ERATO 蓮尾メタ数理システムデザインプロジェクト
ERATO Metamathematics for Systems Design Project
国立情報学研究所 & 科学技術振興機構 National Institute of Informatics & Japan Science and Technology Agency

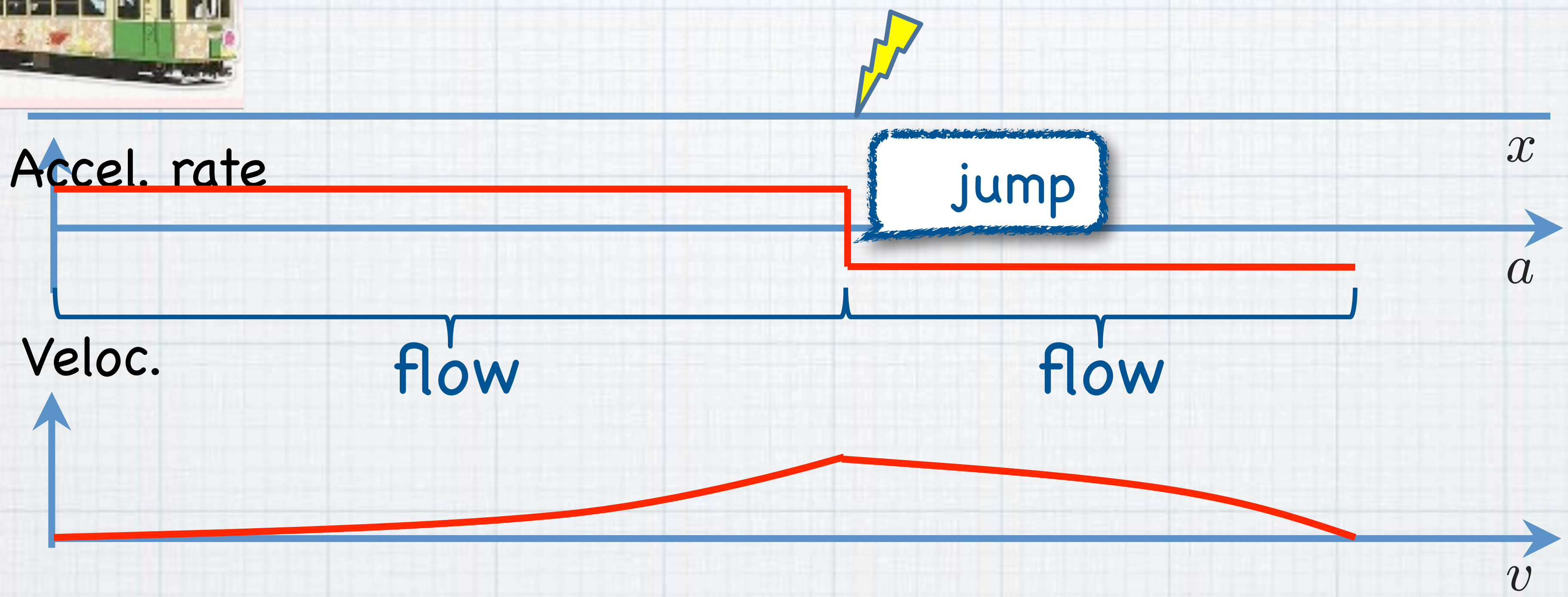
Nonstandard Static Analysis

Literal Transfer of Deductive Verification Frameworks from Discrete to Hybrid

Ichiro Hasuo

National Institute of Informatics
Tokyo, Japan

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

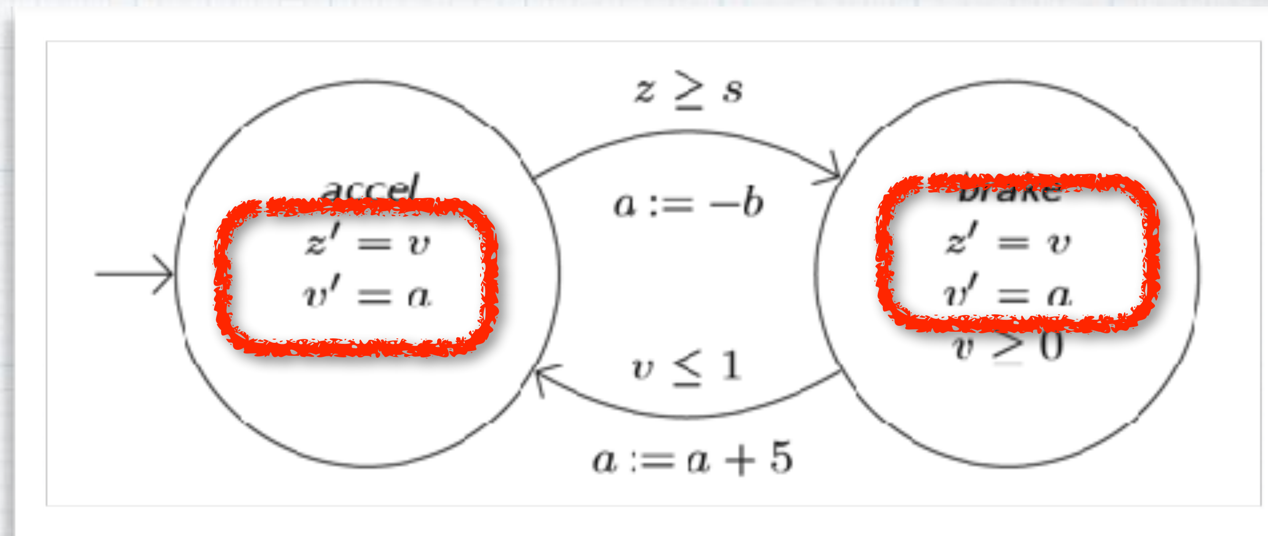
Continuous
"**flow**"

Hybrid!

Control theory
(applied analysis)

Formal Verification Approaches

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



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

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

* Differential equations, explicitly
  ➜ distinction jump vs. flow

"Turn Flow into Jump"

$$
\begin{aligned}
&t := 0\ ;\\
&\texttt{while}\ (t \le 1)\ \mathrm{do}\ \{\\
&\quad t := t + \mathrm{dt}\\
&\}
\end{aligned}
$$

[Suenaga & Hasuo, ICALP'11]
[Hasuo & Suenaga, CAV'12]
[Suenaga, Sekine & Hasuo, POPL'13]
[Kido, Chaudhuri & Hasuo, VMCAI'16]

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

Program Verif. Techniques

* Esp. invariant discovery

Static Analysis

Nonstandard Static Analysis

Nonstandard Analysis

Infinitesimal dt

Part Zero:

Deductive Verification & Static Analysis by Hoare Logic

Hoare Logic

Sir Antony Hoare
(1934.1.11-)
Microsoft Research, Cambridge

* [Hoare, 1969]
* Also called: "Program logic" "Floyd-Hoare logic"
* Related: Dynamic logic, Kleene algebra with tests
* A system that derives Hoare triples

{ A } P { B }

E.g.: { n=2 } n:=n+1 { n=3 }

"precondition" — A

program — P

"postcondition" — B

Deriv. Rules of Hoare Logic

$$\frac{}{\{\,A[a/x]\,\}\ x{:=}a\ \{\,A\,\}}\ \text{(Assign)}$$

$$\frac{\{\,A\,\}\ P_1\ \{\,C\,\} \qquad \{\,C\,\}\ P_2\ \{\,B\,\}}{\{\,A\,\}\ P_1;\ P_2\ \{\,B\,\}}\ \text{(SeqComp)}$$

$$\frac{\{\,A \wedge b\,\}\ P_1\ \{\,B\,\} \qquad \{\,A \wedge \neg b\,\}\ P_2\ \{\,B\,\}}{\{\,A\,\}\ \text{if } b \text{ then } P_1 \text{ else } P_2\ \{\,B\,\}}\ \text{(If)}$$

$$\frac{\{\,A \wedge b\,\}\ P_1\ \{\,A\,\}}{\{\,A\,\}\ \text{while } b\ P_1\ \{\,A \wedge \neg b\,\}}\ \text{(While)}$$

Deriv. Rules of Hoare Logic

$$\frac{A \Rightarrow A' \quad \{A'\}\ P\ \{B'\} \quad B' \Rightarrow B}{\{A\}\ P\ \{B\}}\ (\text{Conseq})$$

Deriv. Rules of Hoare Lo

A is a **loop invariant**!

$$\frac{\{A \land b\}\ P_1\ \{A\}}{\{A\}\ \text{while } b\ P_1\ \{A \land \neg b\}}\ \text{(While)}$$

Out of the loop ➜
b must be false

- E.g.:

$$\frac{\left\{ \begin{array}{l} k*(n!)=N! \\ \land\ n>0 \end{array} \right\}\ \begin{array}{l} k:=k*n; \\ n:=n-1 \end{array}\ \left\{ k*(n!)=N! \right\}}{\left\{ k*(n!) = N! \right\}\ \begin{array}{c} \text{while } (n>0) \\ k:=k*n; \\ n:=n-1 \end{array}\ \left\{ \begin{array}{l} k*(n!)=N! \\ \land\ n=0 \end{array} \right\}}\ \text{(While)}$$

Proof by Hoare Logic

* Goal: derive the Hoare triple

$$\{\ k=1 \wedge n=N\ \}\ \ \texttt{while (n>0)}\ \texttt{k:=k*n;}\ \texttt{n:=n-1}\ \ \{\ k = N!\ \}$$

```
{ k=1 ∧ n=N }   while (n>0)
                  k:=k*n;
                  n:=n-1       { k = N! }
```

Proof by Hoare Logic

(Assign)

$\{ k*n*((n-1)!)=N! \wedge n-1\geq 0 \}$ $k:=k*n$ $\{ k*((n-1)!)=N! \wedge n-1\geq 0 \}$

(Assign)

$\{ k*((n-1)!)=N! \wedge n-1\geq 0 \}$ $n:=n-1$ $\{ k*(n!)=N! \wedge n\geq 0 \}$

(SeqComp)

$k*(n!)=N! \wedge n\geq 0 \wedge n>0 \Rightarrow k*n*((n-1)!)=N! \wedge n-1\geq 0$

$\{ k*n*((n-1)!)=N! \wedge n-1\geq 0 \}$ $k:=k*n;$ $n:=n-1$

loop invariant

(Conseq)

$\{ k*(n!)=N! \wedge n\geq 0 \wedge n>0 \}$ $k:=k*n;$ $n:=n-1$ $\{ \underline{k*(n!)=N!} \wedge n\geq 0 \}$

(While)

$k=1 \wedge n=N \Rightarrow k*(n!) = N! \wedge n\geq 0$

$\{ k*(n!) = N! \wedge n\geq 0 \}$ while $(n>0)$ $k:=k*n;$ $n:=n-1$ $\{ k*(n!)=N! \wedge n\geq 0 \wedge \neg(n>0) \}$

$k*(n!)=N! \wedge n\geq 0 \wedge \neg(n>0) \Rightarrow k=N!$

(Conseq)

$\{ k=1 \wedge n=N \}$ while $(n>0)$ $k:=k*n;$ $n:=n-1$ $\{ k = N! \}$

Soundness, Completeness

- **Soundness**: "What is derived is true"
  - "No lies"
  - Derivation power is not too much
  - Indispensable (unsound ➜ not "formal verification"!)
- **Completeness**: "What is true can be derived"
  - Derivation power is as strong as possible
  - Often unavailable (no help... Gōdel's incompleteness)

- Hoare logic is **sound** and **relatively complete**

Deductive Verification, Static Analysis

- Hoare logic
  - A prototype of **deductive verification** **frameworks**
  - **Static analysis** (instead of dynamic)
    - Doesn't execute the program
    - Loop invariant =>
      "however many times the loop iterates"

Today's Talk: Framework [Suenaga&H., ICALP'11]

The standard textbook [Winskel]

$\mathbf{While}^{\mathrm{dt}}$

Programming lang.

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

$\mathbf{Assn}^{\mathrm{dt}}$

First-order assertion lang.

∃z(x=2*z ∧ y=3*z)

$\mathbf{Hoare}^{\mathrm{dt}}$

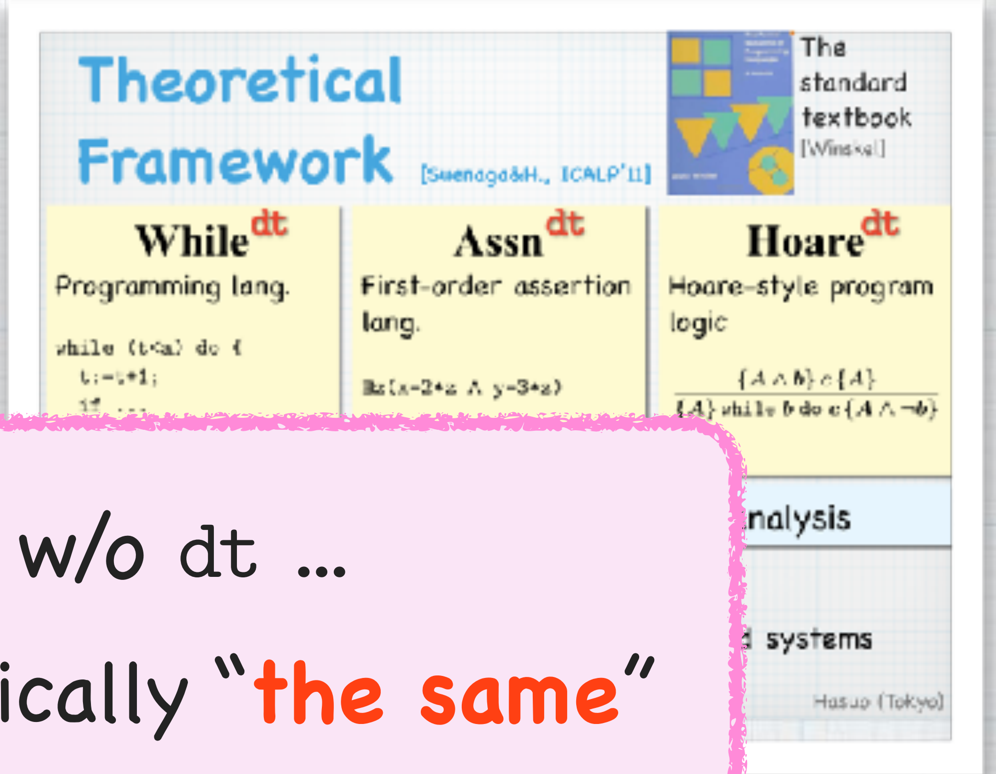
Hoare-style program logic

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

Rigorous semantics by non-standard analysis

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

Outline



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

w/ or w/o dt ...

➜ logically "**the same**"

- 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!

[Suenaga & Hasuo, ICALP'11]
[Hasuo & Suenaga, CAV'12]
[Suenaga, Sekine & Hasuo, POPL'13]
[Kido, Chaudhuri & Hasuo, VMCAI'16]

Part I:

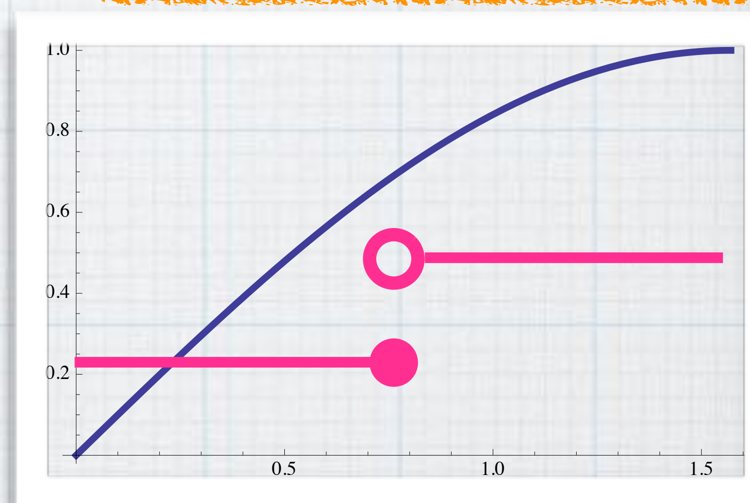
Theoretical Foundations

Nonstandard Analysis

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

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

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



* Cf. Leibniz's **monad**

* Done naively ➜ contradiction!

Logical foundation via an ultrafilter

[Robinson,1960]

Hyperreals

= Reals + Infinitesimals + ...

Defn.
The set of *hyperreal numbers* is

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

Ignore

0th section
1st section
2nd section

- **Operations:** sectionwise

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

- Reals are hyperreals

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

Hyperreals

= Reals + Infinitesimals + ...

Defn.
The set of *hyperreal numbers* is

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

* **Predicates:** sectionwise, "for almost all i"

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

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

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

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

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

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

OK! ✓ ✗ ✗ ✗ ✗ ✓ ✓ ...

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

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"

$\iff$ $\{i \in \mathbb{N} \mid a_i \not< b_i\}$ is finite

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

Prop. $\omega = \left[\,(1, 2, 3, \ldots)\,\right]$ is infinite.

Trouble… Resolved

$$\mathbf{0} \quad \overset{>}{\underset{<}{=}} \quad \big[(\,\mathbf{1}, \mathbf{-1},\ \mathbf{1}, \mathbf{-1}, \ldots)\big]$$

??

- Meaning of "almost every i" extended
  - … so that

For each $S \subset \mathbb{N}$, exactly one of

$$S \quad \text{and} \quad \mathbb{N} \setminus S$$

is "almost all i."

- ➜ Ultrafilter!

Defn.
The set of *hyperreal numbers* is

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

Filters & Ultrafilters

Given $X \subseteq \mathbb{N}$ is "yes, almost all!" or "no!"

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

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

Defn.
A *filter* $\mathcal{F} \subseteq \mathcal{P}(\mathbb{N})$ is that which satisfies Cond. 2.–4.

Prop.
$$\mathcal{F}_c := \{S \subseteq \mathbb{N} \mid \mathbb{N} \setminus S \text{ is finite}\}$$
is a filter (the *cofinite/Frechet* filter).

Prop.
Any filter $\mathcal{F}'$ can be extended to an ultrafilter $\mathcal{F} \supseteq \mathcal{F}'$. (By Zorn's lemma)

Cor.
There is an ultrafilter $\mathcal{F}$ such that $\mathcal{F}_c \subseteq \mathcal{F}$.

Fix one such

Hype

= 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
$$X \quad \text{and} \quad \mathbb{N} \setminus X$$
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$^{\mathrm{dt}}$

Programming lang.

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

Assn$^{\mathrm{dt}}$

First-order assertion lang.

∃z(x=2*z ∧ y=3*z)

Hoare$^{\mathrm{dt}}$

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

Syntax

While + dt

While$^{\mathrm{dt}}$

$$
\begin{array}{llll}
\mathrm{AExp} \ni & a & ::= & x \mid \mathtt{c}_r \mid a_1 \;\mathtt{aop}\; a_2 \mid \mathtt{dt} \\
& & & \text{where } \mathtt{c}_r \text{ is a const. for } r \in \mathbb{R},\ \mathtt{aop} \in \{+, -, \cdot, \hat{}\,, /\} \\
\mathrm{BExp} \ni & b & ::= & \mathtt{true} \mid \mathtt{false} \mid b_1 \land b_2 \mid \neg b \mid a_1 < a_2 \\
\mathrm{Cmd} \ni & c & ::= & \mathtt{skip} \mid x := a \mid c_1 ; c_2 \\
& & & \mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ b\ \mathtt{do}\ c
\end{array}
$$

Assn$^{\mathrm{dt}}$

$$
\begin{array}{lll}
A & ::= & \mathtt{true} \mid \mathtt{false} \mid A_1 \land A_2 \mid \neg A \mid a_1 < a_2 \mid \\
& & \forall x \in {}^*\mathbb{N}.\, A \mid \forall x \in {}^*\mathbb{R}.\, A
\end{array}
$$

Hoare$^{\mathrm{dt}}$

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

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

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

NII, JP)

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	$::=$	$\mathtt{true} \mid \mathtt{false} \mid b_1 \wedge b_2 \mid \neg b \mid a_1 < a_2$
Cmd $\ni$	c	$::=$	$\mathtt{skip} \mid x := a \mid c_1; c_2$
			$\mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ b\ \mathtt{do}\ c$

Assn$^{\mathrm{dt}}$

Assn, *-transformed

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

Hoare$^{\mathrm{dt}}$

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

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

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

Syntax

While^{dt}

While + dt

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

Thm.
HOARE^{dt} rules are sound and relatively complete.

Hoare^{dt}

Precisely the same ru...

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

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

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

NII, JP)

Denotational Semantics: Challenge

```
t := 0 ;
while (t ≤ 1) do {
  t := t + dt
}
```

$$t = 1 + \mathrm{dt}$$

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

$$\perp \ \text{(divergence)}$$

- 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, ...) ;
```

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

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

0th section

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

```
t = 1
```

1st section

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

```
t = 1
```

2nd section

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

```
t = 1
```

...

...

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := (0,0,0,...);
while (t < (1,1,1,...))
```

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

```
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, 1/2, 1/3, ...) ;
```

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

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,...))
```

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

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

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)
```

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

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 + $\frac{1}{2}$;

2nd section

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

...

$\perp$ | $\perp$ | $\perp$ ...

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

```
t := (0,0,0,...);
while (true)
```

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

$$\texttt{t} = (\bot,\bot,\bot,\ldots)$$

Denotational Semantics

* Execute sectionwise and bundle up the outcomes!

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

$\bot$

Denot[...]

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

Hyperstate (stores hyperreals)

$$\begin{array}{lcl} [\![x]\!]\sigma & := & \sigma(x) \\ [\![a_1\ \mathtt{aop}\ a_2]\!]\sigma & := & [\![a_1]\!]\sigma\ \mathtt{aop}\ [\![a_2]\!]\sigma \\ [\![\mathrm{dt}]\!]\sigma & := & \omega^{-1} = \left[\,(1, \tfrac{1}{2}, \tfrac{1}{3}, \ldots)\,\right] \end{array}$$

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

$[\![\mathtt{fa}$ ... $[\![\neg$ ...

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

$$e|_i \quad :\equiv \quad e\left[\,\frac{1}{i+1}\,/\,\mathrm{dt}\,\right]\ .$$

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

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

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

Bundled up

Section of a program

Applied to a section of a memory state

"Sectionwise Lemmas"

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

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

Sectionwise Satisfaction Lemma. For any hyperstate $\boldsymbol{\sigma}$ and an $\textsc{Assn}^{\mathrm{dt}}$ formula $\boldsymbol{\varphi}$:

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

Łos' Theorem

"Sectionwise Lemmas"

Lem. (Sectionwise validity of Hoare triples)

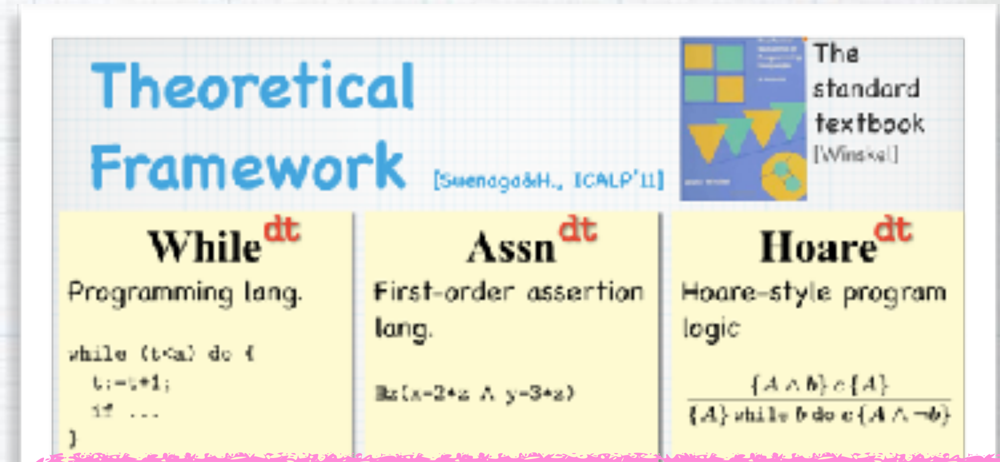
$$\models \{\boldsymbol{A}\}\boldsymbol{c}\{\boldsymbol{B}\} \iff \models \{\boldsymbol{A}|_{\boldsymbol{i}}\}\, \boldsymbol{c}|_{\boldsymbol{i}}\, \{\boldsymbol{B}|_{\boldsymbol{i}}\} \text{ for almost every } \boldsymbol{i}.$$

Interface for **transferring** static analysis techniques

Q. Is a **While**^{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

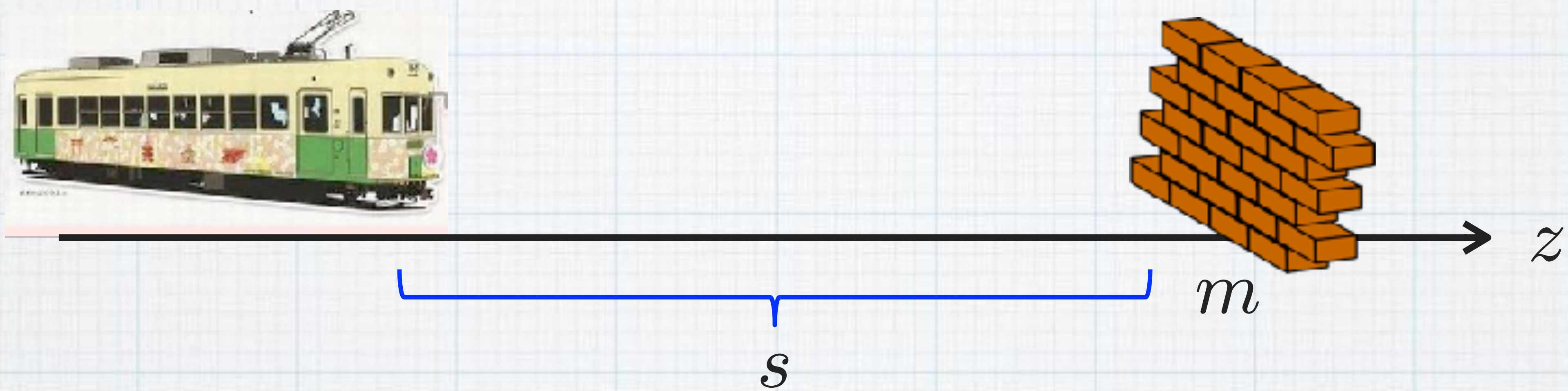
- **Static analysis techniques, transferred as they are**
  - Phase split [Sharma,Dillig,Dillig,Aiken; CAV'11] [Balakrishnan,Sankaranarayanan,Ivancic,Gupta; EMSOFT'09] [Gopan,Reps; SAS'07]
  - Differential invariant [Platzer,Clarke; CAV'08]
  - ... and more!

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
…

$$
\begin{array}{l}
\texttt{while } v > 0 \texttt{ do } \{ \\
\quad t := 0\,; \\
\quad \texttt{if } m - z < s \texttt{ then } a := -b \texttt{ else } a := a_0\,; \\
\quad \texttt{while } t < \varepsilon \texttt{ do } \{ \\
\qquad t := t + \mathrm{dt}\,; \\
\qquad v := v + a \cdot \mathrm{dt}\,; \\
\qquad z := z + v \cdot \mathrm{dt} \\
\quad \}\}
\end{array}
$$

ETCS_0

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

What We'll Be Doing (with dt's around)

(Assign)

$$\left\{ \begin{array}{l} k*n*((n-1)!)=N! \\ \wedge\ n-1\geq 0 \end{array} \right\}\ k:=k*n\ \left\{ \begin{array}{l} k*((n-1)!)=N! \\ \wedge\ n-1\geq 0 \end{array} \right\}$$

(Assign)

$$\left\{ \begin{array}{l} k*((n-1)!)=N! \\ \wedge\ n-1\geq 0 \end{array} \right\}\ n:=n-1\ \left\{ \begin{array}{l} k*(n!)=N! \\ \wedge\ n\geq 0 \end{array} \right\}$$

(SeqComp)

$$k*(n!)=N! \wedge n\geq 0 \wedge n>0 \Rightarrow k*n*((n-1)!)=N! \wedge n-1\geq 0$$

$$\left\{ \begin{array}{l} k*n*((n-1)!)=N! \\ \wedge\ n-1\geq 0 \end{array} \right\}\ k:=k*n;\ n:=n-1\ \left\{ \begin{array}{l} k*(n!)=N! \\ \wedge\ n\geq 0 \end{array} \right\}$$

loop invariant

(Conseq)

$$\left\{ \begin{array}{l} k*(n!)=N! \\ \wedge\ n\geq 0 \\ \wedge\ n>0 \end{array} \right\}\ k:=k*n;\ n:=n-1\ \left\{ \begin{array}{l} k*(n!)=N! \\ \wedge\ n\geq 0 \end{array} \right\}$$

(While)

$$k=1 \wedge n=N \Rightarrow k*(n!) = N! \wedge n\geq 0$$

$$\left\{ \begin{array}{l} k*(n!) = N! \\ \wedge\ n\geq 0 \end{array} \right\}\ \text{while } (n>0)\ k:=k*n;\ n:=n-1\ \left\{ \begin{array}{l} k*(n!)=N! \\ \wedge\ n\geq 0 \\ \wedge\ \neg(n>0) \end{array} \right\}$$

$$k*(n!)=N! \wedge n\geq 0 \wedge \neg(n>0) \Rightarrow k=N!$$

(Conseq)

$$\{\ k=1 \wedge n=N\ \}\ \text{while } (n>0)\ k:=k*n;\ n:=n-1\ \{\ k = N!\ \}$$

```
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:

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

while b_g do [...(if ...)...]

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.

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

$\Leftrightarrow$ 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)

$\Rightarrow$ 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 $\Updownarrow$

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

Transferring Static Analysis Strategies

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

$\Leftrightarrow$ 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)

$\Rightarrow$ std. ver.

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

⋮

sectionwise $\Updownarrow$

$[\![\, \texttt{while } b_g \texttt{ do } h[b_c] \,]\!] = [\![\, \texttt{while } (b_g \wedge \neg b_s) \texttt{ do } h[\texttt{false}]\,;\ \texttt{while } (b_g \wedge b_s) \texttt{ do } h[\texttt{true}] \,]\!]$

* 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}
```

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;
```

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

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

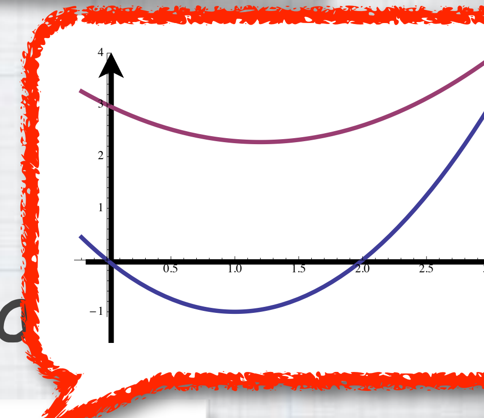
QE Invariant

Lem. In $\textsc{Hoare}^{dt}$,

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

$$\texttt{while}\ b\ \texttt{do}\ x := a\ \ \{A\}\ .$$

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;
```

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

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

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;
```

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

```
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 $\lfloor x0/a \rfloor$ or $\lceil x0/a \rceil$
* ➜ 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. $\boldsymbol{a}$ is *closed*
2. $\boldsymbol{r} \mapsto [\![\boldsymbol{a}[\boldsymbol{r}/\mathtt{dt}]]\!]$ is continuous at $\boldsymbol{r}=\boldsymbol{0}$,

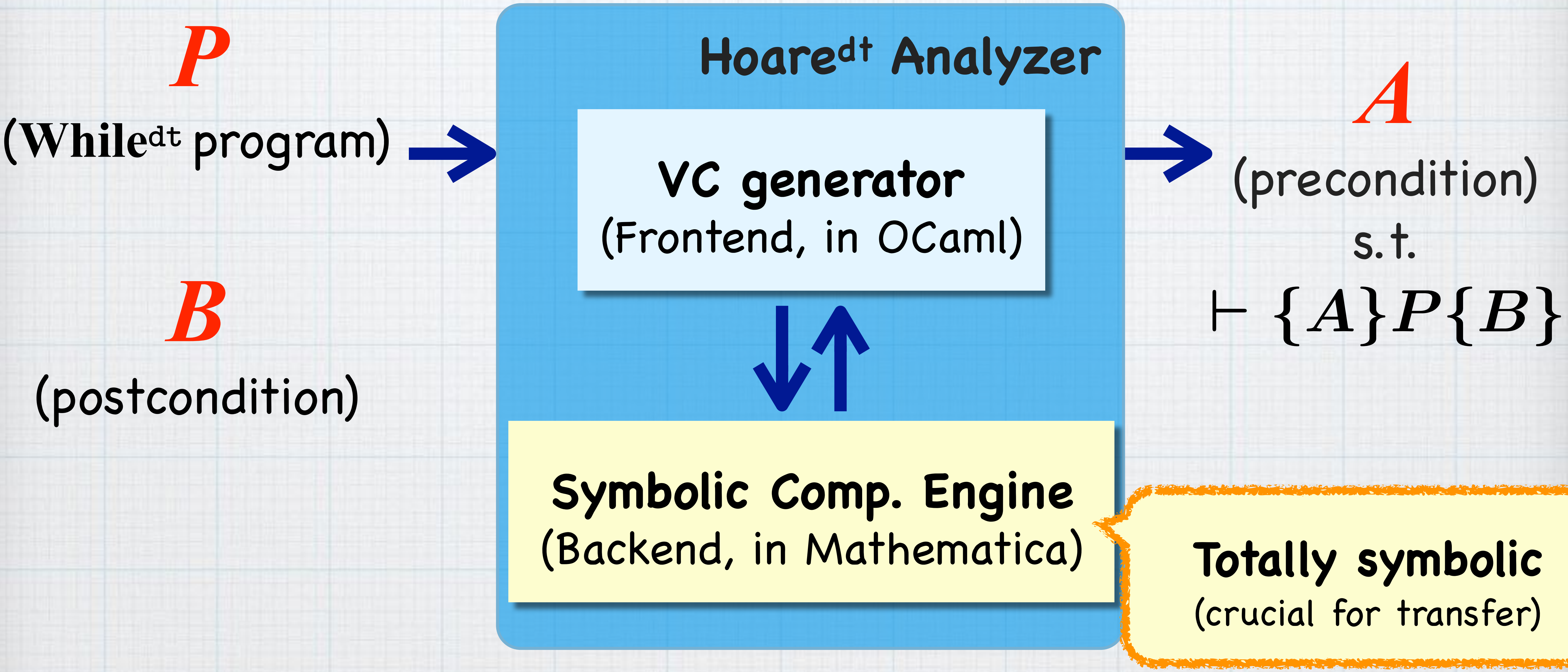
then $\models \boldsymbol{a}[0/\mathtt{dt}] < \boldsymbol{0} \Longrightarrow \boldsymbol{a} < \boldsymbol{0}$.

Strategy 7

"Cast to shadow"

(Eliminates `dt`, strengthens the precond.)

Prototype Automatic Prover



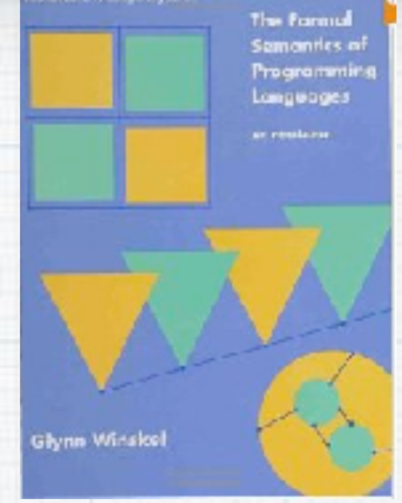
- Fujitsu HX600 with Quad Core AMD Opteron 2.3GHz CPU, 32GB memory. Mathematica 7.0 for Linux x86 (64-bit)
  - ETCS: 40.96 sec.
  - Bouncing ball: runs with one manual insertion of invariants

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?

Today's Talk: Framework

[Suenaga&H., ICALP'11]



The standard textbook [Winskel]

$\mathbf{While}^{\mathrm{dt}}$

Programming lang.

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

$\mathbf{Assn}^{\mathrm{dt}}$

First-order assertion lang.

```
∃z(x=2*z ∧ y=3*z)
```

$\mathbf{Hoare}^{\mathrm{dt}}$

Hoare-style program logic

$$\frac{\{A \land b\}\; c\; \{A\}}{\{A\}\; \texttt{while}\; b\; \texttt{do}\; c\; \{A \land \neg b\}}$$

Rigorous semantics by non-standard analysis

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

Nonstandard Static Analysis: Conclusions

We're hiring!

- Discrete + dt => continuous/hybrid
  - Rigorous semantics by NSA
  - Deductive verification & static analysis are still valid
- Stream/signal processing (POPL'13), abstract interpretation (VMCAI'16)
- Pro: everything is discrete
  Con: everything is discrete
  - Scalability is an issue
    => rather a theoretical vehicle?

[Suenaga & Hasuo, ICALP'11]
[Hasuo & Suenaga, CAV'12]
[Suenaga, Sekine & Hasuo, POPL'13]
[Kido, Chaudhuri & Hasuo, VMCAI'16]

Thank you for your attention!

Ichiro Hasuo (NII, Tokyo)
http://group-mmm.org/~ichiro/