Some Title

Bernd Finkbeiner, Felix Klein, Ruzica Piskac, Mark Santolucito

Abstract. TODO

1 Introduction

We present a Temporal Stream Logic (TSL), to argue about function application and control flow over streams. We use this logic to sythesize FRP programs in language called Casual Commutative Normal Form.

To show soundness of the synthesis procedure, we follow the paths through Fig. 1. We first provide a semantics of TSL to intepret TSL specifications as infinite input/output streams. We use this as a common formalism to argue about the behavior of the synthesized FRP program, which can also be expressed as an infinite stream of input/output values. To show that (TSL → IO) = (CCNF → IO), we use intermidiary Mealy Machines. We first show (TSL → Mealy → IO) = (TSL → IO), then show (Mealy → CCNF → IO) = (Mealy → IO).

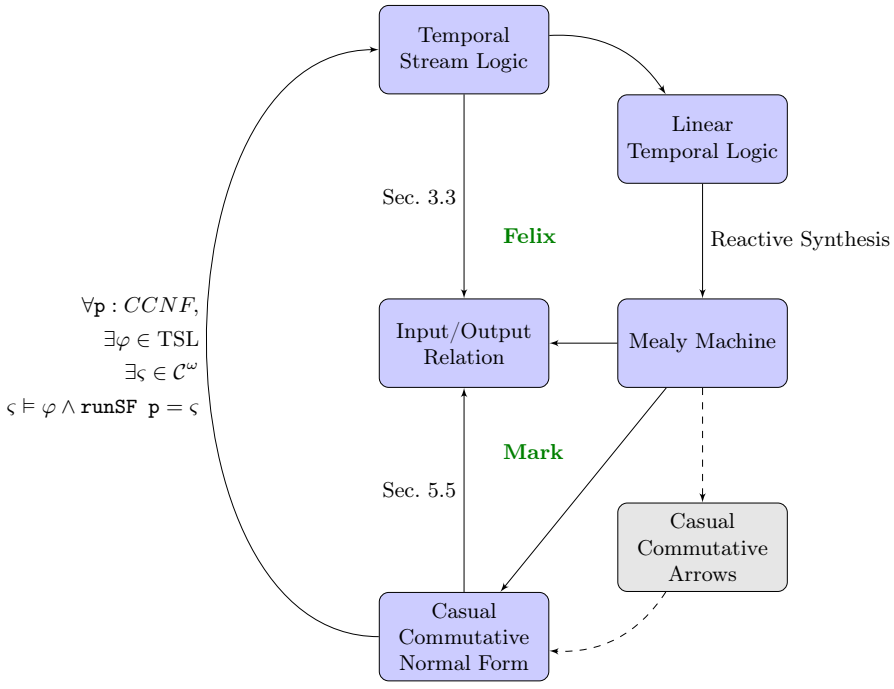

Fig. 1. A map of the relations explained in this paper.

2 Preliminaries

3 Temporal Stream Logic

3.1 Values, Signals, Functions and Predicates

- The set $\mathcal{V}$ denotes a possibly infinite set of values.
- A signal $s\colon \mathcal{T}\!\mathit{ime} \to \mathcal{V}$ is a function fixing a value of $\mathcal{V}$ at each point in time. We assume that time is discrete, i.e., $\mathcal{T}\!\mathit{ime} = \mathbb{N}$.
- $\mathcal{S}_I$ denotes a finite set of input signals.
- $\mathcal{S}_O$ denotes a finite set of output signals.
- $\mathcal{S} = \mathcal{S}_I \uplus \mathcal{S}_O$ is the set of all signals.
- An n-ary function $f\colon \mathcal{V}^n \to \mathcal{V}$ determines a new value from n given values. We denote the set of all functions (of arbitrary arity) by $\mathcal{F}$.
- An n-ary predicate $p\colon \mathcal{V}^n \to \mathcal{B}$ checks a truth statement on n given values. We denote the set of all predicates (of arbitrary arity) by $\mathcal{P}$.

We consider the set of input signals $\mathcal{S}_I$ as uncontrollable, i.e., they are inputs from the outside and can only be read. On the other hand, the output signals $\mathcal{S}_O$ can be read and written, where, if an output is written to at time t, then the written value can be read at time $t+1$. We do not distinguish between pure outputs, i.e., outputs that are write-only, and read+write outputs, which simplifies our overall setting. To model internal signals, we just use normal output signals, that are then projected away afterwards.

3.2 Terms

Let a set of names $\mathcal{N} = \mathcal{N}_I \uplus \mathcal{N}_O \uplus \mathcal{N}_F \uplus \mathcal{N}_P$, partitioned into input signal names, output signal names, function names and predicate names, respectively, be given. We distinguish between the following types of terms:

Function Terms:

$$\tau_F \quad := \quad \mathtt{s} \quad | \quad \mathtt{f}(\tau_F^0, \tau_F^1, \ldots, \tau_F^{n-1})$$

A function term is either a signal name $\mathtt{s} \in \mathcal{N}_I \cup \mathcal{N}_O$ or a function name $\mathtt{f} \in \mathcal{N}_F$, which corresponds to an n-ary function and which is applied to n sub-function terms τ_F^0 to τ_F^{n-1}. The set of all function terms is denoted by $\mathcal{T}_F$.

Predicate Terms:

$$\tau_P \quad := \quad \mathtt{p}(\tau_F^0, \tau_F^1, \ldots, \tau_F^{n-1})$$

A predicate term consists of a predicate name $\mathtt{p} \in \mathcal{N}_P$, which corresponds to an n-ary predicate and which is applied to sub-function terms $\tau_F^0, \tau_F^1, \ldots, \tau_F^{n-1} \in \mathcal{T}_F$. The set of all predicate terms is denoted by $\mathcal{T}_P$.

Update Terms:

$$\tau_U \quad := \quad [\, \mathtt{s} \lhd \tau_F \,]$$

An update term consists of a signal name $\mathtt{s} \in \mathcal{N}_O$ and a function term $\tau_F \in \mathcal{T}_F$. The set of all update terms is denoted by $\mathcal{T}_\lhd$.

We give semantics to a term τ by giving semantics to signal, function and predicate names appearing inside τ. To this end, we use an assignment function $\langle\cdot\rangle \colon \mathcal{N} \to \mathcal{S} \cup \mathcal{F} \cup \mathcal{P}$, which assigns each name either a signal, a function, or a predicate, respectively.

3.3 Computations

Let $\langle\cdot\rangle \colon \mathcal{N} \to \mathcal{S} \cup \mathcal{F} \cup \mathcal{P}$ be given. A computation step $c \colon \mathcal{N}_O \to \mathcal{T}_F$ is a function assigning each output signal name $\mathtt{s}$ a function term $\tau_F \in \mathcal{T}_F$. We denote the set of all computation steps by $\mathcal{C}$. The set of all infinite sequences of computation steps, which we call a computation, is defined as $\mathcal{C}^\omega$. Respectively, we denote a computation as a function $\varsigma \colon \mathcal{T}\!\mathit{ime} \to \mathcal{C}$.

Let $\iota \colon \mathcal{N}_O \to \mathcal{V}$ be an initialization function, assigning each output signal name an initial value. We define the semantics to a computation $\varsigma \in \mathcal{C}^\omega$, denoted by $[\![\varsigma]\!]$, via defining an assignment of each signal name $\mathtt{s} \in \mathcal{N}_O$ to a signal $s \colon \mathcal{T}\!\mathit{ime} \to \mathcal{V}$. The signal s is defined inductively over $t \in \mathcal{T}\!\mathit{ime}$ via $s(t) = \eta(\varsigma, t, \varsigma(t)(\mathtt{s}))$, where the function $\eta(\varsigma, t, \tau)$ is defined by:

$$\eta(\varsigma, t, \mathtt{s}) = \begin{cases} \langle \mathtt{s} \rangle(t) & \text{if } \mathtt{s} \in \mathcal{N}_I \\ \iota(\mathtt{s}) & \text{if } \mathtt{s} \in \mathcal{N}_O \ \wedge \ t = 0 \\ \eta(\varsigma, t-1, \varsigma(t-1)(\mathtt{s})) & \text{if } \mathtt{s} \in \mathcal{N}_O \ \wedge \ t > 0 \end{cases}$$

$$\eta(\varsigma, t, \mathtt{x}(\tau_0, \tau_1, \ldots, \tau_{m-1})) = \langle \mathtt{x} \rangle(\eta(\varsigma, t, \tau_0), \eta(\varsigma, t, \tau_1), \ldots, \eta(\varsigma, t, \tau_{m-1}))$$

Note that the semantics of a computation ς are independent from $\langle \mathcal{N}_O \rangle$. Hence, for fixed $\mathcal{F}$ and $\mathcal{P}$, we can consider $[\![\varsigma]\!]$ as a function:

$$[\![\varsigma]\!]_{\mathcal{F},\mathcal{P}} \colon (\mathcal{N}_I \to \mathcal{S}_I) \to (\mathcal{N}_O \to \mathcal{S}_O)$$

3.4 TSL Syntax

$$\varphi \quad := \quad \tau \in \mathcal{T}_P \cup \mathcal{T}_\lhd \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi \quad | \quad \bigcirc \varphi \quad | \quad \varphi \,\mathcal{U}\, \varphi$$

3.5 TSL Semantics

Let $\langle\cdot\rangle$, ι, and $\varsigma \in \mathcal{C}^\omega$ given. The validity of a TSL formula φ with respect to ς is defined inductively over $t \in \mathcal{T}\!\mathit{ime}$ via:

$$\begin{aligned}
\varsigma, t \vDash \mathtt{p}(\tau_0, \tau_1, \ldots, \tau_{m-1}) \quad &:\Leftrightarrow \quad \eta(\varsigma, t, \mathtt{p}(\tau_0, \tau_1, \ldots, \tau_{m-1})) \\
\varsigma, t \vDash [\,\mathtt{s} \lhd \tau\,] \quad &:\Leftrightarrow \quad \varsigma(t)(\mathtt{s}) = \tau \\
\varsigma, t \vDash \neg\psi \quad &:\Leftrightarrow \quad \varsigma, t \nvDash \psi \\
\varsigma, t \vDash \vartheta \wedge \psi \quad &:\Leftrightarrow \quad \varsigma, t \vDash \vartheta \ \wedge\ \varsigma, t \vDash \psi \\
\varsigma, t \vDash \bigcirc\psi \quad &:\Leftrightarrow \quad \varsigma, t+1 \vDash \psi \\
\varsigma, t \vDash \vartheta\,\mathcal{U}\,\psi \quad &:\Leftrightarrow \quad \exists t'' \geq t.\ \ \varsigma, t'' \vDash \vartheta \ \wedge\ \forall t \leq t' < t''.\ \varsigma, t' \vDash \psi
\end{aligned}$$

We say that ς satisfies φ, denoted by $\varsigma \vDash \varphi$, if $\varsigma, 0 \vDash \varphi$. The language $\mathcal{L}(\varphi)$ of φ is defined as $\mathcal{L}(\varphi) = \{\varsigma \in \mathcal{C}^\omega \mid \varsigma \vDash \varphi\}$.

3.6 Realizability

We are interested in the following realizability problem. Given a TSL formula φ, is there some computation ς such that $\varsigma \vDash \varphi$, independent of the choice of $\langle\cdot\rangle$ and ι, i.e.,

$$\exists \varsigma \in \mathcal{C}^\omega.\ \ \forall \langle\cdot\rangle \colon \mathcal{N} \to \mathcal{S} \cup \mathcal{F} \cup \mathcal{P}.\ \ \forall \iota \colon \mathcal{N}_O \to \mathcal{V}.\quad \varsigma \vDash \varphi$$

If such a ς exists, we also say ς realizes φ.

3.7 Decidability

Theorem 1. *The realizability problem of TSL is undecidable.*

Proof. We give a reduction from the Post Correspondence Problem (PCP): given two finite lists $w_0 w_1 \ldots w_n$ and $v_0 v_1 \ldots v_n$ of equal length, each containing n finite words over some finite alphabet Σ, is there some finite sequence $i_0 i_1 \ldots i_k \in \mathbb{N}^*$ such that4 $w_{i_0} w_{i_1} \ldots w_{i_k} = v_{i_0} v_{i_1} \ldots v_{i_k}$. The problem is undecidable [**?**].

We reduce PCP to the realizability question of TSL, where we translate an arbitrary instance of PCP to a TSL formula φ, which is realizable if and only if there is a solution to the PCP instance. To this end, let $n \in \mathbb{N}$, $w_0 w_1 \ldots w_n \in \Sigma^*$ and $v_0 v_1 \ldots v_n \in \Sigma^*$ be given. We fix $\mathcal{N}_P = \{\mathtt{p}\}$ for some unary predicate p, $\mathcal{N}_F = \Sigma \cup \{\mathtt{X}\}$, where every $\mathtt{f} \in \Sigma$ corresponds to a unary function and $\mathtt{X}$ corresponds to a 0-nary function, $\mathcal{N}_I = \emptyset$ and $\mathcal{N}_O = \{\mathtt{A}, \mathtt{B}\}$. We define φ via:

$$\begin{aligned}
\varphi \ = \quad & \Big([\,\mathtt{A} \lhd \mathtt{X}\,] \ \wedge\ [\,\mathtt{B} \lhd \mathtt{X}\,] \Big) \ \wedge \\
& \bigcirc\Box \Big(\bigvee_{j=0}^{n} \big([\,\mathtt{A} \lhd \mu(w_j, \mathtt{A})\,] \wedge [\,\mathtt{B} \lhd \mu(v_j, \mathtt{B})\,] \big) \Big) \ \wedge \\
& \bigcirc\bigcirc\Diamond \Big(\mathtt{p}(\mathtt{A}) \leftrightarrow \mathtt{p}(\mathtt{B}) \Big)
\end{aligned}$$

where $\mu(x_0x_1 \dots x_m, s) = x_0(x_1(\dots x_m(s)\dots))$.

Intuitively, we first assign the signals $\mathtt{A}$ and $\mathtt{B}$ a constant base value. Then, from the next time step on, we have to pick pairs (w_j, v_j) in every time step. Our choice is stored in the signals $\mathtt{A}$ and $\mathtt{B}$, respectively. Finally, we check that the constructed sequences of function applications are equal at some point in time, where we use the universally quantified predicate $\mathtt{p}$ to check equality.

The TSL formula φ is realizable if and only if there is an index sequence $i_0i_1 \dots i_k$ such that $w_{i_0}w_{i_1} \dots w_{i_k} = v_{i_0}v_{i_1} \dots v_{i_k}$:

"$\Rightarrow$": Assume that φ is realizable, i.e., there is some computation $\varsigma = c_0c_1 \dots$ that satisfies φ independent of the choices for $\langle\cdot\rangle$ and ι. We fix $\iota\colon \{\mathtt{A}, \mathtt{B}\} \to \{\varepsilon\}$, $\langle \mathtt{X} \rangle = \varepsilon$ and $\langle x \rangle\colon \Sigma^* \to \Sigma^*$ with $\langle x \rangle(w) = wx$ for all $w \in \Sigma^*$ and $x \in \Sigma$. We do not fix any assignment to $\mathtt{p}$. Nevertheless, by the satisfiability of φ, there is a position $m > 1$ at which $\mathtt{p}(\mathtt{A}) \leftrightarrow \mathtt{p}(\mathtt{A})$ is satisfied, independent of the predicate assigned to $\mathtt{p}$. We obtain that $[\![\varsigma]\!](\mathtt{A})(m) = [\![\varsigma]\!](\mathtt{B})(m)$, since otherwise there would be a predicate that detects the difference[1]. As there is no other influence on ς, depending on the choice of $\mathtt{p}$, we obtain that the semantics of ς

$$[\![\varsigma]\!] \;=\; \{[\![\varsigma]\!](\mathtt{A}) \mapsto a_0, [\![\varsigma]\!](\mathtt{B}) \mapsto b_0\}\, \{[\![\varsigma]\!](\mathtt{A}) \mapsto a_1, [\![\varsigma]\!](\mathtt{B}) \mapsto b_1\} \dots$$

are well defined (even without fixing $\mathtt{p}$) and induce the sequences $a_0a_1 \dots \in \Sigma^\omega$ and $b_0b_1 \dots \in \Sigma^\omega$. First, we observe that $a_0 = a_1 = b_0 = b_1 = \epsilon$ by the definition of φ and the choice of ι. From the choice of $\langle\cdot\rangle$, we also obtain that $a_t = a_{t-1}w_{i_t}$ and $b_t = b_{t-1}v_{i_t}$ for every $t > 1$ and some $0 \leq i_t \leq n$. A simple induction shows that every $a_t = w_{i_2}w_{i_3} \dots w_{i_t}$ and $b_t = v_{i_2}v_{i_3} \dots v_{i_t}$ for every $t > 1$. It follows that $w_{i_2}w_{i_3} \dots w_{i_{m-1}} = v_{i_2}v_{i_3} \dots v_{i_m}$ from equality of a_m and b_m at position m. This concludes this part of the proof, since we have that $i_2i_3 \dots i_m$ is a solution for the PCP instance.

"$\Leftarrow$": Now, assume that there is a solution $i_0i_1 \dots i_k$ to the PCP instance. We construct the computation $\varsigma = c_0c_1 \dots$ with $c_0(\mathtt{A}) = c_0(\mathtt{B}) = \mathtt{X}$ and for all $t > 0$: $c_t(\mathtt{A}) = \mu(w_{(t-1) \bmod (k+1)}, \mathtt{A})$ and $c_t(\mathtt{B}) = \mu(w_{(t-1) \bmod (k+1)}, \mathtt{B})$. It is straightforward to see that ς satisfies

$$[\,\mathtt{A} \lhd \mathtt{X}\,],\ [\,\mathtt{B} \lhd \mathtt{X}\,] \text{ and } \bigcirc\Box\Big(\bigvee_{j=0}^{n} \big([\,\mathtt{A} \lhd \mu(w_j, \mathtt{A})\,] \wedge [\,\mathtt{B} \lhd \mu(v_j, \mathtt{B})\,]\big)\Big).$$

Thus, it just remains to argue that ς satisfies $\bigcirc\bigcirc\Diamond(\mathtt{p}(\mathtt{A}) \leftrightarrow \mathtt{p}(\mathtt{B}))$. To this end, let $j_0j_1 \dots = (i_0i_1 \dots i_k)^\omega$. Then a simple induction shows that $[\![\varsigma]\!](\mathtt{A})(t) = \eta(\varsigma, t, \mu(w_{j_0}w_{j_1} \dots w_{j_{t-2}}, \mathtt{X}))$ and $[\![\varsigma]\!](\mathtt{B})(t) = \eta(\varsigma, t, \mu(v_{j_0}v_{j_1} \dots v_{j_{t-2}}, \mathtt{X}))$ for all $t > 1$ and all choices of $\langle\cdot\rangle$ and ι. Now, consider that especially for $t = k + 2$ we have that

$$w_{j_0}w_{j_1} \dots w_{j_{t-2}} = w_{i_0}w_{i_1} \dots w_{i_k} = v_{i_0}v_{i_1} \dots v_{i_k} = v_{j_0}v_{j_1} \dots v_{j_{t-2}}$$

[1] Remember that we do not have any inputs here, i.e., $\mathcal{N}_I = \emptyset$. Hence, the semantics $[\![\varsigma]\!]\colon (\mathcal{N}_I \to \mathcal{S}_I) \to (\mathcal{N}_O \to \mathcal{S}_O)$ of ς can be simplified to $[\![\varsigma]\!]\colon \mathcal{N}_O \to \mathcal{S}_O$.

and, thus, also $[\![\varsigma]\!](\mathtt{A})(k+2) = [\![\varsigma]\!](\mathtt{B})(k+2)$, independent of the choice of $\langle\cdot\rangle$ and ι. As this implies that $p([\![\varsigma]\!](\mathtt{A})(k+2)) = p([\![\varsigma]\!](\mathtt{B})(k+2))$ for any unary predicate $p \in \mathcal{P}$, it proves that ς satisfies $\mathtt{p}(\mathtt{A}) \leftrightarrow \mathtt{p}(\mathtt{B})$ at position $k+2$. Hence, ς also satisfies $\bigcirc\bigcirc\Diamond(\mathtt{p}(\mathtt{A}) \leftrightarrow \mathtt{p}(\mathtt{B}))$, which concludes the proof. □

3.8 From TSL to TSL_2

Let $\mathcal{F}_2 \subseteq \mathcal{F}$ and $\mathcal{P}_2 \subseteq \mathcal{P}$ be the sets of binary functions and predicates, respectively. We define TSL_2 as sub-logic of TSL, which only contains terms that are constructed from signals, binary functions and binary predicates.

We show that TSL and TSL_2 are equally expressive, by giving a transformation from a TSL φ to an equi-realizable TSL_2 formula ψ.

Theorem 2. *TSL and TSL_2 are equally expressive with respect to the realizability problem.*

Proof. The transformation proceeds in three steps. First, we eliminate all 0-nary functions and predicates from φ. Then, we eliminate all unary function and predicates. Finally we go down to TSL_2. We denote the intermediate logics by $\mathrm{TSL}_{>0}$ and $\mathrm{TSL}_{>1}$, respectively.

From TSL to $\mathrm{TSL}_{>0}$: Let φ be a TSL formula, $\mathcal{N}_F^0$ be the set of 0-nary function names appearing in φ and $\mathcal{N}_P^0$ be the set of 0-nary predicate names appearing in φ. First, we remove all names of $\mathcal{N}_F^0 \cup \mathcal{N}_P^0$ from $\mathcal{N}_F$ and $\mathcal{N}_P$, respectively, and, instead, add them to the set of output signal names $\mathcal{N}_O$. Then, for each $\mathtt{p} \in \mathcal{N}_P^0$, we create a fresh name $\mathtt{p}'$ and add them to the predicate names $\mathcal{N}_P$. Now, let

$$\kappa_0(\varphi) = \bigwedge_{x \in \mathcal{N}_F^0 \cup \mathcal{N}_P} \big(\square[\, x \lhd x \,]\big) \;\wedge\; \kappa_0'(\varphi),$$

where $\kappa_0'(\varphi)$ results from φ via replacing every 0-nary predicate $\mathtt{p} \in \mathcal{N}_P^0$ by the unary predicate $\mathtt{p}'(\mathtt{p})$. We show that φ is realizable if and only if $\kappa_0(\varphi)$ is realizable:

"⇒": Assume φ is realizable, i.e., there is a computation $\varsigma \in \mathcal{C}^\omega$ such that $\varsigma \vDash \varphi$, for any choice of $\langle\cdot\rangle$ and ι. We define the computation ς' for all $t \in \mathcal{T}\!\mathit{ime}$ via $\varsigma'(t)(\mathtt{s}) = \mathtt{s}$, if $\mathtt{s} \in \mathcal{N}_F^0 \cup \mathcal{N}_P^0$, and $\varsigma'(t)(\mathtt{s}) = \varsigma(t)(\mathtt{s})$, otherwise, i.e., the newly created output signal names are never changed during the whole computation. We claim that ς' realizes $\kappa_0(\varphi)$. Thus, for the sake of contradiction assume that ς' does not satisfy $\kappa_0(\varphi)$ for some fixed $\langle\cdot\rangle'$ and ι'.

First, by the definition of ς', the computation must satisfy $\square[\, x \lhd x \,]$ for all $x \in \mathcal{N}_F^0 \cup \mathcal{N}_P^0$, since the value of every x is never changed over time and is initially fixed by ι'. But then consider the following choices of $\langle\cdot\rangle$ and ι: let $\iota = \iota'$[2], $\langle \mathtt{p} \rangle = \langle \mathtt{p}' \rangle'(\iota'(\mathtt{p}))$ for all $\mathtt{p} \in \mathcal{N}_P^0$, $\langle \mathtt{f} \rangle = \iota'(\mathtt{f})$ for all $\mathtt{f} \in \mathcal{N}_F^0$ and $\langle x \rangle = \langle x \rangle'$, for all

[2] Note that our changes with respect to the name sets are compatible with this definition.

$x \notin \mathcal{N}_P^0 \cup \mathcal{N}_F^0$. A simple induction shows that ς does not satisfy φ, since every evaluation of predicates in φ results in the same truth value as the corresponding evaluation of predicates in $\kappa_0'(\varphi)$. However, this contradicts our assumption that φ is satisfied for all possible $\langle\cdot\rangle$ and ι, which then proves the correctness our claim.

"$\Leftarrow$": Now assume that $\kappa_0(\varphi)$ is realizable and let $\varsigma' \in \mathcal{C}^\omega$ be the computation that satisfies $\kappa_0(\varphi)$. We define the computation ς for all $t \in \mathcal{T}\!\mathit{ime}$ via $\varsigma(t)(\mathtt{s}) = \tau$, where τ results from $\varsigma'(t)(\mathtt{s})$ via replacing all occurrences of $\mathtt{p}'(\mathtt{p})$ by $\mathtt{p}$. We claim that ς realizes φ, and again, let us assume this would not be the case, i.e., there are some $\langle\cdot\rangle$ and ι such that φ is not satisfied. Hence, consider ι' with $\iota'(x) = \langle x\rangle$ for all $x \in \mathcal{N}_P^0 \cup \mathcal{N}_F^0$ and $\iota'(x) = \iota(x)$ for all $x \in \mathcal{S}_O \setminus (\mathcal{N}_P^0 \cup \mathcal{N}_F^0)$, and $\langle\cdot\rangle'$ with $\langle\mathtt{p}'\rangle' = p$ for all $\mathtt{p} \in \mathcal{N}_P^0$, where $p(b) = \langle\mathtt{p}\rangle$ for all $b \in \mathcal{B}$, and $\langle x\rangle' = \langle x\rangle$ for all $x \notin \mathcal{N}_P^0$. With these definitions, every predicate evaluation in φ is equivalent to the corresponding predicate evaluation in $\kappa_0(\varphi)$. Thus, a simple induction shows that ς' does not satisfy $\kappa_0'(\varphi)$ and, hence, also not $\kappa_0(\varphi)$. However this contradicts our assumption, finally proving our claim. u

From $TSL_{>0}$ to $TSL_{>1}$: Let φ be a $\mathrm{TSL}_{>0}$ formula and $\mathcal{N}^1$ be the set of unary functions and predicates appearing in φ. Now let $\kappa_1(\varphi)$ be the $\mathrm{TSL}_{>1}$ formula, which results from φ by replacing every term $x(\tau)$ with $x \in \mathcal{N}^1$ and $\tau \in \mathcal{T}_F$, that appears in φ, by $x(\tau, \tau)$. Thereby note that κ_1 is defined on terms and TSL formulas, simultaneously. We show that φ is realizable if and only if $\kappa_1(\varphi)$ is realizable:

"$\Rightarrow$": Assume φ is realizable and let $\varsigma \in \mathcal{C}^\omega$ be the computation such that $\varsigma \vDash \varphi$ independently of the choices for $\langle\cdot\rangle$ and ι. We define the computation ς' for all $t \in \mathcal{T}\!\mathit{ime}$ via $\varsigma'(t) = \kappa_1(\varsigma(t))$, i.e., we apply the same transformation on the terms of ς, as we applied on the terms of φ. We claim that ς' realizes $\kappa_1(\varphi)$, but for the sake of contradiction assume that this is not the case. Hence, there are $\langle\cdot\rangle'$ and ι', for which ς' does not satisfy $\kappa_1(\varphi)$. We choose $\iota = \iota'$ and $\langle x\rangle(v) = \langle x\rangle'(v, v)$ for all $v \in \mathcal{V}$, if $x \in \mathcal{N}^1$, and $\langle x\rangle = \langle x\rangle$, otherwise. A simple induction shows that for these choices of $\langle\cdot\rangle$ and ι, ς does not satisfy φ. However, this contradicts our assumption, which proofs our claim.

"$\Leftarrow$": Now, assume $\kappa_1(\varphi)$ is realizable and let $\varsigma' \in \mathcal{C}^\omega$ be the corresponding realizing computation. We define the computation ς for all $t \in \mathcal{T}\!\mathit{ime}$ via $\varsigma(t) = \kappa_1^*(\varsigma'(t))$, where κ_1^* replaces every term $x(\tau, \tau')$ with $x \in \mathcal{N}^1$ and $\tau, \tau' \in \mathcal{T}_F$ by $x(\tau)$. We claim that ς realizes φ, but for the sake of contradiction assume the opposite. Thus, there are $\langle\cdot\rangle$ and ι such that ς does not satisfy φ. We choose $\iota' = \iota$ and $\langle x\rangle'(v, v') = \langle x\rangle(v)$, for all $x \in \mathcal{N}^1$ and $v, v' \in \mathcal{V}$. For $x \notin \mathcal{N}^1$, we choose $\langle x\rangle' = \langle x\rangle$. A simple induction shows that for the given choices of $\langle\cdot\rangle'$ and ι' we have $\varsigma' \nvDash \kappa_1(\varphi)$. This contradicts our assumption and proves our claim.

From $TSL_{>1}$ to TSL_2: Finally, let φ be a $\mathrm{TSL}_{>1}$ formula and let $\mathcal{N}^2$ be the set of binary function names and predicate names appearing in φ. Now, consider the transformed formula $\kappa_2(\varphi)$, where we replace every term $x(\tau_0, \tau_1, \ldots, \tau_{m-1})$ of φ with $m > 2$ and $x \in (\mathcal{N}_F \cup \mathcal{N}_P) \setminus \mathcal{N}^2$ by $x(\tau_0, \mathtt{f}(\tau_1, \ldots, \mathtt{f}(\tau_{m-2}, \tau_{m-1}) \ldots))$, where

$\mathtt{f} \in \mathcal{N}_F$ is a fresh binary function name. As before, κ_3 is defined on both: terms and formulas. We show that φ is realizable if and only if $\kappa_2(\varphi)$ is realizable:

"$\Rightarrow$": Assume φ is realizable and let $\varsigma \in \mathcal{C}^\omega$ be the computation that realizes φ. We define the computation ς' for all $t \in \mathcal{T}\!\mathit{ime}$ via $\varsigma'(t) = \kappa_2(\varsigma(t))$. We claim that ς' realizes $\kappa_2(\varphi)$, but for the sake of contradiction assume the opposite, i.e., that there are $\langle\cdot\rangle'$ and ι' such that ς' does not satisfy $\kappa_2(\varphi)$. We choose $\iota = \iota'$, $\langle x\rangle(v_0, v_1, \ldots, v_{m-1}) = \langle x\rangle'(v_0, \langle \mathtt{f}\rangle'(v_1, \ldots, \langle \mathtt{f}\rangle'(v_{m-2}, v_{m-1})\ldots))$ for all $v_0, v_1, \ldots, v_{m-1} \in \mathcal{V}$, if $x \in (\mathcal{N}_F \cup \mathcal{N}_P) \setminus \mathcal{N}^2$, and $\langle x\rangle = \langle x\rangle'$, otherwise. A simple induction shows that for these choices for $\langle\cdot\rangle'$ and ι, $\varsigma \not\models \varphi$. However, this contradicts our assumption, and, hence, proves our claim.

"$\Leftarrow$": Now, assume that $\kappa_2(\varphi)$ is realizable by the computation $\varsigma' \in \mathcal{C}^\omega$. Consider the computation ς, defined for all $t \in \mathcal{T}\!\mathit{ime}$ via $\varsigma(t) = \kappa_2^*(\varsigma'(t))$, where κ_2^* replaces every term $x(\tau_0, \mathtt{f}(\tau_1, \ldots, \mathtt{f}(\tau_{m-2}, \tau_{m-1})\ldots))$ with $x \in \mathcal{N}^1$ and $\tau_0, \tau_1, \ldots, \tau_{m-1} \in \mathcal{T}_F$, by $x(\tau_0, \tau_1, \ldots, \tau_{m-1})$. We claim that ς realizes φ, but for the sake of contradiction assume the opposite, i.e., there are $\langle\cdot\rangle$ and ι, for which $\varsigma \not\models \varphi$. We choose $\iota' = \iota$, $\langle \mathtt{f}\rangle'(v, v') = (v, v')$[3] for all $v, v' \in \mathcal{V}$, $\langle x\rangle'(v_0, (v_1, \ldots, (v_{m-2}, v_{m-1})\ldots)) = \langle x\rangle(v_0, v_1, \ldots v_{m-1})$ for all $x \in \mathcal{N}^1$ and $v_0, v_1, \ldots, v_{m-1} \in \mathcal{V}$, and $\langle x\rangle' = \langle x\rangle$ for all remaining names x. A simple induction shows that ς' does not satisfy $\kappa_2(\varphi)$ for these choices of $\langle\cdot\rangle$ and ι. This contradicts our assumption and, hence, finishes our proof. □

3.9 Incompleteness of the Under-Approximation

Let `p` be the predicate name of a unary predicate and let `X` be the function name of a 0-nary function. Consider the TSL formula $\varphi = \mathtt{p}(\mathtt{X}) \rightarrow \bigcirc\, \mathtt{p}(\mathtt{X})$. Since predicate evaluations are time invariant, the formula is realizable. However, for the under-approximation, we interpret $\mathtt{p}(\mathtt{X})$ as inputs, which are allowed to change over time, since in general signals, which are passed as arguments to a predicate, may change over time. Thus, the synthesizer would return an unrealizability result for φ.

Note that the special case of unary predicates and 0-nary functions may be ruled out by the translation to LTL. However, the same behavior can also be produced by more complex examples, where it is non-trivial to identify restrictions on the input behavior, caused by the time invariance property. As the decidability problem is still open, it is also open, whether all possible cases can always be captured by a translation to LTL in general.

Another observation is that for completeness the synthesizer may use terms, which do not appear as a full term in the formula, but only as a sub-term. For an example consider the following realizable formula

$$\varphi \;=\; \mathtt{p}(\mathtt{f}(\mathtt{X})) \;\rightarrow\; \square\neg[\,\mathtt{s} \lhd \mathtt{f}(\mathtt{X})\,] \;\wedge\; \diamondsuit\, \mathtt{p}(\mathtt{s}),$$

where the only way to satisfy the predicate `p` eventually is to assign `s` the constant `X` in a first step, and then `f` in a later step, separately.

[3] the binary tuple constructor

Nevertheless, we do not know yet, whether the synthesizer may need to use terms that do not appear in any form in the given specification. In other words, it may be that it is sufficient to restrict to all possible sub-terms, which would be a finite set. To answer this questions, further investigations are needed. Also note that our undecidability proof only needs to assign terms, that appear as full terms in the formula.

3.10 Example

Let $\mathcal{N}_I = \{\texttt{press}\}$ and $\mathcal{N}_O = \{\texttt{pos}, \texttt{vel}, \texttt{score}\}$ be the inputs and outputs, respectively, and let $\texttt{plus}, \texttt{minus} \in \mathcal{N}_F$ be binary functions and $\texttt{one}, \texttt{zero} \in \mathcal{N}_F$ be 0-nary functions. Furthermore, let $\texttt{leftmost}, \texttt{rightmost}, \texttt{center}, \texttt{event} \in \mathcal{N}_P$ be unary predicates.

- $\square\,\big(\texttt{leftmost}(\texttt{pos}) \rightarrow \Diamond\,[\,\texttt{pos} \lhd \texttt{plus}(\texttt{pos}, \texttt{vel})\,]\big)$
- $\square\,\big(\texttt{rightmost}(\texttt{pos}) \rightarrow \Diamond\,[\,\texttt{pos} \lhd \texttt{minus}(\texttt{pos}, \texttt{vel})\,]\big)$
- $\square\,\big(\neg\,\texttt{leftmost}(\texttt{pos}) \wedge \neg\,\texttt{rightmost}(\texttt{pos}) \rightarrow \Diamond\,([\,\texttt{pos} \lhd \texttt{minus}(\texttt{pos}, \texttt{vel})\,] \vee [\,\texttt{pos} \lhd \texttt{plus}(\texttt{pos}, \texttt{vel})\,])\big)$
- $\square\,\big(\texttt{leftmost}(\texttt{pos}) \wedge \bigcirc \neg\,\texttt{leftmost}(\texttt{pos}) \rightarrow (\neg\,[\,\texttt{pos} \lhd \texttt{minus}(\texttt{pos}, \texttt{vel})\,]\ \mathcal{U}\ \texttt{rightmost}(\texttt{pos}))\big)$
- $\square\,\big(\texttt{rightmost}(\texttt{pos}) \wedge \bigcirc \neg\,\texttt{rightmost}(\texttt{pos}) \rightarrow (\neg\,[\,\texttt{pos} \lhd \texttt{plus}(\texttt{pos}, \texttt{vel})\,]\ \mathcal{U}\ \texttt{leftmost}(\texttt{pos}))\big)$
- $\square\,\big(\texttt{center}(\texttt{pos}) \wedge \texttt{event}(\texttt{press}) \rightarrow [\,\texttt{score} \lhd \texttt{plus}(\texttt{score}, \texttt{one})\,] \wedge [\,\texttt{vel} \lhd \texttt{plus}(\texttt{vel}, \texttt{one})\,]\big)$
- $\square\,\big(\neg\,\texttt{center}(\texttt{pos}) \wedge \texttt{event}(\texttt{press}) \rightarrow [\,\texttt{score} \lhd \texttt{zero}\,] \wedge [\,\texttt{vel} \lhd \texttt{zero}\,]\big)$

Assume:

- $\square\,\big([\,\texttt{pos} \lhd \texttt{plus}(\texttt{pos}, \texttt{vel})\,] \rightarrow \bigcirc \neg\,\texttt{leftmost}(\texttt{pos})\big)$
- $\square\,\big([\,\texttt{pos} \lhd \texttt{minus}(\texttt{pos}, \texttt{vel})\,] \rightarrow \bigcirc \neg\,\texttt{rightmost}(\texttt{pos})\big)$

Implicit:

- $\square\,\big([\,\texttt{pos} \lhd \texttt{plus}(\texttt{pos}, \texttt{vel})\,] \vee [\,\texttt{pos} \lhd \texttt{minus}(\texttt{pos}, \texttt{vel})\,] \vee [\,\texttt{pos} \lhd \texttt{pos}\,]\big)$
- $\ldots$

4 Term Annotated Mealy Machines:

Using a similar notation as for TSL, we define a Term Annotated Mealy Machine (TA$\mathbb{M}$) as the tuple

$$\mathcal{M} = (\mathcal{N}, \mathcal{T}_P, \mathcal{T}_F, M, m_I, \delta, \ell),$$

where,

- $\mathcal{N} = \mathcal{N}_I \uplus \mathcal{N}_O \uplus \mathcal{N}_F \uplus \mathcal{N}_P$ is a finite set of input signal, output signal, function and predicate names, respectively,
- $\mathcal{T}_P$ is a finite set of predicate terms, constructed from names in $\mathcal{N}$,
- $\mathcal{T}_F$ is a finite set of function terms, constructed from names in $\mathcal{N}$,
- M is a finite set of states,
- $m_I \in M$ is the initial state,
- $\delta\colon 2^{\mathcal{T}_P} \times M \to M$ is a transition function mapping from a state and a predicate evaluation to a successor state, and
- $\ell\colon 2^{\mathcal{T}_P} \times M \to \{\mathcal{N}_O \times \mathcal{T}_F\}$ is a labeling function assigning each state and each possible predicate evaluation a set of updates (assignments from output signal names to terms).

F: what's the thing with the { } notation? Why not $\mathcal{N}_O \to \mathcal{T}_F$?

A run $r = (m_0, c_0)(m_1, c_1)(m_2, c_2) \ldots \in (M \times \mathcal{C})^\omega$ of a TA$\mathbb{M}$ $\mathcal{M}$ is an infinite sequence of states and computation steps such that $m_0 = m_I$ and

$$\forall t \in \mathcal{T}\mathit{ime}.\ \exists P \subseteq \mathcal{T}_P.\ m_{t+1} = \delta(P, m_t) \wedge \{\forall \mathtt{s} \in \mathcal{N}_O.\ (s, c_{t+1}(\mathtt{s}))\} = \ell(P, m_t)$$

M: why not $\exists P \in 2^{\mathcal{T}_P}$

Consider that a run r induces a computation $\varsigma_r = c_0 c_1 \ldots \in \mathcal{C}^\omega$. The set of all possible computations ς_r, induced via runs r of $\mathcal{M}$, defines the language of $\mathcal{M}$, denoted by $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{C}^\omega$.

F: right, should be $P \subseteq \mathcal{T}_P$

5 Synthesizing FRP

5.1 FRP Program *p*:

The semantics of an FRP program are given by its implementation. We choose to examine a particular class of computations that can be used for FRP programs, called Casual Commutative Arrows (CCA). This restricts the implementation so that it must obey a set of laws, allowing us to reason about these programs more abstractly than for a single implementation.

A CCA program is a term that types with SF a b. We present here the syntax and typing rules of a modified version of CCA. We extend the syntax and typing rules with predefined functions and predicates. We also restrict the type space to only the types A and Bool, since the logic is also untyped.Since our synthesis only allows for predefined functions and predicates we include these as expressions.

M: adding types is way to large for this paper

Note the arrow combinators have empty typing environments meaning they are closed terms with no free variables. This ensures an arrow can be evaluated over an input stream with no other parameterization - matching our working concept of a stream function in this paper.

F: not really true, better: "... is independent of the implementation of ..." M: not true?

5.2 Syntax

User Provided U ::= f $\in \mathcal{N}_F$ | p $\in \mathcal{N}_P$
Values ::= x | y | z | ... Types A,B ::= 1 | A x B | A $\to$ B | A $\rightsquigarrow$ B
Expressions M , N ::= () | U | (M , N) | λ V. M | M N | trace M
Programs P, Q ::= arr M | P >>> Q | first P | loop P | init M
Environment Γ::= x0 : A0, ..., xn : An

M: do we want terms or names here?

F: names should be right, but user also provides implementation, i.e., $\langle f \rangle$, $\langle p \rangle$. Does this change something here?

5.3 Typing Rules

$$\text{(FXN)}\ \frac{f \in \mathcal{N}_F}{f\colon A \to A \in \Gamma} \tag{1}$$

$$\text{(PRED)}\ \frac{p \in \mathcal{N}_P}{p\colon A \to \mathit{Bool} \in \Gamma} \tag{2}$$

F: shouldn't it be $f\colon A \times A \to A$ (binary)?'

5.4 From Mealy Machines to FRP

Rutten showed in [**?**]a translation from causal stream functions to Mealy Machines. Here we give a translation from our particular Mealy Machines, TA$\mathbb{M}$, to a specific instance of causal stream functions, CCA. CCA is a subset of all causal stream functions, so the existance of the translation is expected. We provide it here to argue about the larger coorespondance betweeen TSL and CCA. We will call this translation MFRP : TA$\mathbb{M} \to FRP$.

CCNF is in general an optimization of a CCA program, but a robust normalization procedure has been elusive in past work. Recently, Liu et al.[**?**], propose a normal form by approach. We achieve a similar result here, as the synthesis produces a normal form by construction.

For synthesis, we in fact translate the Mealy machine to the normal form of CCA, Causal Commutative Normal Form (CCNF). All CCNF programs are either of the form `arr f` or `loop (f >>> second (init i))` (shortened to `loopD i f`. For the completeness and soundness of the CCA to CCNF transformation see [**?**]. We distinguish three seperate cases in the translation.

For $\mathcal{M} = (\mathcal{N}, \mathcal{T}_P, \mathcal{T}_F, M, m_I, \delta, \ell) \in$ TA$\mathbb{M}$, We use the CCNF, `loopD i f`, with `i` and `f` defined as follows.

F: this translation is to Mealy machines on values with possibly infinitely many states and and the translation is straightforward. Hence, it does not match our setting that well. I still don't see why we need this reference. (because he is on the PC)

M: move this paragraph

f :: $(\mathcal{N}_I, (\mathcal{N}_O, M)) \to (\mathcal{N}_O, M)$
f $(s_i, (s_o$,m) = **if**
| $p_0(s) \wedge q(0, m) \to (f_0(s_o), t_0(m))$
| $p_1(s) \wedge q(1, m) \to (f_1(s_o), t_1(m))$

M: types dont match, shouldnt be $\mathcal{N}_O$, but the actual type of the value on those wires

$|$ ...
$|$ $p_n(s) \wedge q(|M|, m) \rightarrow (f_{|2^{\mathcal{P}}|*|M|}(s_o), t_{|M|}(m))$
where s $=$ $s_i \cup s_o$

```
i :: (N_O, M)
i =  (initValue , m_I)
```

Where

- $p_i \in 2^{\mathcal{P}}$ is the predicate evaluation as the obvious boolean combinations
- $f_i \in \mathcal{T}_F$ is the the tupled function terms for all output signals
- `q i m = i == (toInt m)` checks the value of current state signal
- $t_i \in \delta$ is the state transition function
- `initValue` is the value for $\mathcal{N}_O$are time 0

Recall that in TSL and TA$\mathbb{M}$, output signals can be written at current time, t, and read from at time $t+1$. To implement this in the FRP program, we will use the `loop` and `init` combinators so that $s = \mathcal{N}_I \cup \mathcal{N}_O$. Since the body of `f` may require output values at time $t = 0$, we use `initValue` provide initial values to $\mathcal{N}_O$ with the `init` arrow combinator. The `loop` combinator pipes the output values back to the input to allow them to be read at time $t+1$.

The loop also captures an embedding the mealy states, M, into the signal level of the FRP program. We have the additional predicates q_i which query the current TA$\mathbb{M}$ state, m, and the state transitions v_i which update the current state signal m.

This straight-forward encoding ensures that

$$\forall t \in \mathcal{T}\boldsymbol{ime}, \forall m \in M, \forall s_i \in \mathcal{N}_I$$
$$\ell(P, m) = \{\forall s_o \in \mathcal{N}_O.\ (s_o, f_i(s_i))\} \wedge$$
$$t_i(m) = \delta(P, m)$$

M: where did we establish this?

Since δ is a total function, we will always have compelte pattern matching over t_i. The number of conditionals is bounded by $|2^{\mathcal{T}_P}| * |M|$, as all predicate evaluations might induce unique signal updates for each state.

5.5 From FRP to input/ouput relation

Just as FOL gives a translation from proofs to programs, but does not imply lazy or strict semanticsi, we do not get an operational semantics from TSL. A CCA implementation comes equipped with a function `runSF :: SF a b -> [a] -> [b]`. Applying `runSF` to the CCA program will yield an infinite mapping from inputs to outputs. The casuality garuntee from CCA ensures that the map only depends on the past and present - it is actually possible to realize this mapping without looking into the future.

Since runSF is implementation dependant, we provide one instance here using a continuation style sematics. Similar to the translation from TSL to input/output relation in Sec. 3.3, this coinductively defines the output as input is processed. The full Arrow instance is provided in FRP.hs.

```
newtype SF a b = SF { unSF :: (a -> (b, SF a b)) }

runSF :: SF a b -> [a] -> [b]
runSF (SF f) (x:xs) =
  let (y, f') = f x
  in y `seq` (y : runSF f' xs)
```

For the sake of brevity, we assume the standard operational semantics for Haskell to evaluate this expression.

5.6 Synthesis is complete

We now show that the input/ouput interpretation of a FRP program is same as the input/output interpretation of TA$\mathbb{M}$ as given in Sec. **??**. For this we will treat `runSF` as leaving `f x` as an unevaluated thunk - or update assignment. Since MFRP ensures we only apply a single update at each time step, taken verbatim from TA$\mathbb{M}$, thei sequence of these thunks will be exactly a $\varsigma \in \mathcal{C}^\omega$.

Formally, in a common notation, we can now show

M: ς is a list of computations, right?

$$\forall \mathcal{M} \in \text{TA}\mathbb{M}, \forall \varsigma = (c_0, c_1, ...) \in \mathcal{L}(\mathcal{M}), \forall t \in \mathcal{Time}, \forall s \in \mathcal{N}_O$$
$$\text{sample s t (runSF M2FRP(}\ \mathcal{M})) = c_t(s)$$

[Insert proof by coinduction for equality of streams - read Rutten06 how to]

Along with the input.output equivelence of TSL and Mealy machines, we now conclude by transivtivy of equality that the synthesized FRP program realizes the same input/output relation as the TSL specification.

5.7 FRP to TSL map

Although not all TSL specifications have a cooresponding FRP program. A TSL specification can, in theory, produce a TA$\mathbb{M}$ with infintely many states, M. Since our translation from TA$\mathbb{M}$ to FRP program encodes the checking of these states as concrete predicates, and the program must be of finite size, we cannot encode all specifications.

M: But is this just a limitation to MFRP? What if we were to have higher-order predicates?

Formally, this question is:

$$\begin{aligned} &\forall \mathtt{p} : CCNF, \\ &\exists \varphi \in \text{TSL} \\ &\exists \varsigma \in \mathcal{C}^\omega \\ &\varsigma \vDash \varphi \wedge \mathtt{runSF\ p} = \varsigma \end{aligned}$$

M: Are we limited by the static strucure of arrows? I don't think so, TSL can't express dynamic structure. how to say this formally?