[1] KS and IH. **Programming with Infinitesimals: A While-Language for Hybrid System Modeling**. Proc. ICALP 2011, Track B.
[2] IH and KS. **Exercises in Nonstandard Static Analysis of Hybrid Systems**. Proc. CAV 2012.

Nonstandard Static Analysis

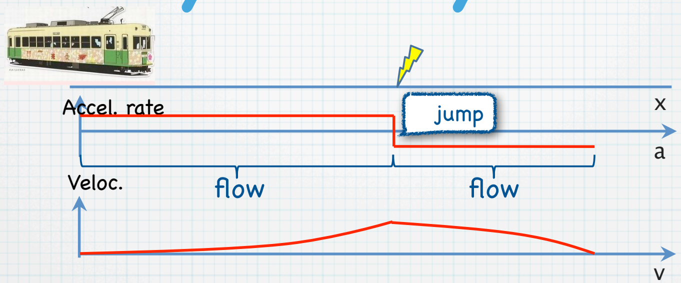
Transfer Verification to Hybrid Systems

Ichiro Hasuo
University of Tokyo (JP)

Kohei Suenaga
Kyoto University (JP)

東京大学 THE UNIVERSITY OF TOKYO
京都大学 KYOTO UNIVERSITY

Hybrid System



* Flow & jump
  * Digital control in a physical environment
  * Component of cyber-physical systems

Hybrid System

Formal verification
(computer science)

Hybrid!

- Flow?
- With minimal cost?

Discrete
"jump"

and

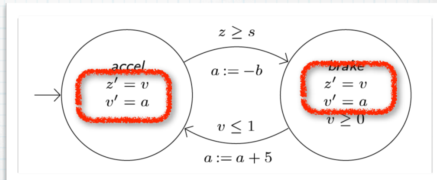
Continuous
"flow"

Hybrid!

Control theory
(applied analysis)

Formal Verification Approaches

* Hybrid automata
  [Alur, Henzinger, ...; '90s–]



* Differential dynamic logic
  [Platzer & others, '07–]

$$[\dot{x} = 1 \text{ while } x \le 3]\varphi$$

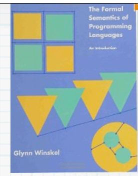
* Differential equations, explicitly
  ➜ distinction jump vs. flow

"Turn Flow into Jump"

$t := 0$;
while $(t \le 1)$ do {
  $t := t + \mathrm{dt}$
}

- Infinitesimal number dt
  - "Infinitely small" :
    0 < dt < r
    for **any** positive real r
- t = 1 after the execution?
- Non-standard analysis!
  [Robinson '60s]

Theoretical Framework [Suenaga&H., ICALP'11]



The standard textbook [Winskel]

While$^{\mathrm{dt}}$	**Assn**$^{\mathrm{dt}}$	**Hoare**$^{\mathrm{dt}}$

Programming lang.
`while (t<a) do {`
`  t:=t+1;`
`  if ...`
`}`
	First-order assertion lang.
	`∃z(x=2*z ∧ y=3*z)`
		Hoare-style program logic
		$\dfrac{\{A\wedge b\}\,c\,\{A\}}{\{A\}\ \mathtt{while}\ b\ \mathtt{do}\ c\ \{A\wedge\neg b\}}$

Rigorous semantics by non-standard analysis

- **Hoare**$^{\mathrm{dt}}$: sound and relatively complete
- Program verification/static analysis of hybrid systems
- Actual verification with NSA

Program Verif. Techniques
- Esp. invariant discovery

Static Analysis

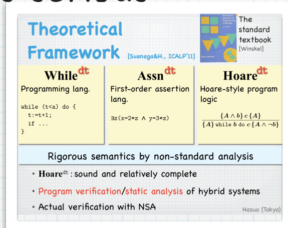
Nonstandard Static Analysis

Nonstandard Analysis

Infinitesimal dt

Nonstandard Static Analysis

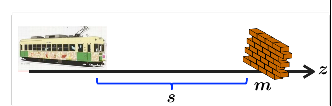
- Towards serious use of



- Static analysis techniques transferred to hybrid appl.

Exactly as they are!

- Leading example: ETCS



Prototype Automatic Prover

P (While^{dt} program) →

B (postcondition)

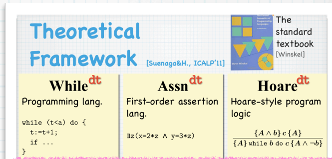
Hoare^{dt} Analyzer

- **VC generator** (Frontend, in OCaml)
- ↓↑
- **Symbolic Comp. Engine** (Backend, in Mathematica)

→ ***A*** (precondition) s.t. $\vdash \{A\}P\{B\}$

For reals, not hyperreals
➜ justified by the **transfer principle**

Outline



w/ or w/o dt ...
➜ logically "the same"

* **Theoretical foundations**
  * While^{dt}, Assn^{dt}, Hoare^{dt}
  * Rigorous semantics via NSA
  * Transfer principle, "sectionwise lemmas"
* **Static analysis techniques, transferred as they are**
  * Phase split [Sharma,Dillig,Dillig,Aiken; CAV'11] [Balakrishnan,Sankaranarayanan,Ivancic,Gupta; EMSOFT'09] [Gopan,Reps; SAS'07]
  * Differential invariant [Platzer,Clarke; CAV'08]
  * ... and more!

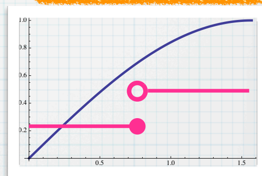
Part I: Theoretical Foundations

Nonstandard Analysis

* Analysis with an **infinitesimal** δ, e.g.

"Infinitely small" $0 < \delta < r$ $(\forall r \in \mathbb{R}_+)$

$$f \text{ is continuous} \iff \left(\begin{array}{l} |x - x'| \text{ is infinitesimal} \\ \implies |f(x) - f(x')| \text{ is infinitesimal} \end{array} \right)$$



* Cf. Leibniz's **monad**
* Done naively ➜ contradiction!

Logical foundation via an ultrafilter
[Robinson,1960]

Hyperreals

= Reals + Infinitesimals + ...

Defn.
The set of *hyperreal numbers* is

$$^*\mathbb{R} \;:=\; \mathbb{R}^{\mathbb{N}} / \sim_{\mathcal{F}} \;\ni\; \big[\,(a_0, a_1, a_2, \dots)\,\big]$$

Ignore

0th section / 1st section / 2nd section

* Operations: sectionwise

$$\begin{array}{ll} & [(a_0, a_1, \dots)] \\ + & [(b_0, b_1, \dots)] \\ = & [(a_0 + b_0, a_1 + b_1, \dots)] \end{array}$$

* Reals are hyperreals

$\mathbb{R} \hookrightarrow {}^*\mathbb{R}$,
$r \mapsto \big[\,(r, r, \dots)\,\big]$

Hyperreals

= Reals + Infinitesimals + ...

Defn.
The set of *hyperreal numbers* is

$$^*\mathbb{R} \;:=\; \mathbb{R}^{\mathbb{N}} / \sim_{\mathcal{F}} \;\ni\; \big[\,(a_0, a_1, a_2, \dots)\,\big]$$

* Predicates: sectionwise, "for almost all i"

$[(a_i)_{i\in\mathbb{N}}] < [(b_i)_{i\in\mathbb{N}}]$
$\iff$ $a_i < b_i$ "for almost every i"
$\Longleftarrow$ $\{i \in \mathbb{N} \mid a_i \not< b_i\}$ is finite

"For sufficiently large i"
"Except for finitely many i"

Precise defn. is via an ultrafilter $\mathcal{F}$:

$[(a_i)_{i\in\mathbb{N}}] < [(b_i)_{i\in\mathbb{N}}]$
$\iff$ $\{i \in \mathbb{N} \mid a_i < b_i\} \in \mathcal{F}$

Hyperreals

= Reals + Infinitesimals + ...

Defn.
The set of *hyperreal numbers* is
$^*\mathbb{R} \;:=\; \mathbb{R}^{\mathbb{N}} / \sim_{\mathcal{F}}$

$[(a_i)_{i\in\mathbb{N}}] < [(b_i)_{i\in\mathbb{N}}]$
$\iff$ $a_i < b_i$ "for almost every i"
$\Longleftarrow$ $\{i \in \mathbb{N} \mid a_i \not< b_i\}$ is finite

Prop. $\omega^{-1} = \big[\,(1, \frac{1}{2}, \frac{1}{3}, \dots)\,\big]$ is infinitesimal.

$$\omega^{-1} = (1, \tfrac{1}{2}, \tfrac{1}{3}, \dots, \tfrac{1}{N}, \tfrac{1}{N+1}, \dots)$$

OK! ^ ✖ ✖ ✖ ✖ ^ ^ ...

$$\frac{1}{N} = (\tfrac{1}{N}, \tfrac{1}{N}, \tfrac{1}{N}, \dots, \tfrac{1}{N}, \tfrac{1}{N}, \dots)$$

Hyperreals

= Reals + Infinitesimals + ...

Defn.
The set of *hyperreal numbers* is
$^*\mathbb{R} \;:=\; \mathbb{R}^{\mathbb{N}} / \sim_{\mathcal{F}}$

$[(a_i)_{i\in\mathbb{N}}] < [(b_i)_{i\in\mathbb{N}}]$
$\iff$ $a_i < b_i$ "for almost every i"
$\Longleftarrow$ $\{i \in \mathbb{N} \mid a_i \not< b_i\}$ is finite

Prop. $\omega^{-1} = \big[\,(1, \frac{1}{2}, \frac{1}{3}, \dots)\,\big]$ is infinitesimal.

Prop. $\omega = \big[\,(1, 2, 3, \dots)\,\big]$ is infinite.

Hyperreals = Reals + Inf

Ultrafilter (existence by AC)

Defn. An *ultrafilter* $\mathcal{F} \subseteq \mathcal{P}(\mathbb{N})$ is such that:

1. For each $X \subseteq \mathbb{N}$, exactly one of X and $\mathbb{N} \setminus X$ is in $\mathcal{F}$.
2. $X, Y \in \mathcal{F} \Longrightarrow X \cap Y \in \mathcal{F}$
3. $X \in \mathcal{F}, X \subseteq Y \Longrightarrow Y \in \mathcal{F}$
4. $\emptyset \notin \mathcal{F}$

Defn. The set of *hyperreal numbers* is

$$^*\mathbb{R} \;:=\; \mathbb{R}^{\mathbb{N}} / \sim_{\mathcal{F}}$$

$$[(a_i)_{i\in\mathbb{N}}] < [(b_i)_{i\in\mathbb{N}}] \iff \{i \in \mathbb{N} \mid a_i < b_i\} \in \mathcal{F}$$

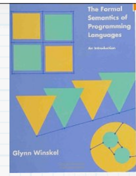
Thm. (Transfer Principle)
A: a first-order formula.
$^*A$: its $*$-*transform*. Then

$$\mathbb{R} \models A \iff {}^*\mathbb{R} \models {}^*A \,.$$

Same as A, except:
$\forall x \in \mathbb{R}$ in A is
$\forall x \in {}^*\mathbb{R}$ in $^*A$

$\mathbb{R}$ and $^*\mathbb{R}$ are "logically the same"

Theoretical Framework [Suenaga&H., ICALP'11]



The standard textbook [Winskel]

While^{dt}
Programming lang.

```
while (t<a) do {
  t:=t+1;
  if ...
}
```

Assn^{dt}
First-order assertion lang.

∃z(x=2*z ∧ y=3*z)

Hoare^{dt}
Hoare-style program logic

$$\frac{\{A \wedge b\}\, c\, \{A\}}{\{A\}\ \mathtt{while}\ b\ \mathtt{do}\ c\ \{A \wedge \neg b\}}$$

Rigorous semantics by non-standard analysis

Hasuo (Tokyo)

Syntax

While^{dt} — While + dt

AExp $\ni$ a ::= $x \mid c_r \mid a_1\ \mathsf{aop}\ a_2 \mid \mathsf{dt}$
where c_r is a const. for $r \in \mathbb{R}$, aop $\in \{+, -, \cdot, \hat{}\,, /\}$
BExp $\ni$ b ::= $\mathsf{true} \mid \mathsf{false} \mid b_1 \wedge b_2 \mid \neg b \mid a_1 < a_2$
Cmd $\ni$ c ::= $\mathsf{skip} \mid x := a \mid c_1; c_2$
$\mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c$

Assn^{dt}

A ::= $\mathsf{true} \mid \mathsf{false} \mid A_1 \wedge A_2 \mid \neg A \mid a_1 < a_2 \mid$
$\forall x \in {}^*\mathbb{N}.\, A \mid \forall x \in {}^*\mathbb{R}.\, A$

Hoare^{dt}

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

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

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

Tokyo)

Syntax

While^{dt} — While + dt

AExp $\ni$ a ::= $x \mid c_r \mid a_1\ \mathsf{aop}\ a_2 \mid \mathsf{dt}$
where c_r is a const. for $r \in \mathbb{R}$, aop $\in \{+, -, \cdot, \hat{}\,, /\}$
BExp $\ni$ b ::= $\mathsf{true} \mid \mathsf{false} \mid b_1 \wedge b_2 \mid \neg b \mid a_1 < a_2$
Cmd $\ni$ c ::= $\mathsf{skip} \mid x := a \mid c_1; c_2$
$\mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c$

Assn^{dt} — Assn, $*$-transformed

A ::= $\mathsf{true} \mid \mathsf{false} \mid A_1 \wedge A_2 \mid \neg A \mid a_1 < a_2 \mid$
$\forall x \in {}^*\mathbb{N}.\, A \mid \forall x \in {}^*\mathbb{R}.\, A$

Hoare^{dt}

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

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

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

Tokyo)

Syntax

While$^{\mathrm{dt}}$

While + dt

AExp $\ni$ a ::= $x \mid c_r \mid a_1 \; \mathtt{aop} \; a_2 \mid \mathtt{dt}$

where c_r is a const. for $r \in \mathbb{R}$, $\mathtt{aop} \in \{+, -, \cdot, \hat{}, /\}$

BExp $\ni$ b ::= $\mathtt{true} \mid \mathtt{false} \mid b_1 \wedge b_2 \mid \neg b \mid a_1 < a_2$

c ::= $\mathtt{skip} \mid x := a \mid c_1 ; c_2$ $\mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ b\ \mathtt{do}\ c$

Thm. HOARE$^{\mathrm{dt}}$ rules are *sound* and *relatively complete*.

Hoare$^{\mathrm{dt}}$

Precisely the same ...

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

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

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

Denotational Semantics: Challenge

$t := 0\,;$
$\mathtt{while}\ (t \leq 1)\ \mathtt{do}\ \{$
$\quad t := t + \mathtt{dt}$
$\}$

$\Downarrow$

$t = 1 + \mathtt{dt}$

$t := 0\,;$
$\mathtt{while}\ (\mathtt{true})\ \mathtt{do}\ \{$
$\quad t := t + \mathtt{dt}$
$\}$

$\Downarrow$

$\perp$ (divergence)

- Semantics by "sectionwise execution"

Denotational Semantics

- Execute sectionwise and bundle up the outcomes!

```
t := 0;
while (t < 1)
  t := t + dt;
```


Denotational Semantics

- Execute sectionwise and bundle up the outcomes!

```
t := 0;
while (t < 1)
  t := t + dt;
```

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := (0,0,0,...);
while (t < (1,1,1,...))
  t := t + (1, 1/2, 1/3, ...) ;
```


Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

0th section	1st section	2nd section	
`t := 0; while (t < 1) t := t + 1 ;`	`t := 0; while (t < 1) t := t + 1/2 ;`	`t := 0; while (t < 1) t := t + 1/3 ;`	...
`t = 1`	`t = 1`	`t = 1`	...

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := (0,0,0,...);
while (t < (1,1,1,...))
  t := t + (1, 1/2, 1/3, ...) ;
```

```
t = (1,1,1,...)
```


Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := 0;
while (t < 1)
  t := t + dt;
```

```
t = 1
```

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := 0;
while (t <= 1)
  t := t + dt;
```


Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := 0;
while (t <= 1)
  t := t + dt;
```


Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

t := (0,0,0,...);
while (t <= (1,1,1,...))
t := t + $(1, \frac{1}{2}, \frac{1}{3}, \ldots)$;

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

0th section	1st section	2nd section	

t := 0;
while (t <= 1)
t := t + 1 ;
	t := 0;
	while (t <= 1)
	t := t + $\frac{1}{2}$;
		t := 0;
		while (t <= 1)
		t := t + $\frac{1}{3}$;
			...

t = 1 + 1	t = 1 + $\frac{1}{2}$	t = 1 + $\frac{1}{3}$	...

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := (0,0,0,...);
while (t <= (1,1,1,...))
  t := t + (1, 1/2, 1/3, ...) ;
```

$$t = (1,1,1,\ldots) + \left(1, \frac{1}{2}, \frac{1}{3}, \ldots\right)$$

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := 0;
while (t <= 1)
  t := t + dt;
```

```
t = 1 + dt
```


Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := 0;
while (true)
  t := t + dt;
```


Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := 0;
while (true)
  t := t + dt;
```

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := (0,0,0,...);
while (true)
  t := t + (1, 1/2, 1/3, ...) ;
```


Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

0th section

```
t := 0;
while (true)
  t := t + 1 ;
```

1st section

```
t := 0;
while (true)
  t := t + 1/2 ;
```

2nd section

```
t := 0;
while (true)
  t := t + 1/3 ;
```

...

⊥ ⊥ ⊥ ...

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := (0,0,0,...);
while (true)
  t := t + (1, 1/2, 1/3, ...) ;
```

```
t = (⊥,⊥,⊥,...)
```


Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := 0;
while (true)
  t := t + dt;
```

⊥

Denot

$$\left[\begin{array}{l} t := 0\,; \\ \texttt{while}\ (t \le 1)\ \texttt{do} \\ \quad t := t + \mathtt{dt} \end{array}\right] \overset{i\text{-th section}}{\longmapsto} \left[\begin{array}{l} t := 0\,; \\ \texttt{while}\ (t \le 1)\ \texttt{do} \\ \quad t := t + \frac{1}{i+1} \end{array}\right]$$

Hyperstate (stores hyperreals)

$$\begin{aligned} [\![x]\!]\sigma &:= \sigma(x) \\ [\![a_1 \;\texttt{aop}\; a_2]\!]\sigma &:= [\![a_1]\!]\sigma \;\texttt{aop}\; [\![a_2]\!]\sigma \\ [\![\mathtt{dt}]\!]\sigma &:= \omega^{-1} = [\,(1, \tfrac{1}{2}, \tfrac{1}{3}, \dots)\,] \end{aligned}$$

$$\begin{aligned} [\![\mathtt{true}]\!]\sigma &:= \mathtt{tt} \\ [\![b_1 \wedge b_2]\!]\sigma &:= [\![b_1]\!]\sigma \wedge [\![b_2]\!]\sigma \\ [\![a_1 < a_2]\!]\sigma &:= [\![a_1]\!]\sigma < [\![a_2]\!]\sigma \end{aligned}$$

$[\![\mathtt{fa}$ … $[\![\neg$ …

Def. The *i-th section* of a WHILE$^{\mathrm{dt}}$ expression e is

$$e|_i \;:\equiv\; e\left[\tfrac{1}{i+1} \,/\, \mathtt{dt}\right] .$$

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

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

$$[\![\texttt{while}\ b\ \texttt{do}\ c]\!]\sigma := \left(\,[\![\,(\texttt{while}\ b\ \texttt{do}\ c)|_i\,]\!](\sigma|_i)\,\right)_{i \in \mathbb{N}}$$

Bundled up

Section of a program

Applied to a section of a memory state

"Sectionwise Lemmas"

Sectionwise Execution Lemma. For any expr. e and $i \in \mathbb{N}$,

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

Sectionwise Satisfaction Lemma. For any hyperstate σ and an ASSN$^{\mathrm{dt}}$ formula φ:

$$\sigma \models \varphi \iff \sigma|_i \models \varphi|_i \quad \text{for almost every } i.$$

Łos'

"Sectionwise Lemmas"

Lem. (Sectionwise validity of Hoare triples)

$$\models \{A\}c\{B\} \iff \models \{A|_i\}\; c|_i\; \{B|_i\} \quad \text{for almost every } i.$$

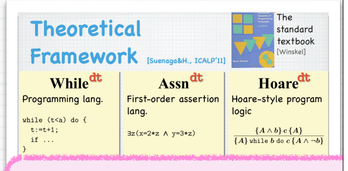
Interface for transferring static analysis techniques

Q. Is a While$^{\mathrm{dt}}$ program executable?

* A. Not exactly.
  * A modeling language
    * Not numerical approx., but exact modeling
    * Advantage: close to a common programming style
  * Static analysis ➜ no need to execute!
    * Mathematical semantics suffices

Outline

Suenaga & H., ICALP'11



- **Theoretical foundations**
  - **While^dt, Assn^dt, Hoare^dt**
  - Rigorous semantics via NSA
  - Transfer principle, "sectionwise lemmas"

w/ or w/o dt ...
→ logically "the same"

Done

H. & Suenaga, CAV'12

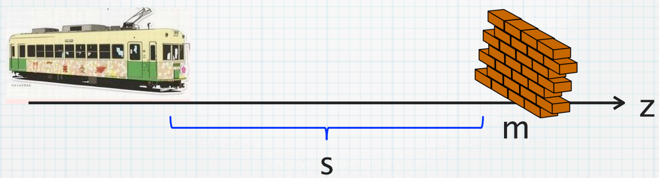
- **Static analysis techniques, transferred as they are**
  - Phase split [Sharma,Dillig,Dillig,Aiken; CAV'11] [Balakrishnan,Sankaranarayanan,Ivancic,Gupta; EMSOFT'09] [Gopan,Reps; SAS'07]
  - Differential invariant [Platzer,Clarke; CAV'08]
  - ... and more!

Part II:

Exercises in Nonstandard Static Analysis

Exercise 1.1

(Tiny) fragment of Euro. Train Ctrl. Sys. (ETCS)



s: big enough
b: big enough
a_0: small enough
...

while $v > 0$ do {
  $t := 0$;
  if $m - z < s$ then $a := -b$ else $a := a_0$;
  while $t < \varepsilon$ do {
    $t := t + \mathrm{dt}$;
    $v := v + a \cdot \mathrm{dt}$;
    $z := z + v \cdot \mathrm{dt}$
  }}

ETCS_0

Q. Find A s.t. $\models \{A\}\,\mathrm{ETCS}_0\,\{z < m\}$


```
while (v > 0) {
  if m - z < s
    then a := -b
    else a := a0;
  t := 0;
  while (t < eps && v > 0) {
    z := z + v * dt;
    v := v + a * dt;
    t := t + dt }}

{z < m}
```

```
while (v > 0 && m - z >= s) {
  a := a0;    t := 0;
  while (t < eps && v > 0) {
    z := z + v * dt;
    v := v + a0 * dt;
    t := t + dt }};
while (v > 0 && m - z < s) {
  a := -b;    t := 0;
  while (t < eps && v > 0) {
    z := z + v * dt;
    v := v - b * dt;
    t := t + dt }}

{z < m}
```

accel.

brake

Strategy1 **"Phase split"**
[Sharma,Dillig,Dillig,Aiken; CAV'11]
[Balakrishnan,Sankaranarayanan,Ivancic,Gupta; EMSOFT'09] [Gopan,Reps; SAS'07]

Phase Split

(Standard Ver., for While & Hoare)

[Sharma,Dillig,Dillig,Aiken; CAV'11]

Defn.
The set of *holed commands* $\mathbf{Cmd}_{[\_]}$ is:

$$\mathbf{Cmd}_{[\_]} \ni h \;::=\; \texttt{if}\ [\_]\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid h;c \mid c;h \mid \texttt{if}\ b\ \texttt{then}\ h\ \texttt{else}\ c \mid \texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ h$$

For each holed command h, its *pre-hole fragment* $\overline{h}$ is:

$$\overline{\texttt{if}\ [\_]\ \texttt{then}\ c_1\ \texttt{else}\ c_2} :\equiv \texttt{skip}$$
$$\overline{h;c} :\equiv \overline{h} \qquad \overline{c;h} :\equiv c;\overline{h}$$
$$\overline{\texttt{if}\ b\ \texttt{then}\ h\ \texttt{else}\ c} :\equiv \texttt{assert}\ b\,;\overline{h}$$
$$\overline{\texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ h} :\equiv \texttt{assert}\ \neg b\,;\overline{h}$$

$\texttt{while}\ b_g\ \texttt{do}\ \boxed{\ldots(\texttt{if}\ldots)\ldots}$

into $\left[\begin{array}{l}\texttt{while}\ b_g \wedge \neg b_s\ \texttt{do}\ \boxed{\ldots}\ ;\\ \texttt{while}\ b_g \wedge b_s\ \texttt{do}\ \boxed{\ldots}\end{array}\right]$

Lem.
If a Boolean expression $b_s \in \mathbf{BExp}$ satisfies

$$\models \{b_s\}\,\overline{h}\,\{b_c\}\ ,\quad \models \{\neg b_s\}\,\overline{h}\,\{\neg b_c\}\ ,\quad \text{and} \quad \models \{b_g \wedge b_s\}\,h[b_c]\,\{\neg b_g \vee b_s\}\ ,$$

then we have

$$[\![\,\texttt{while}\ b_g\ \texttt{do}\ h[b_c]\,]\!] = \left[\!\!\left[\begin{array}{l}\texttt{while}\ (b_g \wedge \neg b_s)\ \texttt{do}\ h[\texttt{false}]\ ;\\ \texttt{while}\ (b_g \wedge b_s)\ \texttt{do}\ h[\texttt{true}]\end{array}\right]\!\!\right] .$$

$h[\,\_\,]$ is a command containing

`if [ _ ] then ... else ...`

Phase Split

(Nonstandard Ver., for While^{dt} & Hoare^{dt})

Defn.
The set of *holed commands* $\mathbf{Cmd}_{[\_]}$ is:

$$\mathbf{Cmd}_{[\_]} \ni h \;::=\; \texttt{if}\ [\_]\ \texttt{then}\ c_1\ \texttt{else}\ c_2 \mid h;c \mid c;h \mid \texttt{if}\ b\ \texttt{then}\ h\ \texttt{else}\ c \mid \texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ h$$

For each holed command h, its *pre-hole fragment* $\overline{h}$ is:

$$\overline{\texttt{if}\ [\_]\ \texttt{then}\ c_1\ \texttt{else}\ c_2} :\equiv \texttt{skip}$$
$$\overline{h;c} :\equiv \overline{h} \qquad \overline{c;h} :\equiv c;\overline{h}$$
$$\overline{\texttt{if}\ b\ \texttt{then}\ h\ \texttt{else}\ c} :\equiv \texttt{assert}\ b\,;\overline{h}$$
$$\overline{\texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ h} :\equiv \texttt{assert}\ \neg b\,;\overline{h}$$

Lem.
If a Boolean expression $b_s \in \mathbf{BExp}$ satisfies

$$\models \{b_s\}\,\overline{h}\,\{b_c\}\ ,\quad \models \{\neg b_s\}\,\overline{h}\,\{\neg b_c\}\ ,\quad \text{and} \quad \models \{b_g \wedge b_s\}\,h[b_c]\,\{\neg b_g \vee b_s\}\ ,$$

then we have

$$[\![\,\texttt{while}\ b_g\ \texttt{do}\ h[b_c]\,]\!] = \left[\!\!\left[\begin{array}{l}\texttt{while}\ (b_g \wedge \neg b_s)\ \texttt{do}\ h[\texttt{false}]\ ;\\ \texttt{while}\ (b_g \wedge b_s)\ \texttt{do}\ h[\texttt{true}]\end{array}\right]\!\!\right] .$$

Proof.

$\models \{b_s\}\,\overline{h}\,\{b_c\}$
$\models \{\neg b_s\}\,\overline{h}\,\{\neg b_c\}$
$\models \{b_g \wedge b_s\}\,h[b_c]\,\{\neg b_g \vee b_s\}$

⇔ sectionwise

$\models \{b_s|_i\}\,\overline{h}|_i\,\{b_c|_i\}$
$\models \{\neg b_s|_i\}\,\overline{h}|_i\,\{\neg b_c|_i\}$
$\models \{b_g|_i \wedge b_s|_i\}\,h|_i[b_c|_i]\,\{\neg b_g|_i \vee b_s|_i\}$

(for almost all i)

⇒ std. ver.

$[\![\,\texttt{while}\ b_g|_i\ \texttt{do}\ h|_i[b_c|_i]\,]\!] = \left[\!\!\left[\begin{array}{l}\texttt{while}\ (b_g|_i \wedge \neg b_s|_i)\ \texttt{do}\ h|_i[\texttt{false}]\ ;\\ \texttt{while}\ (b_g|_i \wedge b_s|_i)\ \texttt{do}\ h|_i[\texttt{true}]\end{array}\right]\!\!\right]$

sectionwise ⇕

$[\![\,\texttt{while}\ b_g\ \texttt{do}\ h[b_c]\,]\!] = \left[\!\!\left[\begin{array}{l}\texttt{while}\ (b_g \wedge \neg b_s)\ \texttt{do}\ h[\texttt{false}]\ ;\\ \texttt{while}\ (b_g \wedge b_s)\ \texttt{do}\ h[\texttt{true}]\end{array}\right]\!\!\right]$

Transferring

Static Analysis Strategies

$\models \{b_s\}\,\overline{h}\,\{b_c\}$
$\models \{\neg b_s\}\,\overline{h}\,\{\neg b_c\}$
$\models \{b_g \wedge b_s\}\,h[b_c]\,\{\neg b_g \vee b_s\}$

⇔ sectionwise

$\models \{b_s|_i\}\,\overline{h}|_i\,\{b_c|_i\}$
$\models \{\neg b_s|_i\}\,\overline{h}|_i\,\{\neg b_c|_i\}$
$\models \{b_g|_i \wedge b_s|_i\}\,h|_i[b_c|_i]\,\{\neg b_g|_i \vee b_s|_i\}$

(for almost all i)

⇒ std. ver.

$[\![\,\texttt{while}\ b_g|_i\ \texttt{do}\ h|_i[b_c|_i]\,]\!] = \left[\!\!\left[\begin{array}{l}\texttt{while}\ (b_g|_i \wedge \neg b_s|_i)\ \texttt{do}\ h|_i[\texttt{false}]\ ;\\ \texttt{while}\ (b_g|_i \wedge b_s|_i)\ \texttt{do}\ h|_i[\texttt{true}]\end{array}\right]\!\!\right]$

sectionwise ⇕

$[\![\,\texttt{while}\ b_g\ \texttt{do}\ h[b_c]\,]\!] = \left[\!\!\left[\begin{array}{l}\texttt{while}\ (b_g \wedge \neg b_s)\ \texttt{do}\ h[\texttt{false}]\ ;\\ \texttt{while}\ (b_g \wedge b_s)\ \texttt{do}\ h[\texttt{true}]\end{array}\right]\!\!\right]$

* Doesn't matter what "std. ver." is
* → **modular method** for transfer

```
while (v > 0) {
  if m - z < s
    then a := -b
    else a := a0;
  t := 0;
  while (t < eps && v > 0) {
    z := z + v * dt;
    v := v + a * dt;
    t := t + dt }}

{z < m}
```

⇒

```
while (v > 0 && m - z >= s) {
  a := a0;   t := 0;
  while (t < eps && v > 0) {
    z := z + v * dt;
    v := v + a0 * dt;
    t := t + dt }};
while (v > 0 && m - z < s) {
  a := -b;   t := 0;
  while (t < eps && v > 0) {
    z := z + v * dt;
    v := v - b * dt;
    t := t + dt }}

{z < m}
```

accel.

brake

Strategy 1 **"Phase split"**

[Sharma,Dillig,Dillig,Aiken; CAV'11]
[Balakrishnan,Sankaranarayanan,Ivancic,Gupta; EMSOFT'09]
[Gopan,Reps; SAS'07]

```
while (v > 0 && m - z >= s) {
  a := a0;   t := 0;
  while (t < eps && v > 0) {
    z := z + v * dt;
    v := v + a0 * dt;
    t := t + dt }};
while (v > 0 && m - z < s) {
  a := -b;   t := 0;
  while (t < eps && v > 0) {
    z := z + v * dt;
    v := v - b * dt;
    t := t + dt }}

{z < m}
```

```
if (v > 0)
  then
    while (m - z >= s) {
      a := a0;   t := 0;
      while (t < eps) {
        z := z + v * dt;
        v := v + a0 * dt;
        t := t + dt }}
  else skip;
while (v > 0) {
  a := -b;
```

Strategy 4
"Differential invariant"
[Platzer,Clarke; CAV'08]

Startegies 2,3
"Superfluous guard elim."   "Time elapse"

```
if (v > 0)
  then
    while (m - z >= s) {
      a := a0;   t := 0;
      while (t < eps) {
        z := z + v * dt;
        v := v + a0 * dt;
        t := t + dt }}
  else skip;
while (v > 0) {
  a := -b;
  z := z + v * dt;
  v := v - b * dt }

{z < m}
```

```
if (v > 0)
  then
    while (m - z >= s) {
      a := a0;   t := 0;
      while (t < eps) {
        z := z + v * dt;
        v := v + a0 * dt;
        t := t + dt }}
  else skip;
```
$$\left\{ \begin{array}{l} (v > 0 \lor m > z) \land \\ (b^2\mathrm{dt}^2 + 4b\mathrm{dt}v + 8bz + 4v^2 < 8bm \\ \quad \lor\, b\mathrm{dt}v + 2bz + v^2 \leq 2bm) \end{array} \right\}$$
```
while (v > 0) {
  a := -b;
  z := z + v * dt;
  v := v - b * dt }
```

Strategy 5
"QE Invariant"

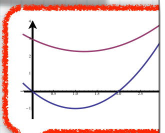
QE Invariant

Lem. In HOARE$^{\mathrm{dt}}$,

$$\vdash \left\{ \begin{array}{l} (\neg b \Rightarrow A) \land \\ \forall y \in {}^*\mathbb{N}.\big(\big(b[a/x]^y \land \neg b[a/x]^{y+1}\big) \Rightarrow A[a/x]^{y+1}\big) \end{array} \right\} \ \texttt{while}\ b\ \texttt{do}\ x := a\ \ \{A\}\ .$$

quantifier must go!
(to manage complexity)

- **Quantifier elimination**
  - Tarski, CAD algorithm, `Resolve` in Mathematica
  - e.g. $\models \forall x \in \mathbb{R}.\, (x^2 + ax + b > 0) \iff a^2 - 4b < 0$
  - then $\models \forall x \in {}^*\mathbb{R}.\, (x^2 + ax + b > 0) \iff a^2 - 4b < 0$



by **transfer**!


```
if (v > 0)
  then
    while (m - z >= s) {
      a := a0;   t := 0;
      while (t < eps) {
        z := z + v * dt;
        v := v + a0 * dt;
        t := t + dt }}
  else skip;
while (v > 0) {
  a := -b;
  z := z + v * dt;
  v := v - b * dt }

{z < m}
```

```
if (v > 0)
  then
    while (m - z >= s) {
      a := a0;   t := 0;
      while (t < eps) {
        z := z + v * dt;
        v := v + a0 * dt;
        t := t + dt }}
  else skip;
```
$$\left\{ \begin{array}{l} (v > 0 \lor m > z) \land \\ (b^2\mathrm{dt}^2 + 4b\mathrm{dt}v + 8bz + 4v^2 < 8bm \\ \quad \lor\, b\mathrm{dt}v + 2bz + v^2 \leq 2bm) \end{array} \right\}$$
```
while (v > 0) {
  a := -b;
  z := z + v * dt;
  v := v - b * dt }
```

Strategy 5
"QE Invariant"

```
if (v > 0)
  then
    while (m - z >= s) {
      a := a0;    t := 0;
      while (t < eps) {
        z := z + v * dt;
        v := v + a0 * dt;
        t := t + dt }}
  else skip;
```

$\{ (v > 0 \lor m > z) \land (b^2\mathrm{dt}^2 + 4b\mathrm{dt}v + 8bz + 4v^2 < 8bm \lor b\mathrm{dt}v + 2bz + v^2 \leq 2bm) \}$

```
while (v > 0) {
  a := -b;
  z := z + v * dt;
  v := v - b * dt }
```

+ some fwd. propagation

```
{ ... (long fml. with dt) }
while (m - z >= s) {
  a := a0;    t := 0;
  while (t < eps) {
    z := z + v * dt;
    v := v + a0 * dt;
    t := t + dt }}
{ ... }
while (v > 0) {
  a := -b;
  z := z + v * dt;
  v := v - b * dt }
```

Strategy 6

"Iteration count"

```
x := 0;
while (x < x0) {
  x := x + a
}
```

iteration: x0/a times?

- approximated by ⌊x0/a⌋ or ⌈x0/a⌉
- ➜ monotonicity reqm. must be discharged

```
{ ... (long fml. with dt) }
while (m - z >= s) {
  a := a0;    t := 0;
  while (t < eps) {
    z := z + v * dt;
    v := v + a0 * dt;
    t := t + dt }}
{ ... }
while (v > 0) {
  a := -b;
  z := z + v * dt;
  v := v - b * dt }
```

long fml. w/o dt, whose core is

$$a_0(2\varepsilon\sqrt{2a_0(m-s-z_0)+v_0^2} + b\epsilon^2 + 2m - 2s - 2z_0) + 2b\varepsilon\sqrt{2a_0(m-s-z_0)+v_0^2} + a_0^2\epsilon^2 + v_0^2 < 2bs$$

the final outcome

Lem. If:

1. a is *closed*
2. $r \mapsto [\![a[r/\mathrm{dt}]]\!]$ is continuous at $r = 0$,

then $\models a[0/\mathrm{dt}] < 0 \implies a < 0$.

Strategy 7

"Cast to shadow"

(Eliminates dt, strengthens the precond.)

Prototype Automatic Prover

P (While[dt] program) →

Hoare[dt] Analyzer

VC generator (Frontend, in OCaml)

↓↑

Symbolic Comp. Engine (Backend, in Mathematica)

→ A (precondition) s.t. $\vdash \{A\}P\{B\}$

B (postcondition)

Totally symbolic (crucial for transfer)

- Fujitsu HX600 with Quad Core AMD Opteron 2.3GHz CPU, 32GB memory. Mathematica 7.0 for Linux x86 (64-bit)
  - ETCS: 40.96 sec.
  - Bouncing ball: runs with one manual insertion of invariants

Related Work

- **Deductive verification** of hybrid sys. [Platzer, '10] [Platzer, LICS'12]
  - Automatic prover KeYmaera
- **Static analysis techniques**
  - A **LOT** in CAV, SAS, VMCAI, ...
  - Applied to hybrid systems (w/ diff. eq.) [Rodriguez-Carbonell, Tiwari; HSCC'05] [Sankaranarayanan; HSCC'10] [Sankaranarayanan, Sipma, Manna; Formal Methods Sys. Design '08]
- Use of **NSA for hybrid systems** [Benveniste, Bourke, Caillaud, Pouzet; J. Comput. Syst. Sci. '12] [Bliudze, Krob; Fundam. Inform. '09] [Gamboa, Kaufmann; J. Autom. Reason. '01]
- **Continuous techniques applied to discrete appl.** [Chaudhuri, Gulwani, Lublinerman, NavidPour; FSE '11]
  - Not contending! Combination?

Conclusions

Nonstandard Static Analysis

While^{dt}
Programming lang.

```
while (t<a) do {
  t:=t+1;
  if ...
}
```

Assn^{dt}
First-order assertion lang.

```
∃z(x=2*z ∧ y=3*z)
```

Hoare^{dt}
Hoare-style program logic

$$\frac{\{A \wedge b\}\, c\, \{A\}}{\{A\}\, \mathtt{while}\ b\ \mathtt{do}\ c\, \{A \wedge \neg b\}}$$

Rigorous semantics by non-standard analysis

- Tool's effectivity. More heuristics?
- (Any discrete frmwk.)^{dt} ?
- Simulink as stream processing?
- With (explicit) differential equations?

Thank you for your attention!

Ichiro Hasuo (Dept. CS, U Tokyo)
http://www-mmm.is.s.u-tokyo.ac.jp/~ichiro/