rule describes states in which b has become false and therefore evolution terminates immediately.

Reduction of compound terms

$$\frac{P \xrightarrow{A,V} P' \quad P' \neq \checkmark}{P; Q \xrightarrow{A,V} P'; Q} \quad \frac{P \xrightarrow{A,V} \checkmark \quad \max \text{dom} A < \infty}{P; Q \xrightarrow{A,V} Q} \quad \frac{P \xrightarrow{A,V} P' \quad \max \text{dom} A = \infty}{P; Q \xrightarrow{A,V} P'}$$

$$\frac{P \xrightarrow{A,V} R \quad A_{\min \text{dom} A}(b) = 1}{\text{if } b \text{ then } P \text{ else } Q \xrightarrow{A,V} R} \quad \frac{Q \xrightarrow{A,V} R \quad A_{\min \text{dom} A}(b) = 0}{\text{if } b \text{ then } P \text{ else } Q \xrightarrow{A,V} R}$$

$$\frac{P \xrightarrow{A,V} P'}{P \sqcup Q \xrightarrow{A,V} P'} \quad \frac{Q \xrightarrow{A,V} Q'}{P \sqcup Q \xrightarrow{A,V} Q'}$$

$$\frac{[\mu X. P/X] P \xrightarrow{A,V} Q}{\mu X. P \xrightarrow{A,V} Q}$$

$$\frac{\begin{array}{l} \text{Var}(A) \cup \text{Var}(B) \subseteq \text{Var}(C) \\ C|_{\text{Var}(A) \cup \{r,n\}} = A \\ C|_{\text{Var}(B) \cup \{r,n\}} = B \\ \text{const}^\circ(\neg r \wedge \neg n, C, 1) \\ V_1 \cap V_2 = \emptyset \\ P \xrightarrow{A,V_1} \checkmark \quad Q \xrightarrow{B,V_2} \checkmark \end{array}}{P \parallel Q \xrightarrow{C, V_1 \cup V_2} \checkmark} \quad \frac{\begin{array}{l} \text{Var}(A) \cup \text{Var}(B) \subseteq \text{Var}(C) \\ C|_{\text{Var}(A) \cup \{r,n\}} = A \\ C|_{\text{Var}(B) \cup \{r,n\}} = B \\ \text{const}^\circ(\neg r \wedge \neg n, C, 1) \\ V_1 \cap V_2 = \emptyset \\ P \xrightarrow{A,V_1} P' \quad Q \xrightarrow{B,V_2} Q' \\ P' \neq \checkmark, Q' \neq \checkmark \end{array}}{P \parallel Q \xrightarrow{C, V_1 \cup V_2} P' \parallel Q'}$$

$$\frac{\begin{array}{l} \text{Var}(A) \cup \text{Var}(B) \subseteq \text{Var}(C) \\ C|_{\text{Var}(A) \cup \{r,n\}} = A \\ C|_{\text{Var}(B) \cup \{r,n\}} = B \\ \text{const}^\circ(\neg r \wedge \neg n, C, 1) \\ V_1 \cap V_2 = \emptyset \\ P \xrightarrow{A,V_1} P' \quad Q \xrightarrow{B,V_2} \checkmark \\ P' \neq \checkmark \end{array}}{P \parallel Q \xrightarrow{C, V_1 \cup V_2} P'} \quad \frac{\begin{array}{l} \text{Var}(A) \cup \text{Var}(B) \subseteq \text{Var}(C) \\ C|_{\text{Var}(A) \cup \{r,n\}} = A \\ C|_{\text{Var}(B) \cup \{r,n\}} = B \\ \text{const}^\circ(\neg r \wedge \neg n, C, 1) \\ V_1 \cap V_2 = \emptyset \\ P \xrightarrow{A,V_1} \checkmark \quad Q \xrightarrow{B,V_2} Q' \\ Q' \neq \checkmark \end{array}}{P \parallel Q \xrightarrow{C, V_1 \cup V_2} Q'}$$

$$\frac{\begin{array}{l} V \subseteq V' \\ \text{const}(V' \setminus V, B) \\ V' \cup \text{Var}(A) \subseteq \text{Var}(B) \\ B|_{\text{Var}(A) \cup \{r,n\}} = A \\ \text{const}^\circ(r \wedge n, A, 1) \\ P \xrightarrow{A,V} P' \quad P' \neq \checkmark \end{array}}{P \parallel Q \xrightarrow{B,V'} P' \parallel Q} \quad \frac{\begin{array}{l} V \subseteq V' \\ \text{const}(V' \setminus V, B) \\ V' \cup \text{Var}(A) \subseteq \text{Var}(B) \\ B|_{\text{Var}(A) \cup \{r,n\}} = A \\ \text{const}^\circ(r \wedge n, A, 1) \\ Q \xrightarrow{A,V} Q' \quad Q' \neq \checkmark \end{array}}{P \parallel Q \xrightarrow{B,V'} P \parallel Q'} \quad \frac{\begin{array}{l} V \subseteq V' \\ \text{const}(V' \setminus V, B) \\ V' \cup \text{Var}(A) \subseteq \text{Var}(B) \\ B|_{\text{Var}(A) \cup \{r,n\}} = A \\ \text{const}^\circ(r \wedge n, A, 1) \\ P \xrightarrow{A,V} \checkmark \end{array}}{P \parallel Q \xrightarrow{B,V'} Q} \quad \frac{\begin{array}{l} V \subseteq V' \\ \text{const}(V' \setminus V, B) \\ V' \cup \text{Var}(A) \subseteq \text{Var}(B) \\ B|_{\text{Var}(A) \cup \{r,n\}} = A \\ \text{const}^\circ(r \wedge n, A, 1) \\ Q \xrightarrow{A,V} \checkmark \end{array}}{P \parallel Q \xrightarrow{B,V'} P}$$

4 A DC semantics of Hybrid Communicating Sequential Processes

Given a process P , $\llbracket P \rrbracket$, with some subscripts to be specified below, is a DC formula which defines the class of DC interpretations that represent runs of P .

Process variables and their corresponding DC temporal variables Real-valued process variables x are modelled by a pairs of DC temporal variables x and x' , which are meant to store the value of x at the

beginning and at the end of the reference interval, respectively. The axiom

$$\Box \forall z \neg (x' = z \wedge x \neq z).$$

entails that the values of x and x' are determined by the beginning and the end point of the reference interval, respectively. It can be shown that

$$\models_{DC} \Box \forall z \neg (x' = z \wedge x \neq z) \Rightarrow \Box (x = c \Rightarrow \neg (x \neq c \wedge \top)) \wedge \Box (x' = c \Rightarrow \neg (\top \wedge x' \neq c))$$

This is known as the locality principle in ITL about x . About primed variables x' , the locality principle holds wrt the endpoints of reference intervals. Boolean process variables are similarly modelled by propositional temporal letters. For the sake of brevity we put

$$\text{loc}(X) \Leftarrow \bigwedge_{\substack{x \in X \\ x \text{ is real}}} \Box \forall z \neg (x' = z \wedge x \neq z) \wedge \bigwedge_{\substack{x \in X \\ x \text{ is boolean}}} \Box \neg ((x' \wedge \neg x) \vee (\neg x' \wedge x))$$

In the sequel, given a DC term e or formula φ written using only unprimed variables, e' and φ' stand for the result of replacing all these variables by their respective primed counterparts.

Time derivatives of process variables As mentioned above, terms of the form \dot{x} where x is a process variable are treated as distinct process variables and are modelled by their respective temporal variables \dot{x} and \dot{x}' . The requirement on \dot{x} to be interpreted as the time derivative of x is incorporated in the semantics of continuous evolution statements.

Computation time and the parameters $\llbracket \cdot \rrbracket$ As explained in Section 3, we allow stretches of time that are dedicated to computation steps and are marked by the auxiliary boolean process variable r . Such stretches of time are conveniently excluded when calculating the duration of process execution. To this end, in DC formulas, we use a state variable R which indicates the time taken by computation steps by the reference process. Similarly, a state variable N indicates time for computation steps by which any process that runs in parallel with the reference one, including the reference one. R and N match the auxiliary variables r and n from the operational semantics and, just like r and n , are supposed to satisfy the condition $R \Rightarrow N$. We assume that all continuous evolution becomes temporarily suspended during intervals in which computation is performed, with the relevant real quantities and their derivatives remaining frozen. To guarantee the atomicity of assignment, computation intervals of different processes are not allowed to overlap. As it becomes clear in the DC semantics of \parallel below, P_i of $P_1 \parallel P_2$ are each given its own variable R_i , $i = 1, 2$, to mark computation time, and R_1 and R_2 are required to satisfy the constraints $\neg(R_1 \wedge R_2)$ and $R_1 \vee R_2 \Leftrightarrow R$ where R is the variable which marks computation times for the whole of $P_1 \parallel P_2$.

The semantics $\llbracket P \rrbracket_{R,N,V}$ of a HCSP term P is given in terms of the DC temporal variables which correspond to the process variables occurring in P , the state variables R and N , and the set of variables V which are controlled by P 's immediately enveloping \parallel -operand.

Assignment To express that the process variables from $X \subseteq V$ may change at the end of the reference interval only, and those from $V \setminus X$ remain unchanged, we write

$$\text{const}(V, X) \Leftarrow \bigwedge_{\substack{x \in V \setminus X \\ x \text{ is real}}} \Box (x' = x) \wedge \bigwedge_{\substack{x \in V \setminus X \\ x \text{ is boolean}}} \Box (x \Leftrightarrow x') \wedge \bigwedge_{\substack{x \in X \\ x \text{ is real}}} \Box^\circ (x' = x) \wedge \bigwedge_{\substack{x \in X \\ x \text{ is boolean}}} \Box^\circ (x' \Leftrightarrow x).$$

The meaning of simultaneous assignment is as follows:

$$\llbracket x_1, \dots, x_n := e_1, \dots, e_n \rrbracket_{R,N,V} \Leftarrow [R]_{\text{fin}} \wedge \text{const}(V, \{x_1, \dots, x_n\}) \wedge \bigwedge_{\substack{i=1, \dots, n \\ x_i \text{ is real}}} x'_i = e_i \wedge \bigwedge_{\substack{i=1, \dots, n \\ x_i \text{ is boolean}}} x'_i \Leftrightarrow e_i.$$

Parallel composition Consider processes P_1 and P_2 and $V \supseteq \text{Var} := (P_1 \parallel P_2)$. Let $\bar{1} = 2, \bar{2} = 1$. Let

$$\exists_{\parallel}(R, R_1, R_2, V, P_1, P_2)\varphi \Leftrightarrow \exists R_1 \exists R_2 \left(\begin{array}{l} [(R_1 \vee R_2 \Leftrightarrow R) \wedge \neg(R_1 \wedge R_2)]^0 \wedge \\ \bigwedge_{i=1}^2 \square([R_i] \Rightarrow \text{const}(V \setminus \text{Var} := (P_i))) \wedge \varphi \end{array} \right).$$

$\exists_{\parallel}(R, R_1, R_2, V_1, V_2)$ means that

- the R -subintervals for the computation steps of $P_1 \parallel P_2$ can be divided into R_1 - and R_2 -subintervals to mark the computation steps of some sub-processes P_1 and P_2 of P which run in parallel;
- the variables which are not controlled by P_i remain unchanged during P_i 's computation steps, $i = 1, 2$, and, finally,
- some property φ , which can involve R_1 and R_2 , holds.

The universal dual \forall_{\parallel} of \exists_{\parallel} is defined in the usual way. Let V_i abbreviate $\text{Var} := (P_i)$. Now we can define $\llbracket P_1 \parallel P_2 \rrbracket_{R, N, V}$ as

$$\exists_{\parallel}(R, R_1, R_2, V, P_1, P_2) \bigvee_{i=1}^2 \left(\begin{array}{l} \llbracket P_i \rrbracket_{R_i, N, V_i} \wedge ([N \wedge \neg R_i]_{\text{fin}}^0 \wedge \llbracket P_i \rrbracket_{R_{\bar{i}}, N, V_{\bar{i}}} \wedge [\neg R_{\bar{i}}]^0) \vee \\ ([N \wedge \neg R_i]_{\text{fin}}^0 \wedge \llbracket P_i \rrbracket_{R_i, N, V_i}) \wedge (\llbracket P_i \rrbracket_{R_{\bar{i}}, N, V_{\bar{i}}} \wedge [\neg R_{\bar{i}}]^0) \end{array} \right) \quad (5)$$

To understand the four disjunctive members of Φ above, note that $P_1 \parallel P_2$ always starts with some action on behalf of either P_1 , or P_2 , or both, in the case of continuous evolution. Hence (at most) one of P_i , $i = 1, 2$, needs to allow negligible time for P_i 's first step. This is expressed by a $[N \wedge \neg R_i]_{\text{fin}}^0$ before $\llbracket P_i \rrbracket_{R_i, N, V_i}$. The amount of time allowed is finite and may be 0 in case both P_1 and P_2 start with continuous evolution in parallel. This makes it necessary to consider two cases, depending on which process starts first. If P_i terminates before P_i , then a $[N \wedge \neg R_i]^0$ interval follows $\llbracket P_i \rrbracket_{R_{\bar{i}}, N, V_{\bar{i}}}$. This generates two more cases to consider, depending on the value of i .

The definitions of $\llbracket x_1, \dots, x_n := e_1, \dots, e_n \rrbracket_{R, N, V}$ and $\llbracket P_1 \parallel P_2 \rrbracket_{R, N, V}$ already appear in (4) and (5). Here follow the definitions for the rest of the basic constructs:

$$\begin{aligned} \llbracket \text{await } b \rrbracket_{R, N, V} &\Leftrightarrow \text{const}(V) \wedge ([\neg R] \vee \ell = 0) \wedge \boxplus^\circ \neg \bar{b}' \wedge (\bar{b}' \vee \ell = \infty) \\ \llbracket \langle F(\dot{x}, x) = 0 \wedge b \rangle \rrbracket_{R, N, V} &\Leftrightarrow \left(\begin{array}{l} \text{const}(V \setminus \{\dot{x}, x\}) \wedge [\neg R] \wedge \\ \square \left(\begin{array}{l} [N] \Rightarrow \text{const}(\{\dot{x}, x\}) \wedge \\ \forall ub \square(\dot{x} \leq ub) \Rightarrow x' \leq x + ub \int \neg N \wedge \\ \forall lb \square(\dot{x} \geq lb) \Rightarrow x' \geq x + lb \int \neg N \wedge \\ F(\dot{x}, x) = 0 \end{array} \right) \wedge \boxplus^\circ b \end{array} \right) \\ &\quad \wedge (\neg b \wedge \ell = 0) \end{aligned}$$

$$\begin{aligned} \llbracket P; Q \rrbracket_{R, N, V} &\Leftrightarrow (\llbracket P \rrbracket_{R, N, V} \wedge [N \wedge \neg R]_{\text{fin}}^0 \wedge \llbracket Q \rrbracket_{R, N, V}) \\ \llbracket P \sqcup Q \rrbracket_{R, N, V} &\Leftrightarrow \llbracket P \rrbracket_{R, N, V} \vee \llbracket Q \rrbracket_{R, N, V} \end{aligned}$$

$$\llbracket \text{if } b \text{ then } P \text{ else } Q \rrbracket_{R, N, V} \Leftrightarrow (b \wedge \llbracket P \rrbracket_{R, N, V}) \vee (\neg b \wedge \llbracket Q \rrbracket_{R, N, V})$$

$$\llbracket \text{while } b \text{ do } P \rrbracket_{R, N, V} \Leftrightarrow (b \wedge \llbracket P \rrbracket_{R, N, V} \wedge [\neg R]^0)^* \wedge (\neg b \wedge \ell = 0)$$

To understand $\llbracket \langle F(\dot{x}, x) = 0 \wedge b \rangle \rrbracket_{R, N, V}$, observe that, assuming I to be the DC interpretation in question and $\lambda t. I_t(\dot{x})$ to be continuous, the two inequalities in $\llbracket \langle F(\dot{x}, x) = 0 \wedge b \rangle \rrbracket_{R, N, V}$ express that

$$I_{t_2}(x) - I_{t_1}(x) = \int_{t_1}^{t_2} I_t(\dot{x})(1 - I_t(N))dt$$

at all finite subintervals $[t_1, t_2]$ of the reference intervals. This means that both \dot{x} and x are constant in N -subintervals, and $I_{t_2}(x) - I_{t_1}(x) = \int_{t_1}^{t_2} I_t(\dot{x})dt$ holds at $\neg N$ -subintervals.

Completeness of $\llbracket \cdot \rrbracket$ Given a process term P , every DC interpretation I for the vocabulary of $\llbracket P \rrbracket_N, N, V$ represents a valid behaviour of P with N being true in the subintervals which P uses for computation steps. To realize this, consider HCSP terms P, Q of the syntax

$$\begin{aligned} P, Q &::= \text{skip} \mid A; R \mid \sqcup_i A_i; R_i \mid \text{if } b \text{ then } P \text{ else } Q \mid P \sqcup Q \\ A &::= x := e \mid \text{await } b \mid \langle F(\dot{x}, x) = 0 \wedge b \rangle \end{aligned} \quad (6)$$

where R and R_i stand for a process term with no restrictions on its syntax (e.g., occurrences of **while**-terms are allowed). (6) is the *guarded normal form (GNF)* for HCSP terms, with the guards being the primitive terms of the form A , and can be established by induction on the construction of terms, with suitable equivalences for each combination of guarded operands that \parallel may happen to have. E.g.,

$$\langle F_1(\dot{x}, x) = 0 \wedge b_1 \rangle; P_1 \parallel \langle F_2(\dot{x}, x) = 0 \wedge b_2 \rangle; P_2 \quad (7)$$

is equivalent to

$$\begin{aligned} &\langle F_1(\dot{x}, x) = 0 \wedge F_2(\dot{x}, x) = 0 \wedge b_1 \wedge b_2 \rangle; \\ &\text{if } b_1 \text{ then } \langle F_1(\dot{x}, x) = 0 \wedge b_1 \rangle; P_1 \parallel P_2 \\ &\text{else if } b_2 \text{ then } \langle F_2(\dot{x}, x) = 0 \wedge b_2 \rangle; P_2 \\ &\text{else } P_1 \parallel P_2 \end{aligned} \quad (8)$$

Note that $\sqcup_i A_i; R_i$ is a modest generalization of *IO* as defined in (2). Some combinations of operands of \parallel require external choice to be extended this way, with the intended meaning being that if none of the A_i s which have the forms $ch?x$ and $ch!e$ is ready to execute, then some other option can be pursued immediately. For example, driving \parallel inwards may require using that

$$\begin{aligned} (ch_1?x \rightarrow P_1 \parallel ch_2!e \rightarrow P_2 \parallel ch_3?y \rightarrow P_3) \parallel ch_1!f; Q_1 \parallel ch_2?z; Q_2 &\equiv \\ ((x := f; (P_1 \parallel Q_1) \parallel ch_2?z; Q_2) \sqcup (z := e; (P_2 \parallel Q_2) \parallel ch_1!f; Q_1)) \parallel \\ ch_3?y; P_3 \parallel ch_1!f; Q_1 \parallel ch_2?z; Q_2. \end{aligned}$$

On the RHS of \equiv above, one of the assignments and the respective subsequent process are bound to take place immediately in case the environment is not ready to communicate over ch_3 .

The GNF renders the correspondence between the semantics of guards and the A s which appear in the rules for $\xrightarrow{A, V}$ explicit, thus making obvious that any finite prefix of a valid behaviour satisfies some *chop*-sequence of guards that can be generated by repeatedly applying the GNF a corresponding number of times and then using the distributivity of *chop* over disjunction, starting with the given process term, and then proceeding to transform the R -parts of guarded normal forms that appear in the process. The converse holds too. This entails that the denotational semantics is equivalent to the operational one.

5 Reasoning about Hybrid Communicating Sequential Processes with DC Hoare triples

We propose reasoning in terms of triples of the form

$$\{A\}P\{G\}_V \quad (9)$$

where A and G are DC formulas, P is a HCSP term, and V is a set of program variables. V is supposed to denote the variables whose evolution needs to be specified in the semantics of P , e.g., an assignment $x := e$ in P is supposed to leave the values of the variables $y \neq x$ unchanged. This enables deriving, e.g., $\{y = 0\}x := 1\{y = 0\}_{\{x, y\}}$, which would be untrue for a y that belongs to a process that runs in parallel with the considered one and is therefore not a member of V . Triple (9) is valid, if

$$\models \text{loc}(V) \wedge [R \Rightarrow N]^0 \wedge (A \frown \top) \Rightarrow \neg(\llbracket P \rrbracket_{R, N, V} \wedge \neg G \frown \top) \quad (10)$$

Since R and N typically occur in $\llbracket P \rrbracket_{R, N, V}$, triples (9) can have occurrences of R and N in A and G too, with their intended meanings.

Next we propose axioms and rules for deriving triples about processes with each of the HCSP constructs as the main one in them. For P of one of the forms **skip**, $x_1, \dots, x_n := e_1, \dots, e_n$, and $\langle \mathcal{F}(\dot{x}, x) = 0 \wedge b \rangle$, we introduce the axioms

$$\{\top\}P\{\llbracket P \rrbracket_{R,N,V}\}_V.$$

where V can be any set of process variables such that $V \supseteq \text{Var}_{:=}(P)$. Here follow the rules for reasoning about processes which are built using each of the remaining basic constructs:

$$\begin{array}{ll} \text{(seq)} & \frac{\{A\}P\{G\}_V \quad \{B\}Q\{H\}_V}{\text{loc}(V) \wedge [R \Rightarrow N]^0 \wedge (A \cap \top) \Rightarrow \neg(G \cap \neg([N \wedge \neg R]^0 \cap B \cap \top))} \{A\}P; Q\{(G \cap [N \wedge \neg R]^0 \cap H)\}_V \\ \text{(\sqcup)} & \frac{\{A\}P\{G\}_V \quad \{B\}Q\{H\}_V}{\{A \wedge B\}P \sqcup Q\{G \vee H\}_V} \\ \text{(if)} & \frac{\{A \wedge b\}P\{G\}_V \quad \{A \wedge \neg b\}Q\{G\}_V}{\{A\}\text{if } b \text{ then } P \text{ else } Q\{G\}_V} \\ \text{(while)} & \frac{\{A\}P\{G\}_V}{\text{loc}(V) \wedge [R \Rightarrow N]^0 \wedge (A \cap \top) \Rightarrow \neg(G \wedge \ell < \infty \cap \neg \diamond_l^c([\neg R]^0 \cap A))} \{A\}\text{while } b \text{ do } P\{((b \wedge G \cap [\neg R]^0)^* \cap \neg b \wedge \ell = 0)\}_V \end{array}$$

Parallel composition The established pattern suggests the following proof rule (\parallel):

$$\frac{\{A_1\}P_1\{G_1\}_{\text{Var}_{:=}(P_1)} \quad \{A_2\}P_2\{G_2\}_{\text{Var}_{:=}(P_2)}}{\left\{ \forall_{\parallel}(R, R_1, R_2, V, P_1, P_2) \left(\bigvee_{i=1}^2 \neg([N \wedge \neg R_i]_{\text{fin}}^0 \cap \neg[R_i/R]A_i \cap \top) \wedge ([R_i/R]A_i \cap \top) \right) \right\} P_1 \parallel P_2 \left\{ \exists_{\parallel}(R, R_1, R_2, V, P_1, P_2) \left(\bigvee_{i=1}^2 \frac{G_i \wedge ([N \wedge \neg R_i]_{\text{fin}}^0 \cap G_i \cap \neg[R_i/R]A_i)}{([N \wedge \neg R_i]_{\text{fin}}^0 \cap G_i) \wedge (G_i \cap \neg[R_i/R]A_i)} \right) \right\}_V}$$

This rule can be shown to be complete as it straightforwardly encodes the semantics of \parallel . However, it is not convenient for proof search as it only derives triples with a special form of the condition on the righthand side and actually induces the use of $\llbracket P_i \rrbracket_{R_i, N, \text{Var}_{:=}(P_i)}$ as G_i , which typically give excess detail. We discuss a way around this inconvenience below, together with the modifications of the setting which are needed in order to handle general HCSP fixpoints $\mu X.P$.

General rules Along with the process-specific rules, we also need the rules

$$\begin{array}{ll} \text{(N)} & \frac{\text{loc}(V) \wedge [R \Rightarrow N]^0 \wedge \diamond_l^c A \Rightarrow G}{\{A\}P\{G\}_V} \quad \text{Var}_{:=}(P) \subset V \\ \text{(K)} & \frac{\{A\}P\{G \Rightarrow H\}_V \quad \{B\}P\{G\}_V \quad \text{loc}(V) \wedge [R \Rightarrow N]^0 \wedge \diamond_l^c A \Rightarrow \diamond_l^c B}{\{A\}P\{H\}_V} \end{array}$$

These rules are analogous to the modal form N of Gödel's generalization rule (also known as the necessitation rule) and the modal axiom **K**.

Soundness and Completeness

The soundness of the proof rules is established by a straightforward induction on the construction of proofs with the definition of $\llbracket \cdot \rrbracket_{R,N,V}$. The system is also complete relative to validity in DC. This effectively means that we allow all valid DC formulas as axioms in proofs, or, equivalently, given some sufficiently powerful set of proof rules and axioms for the inference of valid DC formulas in DC with infinite intervals, our proof rules about triples suffice for deriving all the *valid triples*. Such systems are beyond the scope of our work.

A Hilbert-style proof system for ITL with infinite intervals was proposed and shown to be complete with respect to an abstractly defined class of time models (linearly ordered commutative groups) in [13], building on similar work about finite intervals from [8].

The deductive completeness of our proof system boils down to the possibility to infer triples of the form $\{\top\}P\{G\}_V$ for any given term P and a certain strongest corresponding G , which turns out to be the DC formula $\llbracket P \rrbracket_{R,N,V}$ that we call the semantics of P . Then the validity of $\{\top\}P\{\llbracket P \rrbracket_{R,N,V}\}_V$ is used to derive any valid triple about P by a simple use of the proof rules **K** and **N**. The completeness now follows from the fact that $\llbracket P \rrbracket_{R,N,V}$ defines the class of all valid behaviours of P .

Proposition 1 *The triple*

$$\{\top\}P\{\llbracket P \rrbracket_{R,N,V}\}_V \quad (11)$$

is derivable for all process terms P and all V such that $\text{Var}_{:=}(P) \supseteq V$.

Proof: Induction on the construction of P . The triple (11) is an axiom for P being the forms **skip**, $x_1, \dots, x_n := e_1, \dots, e_n$, and $\langle \mathcal{F}(\dot{x}, x) = 0 \wedge b \rangle$. For processes P of other forms, the induction step follows by single applications of the corresponding rules to the instances of (11) which are assumed to hold for P 's subprocesses. \dashv

Corollary 2 (relative completeness of the Hoare-style proof system) *Let A , G and P be such that (10) is valid. Then (9) is derivable in the extension of the given proof system by all DC theorems.*

Proof: By **N** we first derive

$$\{A\}P\{\llbracket P \rrbracket_{R,N,V} \Rightarrow G\}_V.$$

Now, using (11) from Proposition 1, the validity of $\diamond_{\ell}^c A \Rightarrow \diamond_{\ell}^c \top$ in DC and **K**, we derive (9). \dashv

6 General fixpoints and bottom-up proof search in HCSP

To avoid the constraints on the form of the conclusion of rule (\parallel), we propose a set of rules which correspond to the various possible forms of the operands of the designated \parallel in the considered HCSP term. These rules enable bottom-up proof search much like when using the rules for (just) CSP constructs, which is key to the applicability of classical Hoare-style proof. We propose separate rules for each combination of main connectives in the operands of \parallel , except \parallel itself and the fixpoint construct. For instance, the equivalence between (7) and (8) suggests the following rule for this particular combination of \parallel with the other connectives:

$$\frac{\begin{array}{l} \{P\}\langle F_1(\dot{x}, x) = 0 \wedge F_2(\dot{x}, x) = 0 \wedge b_1 \wedge b_2 \rangle\{R\} \\ \{R \wedge b_1 \wedge \neg b_2\}\langle F_1(\dot{x}, x) = 0 \wedge b_1 \rangle; P_1 \parallel P_2\{Q\} \\ \{R \wedge b_2 \wedge \neg b_1\}P_1 \parallel \langle F_2(\dot{x}, x) = 0 \wedge b_2 \rangle; P_2\{Q\} \\ \{R \wedge \neg b_1 \wedge \neg b_2\}P_1 \parallel P_2\{Q\} \end{array}}{\{P\}\langle F_1(\dot{x}, x) = 0 \wedge b_1 \rangle; P_1 \parallel \langle F_2(\dot{x}, x) = 0 \wedge b_2 \rangle; P_2\{Q\}}$$

Rules like the above one use the possibility to drive \parallel inwards by equivalences like that between (7) and (8), which can be derived for all combinations of the main connectives of \parallel 's operands, except for loops, and indeed can be used to eliminate \parallel . For **while**-loops, the GNF contains a copy of the loop on the RHS of *chop*:

$$\text{while } b \text{ do } P \equiv \text{if } b \text{ then } (P; \text{while } b \text{ do } P) \text{ else skip.} \quad (12)$$

Tail-recursive instances of $\mu X.G$ come handy in completing the elimination of \parallel in such cases by standard means, namely, by treating equivalences such as (12) as the equations leading to definitions such as (3).

To handle general recursion in our setting, we need to take care of the special way in which we treat A from $\{A\}P\{G\}$ in (10). In the rule for $\{A\}\text{while } b \text{ do } P\{G\}$ clipping of initial G -subintervals of an

A -interval are supposed to leave us with suffix subintervals which satisfy $(A \cap \top)$, to provide for successive executions of P . With X allowed on the LHS of *chop* in the P of $\mu X.P$, special care needs to be taken for this to be guaranteed. To this end, instead of $(A \cap \top)$, $\Diamond_l^c A$ is used to state that A holds at an interval that starts at the same time point as the reference one, and is not necessarily its subinterval. This is needed for reasoning from the viewpoint of intervals which accommodate nested recursive executions. The meaning of triples (9) becomes

$$\models \text{loc}(V) \wedge [R \Rightarrow N]^0 \wedge \Diamond_l^c A \wedge \llbracket P \rrbracket_{R,N,V} \Rightarrow G. \quad (13)$$

In this setting, $\mu X.P$ admits the proof rule, where X does not occur in A :

$$(\mu) \quad \frac{\text{loc}(V) \wedge [R \Rightarrow N]^0 \wedge \Diamond_l^c A \wedge G \Rightarrow [\Diamond_l^c A \wedge X/X]G \quad \{A\}P\{G\}_V}{\{A\}\mu X.P\{\mu X.G\}_V}$$

This rule subsumes the one for **while** – **do**, but only as part of a suitably revised variant of the whole proof system wrt (13). E.g., the rule for sequential composition becomes

$$\frac{\begin{array}{l} \{A\}P\{G\}_V \\ \{B\}Q\{H\}_V \\ \text{loc}(V) \wedge [R \Rightarrow N]^0 \wedge \Diamond_l^c A \Rightarrow \neg(G \wedge \ell < \infty \cap \neg \Diamond_l^c ([N \wedge \neg R]^0 \cap B)) \end{array}}{\{A\}P; Q\{(G \cap [N \wedge \neg R]^0 \cap H)\}_V}$$

7 Related work

Our work was influenced by the studies on DC-based reasoning about process-algebraic specification languages in [14, 18, 38, 17]. In a previous paper we proposed a calculus for HCSP [22], which was based on DC in a limited way and lacked compositionality. In [37] and [12] we gave other variants of compositional and sound calculi for HCSP with different assertion and temporal condition formats. Completeness was not considered. The approach in [12] and in this paper is largely drawn from [10] where computation time was proposed to be treated as negligible in order to simplify delay calculation, and the operator of projection was proposed to facilitate writing requirements with negligible time ignored. Hoare-style reasoning about real-time systems was also studied in the literature with explicit time logical languages [21]. However, our view is that using temporal logic languages is preferable. Dedicated temporal constructs both lead to more readable specifications, and facilitate the identification of classes of requirements that can be subjected to automated analysis. Another approach to the verification of hybrid systems is Platzer’s Differential Dynamic Logic [32]. However, the hybrid programs considered there have limited functionality. Communication, parallelism and interrupts are not handled. For logic compositionality, assume-guarantee reasoning has been studied for communication-based concurrency in CSP without timing in [25, 30].

Both in our work and in alternative approaches such as [25], the treatment of continuous evolution is somewhat separated from the analysis of the other basic process-algebraic constructs. Indeed we make a small step forward here by fully expressing the meaning of differential law-governed evolution in DC, which is theoretically sufficient to carry out all the relevant reasoning in the logic. Of course, the feasibility of such an approach is nowhere close to the state of art in the classical theory of ordinary differential equations. Indeed it would typically lead to formalized accounts of classical reasoning. Techniques for reasoning about the ODE-related requirements are the topic of separate studies, see, e.g., [33, 34, 23].

Concluding remarks

We have presented a weakly monotonic time-based semantics and a corresponding Hoare style proof system for HCSP with both the semantics and the temporal conditions in triples being in first-order DC with infinite intervals and extreme fixpoints. The proof system is compositional but the proof rule for parallel composition introduces complications because of the special form of the triples that it derives. However, we have shown that HCSP equivalences that can serve as elimination rules for \parallel can also be used to derive proof rules for

\parallel which do not bring the above difficulty and indeed are perfectly compatible with standard bottom-up proof search. Interestingly, the informal reading of the derived rules for \parallel together with the ones which are inherited from CSP, does not require the mention of weakly monotonic time technicalities. This means that the use of this special semantics can be restricted to establishing the soundness of practically relevant proof systems and awareness of its intricacies is not essential for applying the system. The meaning of triples we propose subsumes classical pre-/postcondition Hoare triples and triples linking (hybrid) temporal conditions in a streamlined way. This is a corollary of the choice to use assumptions which hold at an arbitrary initial subintervals, which is also compatible with reasoning about invariants A in terms of statements of the form $(A \frown \top) \Rightarrow \neg(\llbracket P \rrbracket \wedge \neg(A \frown \top))$.

References

- [1] M. Abadi and L. Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, 1993.
- [2] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, November 1983.
- [3] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems, LNCS 736*, pages 209–229, 1992.
- [4] C. Zhou and M. R. Hansen. *Duration Calculus. A Formal Approach to Real-Time Systems*. Springer, 2004.
- [5] C. Zhou, C. A. R. Hoare, and A. P. Ravn. A Calculus of Durations. *Information Processing Letters*, 40(5):269–276, 1991.
- [6] C. Zhou, V. H. Dang, and X. Li. A Duration Calculus with Infinite Intervals. In *Fundamentals of Computation Theory, LNCS 965*, pages 16–41. Springer, 1995.
- [7] A. Cau, B. Moszkowski, and H. Zedan. ITL web pages. URL: <http://www.antonio-cau.co.uk/ITL/>.
- [8] B. Dutertre. On First-order Interval Temporal Logic. Report CSD-TR-94-3, Department of Computer Science, Royal Holloway, University of London, 1995.
- [9] V. Goranko, A. Montanari, and G. Sciavicco. A road map of interval temporal logics and duration calculi. *Journal of Applied Non-Classical Logics*, 14(1-2):9–54, 2004.
- [10] D. P. Guelev and Dang Van Hung. Prefix and Projection onto State in Duration Calculus. In *Proceedings of TPTS’02, ENTCS 65(6)*. Elsevier Science, 2002.
- [11] D. P. Guelev and Dang Van Hung. A Relatively Complete Axiomatisation of Projection onto State in the Duration Calculus. *Journal of Applied Non-classical Logics, Special Issue on Interval Temporal Logics and Duration Calculi*, 14(1-2):151–182, 2004.
- [12] D. P. Guelev, S. Wang, N. Zhan, and C. Zhou. Super-dense computation in verification of hybrid CSP processes. In *FACS 2013, LNCS 8348*, pages 13–22. Springer, 2013.
- [13] H. Wang and Q. Xu. Completeness of Temporal Logics over Infinite Intervals. *Discrete Applied Mathematics*, 136(1):87–103, 2004.
- [14] H. Zhu and J. He. A DC-based Semantics for Verilog. Technical Report 183, UNU/IIST, P.O. Box 3058, Macau, 2000.

- [15] J. Y. Halpern and Y. Shoham. A Propositional Logic of Time Intervals. In *Proceedings of LICS'86*, pages 279–292. IEEE Computer Society Press, 1986.
- [16] M. R. Hansen and C. Zhou. Chopping a Point. In *BCS-FACS 7th refinement workshop*, Electronic Workshops in Computing. Springer, 1996.
- [17] A. E. Haxthausen and X. Yong. Linking DC together with TRSL. In *IFM'00, LNCS 1945*, pages 25–44, 2000.
- [18] J. He and Q. Xu. Advanced features of duration calculus and their applications in sequential hybrid programs. *Formal Asp. Comput.*, 15(1):84–99, 2003.
- [19] Jifeng He. From csp to hybrid systems. In A. W. Roscoe, editor, *A Classical Mind*, pages 171–189. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1994.
- [20] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of LICS'96*, pages 278–292. IEEE Computer Society Press, 1996.
- [21] J. Hooman. Extending hoare logic to real-time. *Formal Asp. Comput.*, 6(6A):801–826, 1994.
- [22] J. Liu, J. Lv, Z. Quan, N. Zhan, H. Zhao, C. Zhou, and L. Zou. A calculus for hybrid CSP. In *Proceedings of APLAS'10*, pages 1–15. Springer-Verlag, 2010.
- [23] J. Liu, N. Zhan, and H. Zhao. Computing semi-algebraic invariants for polynomial dynamical systems. In *Proceedings of EMSOFT'11*, pages 97–106. ACM, 2011.
- [24] Z. Manna and A. Pnueli. Verifying hybrid systems. In *Hybrid Systems, LNCS 736*, pages 4–35. Springer, 1992.
- [25] J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Trans. Software Eng.*, 7(4):417–426, 1981.
- [26] B. Moszkowski. Temporal Logic For Multilevel Reasoning About Hardware. *IEEE Computer*, 18(2):10–19, 1985.
- [27] B. Moszkowski. *Executing Temporal Logic Programs*. Cambridge University Press, 1986. URL: <http://www.cse.dmu.ac.uk/cau/papers/tempura-book.pdf>.
- [28] Ernst-Rüdiger Olderog and C. A. R. Hoare. Specification-oriented semantics for communicating processes. In *ICALP 1983, Proceedings*, volume 154 of *LNCS*, pages 561–572. Springer, 1983.
- [29] P. K. Pandya and Dang Van Hung. Duration Calculus of Weakly Monotonic Time. In *FTRTFT'98, LNCS 1486*, pages 55–64. Springer, 1998.
- [30] P. K. Pandya and M. Joseph. P - A logic - A compositional proof system for distributed programs. *Distributed Computing*, 5:37–54, 1991.
- [31] Paritosh K. Pandya. Some extensions to propositional mean-value calculus: Expressiveness and decidability. In Hans Kleine Büning, editor, *Computer Science Logic, 9th International Workshop, CSL '95, Annual Conference of the EACSL, Paderborn, Germany, September 22-29, 1995, Selected Papers*, volume 1092 of *Lecture Notes in Computer Science*, pages 434–451. Springer, 1995.
- [32] A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning*, 41(2):143–189, 2008.
- [33] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *HSCC'04, LNCS 2993*, pages 477–492. Springer, 2004.
- [34] S. Sankaranarayanan, H. Sipma, and Z. Manna. Constructing invariants for hybrid systems. In *HSCC'04, LNCS 2993*, pages 539–554. Springer, 2004.

- [35] Y. Venema. A Modal Logic for Chopping Intervals. *Journal of Logic and Computation*, 1(4):453–476, 1991.
- [36] Y. Venema. *Many-Dimensional Modal Logics*. Ph.D. thesis, University of Amsterdam, 1991.
- [37] S. Wang, N. Zhan, and D. Guelev. An assume/guarantee based compositional calculus for hybrid CSP. In *TAMC 2012, LNCS 7287*, pages 72–83. Springer, 2012.
- [38] X. Yong and C. George. An operational semantics for timed RAISE. In *FM’99, LNCS 1709*, pages 1008–1027. Springer, 1999.
- [39] C. Zhou, J. Wang, and A. P. Ravn. A formal description of hybrid systems. In *Hybrid Systems III, LNCS 1066*, pages 511–530. Springer, 1995.
- [40] Chaochen Zhou and Michael R. Hansen. *Duration Calculus - A Formal Approach to Real-Time Systems*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2004.