Declarative Synchronous Multithreaded Programming

Blanca Mancilla and John Plaice

School of Computer Science and Engineering
The University of New South Wales
UNSW SYDNEY NSW 2052, Australia
{mancilla,plaice}@cse.unsw.edu.au

Abstract. We demonstrate how TransLucid can be used as a reactive system. At each instant, there is a set of active ports, where sets of equations, demands and threads are all registered. Each thread defines a sequence of (*state*, *demand*) pairs, and threads may interact through the overall set of equations. The entire system remains fully declarative.

Key words: Synchronous programming, distributed computing, declarative programming, Cartesian programming, multidimensional programming.

1 Introduction

In this paper, we apply the principles of synchronous programming to the extension of the TransLucid language in order to study many aspects of reactive, distributed and mobile computing. The intuitive idea is that certain dimensions have a natural physical interpretation, and that the current set of valid equations varies according to these dimensions. In this manner, "change through time and space" can be modelled as variation through an index space, as is consistent with Cartesian programming.

The synchronous TransLucid system presented in this paper is an extension of the simple TransLucid system [1], which consists of a set of equations and a demand for the evaluation of an expression in a given context. In the synchronous system, the set of equations grows over time. At each instant t, there is a set of open *ports* ($p \in P_t$). For each pair (p, t), new equations may be added to the system, and new demands may be made as well.

Introducing dimension `time` into TransLucid allows its use for the programming of reactive systems, and TransLucid can then be regarded as a super-LUSTRE. Introducing the `port` dimension creates the potential for distributed and mobile computing, however this is not sufficient. Something more is needed, namely the *thread*.

In TransLucid, a thread is a sequence of $(state_t, demand_t)$ pairs where, at each instant t, $state_t$ defines the current context for the thread and $demand_t$ defines the thread's current action. We can envisage a mobile device moving at each instant from one place to another and then executing some action.

Dagstuhl Seminar Proceedings 08271
Topological and Game-Theoretic Aspects of Infinite Computations
http://drops.dagstuhl.de/opus/volltexte/2008/1653

Since a TransLucid system is entirely demand-driven, communication can only take place between threads if they can be made aware of one another. To do this requires a centralised mechanism that can keep track of all of the active ports, threads, equations and demands, which in turn requires adding set-based primitives to TransLucid.

In fact, this paper could be entitled, "Adding sets to TransLucid"! There are sets of instants, sets of ports, sets of sets of equations, sets of demands, and so on. The order of presentation will be bottom-up. We begin (§2) by adding sets to TransLucid expressions. We then (§4) show how *best-fitting* can be used to choose the most appropriate definition for an identifier when there are several possibilities. Once the preliminaries are finished, we (§5) present the overall structure of a synchronous TransLucid system and its semantics. We conclude (§6) with a discussion of future work.

2 Sets as values

TransLucid is a coordination language, and can manipulate typed objects defined in another language. The only primitive types manipulated in the semantics of expressions in the previous publications on TransLucid, such as [1], are `sp` (special), `bool` and `tuple`. In addition, to parse the header file, which defines how concrete syntax is to be translated into abstract syntax, `ustring`, `uchar` and `int` are also needed.

In this section, we add sets to TransLucid, by introducing a new built-in type, called `set`, that plays a dual rôle. At one level, `set` and the standard operations of set enumeration, comprehension, union, intersection and difference, can be viewed as an abstract data type. At another level, a set can be considered to be a container of objects, and one might wish to iterate over these objects and to do some treatment thereon. Rather than providing a special set of iterators over such structures, we provide means for translating a set into an entity varying in a specified dimension.

The *extensional* view of the set provides the standard set of operations:

- $\{E, \ldots, E\}$: creation of set by enumeration of elements;
- $\{E \mid E\}$: set comprehension;
- $S \cup S$: set union;
- $S \cap S$: set intersection;
- $S - S$: set difference.

The *intensional* view of the set allows one to access individual elements by changing the value associated with a specified dimension. There are two operators:

- $h =$ `settoHD.`$d(S)$: Create a hyperdaton varying in dimension d. If $S = \{c_0, \ldots, c_{n-1}\}$ has n elements, then h `@ [`$d : i$`]` equals c_i if $0 \leq i < n$, and `sp<eod>` otherwise.
- $S =$ `HDtoset.`$d(h)$: Create a set from a hyperdaton varying in dimension d. If $h_{[d:i]}$ is non-`eod` for $i = 0..n-1$, and `sp<eod>` elsewhere, then S will be a set with n elements.

3 Regions

In the following sections, we will define context-dependent equations, equation sets, and demands. To do this, we need to define the *region*, which corresponds to a set of contexts.

$$\begin{aligned} \mathit{region} &::= \texttt{[}\ \mathit{range}^{+}\ \texttt{]} \\ \mathit{range} &::= \mathbf{c} : c \texttt{ .. } c \\ &\mid\ \mathbf{c} : c \texttt{ .. } \mathbf{\#c} \\ &\mid\ \mathbf{c} : \mathbf{\#c} \texttt{ .. } c \end{aligned}$$

where the plain c's specify constants and the boldfaced $\mathbf{c}$'s specify dimensions. The multirange specifies a set of contexts, by defining the set of acceptable values for a set of dimensions.

In order to use the context-dependent equations, one needs to be able to compare two regions, to determine if one is included in the other. This is done with the $\sqsubseteq$ operator, defined below.

Let R be a region. The domain of R is the set of left-hand sides in R. Suppose $R = \{\mathbf{c} \mapsto r_{\mathbf{c}} \mid \mathbf{c} \in \operatorname{dom} R\}$, where each $r_{\mathbf{c}}$ is of one of the forms:

- $(c_{\min}, c_{\max})$
- $(c_{\min}, \mathbf{\#c}_{\max})$
- $(\mathbf{\#c}_{\min}, c_{\max})$

Then $\operatorname{ran}_R(\mathbf{c})$ is defined by:

$$\operatorname{ran}_R(\mathbf{c}) = \begin{cases} [c_{\min}, c_{\max}], & r_{\mathbf{c}} = (c_{\min}, c_{\max}) \\ \Big[c_{\min}, \max\big(\operatorname{ran}_R(\mathbf{c}_{\max})\big)\Big], & r_{\mathbf{c}} = (c_{\min}, \mathbf{\#c}_{\max}) \\ \Big[\min\big(\operatorname{ran}_R(\mathbf{c}_{\min})\big), c_{\max}\Big], & r_{\mathbf{c}} = (\mathbf{\#c}_{\min}, c_{\max}) \end{cases}$$

Let two regions R and R', and suppose $\mathbf{c} \in \operatorname{dom} R$ and $\mathbf{c} \in \operatorname{dom} R'$ for some $\mathbf{c}$. We wish to compare $r_{\mathbf{c}}$ and $r'_{\mathbf{c}}$. First

$$r_{\mathbf{c}} \sqsubseteq^{\min}_{\mathbf{c}} r'_{\mathbf{c}} = \begin{cases} c \geq c', & r^{\min}_{\mathbf{c}} = c \text{ and } r'^{\min}_{\mathbf{c}} = c' \\ c \geq \min\big(\operatorname{ran}_R(\mathbf{c}_{\min})\big), & r^{\min}_{\mathbf{c}} = c \text{ and } r'^{\min}_{\mathbf{c}} = \mathbf{\#c}'_{\min} \\ \mathbf{c}_{\min} = \mathbf{c}'_{\min}, & r^{\min}_{\mathbf{c}} = \mathbf{\#c}_{\min} \text{ and } r'^{\min}_{\mathbf{c}} = \mathbf{\#c}'_{\min} \end{cases}$$

$$r_{\mathbf{c}} \sqsubseteq^{\max}_{\mathbf{c}} r'_{\mathbf{c}} = \begin{cases} c \leq c', & r^{\max}_{\mathbf{c}} = c \text{ and } r'^{\max}_{\mathbf{c}} = c' \\ c \leq \max\big(\operatorname{ran}_R(\mathbf{c}_{\max})\big), & r^{\max}_{\mathbf{c}} = c \text{ and } r'^{\max}_{\mathbf{c}} = \mathbf{\#c}'_{\max} \\ \mathbf{c}_{\max} = \mathbf{c}'_{\max}, & r^{\max}_{\mathbf{c}} = \mathbf{\#c}_{\max} \text{ and } r'^{\max}_{\mathbf{c}} = \mathbf{\#c}'_{\max} \end{cases}$$

Now we can compare R and R'. We have that: $R \sqsubseteq R'$ iff $\operatorname{dom} R' \subseteq \operatorname{dom} R$ and $\forall \mathbf{c} \in \operatorname{dom} R'$, $r_{\mathbf{c}} \sqsubseteq^{\min}_{\mathbf{c}} r'_{\mathbf{c}}$ and $r_{\mathbf{c}} \sqsubseteq^{\max}_{\mathbf{c}} r'_{\mathbf{c}}$.

To determine if a context κ is included in a region R, we use the $\in$ operator, defined as follows: $\kappa \in R$ iff $\{\kappa \mid \operatorname{dom} R\} \sqsubseteq R$.

4 Equation sets and best-fit definitions

In the synchronous TransLucid sytem presented in the next section, the overall set of equations is evolving through time and space. For each identifier x, there may be several defining equations for x. Furthermore, for each instant t, the set of equations defining x may be different.

An *equation set* is a set Q of 4-tuples of the form $q = (x_q, \kappa_q, K_q, E_q)$, where

- x_q is the identifier being defined;
- κ_q is the context in which the definition was added to the equation set; currently, we are interested in the value of $\kappa_q(\texttt{time})$, i.e., the timestamp at which q entered the system;
- K_q is a set of contexts in which the definition is valid;
- E_q is the expression.

The concrete syntax for context-dependent equations is as follows:

$$eqn ::= ident \texttt{ @ } region \texttt{ = } expr$$

Suppose we have an equation set Q. Suppose we wish to evaluate identifier x in context κ. We define:

- $Q_x = \{q \in Q \mid x_q = x\}$
- $Q_{x\kappa} = \left\{q \in Q_x \mid \kappa_q(\texttt{time}) \leq \kappa(\texttt{time}) \wedge \kappa \in K_q\right\}$
- The *best-fit definition* is the unique $\overline{q} \in Q_{x\kappa}$ such that $\forall q \in Q_{x\kappa}$, $K_q \sqsubseteq K_{\overline{q}}$, where $\sqsubseteq$ was defined in §3.
- We then write $Q(x, \kappa) = E_{\overline{q}}$, if $\overline{q}$ exists, `sp<undef>` otherwise.

5 Synchronous TransLucid system

A synchronous TransLucid system

$$\mathcal{S} = (H, L, \mathbb{P}, \mathbb{Q}, \mathbb{D}, \mathbb{T}, \mathcal{P}, \mathcal{Q}, \mathcal{D}, \mathcal{T})$$

is a 10-tuple, where:

- H is a header defining the translation from concrete to abstract syntax.
- L is a set of libraries defining the available types and operations.

$$\mathbb{P} : \textbf{Time} \to \textbf{PortID}$$
$$\mathbb{Q} : \textbf{Time} \to \textbf{EqnID}$$
$$\mathbb{D} : \textbf{Time} \to \textbf{DemandID}$$
$$\mathbb{A} : \textbf{Time} \to \textbf{ThreadID}$$

$$\mathcal{P} : \textbf{PortID} \to \textbf{Time} \to \textbf{EqnID} \times \textbf{DemandID} \times \textbf{ThreadID}$$
$$\mathcal{Q} : \textbf{EqnID} \to \textbf{Time} \to \textbf{PortID} \times \textbf{Region} \times \textbf{Eqn}^+$$
$$\mathcal{D} : \textbf{DemandID} \to \textbf{Time} \to \textbf{PortID} \times \textbf{Expr}^+$$
$$\mathcal{A} : \textbf{ThreadID} \to \textbf{Time} \to \textbf{PortID} \times \textbf{Eqn}^+$$

For each port, equation set, demand set and thread, there is a unique identifier u, valid for all of eternity, even if it is decommissioned. This simplifies perfect recall.

Ports. There are two operations for manipulating ports.

- $\mathbf{p} = \texttt{newport}$
- $\texttt{decommissionport}\ \mathbf{p}$

Let t be the current instant. Suppose that:

- $\mathbb{P}_t^{\text{new}}$ is the set of new ports;
- $\mathbb{P}_t^{\text{old}}$ is the set of decommissioned ports.

Then:

$$\mathbb{P}_t = \mathbb{P}_{t-1} \cup \mathbb{P}_t^{\text{new}} - \mathbb{P}_t^{\text{old}}$$
$$\mathcal{P}_{\mathbf{p},t} = (\mathbb{Q}_{\mathbf{p},t}, \mathbb{D}_{\mathbf{p},t}, \mathbb{A}_{\mathbf{p},t})$$

Equation sets. There are four operations for manipulating equation sets.

- $\mathbf{q} = \texttt{neweqn}(R)$
- $\texttt{decommissioneqn}(\mathbf{q})$
- $\texttt{moveeqn}(\mathbf{q}, \mathbf{p})$
- $\texttt{addtoeqn}(\mathbf{q}, Q)$

Let t be the current instant and $\mathbf{p}$ the current port. Suppose that:

- $\mathbb{Q}_{\mathbf{p},t}^{\text{new}}$ is the set of new equation sets;
- $\mathbb{Q}_{\mathbf{p},t}^{\text{old}}$ is the set of decommissioned equation sets;
- $\mathbb{Q}_{\mathbf{p}',\mathbf{p},t}^{\text{move}}$ is the set of equation sets that are moving from port $\mathbf{p}'$ to port $\mathbf{p}$.
- $\mathbb{Q}_{\mathbf{p},t}^{\text{add}}$ is a set of pairs of the form $(\mathbf{q}, Q)$, meaning the equation set $\mathbf{q}$ is being augmented with equation set Q.

Then:

$$\mathbb{Q}_{\mathbf{p},t} = \bigcup_{\mathbf{p}' \in \mathbb{P}_{t-1}} \mathbb{Q}_{t,\mathbf{p}',\mathbf{p}}^{\text{move}} \cup \mathbb{Q}_{\mathbf{p},t-1} \cup \mathbb{Q}_{\mathbf{p},t}^{\text{new}} - \mathbb{Q}_{\mathbf{p},t}^{\text{old}}$$
$$\mathcal{Q}_{\mathbf{q},t} = \mathcal{Q}_{\mathbf{q},t-1} \cup Q, \quad \text{if } \texttt{addtoeqn}(\mathbf{q}, Q)$$

Demands. There are four operations for manipulating demands.

- $\mathbf{d} = \texttt{newdemand}$
- $\texttt{decommissiondemand}(\mathbf{d})$
- $\texttt{movedemand}(\mathbf{d}, \mathbf{p})$
- $\texttt{addtodemand}(\mathbf{d}, E)$

Let t be the current instant and p the current port. Suppose that:

- $\mathbb{D}_{\mathbf{p},t}^{\text{new}}$ is the set of new demands;
- $\mathbb{D}_{\mathbf{p},t}^{\text{old}}$ is the set of decommissioned demands;
- $\mathbb{D}_{\mathbf{p}',\mathbf{p},t}^{\text{move}}$ is the set of demands that are moving from port $\mathbf{p}'$ to port $\mathbf{p}$.
- $\mathbb{D}_{\mathbf{p},t}^{\text{add}}$ is a set of pairs of the form $(\mathbf{d}, E)$, meaning the demand set $\mathbf{d}$ is being augmented with demand E.

Then:

$$\mathbb{D}_{\mathbf{p},t} = \bigcup_{\mathbf{p}' \in \mathbb{P}_{t-1}} \mathbb{D}^{\text{move}}_{\mathbf{p}',\mathbf{p},t} \cup \mathbb{D}_{\mathbf{p},t-1} \cup \mathbb{D}^{\text{new}}_{\mathbf{p},t} - \mathbb{D}^{\text{old}}_{\mathbf{p},t}$$

$$\mathcal{D}_{\mathbf{d},t} = \mathcal{D}_{\mathbf{d},t-1} \cup E, \quad \text{if } \texttt{addtodemand}(\mathbf{d}, E)$$

Threads. There are four operations for manipulating threads.

- $\mathbf{a} = \texttt{newthread}$
- $\texttt{decommissionthread}(\mathbf{a})$
- $\texttt{movethread}(\mathbf{a}, \mathbf{p})$
- $\texttt{addtothread}(\mathbf{a}, Q)$

Let t be the current instant and $\mathbf{p}$ the current port. Suppose that:

- $\mathbb{A}^{\text{new}}_{\mathbf{p},t}$ is the set of new threads;
- $\mathbb{A}^{\text{old}}_{\mathbf{p},t}$ is the set of decommissioned threads;
- $\mathbb{A}^{\text{move}}_{\mathbf{p}',\mathbf{p},t}$ is the set of threads that are moving from port $\mathbf{p}'$ to port $\mathbf{p}$.
- $\mathbb{A}^{\text{add}}_{\mathbf{p},t}$ is a set of pairs of the form $(\mathbf{a}, Q)$, meaning the thread $\mathbf{a}$ is being augmented with equations Q defining variables `state` and `demand`.

Then:

$$\mathbb{A}_{\mathbf{p},t} = \bigcup_{\mathbf{p}' \in \mathbb{P}_{t-1}} \mathbb{A}^{\text{move}}_{\mathbf{p}',\mathbf{p},t} \cup \mathbb{A}_{\mathbf{p},t-1} \cup \mathbb{A}^{\text{new}}_{\mathbf{p},t} - \mathbb{A}^{\text{old}}_{\mathbf{p},t}$$

$$\mathcal{A}_{\mathbf{a},t} = \mathcal{A}_{\mathbf{a},t-1} \cup Q, \quad \text{if } \texttt{addtodemand}(\mathbf{a}, Q)$$

Semantics. The semantics of a system is straightforward:

For each instant t
  For each port $\mathbf{p} \in \mathbb{P}_t$
    For each demand set $\mathbf{d} \in \mathbb{D}_{\mathbf{p},t}$
      For each demand $E \in \mathbf{d}$
        Execute E `@` $[\texttt{time} : t]$
    For each thread $\mathbf{a} \in \mathbb{A}_{\mathbf{p},t}$
      Execute (`demand @ state`) `@` $[\texttt{time} : t, \texttt{thread} : \mathbf{a}]$

6 Conclusion

In a synchronous TransLucid system, it is possible to—completely declaratively—support many different styles of programming. A standard single-loop reactive system can be defined using just a single set of equations and a single set of demands. A software system where the software is updated in real time can be defined using multiple sets of equations. A deterministic multithreaded system can be built using multiple threads.

Future work involves developing real applications and extending the model to situations in which synchronous time is not universally applicable.

References

1. John Plaice and Blanca Mancilla. Cartesian programming: The TransLucid programming language. In *Dagstuhl Seminar 08271*, 2008.