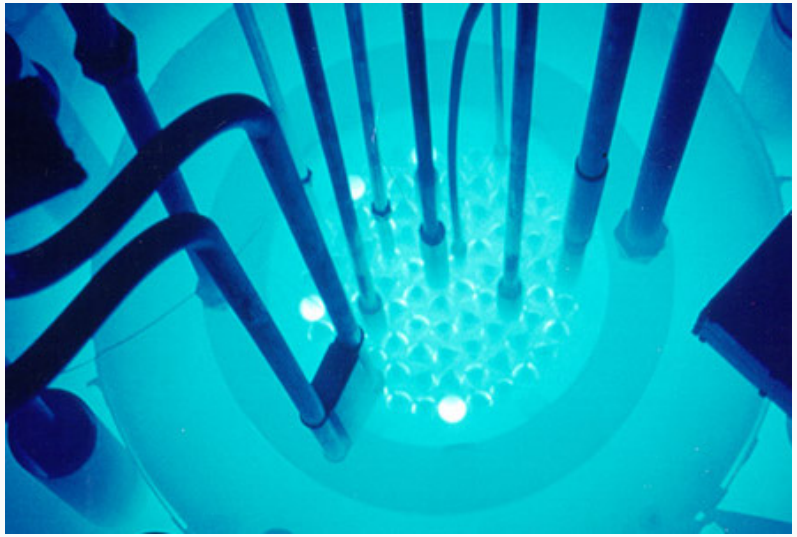
Implementing Hybrid Semantics

Sergey Goncharov[a], **Renato Neves**[b], José Proença[c]

[a]Friedrich-Alexander-Universität Erlangen-Nürnberg
[b]University College London & INESC TEC
[c]CISTER Research Centre, ISEP

Hybrid systems

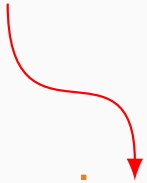
Computational objects that closely interact with physical processes



Hybrid Programming

Hybrid programming

compositionality, Hoare calculi, different styles of semantics . . .



Applies principles and techniques from programming theory to model and analyse hybrid systems

Differential equations are added to the palette of classical program constructs

(Cruise controller)

The main contribution

A mathematical basis for ‘running’ hybrid programs

We introduce an imperative, hybrid while-language

with a runnable operational semantics

and with denotational semantics, connected to the former by

soundness and adequacy

Online prototype implementation $\Longrightarrow$ Lince [1]

[1] `http://arcatools.org/lince`

Related work

Previous work on hybrid semantics

hybrid automata

Most of the previous work on hybrid semantics is inspired by automata theory and Kleene algebra

Differential dynamic logic

Such approaches inherit non-deterministic behaviour which hinders simulation capabilities

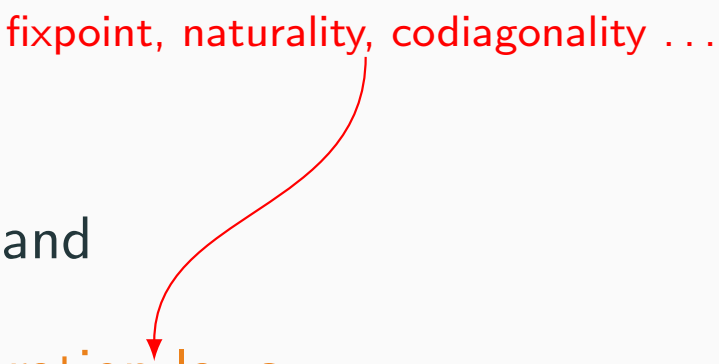
Moreover, they lead to a non-refined view of divergence

$\ldots \; \texttt{while}\,(\texttt{true})\,\texttt{do}\,\{\,\texttt{p}\,\} = 0$

In our work

We aim at a purely hybrid semantics

i.e. hybrid computation as an independent computational paradigm



Iteration still as a least fixpoint and

satisfying standard axiomatic iteration laws

Main features

Operational semantics

The language

Fix a stock of n-variables $\mathcal{X}$

Linear terms

$\mathtt{LTerm}(\mathcal{X}) \ni \mathtt{r} \mid \mathtt{r} \cdot \mathtt{x} \mid \mathtt{t} + \mathtt{s}$

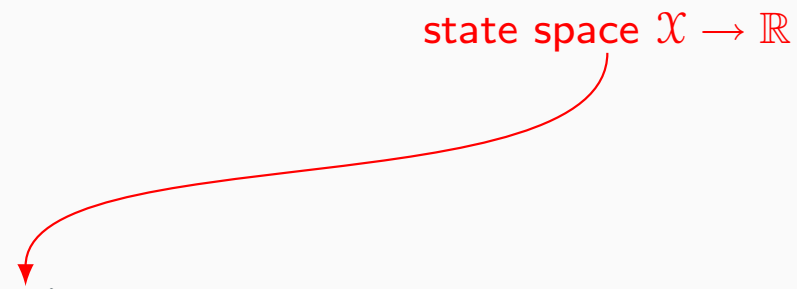
Atomic programs

$\mathtt{At}(\mathcal{X}) \ni \mathtt{x} := \mathtt{t} \mid \mathtt{x}'_1 = \mathtt{t}_1, \ldots, \mathtt{x}'_n = \mathtt{t}_n \texttt{ for } \mathtt{t}$

Hybrid programs

$\mathtt{Prog}(\mathcal{X}) \ni \mathtt{a} \mid \mathtt{p} \,;\, \mathtt{q} \mid \texttt{if } \mathtt{b} \texttt{ then } \mathtt{p} \texttt{ else } \mathtt{q} \mid \texttt{while } \mathtt{b} \texttt{ do } \{\, \mathtt{p} \,\}$

Small-step operational semantics pt. I



Classical programming: $\mathrm{p}, \sigma \longrightarrow \mathrm{p}', \sigma'$

Hybrid programming: $\mathrm{p}, \sigma, \mathtt{t} \longrightarrow \mathrm{p}', \sigma', \mathtt{t}'$

The value $\mathtt{t}$ tells how far we are in time from the time instant at which we evaluate p

Special flag to tell whether we surpassed this time instant

Small-step operational semantics pt. II

Interpretation of $\mathtt{u}$ according to σ

Solution of diff. eq. with initial cond. σ

$$\mathtt{x} := \mathtt{u}, \sigma, t \;\rightarrow\; \mathtt{skip}, \sigma\nabla[\mathtt{u}\sigma/\mathtt{x}], t$$

$$\bar{\mathtt{x}}' = \bar{\mathtt{u}}\;\mathtt{for}\;\mathtt{d}, \sigma, t \;\rightarrow\; \mathtt{stop}, \sigma\nabla[\phi_\sigma(t)/\bar{\mathtt{x}}], 0 \qquad \text{if} \quad t < \mathtt{d}\sigma$$

$$\bar{\mathtt{x}}' = \bar{\mathtt{u}}\;\mathtt{for}\;\mathtt{d}, \sigma, t \;\rightarrow\; \mathtt{skip}, \sigma\nabla[\phi_\sigma(\mathtt{d}\sigma)/\bar{\mathtt{x}}], t - \mathtt{d}\sigma \qquad \text{if} \quad t \geq \mathtt{d}\sigma$$

Unfolds into $\mathtt{x}'_1 = \mathtt{u}_1, \ldots, \mathtt{x}'_n = \mathtt{u}_n$

$$\frac{\mathtt{p}, \sigma, t \;\rightarrow\; \mathtt{stop}, \sigma', t'}{\mathtt{p}; \mathtt{q}, \sigma, t \;\rightarrow\; \mathtt{stop}, \sigma', t'} \qquad \frac{\mathtt{p}, \sigma, t \;\rightarrow\; \mathtt{skip}, \sigma', t'}{\mathtt{p}; \mathtt{q}, \sigma, t \;\rightarrow\; \mathtt{q}, \sigma', t'}$$

Small-step operational semantics pt. III

The reduction process of an infinite while-loop

$$\texttt{while}\ \texttt{true}\ \texttt{do}\ \{\ \texttt{p}\ \}$$

terminates if the latter is non-Zeno

See a step-by-step illustration in the paper (Remark 2)

Definition (Zeno)

A system is Zeno if it iterates infinitely many times in finite time.

Future work

Future work

- Program equivalence:

$$(\mathtt{x}' = 1 \,\mathtt{for}\ 1); (\mathtt{x}' = 1 \ \mathtt{for}\ 1) = \mathtt{x}' = 1 \ \mathtt{for}\ 2$$

- New program constructs: $\mathtt{x}' =_{1/2} 1 \ \mathtt{for}\ 1$

- Robustness: small input variations should not result in big output changes

References i