Introduction to the signal clock calculus

Alexander Steen

Institut für Informatik
Freie Universität Berlin

February 14, 2013

Contents

Signals

Definition (Signal)

Let D be some domain. A *signal* X is a sequence $X = (x_i)_{i \in I}$ of values $x_i \in D$.

A signal X is

- present at instant t, if X carries a value at that instant.
- absent at instant t, if X carries *no* value at that instant (write: $X = \perp$).

Freie Universität Berlin

Signals

Definition (Signal)

Let D be some domain. A *signal* X is a sequence $X = (x_i)_{i \in I}$ of values $x_i \in D$.

A signal X is

- present at instant t, if X carries a value at that instant.
- absent at instant t, if X carries *no* value at that instant (write: $X = \perp$).

Signals

Definition (Signal)

Let D be some domain. A *signal* X is a sequence $X = (x_i)_{i \in I}$ of values $x_i \in D$.

A signal X is

- present at instant t, if X carries a value at that instant.
- absent at instant t, if X carries *no* value at that instant (write: $X = \perp$).

Signals

Example

Example

As an example, three signals X, Y, Z are given by the table below.

X	*true*	*false*	$\perp$	*true*	*true*	$\perp$	*true*	*false*	
Y	*false*	*true*	$\perp$	$\perp$	*true*	$\perp$	*false*	*true*	
Z	*false*	*true*	$\perp$	*true*	*false*	$\perp$	*false*	*true*	
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	$\cdots$

Clocks

Definition (Clock)

Let c be the set of instants, at which the signal X is present.
Then c is called the *clock* X, denoted $\widehat{X}$.

Described formally by clock algebra

$$\mathcal{H} = (U, \cup, \cap, \setminus, \mathbb{O})$$

Clocks

Definition (Clock)

Let c be the set of instants, at which the signal X is present.
Then c is called the *clock* X, denoted $\widehat{X}$.

Described formally by clock algebra

$$\mathcal{H} = (U, \cup, \cap, \setminus, \mathbb{O})$$

Clocks

Boolean clocks

For a Boolean signal C, let

- $[C]$ the set of instants at which C is present and *true*
- $[\neg C]$ the set of instants at which C is present and *false*

$\rightsquigarrow$ It holds that $[C] \cup [\neg C] = \widehat{C}$ and $[C] \cap [\neg C] = \mathbb{O}$.

Freie Universität Berlin

Clocks

Boolean clocks

For a Boolean signal C, let

- $[C]$ the set of instants at which C is present and *true*
- $[\neg C]$ the set of instants at which C is present and *false*

$\rightsquigarrow$ It holds that $[C] \cup [\neg C] = \widehat{C}$ and $[C] \cap [\neg C] = \mathbb{0}$.

Clocks

Boolean clocks

For a Boolean signal C, let

- $[C]$ the set of instants at which C is present and *true*
- $[\neg C]$ the set of instants at which C is present and *false*

$\rightsquigarrow$ It holds that $[C] \cup [\neg C] = \widehat{C}$ and $[C] \cap [\neg C] = \mathbb{O}$.

Clocks

Example

Example

Recall the example signals X, Y, Z.
Its easy to see that $\widehat{X} = \widehat{Z} \subseteq \widehat{Y}$.

X	*true*	*false*	$\perp$	*true*	*true*	$\perp$	*true*	*false*	
Y	*false*	*true*	$\perp$	$\perp$	*true*	$\perp$	*false*	*true*	
Z	*false*	*true*	$\perp$	*true*	*false*	$\perp$	*false*	*true*	
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	$\cdots$

The Signal language

Functions Let $X := f(X_1, X_2, \ldots, X_n)$, s.t.

- X is present $\Leftrightarrow$ all X_i are present
- $x^t = f(x_1^t, x_2^t, \ldots, x_n^t)$ for all $t \in \widehat{X}$

Delay Let $X := Y \ \$ \ n \ \texttt{init} \ \overrightarrow{v}$, s.t.

- X is present $\Leftrightarrow$ Y is present
- $x^{t+n} = y^t$ for $t > n$, v_i otherwise.

The Signal language

Functions Let $X := f(X_1, X_2, \ldots, X_n)$, s.t.

- X is present $\Leftrightarrow$ all X_i are present
- $x^t = f(x_1^t, x_2^t, \ldots, x_n^t)$ for all $t \in \widehat{X}$

Delay Let $X := Y \ \texttt{\$} \ n \ \texttt{init} \ \overrightarrow{v}$, s.t.

- X is present $\Leftrightarrow$ Y is present
- $x^{t+n} = y^t$ for $t > n$, v_i otherwise.

Freie Universität Berlin

The Signal language

Downsampling Let $X := Y$ `when` C, s.t.

- X is present $\Leftrightarrow$ Y present and $C = \mathit{true}$
- $x^t = y^t$ for all $t \in \widehat{X}$

Det. merge Let $X := Y$ `default` Z, s.t.

- X is present $\Leftrightarrow$ Y or Z are present
- $x^t = y^t$ if Y is present, z^t otherwise.

Composition The $|$ operator forces that all its operands equations must hold in parallel

The Signal language

Downsampling Let $X := Y$ `when` C, s.t.

- X is present $\Leftrightarrow$ Y present and $C = \mathit{true}$
- $x^t = y^t$ for all $t \in \widehat{X}$

Det. merge Let $X := Y$ `default` Z, s.t.

- X is present $\Leftrightarrow$ Y or Z are present
- $x^t = y^t$ if Y is present, z^t otherwise.

Composition The | operator forces that all its operands equations must hold in parallel

Freie Universität Berlin

The Signal language

Downsampling Let $X := Y$ `when` C, s.t.

- X is present $\Leftrightarrow$ Y present and $C = \mathit{true}$
- $x^t = y^t$ for all $t \in \widehat{X}$

Det. merge Let $X := Y$ `default` Z, s.t.

- X is present $\Leftrightarrow$ Y or Z are present
- $x^t = y^t$ if Y is present, z^t otherwise.

Composition The $|$ operator forces that all its operands equations must hold in parallel

The Signal clock calculus

Overview

Compilation techniques for

- Extract synchronization contraints
- Extract data-dapendency
- Solve clock equations
- Generate control structure

The Signal clock calculus

Overview

Compilation techniques for

- Extract synchronization contraints
- Extract data-dependency
- Solve clock equations
- Generate control structure

Freie Universität Berlin

The Signal clock calculus

Overview

Compilation techniques for

- Extract synchronization contraints
- Extract data-dependency
- Solve clock equations
- Generate control structure

The Signal clock calculus

Overview

Compilation techniques for

- Extract synchronization contraints
- Extract data-dependency
- Solve clock equations
- Generate control structure

The Signal clock calculus

Overview

Compilation techniques for

- Extract synchronization contraints
- Extract data-dependency
- Solve clock equations
- Generate control structure

The Signal clock calculus

Motivation

We want

- know if specification is consistent
- Automatic code generation
- Code that respects synchronization
- Efficient code

The Signal clock calculus

Motivation

We want

- know if specification is consistent
- Automatic code generation
- Code that respects synchronization
- Efficient code

The Signal clock calculus

Motivation

We want

- know if specification is consistent
- Automatic code generation
- Code that respects synchronization
- Efficient code

The Signal clock calculus
Motivation

We want

- know if specification is consistent
- Automatic code generation
- Code that respects synchronization
- Efficient code

The Signal clock calculus

Synchronizations

Process	**Synchronisation**
$X := f(X_1, X_2, \ldots, X_n)$	$\widehat{X} = \widehat{X_1} = \ldots = \widehat{X_n}$
$X := U \texttt{ when } C$	$\widehat{X} = \widehat{U} \cup [C]$, $\begin{cases} [C] \cup [\neg C] = \widehat{C} \\ [C] \cap [\neg C] = \mathbb{O} \end{cases}$
$X := U \texttt{ default } V$	$\widehat{X} = \widehat{U} \cup \widehat{V}$
$X := Y \texttt{ \$ } n \texttt{ init } \overrightarrow{v}$	$\widehat{X} = \widehat{Y}$

The Signal clock calculus

Data-dependency

Process	**Dependency**
each signal X	$\widehat{X} \xrightarrow{\widehat{X}} X$
$X := f(X_1, X_2, \ldots, X_n)$	$X_i \xrightarrow{\widehat{X}} X$,f.a. $1 \leq i \leq n$
$X := U$ `when` C	$U \xrightarrow{\widehat{U} \cap [C]} X$
$X := U$ `default` V	$U \xrightarrow{\widehat{U}} X \xleftarrow{\widehat{V} \setminus \widehat{U}} V$
$X := Y$ `$` n `init` $\overrightarrow{v}$	none

The Signal clock calculus

Code generation

Generation for e.g. $X := U$ `default` V given by

```
if present(X) then
  if present(U) then
    X := U
  end if
  if present(V) and not present(U) then
    X := V
  end if
end if
```

The Signal clock calculus

Equation systems

Example (Equation system of clocks)

$$\begin{cases} \widehat{C} & = \widehat{C'} \\ \widehat{C'} & = [D] \cup [C_1] \cup \widehat{C} \\ [C] & = \widehat{C_1} = \widehat{C_2} \\ [\neg C] & = \widehat{D} \\ \widehat{C_3} & = \widehat{C_1} = \widehat{C_2} \end{cases}$$

$$\begin{cases} \widehat{C'} & = \widehat{C} \\ \widehat{C_1} & = [C] \\ \widehat{C_2} & = [C] \\ \widehat{C_3} & = [C] \\ \widehat{D} & = [\neg C] \\ \widehat{C} & = [D] \cup [C_1] \cup \widehat{C} \end{cases}$$

The Signal clock calculus

Equation systems

Example (Equation system of clocks)

$$\begin{cases} \widehat{C} & = \widehat{C'} \\ \widehat{C'} & = [D] \cup [C_1] \cup \widehat{C} \\ [C] & = \widehat{C_1} = \widehat{C_2} \\ [\neg C] & = \widehat{D} \\ \widehat{C_3} & = \widehat{C_1} = \widehat{C_2} \end{cases} \quad \begin{cases} \widehat{C'} & = \widehat{C} \\ \widehat{C_1} & = [C] \\ \widehat{C_2} & = [C] \\ \widehat{C_3} & = [C] \\ \widehat{D} & = [\neg C] \\ \widehat{C} & = [D] \cup [C_1] \cup \widehat{C} \end{cases}$$

Freie Universität Berlin

The Signal clock calculus

Equation systems

Example (Equation system of clocks (Cont.))

Solution given by $\begin{cases} \widehat{C'} & = \widehat{C} \\ \widehat{C_1} & = [C] \\ \widehat{C_2} & = [C] \\ \widehat{C_3} & = [C] \\ \widehat{D} & = [\neg C] \end{cases}$

Freie Universität Berlin

The Signal clock calculus

Solving equation systems

Idea:

- Encode equation in a tree structure, called *clock trees*

General approach:

- Start with forest of clock trees
- Fusion of clock trees
- Repeat until inpossible

The Signal clock calculus

Solving equation systems

Idea:

- Encode equation in a tree structure, called *clock trees*

General approach:

- Start with forest of clock trees
- Fusion of clock trees
- Repeat until inpossible

The Signal clock calculus

Solving equation systems

Idea:

- Encode equation in a tree structure, called *clock trees*

General approach:

- Start with forest of clock trees
- Fusion of clock trees
- Repeat until inpossible

The Signal clock calculus

Solving equation systems

Idea:

- Encode equation in a tree structure, called *clock trees*

General approach:

- Start with forest of clock trees
- Fusion of clock trees
- Repeat until inpossible

Conclusion

- Clock calculus: Rich set of techniques
- Extract synchronizations and data-dependency
- Solve clock equations
- Generate executable code

Conclusion

- Clock calculus: Rich set of techniques
- Extract synchronizations and data-dependency
- Solve clock equations
- Generate executable code

Conclusion

- Clock calculus: Rich set of techniques
- Extract synchronizations and data-dependency
- Solve clock equations
- Generate executable code

Conclusion

- Clock calculus: Rich set of techniques
- Extract synchronizations and data-dependency
- Solve clock equations
- Generate executable code

Conclusion

- Clock calculus: Rich set of techniques
- Extract synchronizations and data-dependency
- Solve clock equations
- Generate executable code

References I

Nebut, Mirabelle.
An Overview of the Signal Clock Calculus.
Electron. Notes Theor. Comput. Sci., 88: 39–54, October, 2004.

Amagbegnon, Tochéou et al.
Arborescent Canonical Form of Boolean Expressions, 1994.

Amagbégnon, Pascalin.
Implementation of the data-flow synchronous language SIGNAL.
SIGPLAN Not., 30(6): 163–173, June, 1995.

Freie Universität Berlin