Programming with Infinitesimals: A WHILE-Language for Hybrid System Modeling*

Kohei Suenaga[1] and Ichiro Hasuo[2]

[1] JSPS Research Fellow, Kyoto University, Japan
[2] University of Tokyo, Japan

Abstract. We add, to the common combination of a WHILE-language and a Hoare-style program logic, a constant `dt` that represents an infinitesimal (i.e. infinitely small) value. The outcome is a framework for modeling and verification of *hybrid systems*: hybrid systems exhibit both continuous and discrete dynamics and getting them right is a pressing challenge. We rigorously define the semantics of programs in the language of *nonstandard analysis*, on the basis of which the program logic is shown to be sound and relatively complete.

1 Introduction

Hybrid systems are systems that deal with both discrete and continuous data. They have rapidly gained importance since more and more physical systems—cars, airplanes, etc.—are controlled with computers. Their sensors will provide physical, continuous data, while the behavior of controller software is governed by discrete data. Those information systems which interact with a physical ambience are more generally called *cyber-physical systems (CPS)*; hybrid systems are an important building block of CPSs.

Towards the goal of getting hybrid systems right, the research efforts have been mainly from two directions. *Control theory*—originally focusing on continuous data and their control e.g. via integration and differentiation—is currently extending its realm towards hybrid systems. The de facto standard SIMULINK tool for hybrid system modeling arises from this direction; it employs the block diagram formalism and offers simulation functionality—aiming at *testing* rather than *verification*. The current work is one of the attempts from the other direction—from the *formal verification* community—that advance from discrete to continuous.

Hybrid systems exhibit two kinds of dynamics: continuous *flow* and discrete *jump*. Hence for a formal verification approach to hybrid systems, the challenge is: 1) to incorporate flow-dynamics; and 2) to do so at the lowest possible cost, so that the discrete framework smoothly transfers to hybrid situations. A large body of existing work includes differential equations explicitly in the syntax; see the discussion of related work below. What we propose, instead, is to introduce a constant `dt` for an *infinitesimal* (i.e. infinitely small) value and *turn flow into jump*. With `dt`, the continuous operation of integration can be represented by a `while` loop, to which existing discrete techniques like Hoare-style program logics readily apply. For a rigorous mathematical development we employ *nonstandard analysis (NSA)* beautifully formalized by Robinson.

* We are greteful to Naoki Kobayashi and Toshimitsu Ushio for helpful discussions.

Concretely, in this paper we take the common combination of a WHILE-language, a first-order assertion language and a Hoare logic (e.g. in the textbook [12]); and add a constant dt to obtain a modeling and verification framework for hybrid systems. Its three ingredients are called $\textsc{While}^{\mathsf{dt}}$, $\textsc{Assn}^{\mathsf{dt}}$ and $\textsc{Hoare}^{\mathsf{dt}}$. These are connected by denotational semantics defined in the language of NSA. We prove soundness and relative completeness of the logic $\textsc{Hoare}^{\mathsf{dt}}$. Underlying the technical development is the idea of what we call *sectionwise execution*, illustrated by the following example.

Example 1.1 Let c_{elapse} be the following program; $\equiv$ denotes the syntactic equality.

$$c_{\mathsf{elapse}} \quad :\equiv \quad \Big[\ t := 0\ ; \quad \mathtt{while}\ t \leq 1\ \mathtt{do}\ t := t + \mathtt{dt}\ \Big]$$

The value designated by dt is infinitesimal; therefore the while loop will not terminate within finitely many steps. Nevertheless it is intuitive to expect that after an “execution” of this program (which takes an infinitely long time), the value of t should be infinitely close to 1. How can we turn this intuition into a mathematical argument?

Our idea is to think about *sectionwise execution*. For each natural number i we consider the *i-th section* of the program c_{elapse}, denoted by $c_{\mathsf{elapse}}|_i$. Concretely, $c_{\mathsf{elapse}}|_i$ is defined by replacing the infinitesimal dt in c_{elapse} by $\frac{1}{i+1}$:

$$c_{\mathsf{elapse}}|_i \quad :\equiv \quad \Big[\ t := 0\ ; \quad \mathtt{while}\ t \leq 1\ \mathtt{do}\ t := t + \tfrac{1}{i+1}\ \Big]\ .$$

Informally $c_{\mathsf{elapse}}|_i$ is the “i-th approximation” of the original c_{elapse}.

A section $c_{\mathsf{elapse}}|_i$ does terminate within finite steps and yields $1 + \frac{1}{i+1}$ as the value of t. Now we collect the outcomes of sectionwise executions and obtain a sequence

$$(\,1+1,\ 1+\tfrac{1}{2},\ 1+\tfrac{1}{3},\ \ldots,\ 1+\tfrac{1}{i},\ \ldots)$$

which is thought of as an incremental approximation of the actual outcome of the original program c_{elapse}. Indeed, in the language of NSA, the sequence represents a *hyperreal number r* that is infinitely close to 1.

We note that, as is clear from this example, a program of $\textsc{While}^{\mathsf{dt}}$ is not executable in general. We would rather regard $\textsc{While}^{\mathsf{dt}}$ as a modeling language for hybrid systems, with a merit of being close to a common programming style.

The idea of *turning flow into jump* with dt and NSA seems applicable to other discrete modeling/verification techniques than while-language and Hoare logic. We wish to further explore this potentiality. Adaptation of more advanced techniques for deductive program verification—such as invariant generation and type systems—to the presence of dt is another important direction of future work.

Due to the lack of space all the proofs are deferred to the appendix.

Related work There have been extensive research efforts towards hybrid systems from the formal verification community. Unlike the current work where we turn flow into jump via dt, most of them feature acute distinction between flow- and jump-dynamics.

Hybrid automaton [1] is one of the most successful approaches to verification of hybrid systems. A number of model-checking algorithms have been invented for automatic verification. The deductive approach in the current work—via theorem-proving in

HOARE$^{\mathtt{dt}}$—has an advantage of handling parameters (i.e. universal quantifiers or free variables that range over an infinite domain) well; model checking in such a situation necessarily calls for some abstraction technique. Our logic HOARE$^{\mathtt{dt}}$ is compositional too: a property of a whole system is inferred from the ones of its constituent parts.

Deductive verification of hybrid systems has seen great advancement through a recent series of work by Platzer and his colleagues, including [8, 9]. Their formalism is variations of *dynamic logic*, augmented with differential equations to incorporate flow-dynamics. They have also developed advanced techniques aimed at automated theorem proving, resulting in a sophisticated tool called KeYmaera [10]. We expect our approach (namely incorporating flow-dynamics via `dt`) offer a smoother transfer of existing discrete verification techniques to hybrid applications. Additionally, our preliminary observations suggest that some of the techniques developed by Platzer and others can be translated into the techniques that work in our framework.

There are several *hybrid process algebras* proposed for hybrid system modeling; see [6] for an overview. Some tools have been developed, too, as extensions of process algebra-based verification tools for discrete systems. However, the distinction between flow and jump makes their syntax cryptic and it is not uncommon [6] that flaws are found in widely accepted hybrid process algebras. They are suited for describing nondeterminism and concurrency; this is a feature we wish to add to our framework.

The use of NSA as a foundation of hybrid system modeling is not proposed for the first time; see e.g. [2, 3, 7, 11]. Compared to these existing work, we claim our novelty is a clean integration of NSA and the widely-accepted programming style of `while`-languages, with an accompanying verification framework by HOARE$^{\mathtt{dt}}$.

2 A Nonstandard Analysis Primer

This section is mainly for fixing notations. For more details see e.g. [5].

The type of statements "there exists $i_0 \in \mathbb{N}$ such that, for each $i \geq i_0$, $\varphi(i)$ holds" is typical in analysis. It is often put as "for sufficiently large $i \in \mathbb{N}$." This means: the set $\{i \in \mathbb{N} \mid \varphi(i) \text{ holds}\} \subseteq \mathbb{N}$ belongs to the family $\mathcal{F}_0 := \{S \subseteq \mathbb{N} \mid \mathbb{N} \setminus S \text{ is finite}\}$.

In NSA, the family $\mathcal{F}_0$ is extended to so-called an *ultrafilter* $\mathcal{F}$. The latter is a convenient domain of "i-indexed truth values": notably for each set $S \subseteq \mathbb{N}$, exactly one out of S and $\mathbb{N} \setminus S$ belongs to $\mathcal{F}$.

Definition 2.1 (Ultrafilter) A *filter* is a family $\mathcal{F} \subseteq \mathcal{P}(\mathbb{N})$ such that: 1) $X \in \mathcal{F}$ and $X \subseteq U$ implies $U \in \mathcal{F}$; 2) $X \cap Y \in \mathcal{F}$ if $X, Y \in \mathcal{F}$. A nonempty filter $\mathcal{F} \neq \emptyset$ is said to be *proper* if it does not contain $\emptyset \subseteq \mathbb{N}$; equivalently, if $\mathcal{F} \neq \mathcal{P}(\mathbb{N})$. An *ultrafilter* is a maximal proper filter; equivalently, it is a filter $\mathcal{F}$ such that for each $S \subseteq \mathbb{N}$, exactly one out of S and $\mathbb{N} \setminus S$ belongs to $\mathcal{F}$.

A filter $\mathcal{F}'$ can be always extended to an ultrafilter $\mathcal{F} \supseteq \mathcal{F}'$; this is proved using Zorn's lemma. Since the family $\mathcal{F}_0$ is easily seen to be a filter, we have:

Lemma 2.2 *There is an ultrafilter $\mathcal{F}$ that contains $\mathcal{F}_0 = \{S \subseteq \mathbb{N} \mid \mathbb{N} \setminus S$ is finite$\}$.*

Throughout the rest of the paper we fix such $\mathcal{F}$. Its properties to be noted: 1) $\mathcal{F}$ is closed under finite intersections and infinite unions; 2) exactly one of S or $\mathbb{N} \setminus S$ belongs to $\mathcal{F}$, for each $S \subseteq \mathbb{N}$; and 3) if S is such that $\mathbb{N} \setminus S$ is finite, then $S \in \mathcal{F}$.

We say "$\varphi(i)$ holds for almost every $i \in \mathbb{N}$" for the fact that the set $\{i \mid \varphi(i) \text{ holds}\}$ belongs to $\mathcal{F}$. For its negation we say "for negligibly many i."

Definition 2.3 (Hypernumber $\boldsymbol{d} \in {}^*\mathbb{D}$) For a set $\mathbb{D}$ (typically it is $\mathbb{N}$ or $\mathbb{R}$), we define the set ${}^*\mathbb{D}$ by ${}^*\mathbb{D} := \mathbb{D}^{\mathbb{N}} / \sim_{\mathcal{F}}$. It is the set of infinite sequences on $\mathbb{D}$ modulo the following equivalence $\sim_{\mathcal{F}}$: we define $(d_0, d_1, \dots) \sim_{\mathcal{F}} (d'_0, d'_1, \dots)$ by

$$d_i = d'_i \text{ "for almost every } i\text{," that is, } \{i \in \mathbb{N} \mid d_i = d'_i\} \in \mathcal{F}.$$

An equivalence class $\big[(d_i)_{i\in\mathbb{N}}\big]_{\sim_{\mathcal{F}}} \in {}^*\mathbb{D}$ shall be also denoted by $\big[(d_i)_{i\in\mathbb{N}}\big]$ or $(d_i)_{i\in\mathbb{N}}$ when no confusion occurs. An element $\boldsymbol{d} \in {}^*\mathbb{D}$ is called a *hypernumber*; in contrast $d \in \mathbb{D}$ is a *standard number*. Hypernumbers will be denoted in boldface like $\boldsymbol{d}$.

We say that $(d_i)_{i\in\mathbb{N}}$ is a *sequence representation* of $\boldsymbol{d} \in {}^*\mathbb{D}$ if $\boldsymbol{d} = [(d_i)_i]$. Note that, given $\boldsymbol{d} \in {}^*\mathbb{D}$, its sequence representation is not unique. There is a canonical embedding $\mathbb{D} \hookrightarrow {}^*\mathbb{D}$ mapping d to $[(d, d, \dots)]$; the latter shall also be denoted by d.

Definition 2.4 (Operations and relations on ${}^*\mathbb{D}$) An operation $f : \mathbb{D}^k \to \mathbb{D}$ of any finite arity k (such as $+ : \mathbb{R}^2 \to \mathbb{R}$) has a canonical "pointwise" extension $f : ({}^*\mathbb{D})^k \to {}^*\mathbb{D}$. A binary relation $R \subseteq \mathbb{D}^2$ (such as $<$ on real numbers) also extends to $R \subseteq ({}^*\mathbb{D})^2$.

$$\begin{array}{l} f\big(\,\big[\,(d_i^{(0)})_{i\in\mathbb{N}}\,\big], \dots, \big[\,(d_i^{(k-1)})_{i\in\mathbb{N}}\,\big]\,\big) \;:=\; \big[\,\big(\,f(d_i^{(0)}, \dots, d_i^{(k-1)})\,\big)_{i\in\mathbb{N}}\,\big]\ , \\ \big[\,(d_i)_{i\in\mathbb{N}}\,\big]\ R\ \big[\,(d'_i)_{i\in\mathbb{N}}\,\big] \quad \overset{\text{def.}}{\Longleftrightarrow} \quad d_i\ R\ d'_i \quad \text{for almost every } i. \end{array}$$

These extensions are well-defined since $\mathcal{F}$ is closed under finite intersections.

Example 2.5 (ω and ω^{-1}) By ω we denote the hypernumber $\omega = [\,(1, 2, 3, \dots)\,] \in {}^*\mathbb{N}$. It is bigger than (the embedding of) any (standard) natural number $n = [\,(n, n, n, \dots)\,]$, since we have $n < i$ for all i except for finitely many. The presence of ω shows that $\mathbb{N} \subsetneq {}^*\mathbb{N}$ and $\mathbb{R} \subsetneq {}^*\mathbb{R}$. Its inverse $\omega^{-1} = [\,(1, \frac{1}{2}, \frac{1}{3}, \dots)\,]$ is positive ($0 < \omega^{-1}$) but is smaller than any (standard) positive real number $r > 0$.

These hypernumbers—*infinite* ω and *infinitesimal* ω^{-1}—will be heavily used.

For the set $\mathbb{B} = \{\mathtt{tt}, \mathtt{ff}\}$ of Boolean truth values we have the following. Therefore a "hyper Boolean value" does not make sense.

Lemma 2.6 *Assume that $\mathbb{D}$ is a finite set $\mathbb{D} = \{a_1, \dots, a_n\}$. Then the canonical inclusion map $\mathbb{D} \hookrightarrow {}^*\mathbb{D}$ is bijective. In particular we have ${}^*\mathbb{B} \cong \mathbb{B}$ for $\mathbb{B} = \{\mathtt{tt}, \mathtt{ff}\}$.* □

3 Programming Language WHILE$^{\mathsf{dt}}$

3.1 Syntax

We fix a countable set **Var** of *variables*.

Definition 3.1 (WHILE$^{\mathtt{dt}}$, WHILE) The syntax of our target language WHILE$^{\mathtt{dt}}$ is:

$$\begin{array}{lll} \mathbf{AExp} \ni & a ::= x \mid \mathtt{c}_r \mid a_1 \,\mathtt{aop}\, a_2 \mid \mathtt{dt} \mid \infty \\ & \text{where } x \in \mathbf{Var}, \mathtt{c}_r \text{ is a constant for } r \in \mathbb{R}, \text{and } \mathtt{aop} \in \{+, -, \cdot, ^\wedge\} \\ \mathbf{BExp} \ni & b ::= \mathtt{true} \mid \mathtt{false} \mid b_1 \wedge b_2 \mid \neg b \mid a_1 < a_2 \\ \mathbf{Cmd} \ni & c ::= \mathtt{skip} \mid x := a \mid c_1 ; c_2 \mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ b\ \mathtt{do}\ c \end{array}$$

An expression in $\mathbf{AExp}$ is said to be *arithmetic*; one in $\mathbf{BExp}$ is *Boolean* and one in $\mathbf{Cmd}$ is a *command*. The operator $a^\wedge b$ designates "a to the power of b" and will be denoted by a^b. The operator $^\wedge$ is included as a primitive for the purpose of relative completeness (Thm. 5.4). We will often denote the constant $\mathtt{c}_r$ by r.

By WHILE, we denote the fragment of WHILE$^{\mathtt{dt}}$ without the constants $\mathtt{dt}$ and ∞.

The language WHILE is much like usual programming languages with a `while` construct, such as $\mathbf{IMP}$ in the textbook [12]. Its only anomaly is a constant $\mathtt{c}_r$ for any real number r: although unrealistic from the implementation viewpoint, it is fine because WHILE is meant to be a modeling language. Then our target language WHILE$^{\mathtt{dt}}$ is obtained by adding $\mathtt{dt}$ and ∞: they designate an infinitesimal ω^{-1} and an infinite ω.

The relations $>, \leq, \geq$ and $=$ are definable in WHILE$^{\mathtt{dt}}$: $x > y$ as $y < x$; $\leq$ as the negation of $>$; and $=$ as the conjunction of $\leq$ and $\geq$. So are all the Boolean connectives such as $\vee$ and $\Rightarrow$, using $\neg$ and $\wedge$. We emphasize that $\mathtt{dt}$ is by itself a constant and has nothing to do with a variable t. We could have used a more neutral notation like ∂ in [2]; however the notation $\mathtt{dt}$ turns out to be conveniently intuitive in many examples.

Definition 3.2 (Section of WHILE$^{\mathtt{dt}}$ expression) Let e be an expression of WHILE$^{\mathtt{dt}}$, and $i \in \mathbb{N}$. The *i-th section* of e, denoted by $e|_i$, is obtained by replacing each occurrence of $\mathtt{dt}$ and ∞ in e by the constants $\mathtt{c}_{1/(i+1)}$ and $\mathtt{c}_{i+1}$, respectively. Obviously $e|_i$ is an expression of WHILE.

Example 3.3 (Train control) Our first examples model small fragments of the European Train Control System (ETCS); this is also a leading example in [8]. The following command c_{accel} models a train accelerating at a constant acceleration a, until the time ε is reached. The variable v is for the train's velocity; and z is for its position.

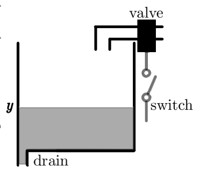
$$\begin{array}{rl} c_{\mathsf{accel}} :\equiv & \left[\mathtt{while}\ t < \varepsilon\ \mathtt{do}\ c_{\mathsf{drive}}\right] \qquad \text{where} \\ c_{\mathsf{drive}} :\equiv & \left[t := t + \mathtt{dt}\,;\ v := v + a \cdot \mathtt{dt}\,;\ z := z + v \cdot \mathtt{dt}\right] \end{array} \tag{1}$$

The following command $c_{\mathsf{chkAndBrake}}$ models a train that, once its distance from the boundary m gets within the safety distance s, starts braking with the force $b > 0$. However the check if the train is within the safety distance is done only every ε seconds.

$$\begin{array}{rl} c_{\mathsf{chkAndBrake}} :\equiv & \left[\mathtt{while}\ v > 0\ \mathtt{do}\ (c_{\mathsf{corr}};\ c_{\mathsf{accel}})\right] \qquad \text{where} \\ c_{\mathsf{corr}} :\equiv & \left[t := 0\,;\ \mathtt{if}\ m - z < s\ \mathtt{then}\ a := -b\ \mathtt{else}\ a := 0\right] \end{array} \tag{2}$$

Example 3.4 (Water-level monitor) Our second example is an adaptation from [1]. Compared to the above train example, it exhibits simpler flow-dynamics (the first derivative is already a constant) but more complex jump-dynamics.

There is a water tank with a constant drain (2 cm per second). When the water level y gets lower than 5 cm the switch is turned on, which eventually opens up the valve but only after a time lag of two seconds. While the valve is open, the water level y rises by 1 cm per second. Once y reaches 10 cm the switch is turned off, which will shut the valve but again after a time lag of two seconds.



In the following command c_{water}, the variable x is a timer for a time lag. The `case` construct is an obvious abbreviation of nested `if` ... `then` ... `else` ....

$$
c_{\mathsf{water}} \;:\equiv\; \left[\begin{array}{l}
x := 0;\ y := 1;\ s := 1;\ v := 1; \\
\texttt{while}\ t < t_{\mathsf{max}}\ \texttt{do}\ \ \{ \\
\quad x := x + \texttt{dt};\quad t := t + \texttt{dt}; \\
\quad \texttt{if}\ v = 0\ \texttt{then}\ y := y - 2\cdot\texttt{dt}\ \texttt{else}\ y := y + \texttt{dt}; \\
\quad \texttt{case}\ \ \{ s = 0 \,\wedge\, v = 0 \,\wedge\, y \le 5 : \quad s := 1;\ x := 0; \\
\qquad\qquad s = 1 \,\wedge\, v = 0 \,\wedge\, x \ge 2 : \quad v := 1; \\
\qquad\qquad s = 1 \,\wedge\, v = 1 \,\wedge\, 10 \le y : \quad s := 0;\ x := 0; \\
\qquad\qquad s = 0 \,\wedge\, v = 1 \,\wedge\, x \ge 2 : \quad v := 0; \\
\qquad\qquad \texttt{else} \qquad\qquad\qquad\qquad \texttt{skip}\ \ \}\}
\end{array}\right] \tag{3}
$$

3.2 Denotational Semantics

We follow [12] and interpret a command of $\textsc{While}^{\mathsf{dt}}$ as a transformer on memory states. Our state can store hyperreal numbers such as the infinitesimal $\omega^{-1} = [\,(1, \frac{1}{2}, \frac{1}{3}, \dots)\,]$, hence is called a *hyperstate*.

Definition 3.5 (Hyperstate, state) A *hyperstate* $\boldsymbol{\sigma}$ is either $\boldsymbol{\sigma} = \bot$ (“undefined”) or a function $\boldsymbol{\sigma} : \mathbf{Var} \to {}^*\mathbb{R}$. A *state* is a standard version of a hyperstate: namely, a *state* σ is either $\sigma = \bot$ or a function $\sigma : \mathbf{Var} \to \mathbb{R}$.

We denote the collection of hyperstates by $\mathbf{HSt}$; that of (standard) states by $\mathbf{St}$.

The definition of (hyper)state as a total function—rather than a partial function with a finite domain—follows [12]. This makes the denotational semantics much simpler. Practically, one can imagine there is a fixed default value (say 0) for any variable.

The following definition is as usual.

Definition 3.6 (State update) Let $\boldsymbol{\sigma} \in \mathbf{HSt}$ be a hyperstate, $x \in \mathbf{Var}$ and $\boldsymbol{r} \in {}^*\mathbb{R}$. We define an *updated hyperstate* $\boldsymbol{\sigma}[x \mapsto \boldsymbol{r}]$ as follows. When $\boldsymbol{\sigma} = \bot$, we set $\bot[x \mapsto \boldsymbol{r}] := \bot$. Otherwise: $\big(\boldsymbol{\sigma}[x \mapsto \boldsymbol{r}]\big)(x) := \boldsymbol{r}$; and for $y \neq x$, $\big(\boldsymbol{\sigma}[x \mapsto \boldsymbol{r}]\big)(y) := \boldsymbol{\sigma}(y)$.

An *updated (standard) state* $\sigma[x \mapsto r]$ is defined analogously.

Definition 3.7 (Sequence representation) Let $(\sigma_i)_{i\in\mathbb{N}}$ be a sequence of (standard) states. It gives rise to a hyperstate—denoted by $[(\sigma_i)_{i\in\mathbb{N}}]$ or simply by $(\sigma_i)_{i\in\mathbb{N}}$—in the following way. We set $(\sigma_i)_{i\in\mathbb{N}} := \bot$ if $\sigma_i = \bot$ for almost all i. Otherwise $[\,(\sigma_i)_{i\in\mathbb{N}}\,] \neq \bot$ and we set $\big[\,(\sigma_i)_{i\in\mathbb{N}}\,\big](x) := \big[\,(\sigma_i(x))_{i\in\mathbb{N}}\,\big]$, where the latter is the hyperreal represented by the sequence $(\sigma_i(x))_i$ of reals. For $i \in \mathbb{N}$ such that $\sigma_i = \bot$, the value $\sigma_i(x)$ is not defined; in this case we use an arbitrary real number (say 0) for $\sigma_i(x)$. This does not affect the resulting hyperstate since $\sigma_i(x)$ is defined for almost all i.

Let $\boldsymbol{\sigma} \in \mathbf{HSt}$ be a hyperstate, and $(\sigma_i)_{i\in\mathbb{N}}$ be a sequence of states. We say $(\sigma_i)_{i\in\mathbb{N}}$ is a *sequence representation* of $\boldsymbol{\sigma}$ if it gives rise to $\boldsymbol{\sigma}$, that is, $\big[(\sigma_i)_{i\in\mathbb{N}}\big] = \boldsymbol{\sigma}$. In what follows we shall often denote a sequence representation of $\boldsymbol{\sigma}$ by $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$. We emphasize that given $\boldsymbol{\sigma} \in \mathbf{HSt}$, its sequence representation $(\boldsymbol{\sigma}|_i)_i$ is not unique.

The denotational semantics of WHILE$^{\mathsf{dt}}$ is a straightforward adaptation of the usual semantics of WHILE, except for the `while` clauses where we use sectionwise execution (see Ex. 1.1). As we see later in Lem. 3.10, however, the idea of sectionwise execution extends to the whole language WHILE$^{\mathsf{dt}}$.

Definition 3.8 (Denotational semantics for WHILE$^{\mathsf{dt}}$) For expressions of WHILE$^{\mathsf{dt}}$, their *denotation*

$$\begin{array}{lll} [\![a]\!] : & \mathbf{HSt} \longrightarrow {}^*\mathbb{R} \cup \{\bot\} & \text{for } a \in \mathbf{AExp}, \\ [\![b]\!] : & \mathbf{HSt} \longrightarrow \mathbb{B} \cup \{\bot\} & \text{for } b \in \mathbf{BExp}, \text{ and} \\ [\![c]\!] : & \mathbf{HSt} \longrightarrow \mathbf{HSt} & \text{for } c \in \mathbf{Cmd} \end{array}$$

is defined as follows. Recall that $\bot$ means “undefined” (cf. Def. 3.5); that $\mathbb{B} = \{\mathtt{tt}, \mathtt{ff}\}$ is the set of Boolean truth values; and that ${}^*\mathbb{B} \cong \mathbb{B}$ (Lem. 2.6).

If $\boldsymbol{\sigma} = \bot$, we define $[\![e]\!]\bot := \bot$ for any expression e. If $\boldsymbol{\sigma} \neq \bot$ we define

$$\begin{array}{lll} [\![x]\!]\boldsymbol{\sigma} & := \boldsymbol{\sigma}(x) & [\![\mathtt{c}_r]\!]\boldsymbol{\sigma} := r \quad \text{for each } r \in \mathbb{R} \\ [\![a_1 \ \mathtt{aop}\ a_2]\!]\boldsymbol{\sigma} & := [\![a_1]\!]\boldsymbol{\sigma}\ \mathtt{aop}\ [\![a_2]\!]\boldsymbol{\sigma} & \\ [\![\mathtt{dt}]\!]\boldsymbol{\sigma} & := \omega^{-1} = \big[\,(1, \frac{1}{2}, \frac{1}{3}, \dots)\,\big] & [\![\infty]\!]\boldsymbol{\sigma} := \omega = \big[\,(1,2,3,\dots)\,\big] \end{array}$$

$$\begin{array}{ll} [\![\mathtt{true}]\!]\boldsymbol{\sigma} := \mathtt{tt} & [\![\mathtt{false}]\!]\boldsymbol{\sigma} := \mathtt{ff} \\ [\![b_1 \wedge b_2]\!]\boldsymbol{\sigma} := [\![b_1]\!]\boldsymbol{\sigma} \wedge [\![b_2]\!]\boldsymbol{\sigma} & [\![\neg b]\!]\boldsymbol{\sigma} := \neg([\![b]\!]\boldsymbol{\sigma}) \\ [\![a_1 < a_2]\!]\boldsymbol{\sigma} := [\![a_1]\!]\boldsymbol{\sigma} < [\![a_2]\!]\boldsymbol{\sigma} & \end{array}$$

$$[\![\mathtt{skip}]\!]\boldsymbol{\sigma} := \boldsymbol{\sigma} \qquad [\![x := a]\!]\boldsymbol{\sigma} := \boldsymbol{\sigma}\big[\,x \mapsto [\![a]\!]\boldsymbol{\sigma}\,\big] \qquad [\![c_1; c_2]\!]\boldsymbol{\sigma} := [\![c_2]\!]\big(\,[\![c_1]\!]\boldsymbol{\sigma}\,\big)$$

$$[\![\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2]\!]\boldsymbol{\sigma} := \begin{cases} [\![c_1]\!]\boldsymbol{\sigma} & \text{if } [\![b]\!]\boldsymbol{\sigma} = \mathtt{tt} \\ [\![c_2]\!]\boldsymbol{\sigma} & \text{if } [\![b]\!]\boldsymbol{\sigma} = \mathtt{ff} \end{cases}$$

$$[\![\mathtt{while}\ b\ \mathtt{do}\ c]\!]\boldsymbol{\sigma} := \Big(\ \big[\!\big[\,(\mathtt{while}\ b\ \mathtt{do}\ c)|_i\,\big]\!\big](\boldsymbol{\sigma}|_i)\ \Big)_{i\in\mathbb{N}}\ , \tag{4}$$
where $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ is an arbitrary sequence representation of $\boldsymbol{\sigma}$ (Def. 3.7)

Here $\mathtt{aop} \in \{+, -, \times, {}^\wedge\}$ and $<$ are interpreted on ${}^*\mathbb{R}$ as in Def. 2.4. For each $e \in \mathbf{AExp} \cup \mathbf{BExp}$, we obviously have $[\![e]\!]\boldsymbol{\sigma} = \bot$ if and only if $\boldsymbol{\sigma} = \bot$. It may happen that $[\![c]\!]\boldsymbol{\sigma} = \bot$ with $\boldsymbol{\sigma} \neq \bot$, due to nontermination of `while` loops.

In the semantics of `while` clauses (4), the section $(\mathtt{while}\ b\ \mathtt{do}\ c)|_i$ is a command of WHILE (Def. 3.2); and $\boldsymbol{\sigma}|_i$ is a (standard) state. Therefore the standard state $\big[\!\big[\,(\mathtt{while}\ b\ \mathtt{do}\ c)|_i\,\big]\!\big](\boldsymbol{\sigma}|_i)$ can be defined by the usual semantics of `while` constructs (see e.g. [12]). That is,

$$[\![\mathtt{while}\ b'\ \mathtt{do}\ c']\!]\sigma = \sigma' \qquad \overset{\text{def.}}{\Longleftrightarrow}$$
$$\left\{ \begin{array}{l} -\ \sigma = \sigma' = \bot; \\ -\ \text{there exists a finite sequence } \sigma = \sigma_0, \sigma_1, \dots, \sigma_n = \sigma' \text{ such that: } [\![b']\!]\sigma_n = \\ \quad \mathtt{ff}; \text{ and for each } j \in [0, n).\ \big(\,[\![b']\!]\sigma_j = \mathtt{tt}\ \&\ [\![c']\!]\sigma_j = \sigma_{j+1}\,\big); \text{ or} \\ -\ \text{such a finite sequence does not exist and } \sigma' = \bot. \end{array} \right. \tag{5}$$

By bundling these up for all i, and regarding it as a hyperstate (Def. 3.7), we obtain the right-hand side of (4). The well-definedness of (4) is proved in Lem. 3.9.

Lemma 3.9 *The semantics of* `while` *clauses (4) is well-defined, being independent of the choice of a sequence representation* $(\boldsymbol{\sigma}|_i)_i$ *of the hyperstate* $\boldsymbol{\sigma}$. □

In proving the lemma it is crucial that: the set $\{x_1, \ldots, x_n\}$ of variables that are relevant to the execution of the command is finite and statically known. This would not be the case with a programming language that allows dynamical creation of fresh variables.

We have chosen not to include the division operator / in $\text{WHILE}^{\mathsf{dt}}$; this is to avoid handling of *division by zero* in the semantics, which is cumbersome but seems feasible.

Here is one of our two key lemmas. Its proof is by induction.

Lemma 3.10 (Sectionwise Execution Lemma) *Let* e *be an arbitrary expression of* $\text{WHILE}^{\mathsf{dt}}$*;* $\boldsymbol{\sigma}$ *be a hyperstate; and* $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ *be an arbitrary sequence representation of* $\boldsymbol{\sigma}$*. We have*

$$[\![e]\!]\boldsymbol{\sigma} = \left[\, \left([\![e|_i]\!](\boldsymbol{\sigma}|_i) \right)_{i\in\mathbb{N}} \right] \enspace .$$

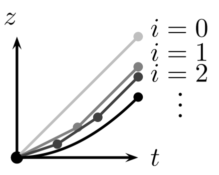
Here the denotational semantics $[\![e|_i]\!]$ *of a* WHILE *expression* $e|_i$ *is defined in a usual way (i.e. like in Def. 3.8; for* `while` *clauses see (5)).* □

Example 3.11 Consider c_{accel} in Ex. 3.3. For simplicity let us fix the parameters: $c_{\mathsf{accel1}} :\equiv [\, t := 0;\ \varepsilon := 1;\ a := 1;\ v := 0;\ z := 0;\ c_{\mathsf{accel}}\,]$. Its i-th section $c_{\mathsf{accel1}}|_i$ has the obvious semantics. For any (standard) state $\sigma \neq \bot$, the real number $([\![c_{\mathsf{accel1}}|_i]\!]\sigma)(z)$—the traveled distance—is easily calculated as

$$\tfrac{1}{i+1}\cdot\tfrac{1}{i+1} + \tfrac{1}{i+1}\cdot\tfrac{2}{i+1} + \cdots + \tfrac{1}{i+1}\cdot\tfrac{i+1}{i+1} = \tfrac{(i+1)(i+2)}{2(i+1)^2} = \tfrac{1}{2}\cdot\tfrac{i+2}{i+1} \enspace .$$

Therefore by (4), for any hyperstate $\boldsymbol{\sigma} \neq \bot$, the hyperreal $([\![c_{\mathsf{accel1}}]\!]\boldsymbol{\sigma})(z)$ is equal to

$$\left[\, \left(1,\ \tfrac{3}{4},\ \tfrac{2}{3},\ \tfrac{5}{8},\ \ldots,\ \tfrac{1}{2}\cdot\tfrac{i+2}{i+1},\ \ldots\right) \right] \enspace ;$$



this is a hyperreal that is infinitely close to $1/2$.

Much like Ex. 1.1, one way to look at this sectionwise semantics is as an incremental approximation. Here it approximates the solution $z = \frac{1}{2}t^2$ of the differential equation $z'' = 1$, obtained via the Riemann integral. See the above figure.

Remark 3.12 (Denotation of `dt`**)** We fixed ω^{-1} as the denotation of `dt`. However there are more infinitesimals, such as $(\pi\omega)^{-1} = (\frac{1}{\pi}, \frac{1}{2\pi}, \frac{1}{3\pi}, \ldots)$ with $(\pi\omega)^{-1} < \omega^{-1}$. The choice of `dt`'s denotation does affect the behavior of the following program $c_{\mathsf{nonintegrable}}$:

$$c_{\mathsf{nonintegrable}} \quad :\equiv \quad \left[\ x := 1\,;\quad \mathtt{while}\ x \neq 0\ \mathtt{do}\ x := x - \mathtt{dt}\ \right] \enspace .$$

When we replace `dt` by $\frac{1}{i+1}$ the program terminates with $x = 0$; hence by our semantics the program yields a non-$\bot$ hyperstate with $x \mapsto 0$. However, replacing `dt` by $\frac{1}{(i+1)\pi}$ with π irrational, the program terminates for no i and it leads to the hyperstate $\bot$.

In fact, indifference to the choice of an infinitesimal value (violated by $c_{\mathsf{nonintegrable}}$) is a typical condition in nonstandard analysis, found e.g. in the characterization of differentiability or Riemann integrability (see [5]). In this sense the program $c_{\mathsf{nonintegrable}}$ is “nonintegrable”; we are yet to see if we can check integrability by syntactic means.

The program $c_{\mathsf{nonintegrable}}$ can be modified into the one with more reasonable behavior, by replacing the guard $x \neq 0$ by $x > 0$. One easily sees that, while different choices of `dt`'s denotation (e.g. ω^{-1} vs. $(\pi\omega)^{-1}$) still lead to different post-hyperstates, the differences lie within infinitesimal gaps. The same is true of all the "realistic" programs that we have looked at.

4 Assertion Language $\textsc{Assn}^{\mathsf{dt}}$

Definition 4.1 ($\textsc{Assn}^{\mathsf{dt}}$, $\textsc{Assn}$) The syntax of our assertion language $\textsc{Assn}^{\mathsf{dt}}$ is:

$$\begin{array}{llll}
\mathbf{AExp} \ni & a & ::= & x \mid \mathtt{c}_r \mid a_1 \;\mathtt{aop}\; a_2 \mid \mathtt{dt} \mid \infty \qquad \text{(the same as in } \textsc{While}^{\mathsf{dt}}\text{)} \\
\mathbf{Fml} \ni & A & ::= & \mathtt{true} \mid \mathtt{false} \mid A_1 \wedge A_2 \mid \neg A \mid a_1 < a_2 \mid \\
 & & & \forall x \in {}^*\mathbb{N}.\, A \mid \forall x \in {}^*\mathbb{R}.\, A \qquad \text{where } x \in \mathbf{Var}
\end{array}$$

An expression in the family $\mathbf{Fml}$ is called an *(assertion) formula*.

We introduce existential quantifiers as notational conventions: $\exists x \in {}^*\mathbb{D}.\, A :\equiv \neg\forall x \in {}^*\mathbb{D}.\, \neg A$, where $\mathbb{D} \in \{\mathbb{N}, \mathbb{R}\}$.

By $\textsc{Assn}$ we designate the language obtained from $\textsc{Assn}^{\mathsf{dt}}$ by: 1) dropping the constants $\mathtt{dt}, \infty$; and 2) replacing the quantifiers $\forall x \in {}^*\mathbb{N}$ and $\forall x \in {}^*\mathbb{R}$ by $\forall x \in \mathbb{N}$ and $\forall x \in \mathbb{R}$, respectively, i.e. by those which range over standard numbers.

Formulas of $\textsc{Assn}^{\mathsf{dt}}$ are the Boolean expressions of $\textsc{While}^{\mathsf{dt}}$, augmented with quantifiers. The quantifier $\forall x \in {}^*\mathbb{N}$ ranging over hyper-*natural* numbers plays an important role in relative completeness of $\textsc{Hoare}^{\mathsf{dt}}$ (Thm. 5.4).

It is essential that in $\textsc{Assn}^{\mathsf{dt}}$ we have only *hyper*quantifiers like $\forall x \in {}^*\mathbb{R}$ and not *standard* quantifiers like $\forall x \in \mathbb{R}$. The situation is much like with the celebrated *transfer principle* in nonstandard analysis [5, Thm. II.4.5]. There the validity of a standard formula φ is transferred to that of its $*$-transform ${}^*\varphi$; and in ${}^*\varphi$ only hyperquantifiers, and no standard quantifiers, are allowed to occur.

Remark 4.2 (Absence of standard quantifiers) The lack of standard quantifiers does restrict the expressive power of $\textsc{Assn}^{\mathsf{dt}}$. Notably we cannot assert that two hypernumbers x, y are infinitely close, that is, $\forall \varepsilon \in \mathbb{R}.\, (\varepsilon > 0 \Rightarrow -\varepsilon < x - y < \varepsilon)$.[3] However this assertion is arguably unrealistic since, to check it against a physical system, one needs measurements of arbitrarily progressive accuracy. The examples in §6 indicate that $\textsc{Assn}^{\mathsf{dt}}$ is sufficiently expressive for practical verification scenarios, too.

Definition 4.3 (Section of $\textsc{Assn}^{\mathsf{dt}}$ expression) Let e be an expression of $\textsc{Assn}^{\mathsf{dt}}$ (arithmetic or a formula), and $i \in \mathbb{N}$. The *i-th section* of e, denoted by $e|_i$, is obtained by: 1) replacing every occurrence of `dt` and ∞ by the constant $\mathtt{c}_{1/(i+1)}$ and $\mathtt{c}_{i+1}$, respectively; and 2) replacing every hyperquantifier $\forall x \in {}^*\mathbb{D}$ by $\forall x \in \mathbb{D}$. Here $\mathbb{D} \in \{\mathbb{N}, \mathbb{R}\}$.

Obviously a section $e|_i$ is an expression of $\textsc{Assn}$.

[3] By replacing $\forall \varepsilon \in \mathbb{R}$ by $\forall \varepsilon \in {}^*\mathbb{R}$ we obtain a legitimate $\textsc{Assn}^{\mathsf{dt}}$ formula, but it is satisfied only when the two hypernumbers x, y are equal.

Definition 4.4 (Semantics of $\textsc{Assn}^{\mathrm{dt}}$) We define the relation $\boldsymbol{\sigma} \models A$ ("$\boldsymbol{\sigma}$ satisfies A") between a hyperstate $\boldsymbol{\sigma} \in \mathbf{HSt}$ and an $\textsc{Assn}^{\mathrm{dt}}$ formula $A \in \mathbf{Fml}$ as usual.

Namely, if $\boldsymbol{\sigma} = \bot$ we define $\bot \models A$ for each $A \in \mathbf{Fml}$. If $\boldsymbol{\sigma} \neq \bot$, the definition is by the following induction on the construction of A.

$$
\begin{array}{lll}
\boldsymbol{\sigma} \models \mathtt{true} & \boldsymbol{\sigma} \not\models \mathtt{false} & \\
\boldsymbol{\sigma} \models A_1 \wedge A_2 & \overset{\text{def.}}{\Longleftrightarrow} & \boldsymbol{\sigma} \models A_1 \ \&\ \boldsymbol{\sigma} \models A_2 \\
\boldsymbol{\sigma} \models \neg A & \overset{\text{def.}}{\Longleftrightarrow} & \boldsymbol{\sigma} \not\models A \\
\boldsymbol{\sigma} \models a_1 < a_2 & \overset{\text{def.}}{\Longleftrightarrow} & [\![a_1]\!]\boldsymbol{\sigma} < [\![a_2]\!]\boldsymbol{\sigma} \qquad \text{where } [\![a_i]\!]\boldsymbol{\sigma} \text{ is as defined in Def. 3.8} \\
\boldsymbol{\sigma} \models \forall x \in {}^*\mathbb{D}.\, A & \overset{\text{def.}}{\Longleftrightarrow} & \boldsymbol{\sigma}[x \mapsto \boldsymbol{d}] \models A \quad \text{for each } \boldsymbol{d} \in {}^*\mathbb{D} \qquad (\mathbb{D} \in \{\mathbb{N}, \mathbb{R}\})
\end{array}
$$

Recall that $\boldsymbol{\sigma}[x \mapsto \boldsymbol{d}]$ denotes an updated hyperstate (Def. 3.6).

An $\textsc{Assn}^{\mathrm{dt}}$ formula $A \in \mathbf{Fml}$ is said to be *valid* if $\boldsymbol{\sigma} \models A$ for any $\boldsymbol{\sigma} \in \mathbf{HSt}$. We denote this by $\models A$. Validity of an $\textsc{Assn}$ formula is defined similarly.

Lemma 4.5 (Sectionwise Satisfaction Lemma) *Let $A \in \mathbf{Fml}$ be an $\textsc{Assn}^{\mathrm{dt}}$ formula; $\boldsymbol{\sigma}$ be a hyperstate; and $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ be an arbitrary sequence representation of $\boldsymbol{\sigma}$. We have*

$$
\boldsymbol{\sigma} \models A \quad \textit{if and only if} \quad \bigl(\boldsymbol{\sigma}|_i \models A|_i \quad \textit{for almost every } i\bigr) \ ,
$$

where the latter relation $\models$ between standard states and $\textsc{Assn}$ formulas is defined in the usual way (i.e. like in Def. 4.4). □

This is our second key lemma. We note that it fails once we allow standard quantifiers in $\textsc{Assn}^{\mathrm{dt}}$. For example, let $A :\equiv (\exists y \in \mathbb{R}.\, 0 < y < x)$ and $\boldsymbol{\sigma}$ be a hyperstate such that $\boldsymbol{\sigma}(x) = \omega^{-1}$. Then we have $\boldsymbol{\sigma}|_i \models A|_i$ for every i but $\boldsymbol{\sigma} \not\models A$.

The validity of an $\textsc{Assn}^{\mathrm{dt}}$ formula A, if A is $(\mathtt{dt}, \infty)$-free, can be reduced to that of an $\textsc{Assn}$ formula. This is the *transfer principle* for $\textsc{Assn}^{\mathrm{dt}}$ which we now describe.

Definition 4.6 ($*$-transform) Let A be an $\textsc{Assn}$ formula. We define its $*$-*transform*, denoted by ${}^*A$, to be the $\textsc{Assn}^{\mathrm{dt}}$ formula obtained from A by replacing every occurrence of a standard quantifier $\forall x \in \mathbb{D}$ by the corresponding hyperquantifier $\forall x \in {}^*\mathbb{D}$.

It is easy to see that: 1) $({}^*A)|_i \equiv A$ for each $\textsc{Assn}$ formula A; 2) $A \equiv {}^*(A|_i)$ for each $\textsc{Assn}^{\mathrm{dt}}$ formula A that is *($\mathtt{dt}$, ∞)-free*—that is, $\mathtt{dt}$ or ∞ does not occur in it. Then the following is an immediate consequence of Lem. 4.5.

Proposition 4.7 (Transfer principle) *1. For each $\textsc{Assn}$ formula A, $\models A$ iff $\models {}^*A$.*
2. For any ($\mathtt{dt}$,∞)-free $\textsc{Assn}^{\mathrm{dt}}$ formula A, the following are equivalent: a) $\models A|_i$ for each $i \in \mathbb{N}$; b) $\models A|_i$ for some $i \in \mathbb{N}$; c) $\models A$. □

5 Program Logic $\textsc{Hoare}^{\mathrm{dt}}$

We now introduce a Hoare-style program logic $\textsc{Hoare}^{\mathrm{dt}}$ that is devised for the verification of $\textsc{While}^{\mathrm{dt}}$ programs. It derives *Hoare triples* $\{A\}c\{B\}$.

Definition 5.1 (Hoare triple) A *Hoare triple* $\{A\}c\{B\}$ of $\textsc{Hoare}^{\mathsf{dt}}$ is a triple of $\textsc{Assn}^{\mathsf{dt}}$ formulas A, B and a $\textsc{While}^{\mathsf{dt}}$ command c.

A Hoare triple $\{A\}c\{B\}$ is said to be *valid*—we denote this by $\models \{A\}c\{B\}$—if, for any hyperstate $\boldsymbol{\sigma} \in \mathbf{HSt}$, $\boldsymbol{\sigma} \models A$ implies $[\![c]\!]\boldsymbol{\sigma} \models B$.

As usual a Hoare triple $\{A\}c\{B\}$ asserts *partial correctness*: if the execution of c starting from $\boldsymbol{\sigma}$ does not terminate, we have $[\![c]\!]\boldsymbol{\sigma} = \bot$ hence trivially $[\![c]\!]\boldsymbol{\sigma} \models B$. The formula A in $\{A\}c\{B\}$ is called a *precondition*; B is a *postcondition*.

The rules of $\textsc{Hoare}^{\mathsf{dt}}$ are the same as usual; see e.g. [12].

Definition 5.2 ($\textsc{Hoare}^{\mathsf{dt}}$) The deduction rules of $\textsc{Hoare}^{\mathsf{dt}}$ are as follows.

$$\frac{}{\{A\}\,\mathtt{skip}\,\{A\}}\ (\textsc{Skip}) \qquad \frac{}{\{\,A[a/x]\,\}\;x := a\,\{A\}}\ (\textsc{Assign})$$

$$\frac{\{A\}\,c_1\,\{C\} \quad \{C\}\,c_2\,\{B\}}{\{A\}\,c_1; c_2\,\{B\}}\ (\textsc{Seq}) \qquad \frac{\{A \wedge b\}\,c_1\,\{B\} \quad \{A \wedge \neg b\}\,c_2\,\{B\}}{\{A\}\,\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\,\{B\}}\ (\textsc{If})$$

$$\frac{\{A \wedge b\}\,c\,\{A\}}{\{A\}\,\mathtt{while}\ b\ \mathtt{do}\ c\,\{A \wedge \neg b\}}\ (\textsc{While}) \qquad \frac{\models A \Rightarrow A' \quad \{A'\}\,c\,\{B'\} \quad \models B' \Rightarrow B}{\{A\}\,c\,\{B\}}\ (\textsc{Conseq})$$

In the rule ($\textsc{Assign}$), $A[a/x]$ denotes the capture-avoiding substitution of a for x in A. Recall that $\mathbf{BExp}$ of $\textsc{While}^{\mathsf{dt}}$ is a fragment of $\mathbf{Fml}$ of $\textsc{Assn}^{\mathsf{dt}}$. Therefore in the rules ($\textsc{If}$) and ($\textsc{While}$), an expression b is an $\textsc{Assn}^{\mathsf{dt}}$ formula.

We write $\vdash \{A\}c\{B\}$ if the triple $\{A\}c\{B\}$ can be derived using the above rules.

Soundness is a minimal requirement of a logic for verification. The proof makes an essential use of the key "sectionwise" lemmas (Lem. 3.10 and Lem. 4.5).

Theorem 5.3 (Soundness) $\vdash \{A\}c\{B\}$ *implies* $\models \{A\}c\{B\}$. □

We also have a "completeness" result. It is called *relative completeness* [4] since completeness is only modulo the validity of $\textsc{Assn}^{\mathsf{dt}}$ formulas (namely those in the ($\textsc{Conseq}$) rule); and checking such validity is easily seen to be undecidable. The proof follows the usual method (see e.g. [12, Chap. 7]); namely via explicit description of weakest preconditions.

Theorem 5.4 (Relative completeness) $\models \{A\}c\{B\}$ *implies* $\vdash \{A\}c\{B\}$. □

6 Verification with $\textsc{Hoare}^{\mathsf{dt}}$

We present a couple of examples. Its details as well as some lemmas that aid finding loop invariants will be presented in another venue, due to the lack of space.

Example 6.1 (Water-level monitor) For the program c_{water} in Ex. 3.4, we would like to prove that the water level y stays between 1 cm and 12 cm. It is not hard to see, after some trials, that what we can actually prove is: $\vdash \{\mathtt{true}\}c_{\mathsf{water}}\{1 - 4 \cdot \mathtt{dt} < y <$

$12 + 2 \cdot \mathtt{dt}\}$. Note that the additional infinitesimal gaps like $4 \cdot \mathtt{dt}$ have no physical meaning. In the proof, we use the following formula A as a loop invariant.

$$\begin{array}{lcl}
A & :\equiv & A_s \wedge A_0 \wedge A_1 \wedge A_2 \wedge A_3 \\
A_s & :\equiv & (s = 0 \vee s = 1) \wedge (v = 0 \vee v = 1) \\
A_0 & :\equiv & s = 1 \wedge v = 1 \quad \Rightarrow \quad 1 - 4 \cdot \mathtt{dt} < y < 10 \\
A_1 & :\equiv & s = 0 \wedge v = 1 \quad \Rightarrow \quad 0 \le x < 2 \quad \wedge \quad 10 \le y < 10 + x + \mathtt{dt} \\
A_2 & :\equiv & s = 0 \wedge v = 0 \quad \Rightarrow \quad 5 < y < 12 + 2 \cdot \mathtt{dt} \\
A_3 & :\equiv & s = 1 \wedge v = 0 \quad \Rightarrow \quad 0 \le x < 2 \quad \wedge \quad 5 - 2x - 2 \cdot \mathtt{dt} < y \le 5
\end{array}$$

Example 6.2 (Train control) Take the program $c_{\mathsf{chkAndBrake}}$ in Ex. 6.2; we aim at the postcondition that the train does not travel beyond the boundary m, that is, $z \le m$. For simplicity let us first consider $c_{\mathsf{constChkAndBrake}} :\equiv (\varepsilon := \mathtt{dt}\,;\ c_{\mathsf{chkAndBrake}})$. This is the setting where the check is conducted constantly. Indeed we can prove that $\vdash \{v^2 \le 2b(z - m)\} c_{\mathsf{constChkAndBrake}} \{z \le m\}$, with a loop invariant $v^2 \le 2b(z - m)$.

The invariant (and the precondition) $v^2 \le 2b(z - m)$ is what is derived in [8] by solving a differential equation and then eliminating quantifiers. Using HOARE$^{\mathtt{dt}}$ we can also derive it: roughly speaking, a differential equation in [8] becomes a recurrence relation in our NSA framework. The details and some general lemmas that aid invariant generation are deferred to another venue.

In the general case where $\varepsilon > 0$ is arbitrary, we can prove $\vdash \{v^2 \le 2b(z - m - v \cdot \varepsilon)\} c_{\mathsf{chkAndBrake}} \{z \le m\}$ in HOARE$^{\mathtt{dt}}$.

An obvious challenge in verification with HOARE$^{\mathtt{dt}}$ is finding loop invariants. It is tempting—especially with "flow-heavy" systems, i.e. those with predominant flow-dynamics—to assert a differential equation's solution as a loop invariant. This does not work: it is a loop invariant only modulo infinitesimal gaps, a fact not expressible in ASSN$^{\mathtt{dt}}$ (Rem. 4.2). We do not consider this as a serious drawback, for two reasons. Firstly, such "flow-heavy" systems could be studied, after all, from the control theory perspective that is continuous in its origin. The formal verification approach is supposed to show its strength against "jump-heavy" systems, for which differential equations are hardly solvable. Secondly, verification goals are rarely as precise as the solution of a differential equation: we would aim at $z \le m$ in Ex. 6.2 but not at $z = \frac{1}{2}at^2$.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theor. Comp. Sci. 138(1), 3–34 (1995)
2. Benveniste, A., Caillaud, B., Pouzet, M.: The fundamentals of hybrid systems modelers. In: Proc. Conference on Decision and Control (CDC). IEEE (2010)
3. Bliudze, S., Krob, D.: Modelling of complex systems: Systems as dataflow machines. Fundam. Inform. 91(2), 251–274 (2009)
4. Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM Journ. Comput. 7(1), 70–90 (1978)
5. Hurd, A.E., Loeb, P.A.: An Introduction to Nonstandard Real Analysis. Academic Press (1985)

6. Khadim, U.: Process Algebras for Hybrid Systems: Comparison and Development. Ph.D. thesis, Technical University Eindhoven (2008)
7. Nakamura, K., Fusaoka, A.: An analysis of the fuller phenomenon on transfinite hybrid automata. In: Majumdar, R., Tabuada, P. (eds.) HSCC. Lect. Notes Comp. Sci., vol. 5469, pp. 450–454. Springer (2009)
8. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reasoning 41(2), 143–189 (2008)
9. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. 20(1), 309–352 (2010)
10. Platzer, A., Quesel, J.D.: KeYmaera: A hybrid theorem prover for hybrid systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR. Lect. Notes Comp. Sci., vol. 5195, pp. 171–178. Springer (2008)
11. Rust, H.: Operational Semantics for Timed Systems: A Non-standard Approach to Uniform Modeling of Timed and Hybrid Systems, Lect. Notes Comp. Sci., vol. 3456. Springer (2005)
12. Winskel, G.: The Formal Semantics of Programming Languages. MIT Press (1993)

A Appendix

A.1 Proofs

Proof of Lem. 2.6 Let $\boldsymbol{d} = [(d_i)_i]$ be an arbitrary element of ${}^*\mathbb{D}$; and for each $j \in [1, n]$, define $S_j := \{i \in \mathbb{N} \mid d_i = a_j\}$. We argue by contradiction. Assume $\boldsymbol{d}$ is distinct from the images of $a_j \in \mathbb{D}$. Then none of S_j belongs to the ultrafilter $\mathcal{F}$; by maximality of $\mathcal{F}$ we have $\mathbb{N} \setminus S_j \in \mathcal{F}$ for each $j \in [1, n]$. Therefore $\bigcap_j (\mathbb{N} \setminus S_j)$ belongs to $\mathcal{F}$; but obviously $\bigcap_j (\mathbb{N} \setminus S_j) = \emptyset$. This cannot be the case since $\mathcal{F}$ is proper. $\square$

Proof of Lem. 3.9 Obvious when $\boldsymbol{\sigma} = \bot$; assume otherwise. Let $(\sigma_i)_i$ and $(\sigma'_i)_i$ be two sequence representations of $\boldsymbol{\sigma}$. We define, for each $i \in \mathbb{N}$

$$\hat{\sigma_i} \;:=\; [\![(\texttt{while}\ b\ \texttt{do}\ c)|_i]\!]\sigma_i \quad \text{and} \quad \hat{\sigma}'_i \;:=\; [\![(\texttt{while}\ b\ \texttt{do}\ c)|_i]\!]\sigma'_i \;\; ;$$

and we are set out to prove $[(\hat{\sigma}_i)_i] = [(\hat{\sigma}'_i)_i]$.

For each $x \in \mathbf{Var}$ we have the equality of hyperreals

$$\big[(\sigma_i(x))_i\big] = \big[(\sigma'_i(x))_i\big] = \boldsymbol{\sigma}(x) \tag{6}$$

by assumption. Therefore for each $x \in \mathbf{Var}$, the set

$$S_x \;:=\; \{i \in \mathbb{N} \mid \sigma_i(x) = \sigma'_i(x)\}$$

belongs to the ultrafilter $\mathcal{F}$. Now let $x_1, \dots, x_n$ be an enumeration of all the variables occurring in b or c; obviously there are only finitely many of such. These are those variables which affect the execution of the command, or are affected by that. Let $S := S_{x_1} \cap \dots \cap S_{x_n}$; then the set S again belongs to $\mathcal{F}$. By the definition of S_x we have:

$$\sigma_i(x_1) = \sigma'_i(x_1)\ , \quad \dots, \quad \sigma_i(x_n) = \sigma'_i(x_n) \qquad \text{for each } i \in S. \tag{7}$$

First consider the case that either $[(\hat{\sigma}_i)_i]$ or $[(\hat{\sigma}'_i)_i]$ is $\bot$. We have

$$\begin{aligned}
& [(\hat{\sigma}_i)_i] = \bot \\
\iff\ & \{i \mid \hat{\sigma}_i = \bot\} \in \mathcal{F} \qquad \text{by Def. 3.7} \\
\iff\ & \{i \in S \mid \hat{\sigma}_i = \bot\} \in \mathcal{F} \qquad \text{since } \mathcal{F} \text{ is a filter and } S \in \mathcal{F} \\
\iff\ & \{i \in S \mid \hat{\sigma}'_i = \bot\} \in \mathcal{F} \qquad \text{since for each } i \in S,\ \hat{\sigma}_i = \bot \text{ iff } \hat{\sigma}'_i = \bot, \text{ by (7)} \\
\iff\ & [(\hat{\sigma}'_i)_i] = \bot \qquad \text{by the same transformation as above.}
\end{aligned}$$

Therefore we have $[(\hat{\sigma}_i)_i] = [(\hat{\sigma}'_i)_i] = \bot$.

Assume otherwise. It suffices to show $[(\hat{\sigma}_i(x))_i] = [(\hat{\sigma}'_i(x))_i]$ for each $x \in \mathbf{Var}$. If $x \notin \{x_1, \dots, x_n\}$ (i.e. if x does not occur in b or c) then obviously $\hat{\sigma}_i(x) = \sigma_i(x)$ and $\hat{\sigma}'_i(x) = \sigma'_i(x)$. Therefore the claim follows from the assumption (6). If $x = x_k$ for some $k \in [1, n]$, for each $i \in S$ we have $\hat{\sigma}_i(x_k) = \hat{\sigma}'_i(x_k)$: this follows immediately from (7), since $x_1, \dots, x_n$ cover all the variables whose value can affect execution of $(\texttt{while}\ b\ \texttt{do}\ c)|_i$. We are done because $S \in \mathcal{F}$. $\square$

Proof of Lem. 3.10 It holds when $\boldsymbol{\sigma} = \bot$, because we have $[\![e|_i]\!]\bot = \bot$. We assume $\boldsymbol{\sigma} \neq \bot$; the proof is by induction on the construction of e.

When $e \equiv x \in \mathbf{Var}$ the claim follows from $\boldsymbol{\sigma}(x) = \big[\, \big((\boldsymbol{\sigma}|_i)(x) \big)_i \,\big]$; the latter holds since $(\boldsymbol{\sigma}|_i)_i$ is a sequence representation of $\boldsymbol{\sigma}$ (Def. 3.7).

When e is a constant $\mathtt{c}_r$, the right-hand side is a sequence that is "almost" $(r, r, \dots)$: its entry is r for almost every i, except for those (negligibly many) i's with $\boldsymbol{\sigma}|_i = \bot$.

When $e \equiv a_1 \;\mathtt{aop}\; a_2$, we have

$$\begin{aligned}
&[\![a_1 \;\mathtt{aop}\; a_2]\!]\boldsymbol{\sigma}\\
&= [\![a_1]\!]\boldsymbol{\sigma} \;\mathtt{aop}\; [\![a_2]\!]\boldsymbol{\sigma} && \text{by the semantics of } \textsc{While}^{\mathtt{dt}}\\
&= \big([\![a_1|_i]\!](\boldsymbol{\sigma}|_i) \big)_{i\in\mathbb{N}} \;\mathtt{aop}\; \big([\![a_2|_i]\!](\boldsymbol{\sigma}|_i) \big)_{i\in\mathbb{N}} && \text{by the induction hypothesis}\\
&= \big([\![a_1|_i]\!](\boldsymbol{\sigma}|_i) \;\mathtt{aop}\; [\![a_2|_i]\!](\boldsymbol{\sigma}|_i) \big)_{i\in\mathbb{N}} && \text{by Def. 2.4}\\
&= \big([\![\, (a_1|_i) \;\mathtt{aop}\; (a_2|_i) \,]\!](\boldsymbol{\sigma}|_i) \big)_{i\in\mathbb{N}} && \text{by the semantics of } \textsc{While}\\
&= \big([\![\, (a_1 \;\mathtt{aop}\; a_2)|_i \,]\!](\boldsymbol{\sigma}|_i) \big)_{i\in\mathbb{N}} \;.
\end{aligned}$$

Here we denoted the hyperstate $\big[\, (\sigma_i)_{i\in\mathbb{N}} \,\big]$ simply by $(\sigma_i)_{i\in\mathbb{N}}$, as we often do elsewhere (cf. Def. 3.7).

When $e \equiv \mathtt{dt}$ the left-hand side is ω^{-1}; and the right-hand side is almost ω^{-1} due to the definition of $\mathtt{dt}|_i$ (Def. 3.2). The same when $e \equiv \infty$.

When e is a Boolean constant `true` or `false`, the proof is the same as when $e \equiv \mathtt{c}_r$. When $e \equiv b_1 \wedge b_2$, the proof is the same as when $e \equiv a_1 \;\mathtt{aop}\; a_2$. So is when $e \equiv \neg b$, and when $e \equiv a_1 < a_2$.

When e is the command `skip` the claim is trivial.

When e is a command $x := a$, we have to show that the hyperstate $[\![x := a]\!]\boldsymbol{\sigma} = \boldsymbol{\sigma}[\, x \mapsto [\![a]\!]\boldsymbol{\sigma} \,]$ coincides with

$$\big([\![(x := a)|_i]\!](\boldsymbol{\sigma}|_i) \big)_{i\in\mathbb{N}} = \big([\![x := (a|_i)]\!](\boldsymbol{\sigma}|_i) \big)_{i\in\mathbb{N}} = \big(\; (\boldsymbol{\sigma}|_i) \left[x \mapsto [\![a|_i]\!](\boldsymbol{\sigma}|_i) \right] \; \big)_{i\in\mathbb{N}} \;.$$

They assign the same value to a variable $y \neq x$, since $(\boldsymbol{\sigma}|_i)_i$ is a sequence representation of $\boldsymbol{\sigma}$. To the variable x they assign $[\![a]\!]\boldsymbol{\sigma}$ and $\big([\![a|_i]\!](\boldsymbol{\sigma}|_i) \big)_{i\in\mathbb{N}}$, respectively; these are the same hyperreal due to the induction hypothesis.

When e is a command $c_1; c_2$, we have

$$\begin{aligned}
&[\![e]\!]\boldsymbol{\sigma} = [\![c_2]\!]([\![c_1]\!]\boldsymbol{\sigma})\\
&= [\![c_2]\!]\Big[\Big([\![c_1|_i]\!](\boldsymbol{\sigma}|_i) \Big)_{i\in\mathbb{N}} \Big] && \text{by the induction hypothesis}\\
&= \Big[\Big([\![c_2|_i]\!]\big([\![c_1|_i]\!](\boldsymbol{\sigma}|_i) \big) \Big)_{i\in\mathbb{N}} \Big] && \text{by the induction hypothesis, } (*)\\
&= \Big[\Big(\; [\![(c_1|_i); (c_2|_i)]\!](\boldsymbol{\sigma}|_i) \; \Big)_{i\in\mathbb{N}} \Big] = \Big[\Big(\; [\![(c_1; c_2)|_i]\!](\boldsymbol{\sigma}|_i) \; \Big)_{i\in\mathbb{N}} \Big] \;.
\end{aligned}$$

In $(*)$, we applied the induction hypothesis to an (obvious) sequence representation $\Big([\![c_1|_i]\!](\boldsymbol{\sigma}|_i) \Big)_{i\in\mathbb{N}}$ of the hyperstate $\Big[\Big([\![c_1|_i]\!](\boldsymbol{\sigma}|_i) \Big)_{i\in\mathbb{N}} \Big]$.

When e is a command `if` b `then` c_1 `else` c_2, assume first that $[\![b]\!]\boldsymbol{\sigma} = \mathtt{tt}$. Then by the induction hypothesis $[\![b|_i]\!](\boldsymbol{\sigma}|_i) = \mathtt{tt}$ for almost every i. Therefore we have

$[\![e|_i]\!](\boldsymbol{\sigma}|_i) = [\![c_1|_i]\!](\boldsymbol{\sigma}|_i)$ for almost every i. We use this fact in the following (†).

$$\begin{aligned}
[\![e]\!]\boldsymbol{\sigma} &= [\![c_1]\!]\boldsymbol{\sigma} && \text{since } [\![b]\!]\boldsymbol{\sigma} = \mathtt{tt} \\
&= \Big[\Big(\, [\![c_1|_i]\!](\boldsymbol{\sigma}|_i)\,\Big)_{i\in\mathbb{N}}\Big] && \text{by the induction hypothesis} \\
&\overset{(\dagger)}{=} \Big[\Big(\, [\![e|_i]\!](\boldsymbol{\sigma}|_i)\,\Big)_{i\in\mathbb{N}}\Big] \enspace .
\end{aligned}$$

The case where $[\![b]\!]\boldsymbol{\sigma} = \mathtt{ff}$ is similar.

Finally, when e is a command `while` b `do` c the claim is the definition (4) itself. This concludes the proof. $\square$

Proof of Lem. 4.5 Obvious when $\boldsymbol{\sigma} = \bot$. Assume $\boldsymbol{\sigma} \neq \bot$; the proof is by induction on the construction of A. In fact most of the cases are the same as in the proof of Lem. 3.10—more specifically the cases for the Boolean expressions therein. The remaining case is when $A \equiv \forall x \in {}^*\mathbb{D}.\, A'$, with $\mathbb{D} \in \{\mathbb{N}, \mathbb{R}\}$.

For the 'if' part, assume that for almost every i we have $\boldsymbol{\sigma}|_i \models (\forall x \in {}^*\mathbb{D}.\, A')|_i$, that is, $\boldsymbol{\sigma}|_i \models \forall x \in \mathbb{D}.\, (A'|_i)$. By the semantics of Assn we have

$$\begin{aligned}
&\text{"for almost every } i\text{:}\quad \text{for all } d \in \mathbb{D}\text{:}\ (\boldsymbol{\sigma}|_i)[x \mapsto d] \models A'|_i\text{,"}\quad \text{that is,} \\
&\big\{\, i \in \mathbb{N} \;\big|\; \text{for all } d \in \mathbb{D},\ (\boldsymbol{\sigma}|_i)[x \mapsto d] \models A'|_i \,\big\} \in \mathcal{F} \enspace .
\end{aligned} \tag{8}$$

We note that this is *not* equivalent to

$$\begin{aligned}
&\text{"for all } d \in \mathbb{D}\text{:}\quad \text{for almost every } i\text{:}\ (\boldsymbol{\sigma}|_i)[x \mapsto d] \models A'|_i\text{,"}\quad \text{that is,} \\
&\big\{\, i \in \mathbb{N} \;\big|\; (\boldsymbol{\sigma}|_i)[x \mapsto d] \models A'|_i \,\big\} \in \mathcal{F} \quad \text{for all } d \in \mathbb{D}\text{:}
\end{aligned}$$

the former implies the latter but the converse fails since $\mathcal{F}$ need not be closed under infinite intersection.

Anyway take an arbitrary $\boldsymbol{d} \in {}^*\mathbb{D}$; and let $(\boldsymbol{d}|_i)_{i\in\mathbb{N}}$ be its arbitrary sequence representation. By (8) we have $(\boldsymbol{\sigma}|_i)\big[x \mapsto (\boldsymbol{d}|_i)\big] \models A'|_i$ for almost every i. Since $\big((\boldsymbol{\sigma}|_i)\big[x \mapsto (\boldsymbol{d}|_i)\big]\big)_{i\in\mathbb{N}}$ is a sequence representation of a hyperstate $\boldsymbol{\sigma}[x \mapsto \boldsymbol{d}]$, we can use the induction hypothesis and conclude $\boldsymbol{\sigma}[x \mapsto \boldsymbol{d}] \models A'$. This holds for an arbitrary $\boldsymbol{d} \in {}^*\mathbb{D}$; thus by Def. 4.4 we have $\boldsymbol{\sigma} \models \forall x \in {}^*\mathbb{D}.\, A'$.

For the 'only if' part, we prove its contraposition. Assume

$$\big\{\, i \in \mathbb{N} \;\big|\; \boldsymbol{\sigma}|_i \models (\forall x \in {}^*\mathbb{D}.\, A')|_i \,\big\} \notin \mathcal{F} \enspace .$$

Then we have

$$\begin{aligned}
&\big\{\, i \in \mathbb{N} \;\big|\; \boldsymbol{\sigma}|_i \not\models (\forall x \in {}^*\mathbb{D}.\, A')|_i \,\big\} \in \mathcal{F} && \text{since } \mathcal{F} \text{ is an ultrafilter;}\quad \text{thus} \\
&S := \big\{\, i \in \mathbb{N} \;\big|\; \text{for some } d \in \mathbb{D},\ (\boldsymbol{\sigma}|_i)[x \mapsto d] \not\models A'|_i \,\big\} \in \mathcal{F} && \text{by the semantics of Assn.}
\end{aligned}$$

For each $i \in S$, let us choose $d_i \in \mathbb{D}$ so that $(\boldsymbol{\sigma}|_i)[x \mapsto d_i] \not\models A'|_i$. This is possible due to the Axiom of Choice. Set $d_i := 0$ for each $i \in \mathbb{N} \setminus S$; and let $\boldsymbol{d} := \big[(d_i)_{i\in\mathbb{N}}\big]$. Then $\big((\boldsymbol{\sigma}|_i)[x \mapsto d_i]\big)_{i\in\mathbb{N}}$ is a sequence representation of a hyperstate $\boldsymbol{\sigma}[x \mapsto \boldsymbol{d}]$. Since $(\boldsymbol{\sigma}|_i)[x \mapsto d_i] \not\models A'|_i$ for each $i \in S \in \mathcal{F}$, by the induction hypothesis we have $\boldsymbol{\sigma}[x \mapsto \boldsymbol{d}] \not\models A'$. Therefore $\boldsymbol{\sigma} \not\models \forall x \in {}^*\mathbb{D}.\, A'$. $\square$

Proof of Prop. 4.7

Lemma A.1 *1. For each* $\textsc{Assn}$ *formula A and $i \in \mathbb{N}$, we have $({}^*A)|_i \equiv A$.*
2. For each $\textsc{Assn}^{\mathtt{dt}}$ *formula A that is* $(\mathtt{dt}, \infty)$*-free—i.e.* $\mathtt{dt}$ *or* ∞ *does not occur in it—we have $A|_i \equiv A|_j$ for each $i, j \in \mathbb{N}$. Moreover $A \equiv {}^*(A|_i)$.* □

Proof. (Of Prop. 4.7) For the 'if' part of the item 1., for an arbitrary state $\sigma \in \mathbf{St}$ take $\boldsymbol{\sigma} := \big[(\sigma)_{i\in\mathbb{N}}\big]$. By the assumption we have $\boldsymbol{\sigma} \models {}^*A$. Therefore by Lem. 4.5 we have $\sigma \models ({}^*A)|_i$; since $A \equiv ({}^*A)|_i$ we have shown the claim.

For the 'only if' part of the item 1., let $\boldsymbol{\sigma} \in \mathbf{HSt}$ be an arbitrary hyperstate. Choose an arbitrary sequence representation $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ of $\boldsymbol{\sigma}$; by the assumption we have $\boldsymbol{\sigma}|_i \models A$, that is, $\boldsymbol{\sigma}|_i \models ({}^*A)|_i$. This is for any $i \in \mathbb{N}$; we are done due to Lem. 4.5.

The item 2. follows immediately from Lem. A.1 and the item 1. □

Proof of Thm. 5.3 We need the following lemma that relates substitution and state update (Def. 3.6).

Lemma A.2 *Let $a \in \mathbf{AExp}$ be an arbitrary arithmetic expression; and $\boldsymbol{\sigma} \in \mathbf{HSt}$.*

1. *For any $a' \in \mathbf{AExp}$ we have: $[\![\, a'[a/x] \,]\!]\boldsymbol{\sigma} = [\![a']\!]\big(\,\boldsymbol{\sigma}[x \mapsto [\![a]\!]\boldsymbol{\sigma}]\,\big)$.*
2. *For any $A \in \mathbf{Fml}$ we have: $\boldsymbol{\sigma} \models A[a/x]$ if and only if $\boldsymbol{\sigma}[x \mapsto [\![a]\!]\boldsymbol{\sigma}] \models A$.*

Proof. Both items are proved by induction on the complexity of expressions a' and A.[4] It is straightforward; the presence of $\mathtt{dt}$ or ∞ does not change the proofs at all. Nevertheless we shall describe some details.

The item 1. is easy. For the item 2., the cases where A is $\mathtt{true}$, $\mathtt{false}$, $A_1 \wedge A_2$ or $\neg A'$ are obvious. When A is $a_1 < a_2$ we use the item 1. Assume A is $\forall y \in {}^*\mathbb{D}.\, A'$. The left-hand side is

$$\boldsymbol{\sigma} \models (\forall y \in {}^*\mathbb{D}.\, A')[a/x] \ \ ;$$

we shall successively transform this into equivalent conditions. First, it is equivalent to

$$\boldsymbol{\sigma} \models \forall y' \in {}^*\mathbb{D}.\, (A'[y'/y][a/x]) \qquad \text{where } y' \in \mathbf{Var} \text{ is fresh,}$$

by the definition of capture-avoiding substitution. That is,

$$\boldsymbol{\sigma}[y' \mapsto \boldsymbol{d}] \models A'[y'/y][a/x] \qquad \text{for each } \boldsymbol{d} \in {}^*\mathbb{D}.$$

Now the formula $A'[y'/y]$ has smaller complexity than $A \equiv \forall y \in {}^*\mathbb{D}.\, A'$; hence by the induction hypothesis, the above is equivalent to

$$\big(\,\boldsymbol{\sigma}[y' \mapsto \boldsymbol{d}]\,\big)\big[\,x \mapsto [\![a]\!](\boldsymbol{\sigma}[y' \mapsto \boldsymbol{d}])\,\big] \models A'[y'/y] \qquad \text{for each } \boldsymbol{d} \in {}^*\mathbb{D}.$$

We have $[\![a]\!](\boldsymbol{\sigma}[y' \mapsto \boldsymbol{d}]) = [\![a]\!]\boldsymbol{\sigma}$ since y' does not occur in a; and $\big(\,\boldsymbol{\sigma}[y' \mapsto \boldsymbol{d}]\,\big)[x \mapsto [\![a]\!]\boldsymbol{\sigma}] = \big(\,\boldsymbol{\sigma}[x \mapsto [\![a]\!]\boldsymbol{\sigma}]\,\big)[y' \mapsto \boldsymbol{d}]$ since $x \neq y'$. Therefore the above is equivalent to

$$\big(\,\boldsymbol{\sigma}[x \mapsto [\![a]\!]\boldsymbol{\sigma}]\,\big)[y' \mapsto \boldsymbol{d}] \models A'[y'/y] \qquad \text{for each } \boldsymbol{d} \in {}^*\mathbb{D},$$

[4] The induction is not strictly on the "construction" of expressions but on their complexity; the latter is, say, measured by the number of connectives occurring in the expressions.

that is,

$$\boldsymbol{\sigma}[x \mapsto [\![a]\!]\boldsymbol{\sigma}] \models \forall y' \in {}^*\mathbb{D}.\, A'[y'/y] \ .$$

This is equivalent to the right-hand side of the claim because the α-equivalent formulas $\forall y' \in {}^*\mathbb{D}.\, A'[y'/y]$ and $\forall y \in {}^*\mathbb{D}.\, A'$ obviously have the same semantics. □

Proof. (Of Thm. 5.3) Assume $\vdash \{A\}c\{B\}$. We show that, for each $\boldsymbol{\sigma} \neq \bot$, we have that $\boldsymbol{\sigma} \models A$ implies $[\![c]\!]\boldsymbol{\sigma} \models B$ (if $\boldsymbol{\sigma} = \bot$ it is obvious). This is by induction on the derivation of $\vdash \{A\}c\{B\}$.

If the last rule is (SKIP), we have $[\![\texttt{skip}]\!]\boldsymbol{\sigma} = \boldsymbol{\sigma}$ hence the claim is obvious.

If the last rule is (ASSIGN), our assumption is that $\boldsymbol{\sigma} \models A[a/x]$. This is equivalent to $\boldsymbol{\sigma}[x \mapsto [\![a]\!]\boldsymbol{\sigma}] \models A$ by Lem. A.2; and to $[\![x := a]\!]\boldsymbol{\sigma} \models A$ by the definition of $[\![x := a]\!]$ (Def. 3.8).

If the last rule is (SEQ), from $\boldsymbol{\sigma} \models A$ it immediately follows that $[\![c_2]\!]([\![c_1]\!]\boldsymbol{\sigma}) \models B$ using the induction hypothesis. We are done since $[\![c_1; c_2]\!]\boldsymbol{\sigma} = [\![c_2]\!]([\![c_1]\!]\boldsymbol{\sigma})$.

If the last rule is (IF), the proof is by cases. Assume $\boldsymbol{\sigma} \models A$. If $\boldsymbol{\sigma} \models b$, then $\boldsymbol{\sigma} \models A \wedge b$ and by the induction hypothesis we have $[\![c_1]\!]\boldsymbol{\sigma} \models B$. In this case we furthermore have $[\![\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2]\!]\boldsymbol{\sigma} = [\![c_1]\!]\boldsymbol{\sigma}$ and we are done. The other case where $\boldsymbol{\sigma} \not\models b$ is the same.

If the last rule is (WHILE), we argue by contradiction. Assume that there is $\boldsymbol{\sigma} \in \mathbf{HSt}$ such that $\boldsymbol{\sigma} \models A$ but $[\![\texttt{while}\ b\ \texttt{do}\ c]\!]\boldsymbol{\sigma} \not\models A \wedge \neg b$. Let $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ be an arbitrary sequence representation of $\boldsymbol{\sigma}$; by Lem. 3.10 and Lem. 4.5 we have

$$\{i \mid \boldsymbol{\sigma}|_i \models A|_i\} \in \mathcal{F} \quad \text{and} \quad \left\{ i \;\middle|\; [\![\texttt{while}\ b|_i\ \texttt{do}\ c|_i]\!](\boldsymbol{\sigma}|_i) \not\models A|_i \wedge \neg\, b|_i \right\} \in \mathcal{F} \ .$$

Therefore their intersection S belongs to $\mathcal{F}$:

$$S \ := \ \left\{ i \;\middle|\; \boldsymbol{\sigma}|_i \models A|_i \quad \& \quad [\![\texttt{while}\ b|_i\ \texttt{do}\ c|_i]\!](\boldsymbol{\sigma}|_i) \not\models A|_i \wedge \neg\, b|_i \right\} \quad \in \mathcal{F} \ . \qquad (9)$$

For each $i \in S$, the (standard) state $[\![\texttt{while}\ b|_i\ \texttt{do}\ c|_i]\!](\boldsymbol{\sigma}|_i)$ is not $\bot$; otherwise it would satisfy any formula. Therefore by the semantics of standard `while` loops (5), there exist $N_i \in \mathbb{N}$ and a sequence $\sigma_{i,0}, \sigma_{i,1}, \dots, \sigma_{i,N_i}$ such that

- $\sigma_{i,0} = \boldsymbol{\sigma}|_i$ and $\sigma_{i,N_i} = [\![\texttt{while}\ b|_i\ \texttt{do}\ c|_i]\!](\boldsymbol{\sigma}|_i)$;
- $[\![b|_i]\!]\sigma_{i,N_i} = \mathrm{ff}$, and $[\![b|_i]\!]\sigma_{i,j} = \mathrm{tt}$ for each $j \in [0, N_i)$; and
- $\sigma_{i,j+1} = [\![c|_i]\!]\sigma_{i,j}$ for each $j \in [0, N_i)$.

Then we have

$$\sigma_{i,0} \models A|_i \quad \text{and} \quad \sigma_{i,N_i} \not\models A|_i \ . \qquad (10)$$

The former is because $\sigma_{i,0} = \boldsymbol{\sigma}|_i$ and $i \in S$ (see (9)); the latter is because $\sigma_{i,N_i} \not\models A|_i \wedge \neg b|_i$ (by (9)) and that $\sigma_{i,N_i} \models \neg b|_i$. By (10), there necessarily exists a natural number $k_i \in [0, N_i)$ such that $\sigma_{i,k_i} \models A|_i$ and $\sigma_{i,k_i+1} \not\models A|_i$. Note also that $\sigma_{i,k_i} \models b|_i$ since $k_i \in [0, N_i)$. We set $\sigma'_i := \sigma_{i,k_i}$; summing up its properties we have

$$\sigma'_i \models A|_i \ , \quad \sigma'_i \models b|_i \quad \text{and} \quad [\![c|_i]\!]\sigma'_i \not\models A|_i \ . \qquad (11)$$

Thus we have found, for each $i \in S$, a state σ'_i such that (11) holds. For each $i \in \mathbb{N} \setminus S$ fix σ'_i to be, say, $\bot$; and define a hyperstate $\boldsymbol{\sigma}' := \left[(\sigma'_i)_{i\in\mathbb{N}}\right]$. We have $\boldsymbol{\sigma}' \models A \wedge b$ by Lem. 4.5; and $[\![c]\!]\boldsymbol{\sigma}' \not\models A$ by Lem. 3.10 and Lem. 4.5. This contradicts the induction hypothesis that $\models \{A \wedge b\}c\{A\}$.

The case where the last rule is (CONSEQ) is obvious. This concludes the proof. □

Proof of Thm. 5.4 Our strategy is as in [12], that is, via explicit description of weakest preconditions in $\text{A}\textsc{ssn}^{\mathsf{dt}}$. The weakest precondition for a `while` clause calls for an encoding of a sequence $x_0, \ldots, x_k$ of values (with an arbitrary length $k+1$) by a fixed number of values. In the usual setting where values are integers, one would use Gödel's β function for this purpose. The following formulas are adaptation of those in [12, Chap. 7].

Definition A.3 We introduce the following notational conventions for $\text{A}\textsc{ssn}$.

$$\begin{array}{lll}
\mathsf{isNat}(x) & :\equiv & \exists y \in \mathbb{N}.\, x = y \\
\mathsf{isInt}(x) & :\equiv & \mathsf{isNat}(x) \lor \mathsf{isNat}(-x) \\
x = y \bmod z & :\equiv & \mathsf{isNat}(x) \land \mathsf{isNat}(y) \land \mathsf{isNat}(z) \land \\
 & & \exists u \in \mathbb{N}. \big(u \cdot z \le y \ \land \ y < (u+1) \cdot z \ \land \ x = y - u \cdot z \big) \\
\beta(x,y,z,u) & :\equiv & u = x \bmod (1 + (1+z) \cdot y) \\
F(u,v) & :\equiv & \mathsf{isNat}(u) \land \exists z \in \mathbb{N}. \\
 & & \big((u = 2 \cdot z \ \Rightarrow \ v = z) \land (u = 2 \cdot z + 1 \ \Rightarrow \ v = -z) \big) \\
\beta^{\pm}(x,y,z,v) & :\equiv & \exists u \in \mathbb{N}. \big(\beta(x,y,z,u) \ \land \ F(u,v) \big)
\end{array}$$

The formula $x = y \bmod z$ reads: x is the remainder when y is divided by z. One easily sees that it implies $0 \le x < z$.

Lemma A.4 *Given any sequence $n_0, \ldots, n_k$ of integers (possibly negative, of any length k), there exist two natural numbers n, m such that: for each natural number $j \in [0, k]$, the* $\text{A}\textsc{ssn}$ *formula $\beta^{\pm}(n,m,j,x) \Leftrightarrow x = n_j$ is valid.*

Thus the formula $\beta^{\pm}$ allows us to encode a sequence of integers by two natural numbers.

Proof. The proof uses the Chinese Remainder Theorem; see [12, Lem. 7.4]. $\square$

In the current work we will also need to encode a sequence $x_0, \ldots, x_k$ of real numbers. For that purpose we use the following formulas.

Definition A.5 We introduce the following notational conventions for $\text{A}\textsc{ssn}$.

$$\begin{array}{lll}
\mathsf{intPart}(x,a) & :\equiv & \mathsf{isInt}(a) \ \land \ 0 \le x - a < 1 \\
\mathsf{fracPart}(x,z) & :\equiv & \exists a \in \mathbb{R}. \big(\mathsf{intPart}(x,a) \ \land \ x = a + z \big) \\
\mathsf{digit}(x,a,b) & :\equiv & \mathsf{isNat}(a) \ \land \ \mathsf{isNat}(b) \ \land \ 0 \le x < 1 \ \land \ \exists c_1, c_2 \in \mathbb{N}. \\
 & & \big(\mathsf{intPart}(x \cdot 2^a, c_1) \ \land \ \mathsf{intPart}(x \cdot 2^{a+1}, c_2) \ \land \ c_2 = 2 \cdot c_1 + b \big) \\
\gamma(x,a,b,y) & :\equiv & \mathsf{isNat}(a) \ \land \ \mathsf{isNat}(b) \ \land \ \forall c \in \mathbb{N}. \forall z \in \mathbb{R}. \\
 & & \big(\mathsf{digit}(x, c \cdot (a+1) + b, z) \Leftrightarrow \mathsf{digit}(y,c,z) \big) \\
\rho(a,b,x,c,d,y) & :\equiv & \exists e \in \mathbb{R}. \exists z \in \mathbb{R}. \begin{pmatrix} \mathsf{intPart}(y,e) \ \land \ \beta^{\pm}(a,b,d,e) \land \\ \mathsf{fracPart}(y,z) \ \land \ \gamma(x,c,d,z) \end{pmatrix}
\end{array}$$

The formula $\mathsf{digit}(x,a,b)$ means: in the eager binary representation of a real number $x \in [0,1)$, the a-th digit is b.[5] Here "eager" means that, for example, the number $1/2$ is represented by $.1000\ldots$ rather than by $.0111\ldots$. Consult the following lemma for the intention of the formulas γ and ρ.

[5] Convention: a binary representation $.d_0 d_1 \ldots$ of a real number $x \in [0,1)$ starts with the "0-th" digit d_0, not with the "first."

Lemma A.6 *1. Let $s_0, \dots, s_k$ be an arbitrary sequence of real numbers in $[0,1)$, of an arbitrary length k. There exists a real number $s \in [0,1)$ such that: for each natural number $j \in [0,k]$, the* ASSN *formula $\gamma(s,k,j,x) \Leftrightarrow x = s_j$ is valid.*

2. Let $r_0, \dots, r_k$ be an arbitrary sequence of real numbers. There exists natural numbers $n, m \in \mathbb{N}$ and a real number $s \in [0,1)$ such that: for each natural number $j \in [0,k]$, the ASSN *formula $\rho(n,m,s,k,j,x) \Leftrightarrow x = r_j$ is valid.*

Proof. 1. Let $.d_{j0}d_{j1}d_{j2}\cdots$ be the eager binary representation of s_j. Let a real number s to be the one represented by $.(d_{00}d_{10}\dots d_{k0})(d_{01}d_{11}\dots d_{k1})(d_{02}d_{12}\dots d_{k2})\dots$, where we put parentheses for the sake of readability. Therein the digit d_{jl} occurs as the $(l \cdot (k+1) + j)$-th digit. It is straightforward to see that this s makes the formula $\gamma(s,k,j,x) \Leftrightarrow x = s_j$ valid.

2. For each $j \in [0,k]$, let n_j be the integer part of r_j; and s_j be the fractional part. Then there exist $n, m \in \mathbb{N}$ that encode $n_0, \dots, n_k$ in the sense of Lem. A.4; and $s \in [0,1)$ that encode $s_0, \dots, s_k$ in the sense of the item 1. Obviously these numbers n, m, s qualify as the ones required in the claim. $\square$

Using (the $*$-transforms of) these shorthands, we explicitly define an ASSN$^{\mathsf{dt}}$ formula $w[\![c,B]\!]$. It is meant to be a weakest precondition for c and B. The definition is much like in [12, Chap. 7].

Definition A.7 ($w[\![c,B]\!]$) For each WHILE$^{\mathsf{dt}}$ command $c \in \mathbf{Cmd}$ and an ASSN$^{\mathsf{dt}}$ formula $B \in \mathbf{Fml}$, we define an ASSN$^{\mathsf{dt}}$ formula $w[\![c,B]\!]$ by the following induction on c.

$$\begin{array}{ll}
w[\![\mathtt{skip}, B]\!] & :\equiv\ B \\
w[\![x := a, B]\!] & :\equiv\ B[a/x] \\
w[\![c_1; c_2, B]\!] & :\equiv\ w[\![\, c_1,\ w[\![c_2, B]\!]\,]\!] \\
w[\![\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2, B]\!] & :\equiv\ \big(\, b \Rightarrow w[\![c_1, B]\!]\,\big) \ \wedge\ \big(\, \neg b \Rightarrow w[\![c_2, B]\!]\,\big) \\
w[\![\mathtt{while}\ b\ \mathtt{do}\ c, B]\!] & :\equiv\ \forall a, b, d \in {}^*\mathbb{N}.\ \forall y \in {}^*\mathbb{R}.
\end{array}$$

$$\left[\begin{array}{l}
\qquad \boxed{\text{“}\mathsf{seq}(a,b,y,d)_0 = \boldsymbol{\sigma}\text{”, (a)}} \wedge \\
\qquad \forall e \in {}^*\mathbb{N}.\ \left(\begin{array}{l}
0 \le e < d \Longrightarrow \\
\qquad \boxed{\text{“}\mathsf{seq}(a,b,y,d)_e \models b\text{”, (b)}} \wedge \\
\qquad \boxed{\text{“}\mathsf{seq}(a,b,y,d)_{e+1} = [\![c]\!]\big(\mathsf{seq}(a,b,y,d)_e\big)\text{”, (c)}}
\end{array}\right) \\
\Longrightarrow \ \boxed{\text{“}\mathsf{seq}(a,b,y,d)_d \models b \vee B\text{”, (d)}}
\end{array}\right]$$

Here idea for the formulas (a–d) are as follows. We let a, b, d and y to encode a sequence of hyperstates (more precisely, the values stored for the relevant variables). The formula (a), for example, asserts that the first in the sequence coincides with the current state $\boldsymbol{\sigma}$.

The “relevant variables” are those which occur in b, c or B. Assume first, for simplicity, that x is the only such. Then the formulas (a–d) are concretely:

$$\begin{array}{ll}
\text{(a)} & {}^*\rho(a,b,y,d,0,x) \\
\text{(b)} & \forall z \in {}^*\mathbb{R}.\ \big(\, {}^*\rho(a,b,y,d,e,z) \Longrightarrow b[z/x]\,\big) \\
\text{(c)} & \forall z, u \in {}^*\mathbb{R}.\ \left[\begin{array}{l} {}^*\rho(a,b,y,d,e,z) \wedge {}^*\rho(a,b,y,d,e+1,u) \ \Longrightarrow \\ \big(\, w[\![c, x = u]\!] \ \wedge\ \neg\, w[\![c, \mathtt{false}]\!]\,\big)[z/x] \end{array}\right] \\
\text{(d)} & \forall z \in {}^*\mathbb{R}.\ \big(\, {}^*\rho(a,b,y,d,d,z) \Longrightarrow (b \vee B)[z/x]\,\big)
\end{array}$$

where ${}^*\rho$ is the $*$-transform (Def. 4.6) of ρ in Def. A.5.

In the general case, let $x_0, \ldots, x_l$ be an enumeration of those variables which occur in b, c or B. The formulas (a–d) are concretely as follows. Notice that l is a fixed number that is known statically from b, c and B; hence despite the presence of ... below, they are concrete ASSN$^{\mathsf{dt}}$ formulas.

(a) ${}^*\rho(a,b,y,d\cdot(l+1)+l,0,x_0) \;\wedge\cdots\wedge\; {}^*\rho(a,b,y,d\cdot(l+1)+l,l,x_l)$

(b) $\forall z_0,\ldots,z_l \in {}^*\mathbb{R}.$
$$\left[\begin{array}{l} \quad {}^*\rho(a,b,y,d\cdot(l+1)+l,e\cdot(l+1),z_0) \\ \quad \wedge\;\cdots \\ \quad \wedge\;{}^*\rho(a,b,y,d\cdot(l+1)+l,e\cdot(l+1)+l,z_l) \\ \Longrightarrow b[z_0/x_0,\ldots,z_l/x_l] \end{array}\right]$$

(c) $\forall z_0,\ldots,z_l,u_0,\ldots,u_l \in {}^*\mathbb{R}.$
$$\left[\begin{array}{l} \quad {}^*\rho(a,b,y,d\cdot(l+1)+l,e\cdot(l+1),z_0) \\ \quad \wedge\;\cdots \\ \quad \wedge\;{}^*\rho(a,b,y,d\cdot(l+1)+l,e\cdot(l+1)+l,z_l) \\ \quad \wedge\;{}^*\rho(a,b,y,d\cdot(l+1)+l,(e+1)\cdot(l+1),u_0) \\ \quad \wedge\;\cdots \\ \quad \wedge\;{}^*\rho(a,b,y,d\cdot(l+1)+l,(e+1)\cdot(l+1)+l,u_l) \\ \Longrightarrow \big(\,w[\![c,\ x_0=u_0\wedge\cdots\wedge x_l=u_l]\!]\;\wedge\;\neg\, w[\![c,\mathtt{false}]\!]\,\big)[z_0/x_0,\ldots,z_l/x_l] \end{array}\right]$$

(d) $\forall z_0,\ldots,z_l \in {}^*\mathbb{R}.$
$$\left[\begin{array}{l} \quad {}^*\rho(a,b,y,d\cdot(l+1)+l,d\cdot(l+1),z_0) \\ \quad \wedge\;\cdots \\ \quad \wedge\;{}^*\rho(a,b,y,d\cdot(l+1)+l,d\cdot(l+1)+l,z_l) \\ \Longrightarrow (b\vee B)[z_0/x_0,\ldots,z_l/x_l] \end{array}\right]$$

It will be shown that $w[\![c,B]\!]$ is indeed a weakest precondition (Prop. A.9). For that we need the following lemma.

Lemma A.8 (Expressivity of ASSN) *Let c be a* WHILE$^{\mathsf{dt}}$ *command, and B be an* ASSN$^{\mathsf{dt}}$ *formula. Then for each $\sigma \in \mathbf{St}$ and $i \in \mathbb{N}$,*

$$[\![c|_i]\!]\sigma \models B|_i \quad \textit{if and only if} \quad \sigma \models \big(\,w[\![c,B]\!]\,\big)|_i \ .$$

From this it follows that: for each ASSN *formula A,*

$$\models \{A\}\,c|_i\,\{B|_i\} \quad \textit{if and only if} \quad \models A \Rightarrow \big(\,w[\![c,B]\!]\,\big)|_i \ .$$

Proof. One easily sees that the formula $\big(\,w[\![c,B]\!]\,\big)|_i$ coincides with the usual definition of a weakest precondition $w[\![c|_i, B|_i]\!]$; see e.g. [12]. The different definition for the `while` clauses is due to the presence of real values stored in states, and nothing more. □

Proposition A.9 (Expressivity of ASSN$^{\mathsf{dt}}$) *For any* WHILE$^{\mathsf{dt}}$ *command c,* ASSN$^{\mathsf{dt}}$ *formula B and $\boldsymbol{\sigma} \in \mathbf{HSt}$, we have*

$$[\![c]\!]\boldsymbol{\sigma} \models B \quad \textit{if and only if} \quad \boldsymbol{\sigma} \models w[\![c,B]\!] \ .$$

From this it immediately follows that: for each ASSN$^{\mathrm{dt}}$ *formula A,*

$$\models \{A\}\, c\, \{B\} \quad \textit{if and only if} \quad \models A \Rightarrow w[\![c, B]\!] \ .$$

Proof. The proof is by induction on the construction of c.

If c is `skip` the claim is obvious since $[\![\mathtt{skip}]\!]\boldsymbol{\sigma} = \boldsymbol{\sigma}$ and $w[\![\mathtt{skip}, B]\!] = B$.

If c is $x := a$ the claim follows from Lem. A.2.

If c is $c_1; c_2$ or $\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2$, the claim is easy using the induction hypothesis.

Finally, for the case where c is $\mathtt{while}\ b\ \mathtt{do}\ c'$, we use Lem. 4.5 and Lem. A.8. Fix a sequence representation $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ of $\boldsymbol{\sigma}$. We have:

$$\begin{array}{lll} [\![\mathtt{while}\ b\ \mathtt{do}\ c']\!]\boldsymbol{\sigma} \models B & & \\ \iff \quad [\![\mathtt{while}\ b|_i\ \mathtt{do}\ c'|_i]\!](\boldsymbol{\sigma}|_i) \models B|_i \quad \text{for almost every } i, & & \text{by Lem. 4.5} \\ \iff \quad \boldsymbol{\sigma}|_i \models \big(w[\![\mathtt{while}\ b\ \mathtt{do}\ c', B]\!]\big)|_i \quad \text{for almost every } i, & & \text{by Lem. A.8} \\ \iff \quad \boldsymbol{\sigma} \models w[\![\mathtt{while}\ b\ \mathtt{do}\ c', B]\!] & & \text{by Lem. 4.5.} \end{array}$$

This concludes the proof. □

The following lemma hinges on the presence of the (CONSEQ) rule. The proof is much like in [12, Chap. 7].

Lemma A.10 *For each* WHILE$^{\mathrm{dt}}$ *command c and each* ASSN$^{\mathrm{dt}}$ *formulas B, we have*

$$\vdash \big\{\, w[\![c, B]\!]\,\big\}\, c\, \{B\} \qquad \textit{in}\ \text{HOARE}^{\mathrm{dt}}.$$

Proof. By induction on c. Obvious when c is `skip`, $x := a$, or $c_1; c_2$.

If c is $\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2$: we have $\vdash \big\{\, w[\![c_j, B]\!]\,\big\}\, c_j\, \{B\}$ by the induction hypothesis, for $j \in \{1, 2\}$. By the definition of $w[\![\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2,\ B]\!]$ we have

$$\begin{array}{l} \models b\ \wedge\ w[\![\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2,\ B]\!]\ \Rightarrow\ w[\![c_1, B]\!] \quad \text{and} \\ \models \neg b\ \wedge\ w[\![\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2,\ B]\!]\ \Rightarrow\ w[\![c_2, B]\!] \ . \end{array}$$

Therefore by the (CONSEQ) rule we obtain

$$\begin{array}{l} \vdash \big\{\, b\ \wedge\ w[\![\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2,\ B]\!]\,\big\}\, c_1\, \{B\} \quad \text{and} \\ \vdash \big\{\, \neg b\ \wedge\ w[\![\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2,\ B]\!]\,\big\}\, c_2\, \{B\} \ . \end{array}$$

By applying the (IF) rule we prove the claim.

If c is $\mathtt{while}\ b\ \mathtt{do}\ c'$: first we prove that

$$\models \big\{\, w[\![\mathtt{while}\ b\ \mathtt{do}\ c',\ B]\!]\ \wedge\ b\,\big\}\, c'\, \big\{\, w[\![\mathtt{while}\ b\ \mathtt{do}\ c',\ B]\!]\,\big\} \ . \tag{12}$$

To see it, assume $\boldsymbol{\sigma} \models w[\![\mathtt{while}\ b\ \mathtt{do}\ c',\ B]\!]\ \wedge\ b$ and let $(\boldsymbol{\sigma}|_i)_{i\in\mathbb{N}}$ be an arbitrary sequence representation of $\boldsymbol{\sigma}$. We have, by Lem. 4.5, $\boldsymbol{\sigma}|_i \models \big(w[\![\mathtt{while}\ b\ \mathtt{do}\ c',\ B]\!]\big)|_i \wedge b|_i$ for almost every i; from which $[\![c'|_i]\!](\boldsymbol{\sigma}|_i) \models \big(w[\![\mathtt{while}\ b\ \mathtt{do}\ c',\ B]\!]\big)|_i$ easily follows by the definition of $w[\![\mathtt{while}\ b\ \mathtt{do}\ c',\ B]\!]$. Thus we conclude $[\![c']\!]\boldsymbol{\sigma} \models w[\![\mathtt{while}\ b\ \mathtt{do}\ c',\ B]\!]$.

We also prove that

$$\models w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!] \wedge \neg b \;\Rightarrow\; B\ . \tag{13}$$

To see this, assume $\boldsymbol{\sigma} \models w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!] \wedge \neg b$. We can also assume that $\boldsymbol{\sigma} \neq \bot$ (otherwise the claim is obvious). By Prop. A.9 we have $[\![\texttt{while}\ b\ \texttt{do}\ c']\!]\boldsymbol{\sigma} \models B$. From $\boldsymbol{\sigma} \not\models b$, it easily follows that $[\![\texttt{while}\ b\ \texttt{do}\ c']\!]\boldsymbol{\sigma} = \boldsymbol{\sigma}$, using Lem. 3.10 and Lem. 4.5. Therefore $\boldsymbol{\sigma} \models B$.

We get back to the main line of the proof.

$$\models w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!] \wedge b \;\Rightarrow\; w[\![c',\, w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!]]\!] \tag{14}$$

by Prop. A.9 and (12)

$$\vdash \big\{\, w[\![c',\, w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!]]\!] \,\big\}\, c' \,\big\{\, w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!] \,\big\} \tag{15}$$

by the induction hypothesis, (‡)

$$\vdash \big\{\, w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!] \wedge b \,\big\}\, c' \,\big\{\, w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!] \,\big\} \tag{16}$$

by (14–15) and the (CONSEQ) rule

$$\vdash \big\{\, w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!] \,\big\}\, \texttt{while}\ b\ \texttt{do}\ c' \,\big\{\, w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!] \wedge \neg b \,\big\} \tag{17}$$

by (16) and the (WHILE) rule

$$\vdash \big\{\, w[\![\texttt{while}\ b\ \texttt{do}\ c',\, B]\!] \,\big\}\, \texttt{while}\ b\ \texttt{do}\ c' \,\{B\} \tag{18}$$

by (13), (17) and the (CONSEQ) rule.

Note that in (‡), we can use the induction hypothesis since the command c' is simpler than $c \equiv \texttt{while}\ b\ \texttt{do}\ c'$. This concludes the proof. □

Relative completeness of HOARE$^{\mathsf{dt}}$ is an immediate corollary of the above results.

Proof. (Of Thm. 5.4) Assume $\models \{A\}c\{B\}$. By Prop. A.9 we have $\models A \Rightarrow w[\![c, B]\!]$; and by Lem. A.10 we have $\vdash \big\{\, w[\![c, B]\!] \,\big\}c\{B\}$. Applying the (CONSEQ) rule we obtain $\vdash \{A\}c\{B\}$. □