Labelled Transition Systems For a Game Graph

Soren Lassen[1] Paul Blain Levy[2]

[1]Google, Inc.

[2]University of Birmingham

March 29, 2009

Outline

1. Adapting LTSs to games
2. Example: the applicative LTS for call-by-push-value
3. Discussion of normal form LTSs

Labelled Transition Systems

Definition

An alphabet $\mathcal{A}$ is a countable set of actions.

Definition

- A LTS over $\mathcal{A}$ is a set S of nodes and a function $S \xrightarrow{\theta} \mathcal{P}(\mathcal{A} \times S)$.
  Coalgebra for $S \mapsto \mathcal{P}(\mathcal{A} \times S)$
- A LTS with divergence over $\mathcal{A}$ is a set S of nodes and a function $S \xrightarrow{\theta} \mathcal{P}((\mathcal{A} \times S) + \{\Uparrow\})$. Coalgebra for $S \mapsto \mathcal{P}((\mathcal{A} \times S) + \{\Uparrow\})$

Can we adapt all this to (alternating) two-player games?

- We must distinguish between Proponent (output) actions and Opponent (input) actions. (cf. Moore and Mealy machines)
- The set of available actions must change through time (cf. typed transition systems).

Game graphs

We replace the definition of alphabet as follows.

Definition

A game graph $\mathcal{M}$ consists of

- a set $\mathcal{M}_{\mathrm{act}}$ of *active modes* (rough idea: mode = type)
- a set $\mathcal{M}_{\mathrm{pass}}$ of *passive modes*
- for each active mode $m \in \mathcal{M}_{\mathrm{act}}$, a countable set $\mathcal{M}_{\mathrm{P}}(m)$ of *Proponent-actions* from m
- a function $\sum_{m \in \mathcal{M}_{\mathrm{act}}} \mathcal{M}_{\mathrm{P}}(m) \xrightarrow{\mathrm{tgt}_{\mathrm{P}}} \mathcal{M}_{\mathrm{pass}}$
- for each passive mode $m \in \mathcal{M}_{\mathrm{pass}}$, a countable set $\mathcal{M}_{\mathrm{O}}(m)$ of *Opponent-actions* from m
- a function $\sum_{m \in \mathcal{M}_{\mathrm{pass}}} \mathcal{M}_{\mathrm{O}}(m) \xrightarrow{\mathrm{tgt}_{\mathrm{O}}} \mathcal{M}_{\mathrm{act}}$

Game graphs

We replace the definition of alphabet as follows.

Definition

A game graph $\mathcal{M}$ consists of

- a set $\mathcal{M}_{\mathrm{act}}$ of *active modes* (rough idea: mode = type)
- a set $\mathcal{M}_{\mathrm{pass}}$ of *passive modes*
- for each active mode $m \in \mathcal{M}_{\mathrm{act}}$, a countable set $\mathcal{M}_{\mathrm{P}}(m)$ of *Proponent-actions* from m
- a function $\sum_{m \in \mathcal{M}_{\mathrm{act}}} \mathcal{M}_{\mathrm{P}}(m) \xrightarrow{\mathrm{tgt}_{\mathrm{P}}} \mathcal{M}_{\mathrm{pass}}$
- for each passive mode $m \in \mathcal{M}_{\mathrm{pass}}$, a countable set $\mathcal{M}_{\mathrm{O}}(m)$ of *Opponent-actions* from m
- a function $\sum_{m \in \mathcal{M}_{\mathrm{pass}}} \mathcal{M}_{\mathrm{O}}(m) \xrightarrow{\mathrm{tgt}_{\mathrm{O}}} \mathcal{M}_{\mathrm{act}}$

These are not transitions systems. (cf. Hyvernat's Janus systems)

LTS over a game graph

Definition

Let $\mathcal{M}$ be a game graph. A LTS with divergence over $\mathcal{M}$ consists of the following:

- for each active mode m, a set $\mathcal{S}_{\mathrm{act}}(m)$ of *active nodes in mode m*
- for each passive mode m, a set $\mathcal{S}_{\mathrm{pass}}(m)$ of *passive nodes in mode m*
- for each active mode m, a function

  $$\mathcal{S}_{\mathrm{act}}(m) \xrightarrow{\theta_{\mathrm{act}}(m)} \mathcal{P}((\sum_{i \in \mathcal{M}_{\mathrm{P}}(m)} \mathcal{S}_{\mathrm{pass}} \mathrm{tgt}_{\mathrm{P}}(m,i)) + \{\Uparrow\})$$

- for each passive mode m, a function

  $$\mathcal{S}_{\mathrm{pass}}(m) \xrightarrow{\theta_{\mathrm{pass}}(m)} \prod_{i \in \mathcal{M}(m)} \mathcal{S}_{\mathrm{act}} \mathrm{tgt}_{\mathrm{O}}(m,i) \; .$$

- For an active node n, we write $n \overset{i}{\Rightarrow} n'$ and $n \Uparrow$
- For a passive node n, we write $n : i$ for the node we move to after inputting i.

LTS wrt an endofunctor

Let $\mathcal{M}$ be a game graph, and let R be an endofunctor on **Set**.

Definition

Let $\mathcal{M}$ be a game graph. A LTS over $\mathcal{M}$ wrt R consists of the following:

- for each active mode m, a set $\mathcal{S}_{\mathsf{act}}(m)$ of *active nodes in mode* m
- for each passive mode m, a set $\mathcal{S}_{\mathsf{pass}}(m)$ of *passive nodes in mode* m
- for each active mode m, a function

  $$\mathcal{S}_{\mathsf{act}}(m) \xrightarrow{\theta_{\mathsf{act}}(m)} R \sum_{i \in \mathcal{M}_{\mathsf{P}}(m)} \mathcal{S}_{\mathsf{pass}} \mathsf{tgt}_{\mathsf{P}}(m, i)$$

- for each passive mode m, a function

  $$\mathcal{S}_{\mathsf{pass}}(m) \xrightarrow{\theta_{\mathsf{pass}}(m)} \prod_{i \in \mathcal{M}(m)} \mathcal{S}_{\mathsf{act}} \mathsf{tgt}_{\mathsf{O}}(m, i) \ .$$

LTS as a coalgebra

Let $\mathcal{M}$ be a game graph, and let R be an endofunctor on **Set**.

Definition

The endofunctor $R_{\mathcal{M}}$ on $\mathbf{Set}^{\mathcal{M}_{\mathsf{act}}} \times \mathbf{Set}^{\mathcal{M}_{\mathsf{pass}}}$ is given by

$$\begin{array}{rcl} \langle \mathcal{S}_{\mathsf{act}}, \mathcal{S}_{\mathsf{pass}} \rangle & \mapsto & \langle \lambda m \in \mathcal{M}_{\mathsf{act}}. R \sum_{i \in \mathcal{M}_{\mathsf{P}}(m)} \mathcal{S}_{\mathsf{pass}} \mathsf{tgt}_{\mathsf{P}}(m, i), \\ & & \lambda m \in \mathcal{M}_{\mathsf{pass}}. \prod_{i \in \mathcal{M}_{\mathsf{O}}(m)} \mathcal{S}_{\mathsf{act}} \mathsf{tgt}_{\mathsf{O}}(m, i) \rangle \end{array}$$

Definition

A LTS over $\mathcal{M}$ wrt R is a coalgebra for $R_{\mathcal{M}}$.

Bisimulation

Let $\mathcal{R}$ be a mode-indexed binary relation between $\mathcal{S}$ and $\mathcal{S}'$, LTSs over a game graph $\mathcal{M}$.
It is a convex bisimulation when the following conditions hold.

Proponent actions are matched

For active nodes $n \; \mathcal{R} \; n'$ in mode m

- if $n \stackrel{i}{\Rightarrow} p$ then there exists p' s.t. $n' \stackrel{i}{\Rightarrow} p'$ and $p \; \mathcal{R} \; p'$
- if $n' \stackrel{i}{\Rightarrow} p'$ then there exists p s.t. $n \stackrel{i}{\Rightarrow} p$ and $p \; \mathcal{R} \; p'$
- $n \Uparrow$ iff $n' \Uparrow$

Opponent actions are matched

For passive nodes $n \; \mathcal{R} \; n'$ in mode m,

- $n : i \; \mathcal{R} \; n' : i$ for each $i \in \mathcal{M}_{\mathrm{O}}(m)$

The largest convex bisimulation is convex bisimilarity.

Call-By-Push-Value Syntax

Types (can also include type recursion)

$$\begin{array}{lll} \text{value types} & A ::= & U\underline{B} \mid \sum_{i\in I} A_i \mid 1 \mid A \times A \\ \text{computation types} & \underline{B} ::= & FA \mid \prod_{i\in I} \underline{B}_i \mid A \to \underline{B} \end{array}$$

Terms (including recursion and countable nondeterminism)

Judgements $\Gamma \vdash^{\mathsf{v}} V : A \quad \Gamma \vdash^{\mathsf{c}} M : \underline{B}$

$$\begin{array}{lll} \text{values} & V ::= & \mathrm{x} \mid \mathtt{thunk}\ M \mid \langle \hat{\imath}, V \rangle \mid \langle V, V \rangle \\ \text{computations} & M ::= & \mathtt{let}\ V\ \mathtt{be}\ \mathrm{x}.\ M \mid \mathtt{return}\ V \mid M\ \mathtt{to}\ \mathrm{x}.\ M \\ & & \mid \lambda \mathrm{x}.M \mid MV \mid \lambda\{i.M_i\}_{i\in I} \mid M\hat{\imath} \mid \mathtt{force}\ V \\ & & \mid \mathtt{pm}\ V\ \mathtt{as}\ \{\langle i, \mathrm{x}\rangle.\ M_i\}_{i\in I} \mid \mathtt{pm}\ V\ \mathtt{as}\ \langle \mathrm{x}, \mathrm{y}\rangle.\ M \\ & & \mid \mathtt{rec}\ \mathrm{x}.\ M \mid \mathtt{choose}\ n \in \mathbb{N}.\ M_n \end{array}$$

Applicative Bisimulation (Abramsky 1990)

A type-indexed relation $\mathcal{R}$ on closed terms is a convex applicative bisimulation when the following hold.

If $M \; \mathcal{R} \; M' : FA$

- $M \Downarrow \texttt{return}\ V$ implies there exists V' s.t. $M' \Downarrow V'$ and $V \; \mathcal{R} \; V' : A$
- $M' \Downarrow \texttt{return}\ V'$ implies there exists V s.t. $M \Downarrow V$ and $V \; \mathcal{R} \; V' : A$
- $M \Uparrow$ iff $M' \Uparrow$

Requirements at other types

- If $\langle \hat{\imath}, V \rangle \; \mathcal{R} \; \langle \hat{\imath}', V' \rangle : \sum_{i \in I} A_i$ then $\hat{\imath} = \hat{\imath}'$ and $V \; \mathcal{R} \; V' : A_{\hat{\imath}}$.
- If $\langle V, W \rangle \; \mathcal{R} \; \langle V', W' \rangle : A \times B$ then $V \; \mathcal{R} \; V' : A$ and $W \; \mathcal{R} \; W' : B$.
- If $V \; \mathcal{R} \; V' : \; U\underline{B}$ then $\texttt{force}\ V \; \mathcal{R} \; \texttt{force}\ V' : \underline{B}$.
- If $M \; \mathcal{R} \; M' : A \to \underline{B}$ then $MV \; \mathcal{R} \; M'V : \underline{B}$ for each $\vdash^{\mathsf{v}} V : \underline{B}$.
- If $M \; \mathcal{R} \; M' : \prod_{i \in I} \underline{B}_i$ then $M\hat{\imath} \; \mathcal{R} \; M'\hat{\imath}$ for each $\hat{\imath} \in I$.

Some conditions are Proponent flavoured

If $M \mathcal{R} M' : FA$

- $M \Downarrow \texttt{return}\ V$ implies there exists V' s.t. $M' \Downarrow V'$ and $V \mathcal{R} V' : A$
- $M' \Downarrow \texttt{return}\ V'$ implies there exists V s.t. $M \Downarrow V$ and $V \mathcal{R} V' : A$
- $M \Uparrow$ iff $M' \Uparrow$

Requirements at other types

- If $\langle \hat{\imath}, V \rangle \mathcal{R} \langle \hat{\imath}', V' \rangle : \sum_{i \in I} A_i$ then $\hat{\imath} = \hat{\imath}'$ and $V = V'$.
- If $\langle V, W \rangle \mathcal{R} \langle V', W' \rangle : A \times B$ then $V \mathcal{R} V' : A$ and $W \mathcal{R} W' : B$.
- If $V \mathcal{R} V' : U\underline{B}$ then $\texttt{force}\ V \mathcal{R} \texttt{force}\ V' : \underline{B}$.
- If $M \mathcal{R} M' : A \rightarrow \underline{B}$ then $MV \mathcal{R} M'V : \underline{B}$ for each $\vdash^{\mathsf{v}} V : \underline{B}$.
- If $M \mathcal{R} M' : \prod_{i \in I} \underline{B}_i$ then $M\hat{\imath} \mathcal{R} M'\hat{\imath}$ for each $\hat{\imath} \in I$.

Some conditions are Opponent flavoured

If $M \mathcal{R} M' : FA$

- $M \Downarrow \texttt{return } V$ implies there exists V' s.t. $M' \Downarrow V'$ and $V \mathcal{R} V' : A$
- $M' \Downarrow \texttt{return } V'$ implies there exists V s.t. $M \Downarrow V$ and $V \mathcal{R} V' : A$
- $M \Uparrow$ iff $M' \Uparrow$

Requirements at other types

- If $\langle \hat{\imath}, V \rangle \mathcal{R} \langle \hat{\imath}', V' \rangle : \sum_{i \in I} A_i$ then $\hat{\imath} = \hat{\imath}'$ and $V = V'$.
- If $\langle V, W \rangle \mathcal{R} \langle V', W' \rangle : A \times B$ then $V \mathcal{R} V' : A$ and $W \mathcal{R} W' : B$.
- If $V \mathcal{R} V' : U\underline{B}$ then $\texttt{force } V \mathcal{R} \texttt{ force } V' : \underline{B}$.
- If $M \mathcal{R} M' : A \rightarrow \underline{B}$ then $MV \mathcal{R} M'V : \underline{B}$ for each $\vdash^{\mathsf{v}} V : \underline{B}$.
- If $M \mathcal{R} M' : \prod_{i \in I} \underline{B}_i$ then $M\hat{\imath} \mathcal{R} M'\hat{\imath}$ for each $\hat{\imath} \in I$.

Ultimate Patterns: for the Proponent actions

Intuition (Abramsky-McCusker)

Every value type A is isomorphic to one of the form $\sum_{i \in I} U\underline{B}_i$

We want to decompose a closed value into

- an ultimate pattern—the tags
- and the filling a value sequence—the rest, consisting of thunks.

Example:

$$\langle i_0, \langle \langle \langle \texttt{thunk}\ M, \texttt{thunk}\ M' \rangle, \texttt{thunk}\ M'' \rangle, \langle i_1, \texttt{thunk}\ M''' \rangle \rangle \rangle$$

We decompose this into

the ultimate pattern $\langle i_0, \langle \langle \langle -_{U\underline{B}}, -_{U\underline{B}'} \rangle, -_{U\underline{B}''} \rangle, \langle i_1, -_{U\underline{B}'''} \rangle \rangle \rangle$

and the filling $\texttt{thunk}\ M$, $\texttt{thunk}\ M'$, $\texttt{thunk}\ M''$, $\texttt{thunk}\ M'''$

The Ultimate Patterns

We write ulpatt(A) for the set of ultimate patterns of type A.

These sets are defined by mutual induction.

- $-_{U\underline{B}} \in \mathrm{ulpatt(UA)}$.
- If $p \in \mathrm{ulpatt(A)}$ and $p' \in \mathrm{ulpatt(A')}$ then $\langle p, p' \rangle \in \mathrm{ulpatt(A \times A')}$.
- If $\hat{\imath} \in I$ and $p \in \mathrm{ulpatt(A_{\hat{\imath}})}$ then $\langle \hat{\imath}, p \rangle \in \mathrm{ulpatt(\sum_{i \in I} A_i)}$.

The Ultimate Patterns

We write ulpatt(A) for the set of ultimate patterns of type A.

These sets are defined by mutual induction.

- $-_{U\underline{B}} \in \mathrm{ulpatt}(\mathrm{UA})$.
- If $p \in \mathrm{ulpatt}(\mathrm{A})$ and $p' \in \mathrm{ulpatt}(\mathrm{A}')$ then $\langle p, p' \rangle \in \mathrm{ulpatt}(\mathrm{A} \times \mathrm{A}')$.
- If $\hat{\imath} \in I$ and $p \in \mathrm{ulpatt}(\mathrm{A}_{\hat{\imath}})$ then $\langle \hat{\imath}, p \rangle \in \mathrm{ulpatt}(\sum_{\mathrm{i} \in \mathrm{I}} \mathrm{A}_{\mathrm{i}})$.

We write $H(p)$ for the sequence of types of the holes of p. They are all U types.

Ultimate Pattern Matching Theorem

Theorem for Closed Values

Any closed value $\vdash V : A$ is $p(\overrightarrow{W})$ for unique $p \in \text{ulpatt(A)}$ and filling $\vdash \overrightarrow{V} : A$.

Theorem for Open Values

Let Γ be a context in which each identifier has a U type.
Any value $\Gamma \vdash^{\mathsf{v}} V : A$ is $p(\overrightarrow{W})$ for unique $p \in \text{ulpatt(A)}$ and filling $\Gamma \vdash \overrightarrow{V} : A$.

Operand List—for Opponent Actions

In the applicative rules, a closed computation gets applied to a list of closed operands until it becomes a computation of F type.

We write OpList($\underline{B}$) for the set of operand lists from $\underline{B}$.

Operand List—for Opponent Actions

In the applicative rules, a closed computation gets applied to a list of closed operands until it becomes a computation of F type.

We write OpList($\underline{B}$) for the set of operand lists from $\underline{B}$.
More generally, OpList($\Gamma \mid \underline{B}$) when the operands are in context Γ.

Operand List—for Opponent Actions

In the applicative rules, a closed computation gets applied to a list of closed operands until it becomes a computation of F type.

We write OpList($\underline{B}$) for the set of operand lists from $\underline{B}$.
More generally, OpList($\Gamma \mid \underline{B}$) when the operands are in context Γ.

- $\texttt{nil}_{FA} \in \mathrm{OpList}(\Gamma \mid FA)$.
- If $\Gamma \vdash^{\mathsf{v}} V : A$ and $o \in \mathrm{OpList}(\Gamma \mid \underline{B})$ then $V :: o \in \mathrm{OpList}(\Gamma \mid A \to \underline{B})$.
- If $\hat{\imath} \in I$ and $o \in \mathrm{OpList}(\Gamma \mid \underline{B}_{\hat{\imath}})$ then $\hat{\imath} :: o \in \mathrm{OpList}(\Gamma \mid \prod_{i \in I} \underline{B}_i)$.

We write $E(o)$ for the end-type of o, which is an F type.

If $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ and $o \in \mathrm{OpList}(\Gamma \mid \underline{B})$ then $\Gamma \vdash^{\mathsf{c}} Mo : E(l)$.

The Applicative Game Graph

We want Proponent actions to be ultimate patterns, and Opponent actions to be operand lists.

Definition of the game graph

- An active mode is an F type.
- A passive mode is a finite sequence of U types.
- A Proponent action from the active mode FA is $p \in \mathrm{ulpatt(A)}$. Its target is $H(p)$.
- An Opponent action from the passive mode $U\underline{B}_0, \ldots, U\underline{B}_{m-1}$ is a pair (j, o) where $j < m$ and $o \in \mathrm{OpList}(\mid \underline{B}_j)$. Its target is $E(o)$.

The Applicative LTS

Definition of the transition system

- An active node in mode FA is a closed computation.
- A passive node in mode $U\underline{B}_0, \ldots, U\underline{B}_{m-1}$ is a sequence of closed values.
- For an active node $\vdash^{\mathsf{c}} M : FA$
  - if $M \Downarrow \mathtt{return}\ V$, then $V = p(\overrightarrow{W})$ and $M \overset{p}{\Rightarrow} \overrightarrow{W}$ in the LTS
  - if $M \Uparrow$, then $M \Uparrow$ in the LTS.
- For a passive node $\vdash^{\mathsf{v}} \overrightarrow{V} : \overrightarrow{U\underline{B}_i}$

$$(\overrightarrow{V}) : (j, o) \stackrel{\text{def}}{=} (\mathtt{force}\ V_j)\ o$$

The Applicative LTS

Definition of the transition system

- An active node in mode FA is a closed computation.
- A passive node in mode $U\underline{B}_0, \ldots, U\underline{B}_{m-1}$ is a sequence of closed values.
- For an active node $\vdash^{\mathsf{c}} M : FA$
  - if $M \Downarrow \mathtt{return}\ V$, then $V = p(\overrightarrow{W})$ and $M \overset{p}{\Rightarrow} \overrightarrow{W}$ in the LTS
  - if $M \Uparrow$, then $M \Uparrow$ in the LTS.
- For a passive node $\vdash^{\mathsf{v}} \overrightarrow{V} : \overrightarrow{U\underline{B}_i}$

$$(\overrightarrow{V}) : (j, o) \stackrel{\text{def}}{=} (\mathtt{force}\ V_j)\ o$$

LTS bisimilarity coincides with applicative bisimilarity.

Proponent actions vs Opponent actions

There is an asymmetry between the actions of the two players.

The set of Proponent actions from *FA* is ulpatt(A). This is a countable set.

Proponent actions vs Opponent actions

There is an asymmetry between the actions of the two players.
The set of Proponent actions from FA is ulpatt(A). This is a countable set.

The set of Opponent actions from $U\underline{B}_0, \ldots, U\underline{B}_{m-1}$ is $\sum_{i<m} \mathrm{OpList}(\underline{B}_i)$.
This is an uncountable set.

Proponent actions vs Opponent actions

There is an asymmetry between the actions of the two players.
The set of Proponent actions from FA is ulpatt(A). This is a countable set.

The set of Opponent actions from $U\underline{B}_0, \ldots, U\underline{B}_{m-1}$ is $\sum_{i<m} \mathrm{OpList}(\underline{B}_i)$.
This is an uncountable set.

So we do not have a game graph.

Proponent actions vs Opponent actions

There is an asymmetry between the actions of the two players.
The set of Proponent actions from FA is ulpatt(A). This is a countable set.

The set of Opponent actions from $U\underline{B}_0, \ldots, U\underline{B}_{m-1}$ is $\sum_{i<m}\mathrm{OpList}(\underline{B}_i)$.
This is an uncountable set.

So we do not have a game graph.

But the target mode of an Opponent action depends only on the tags that appear in it.

Game graph with parameters

Definition

A game graph with parameters $\mathcal{M}$ consists of

- a game graph
- for each active mode m and Opponent action $i \in \mathcal{M}_{\mathsf{O}}(m)$ a set $\mathcal{M}_{\mathsf{O}}^{\mathsf{param}}(m, i)$ of *parameters* for i.

The set of parameters doesn't need to be countable.

The target mode of a Opponent action doesn't depend on the parameters.

LTS over a game graph with parameters

Definition

Let $\mathcal{M}$ be a game graph with parameters. A *LTS with divergence* over $\mathcal{M}$ consists of the following:

- for each active mode m, a set $\mathcal{S}_{\text{act}}(m)$ of *active nodes in mode m*
- for each passive mode m, a set $\mathcal{S}_{\text{pass}}(m)$ of *passive nodes in mode m*
- for each active mode m, a function

  $$\mathcal{S}_{\text{act}}(m) \xrightarrow{\theta_{\text{act}}(m)} \mathcal{P}((\textstyle\sum_{i \in \mathcal{M}_{\text{P}}(m)} \mathcal{S}_{\text{pass}} \text{tgt}_{\text{P}}(m,i)) + \{\Uparrow\})$$

- for each passive mode m, a function

  $$\mathcal{S}_{\text{pass}}(m) \xrightarrow{\theta_{\text{pass}}(m)} \textstyle\prod_{i \in \mathcal{M}(m)} (\mathcal{M}_{\text{O}}^{\text{param}}(m,i) \to \mathcal{S}_{\text{act}} \text{tgt}_{\text{O}}(m,i)) \ .$$

- For an active node n, we write $n \overset{i}{\Rightarrow} n'$ and $n \Uparrow$
- For a passive node n, we write $n : i(a)$ for the node we move to after inputting action i and parameter a.

Ultimate patterns for operand list

Intuition

Every computation type $\underline{B}$ is isomorphic to one of the form $\prod_{i \in I}(U\underline{A}_i \to FB_i)$.

An operand list is a sequence of values and tags. So just like a single value, it can be ultimately pattern matched.

Ultimate patterns for operand list

Intuition

Every computation type $\underline{B}$ is isomorphic to one of the form $\prod_{i \in I}(U\underline{A}_i \to FB_i)$.

An operand list is a sequence of values and tags. So just like a single value, it can be ultimately pattern matched.

We write olup($\underline{B}$) for the set of operand list ultimate patterns.

Ultimate patterns for operand list

Intuition

Every computation type $\underline{B}$ is isomorphic to one of the form $\prod_{i \in I}(U\underline{A}_i \to FB_i)$.

An operand list is a sequence of values and tags. So just like a single value, it can be ultimately pattern matched.

We write olup($\underline{B}$) for the set of operand list ultimate patterns.

These sets are defined by mutual induction.

- $\mathtt{nil}\ _{FA} \in \mathrm{olup}(FA)$.
- If $p \in \mathrm{ulpatt(A)}$ and $q \in \mathrm{olup}(\underline{B})$ then $p :: q \in \mathrm{olup}(A \to \underline{B})$.
- If $\hat{\imath} \in I$ and $q \in \mathrm{olup}(\underline{B}_{\hat{\imath}})$ then $\hat{\imath} :: q \in \mathrm{olup}(\prod_{i \in I} \underline{B}_i)$.

Ultimate patterns for operand list

Intuition

Every computation type $\underline{B}$ is isomorphic to one of the form $\prod_{i \in I}(U\underline{A}_i \to FB_i)$.

An operand list is a sequence of values and tags. So just like a single value, it can be ultimately pattern matched.

We write $\mathrm{olup}(\underline{B})$ for the set of operand list ultimate patterns.

These sets are defined by mutual induction.

- $\texttt{nil}\ _{FA} \in \mathrm{olup}(FA)$.
- If $p \in \mathrm{ulpatt}(\mathrm{A})$ and $q \in \mathrm{olup}(\underline{B})$ then $p :: q \in \mathrm{olup}(A \to \underline{B})$.
- If $\hat{\imath} \in I$ and $q \in \mathrm{olup}(\underline{B}_{\hat{\imath}})$ then $\hat{\imath} :: q \in \mathrm{olup}(\prod_{i \in I} \underline{B}_i)$.

We write $H(q)$ for the sequence of types—all U types—of the holes of q. We write $E(q)$ for the end-type of q, which is an F type.

The Ultimate Pattern Matching Theorem For Operand Lists

Theorem for Closed Operand Lists

Any closed operand list $o \in \mathrm{OpList}(\underline{B})$ is $q(\overrightarrow{W})$ for unique $q \in \mathrm{olup}(\underline{B})$ and filling $\vdash^{\mathsf{v}} \overrightarrow{W} : H(q)$.

Theorem for Open Operand Lists

Let Γ be a context in which each identifier has a U type.
Any closed operand list $o \in \mathrm{OpList}(\Gamma \mid \underline{B})$ is $q(\overrightarrow{W})$ for unique $q \in \mathrm{olup}(\underline{B})$ and filling $\Gamma \vdash^{\mathsf{v}} \overrightarrow{W} : H(q)$.

The Applicative Game Graph, Take Two

Definition of the game graph with parameters

- An active mode is an F type.
- A passive mode is a finite sequence of U types.
- A Proponent action from the active mode FA is $p \in \mathrm{ulpatt}(A)$. Its target is $H(p)$.
- An Opponent action from the passive mode $U\underline{B}_0, \ldots, U\underline{B}_{m-1}$ is a pair (j, q) where $j < m$ and $q \in \mathrm{olup}(\underline{B}_j)$. Its target is $E(q)$.
- A parameter for (j, q) is a value sequence $\vdash^{\mathsf{v}} \overrightarrow{V} : H(q)$.

The Applicative LTS, Take Two

Definition of the transition system

- An active node in mode FA is a closed computation.
- A passive node in mode $U\underline{B}_0, \ldots, U\underline{B}_{m-1}$ is a sequence of closed values.
- For an active node $\vdash^{\mathsf{c}} M : FA$
  - if $M \Downarrow \texttt{return } V$, then $V = p(\overrightarrow{W})$ and $M \stackrel{p}{\Rightarrow} \overrightarrow{W}$ in the LTS
  - if $M \Uparrow$, then $M \Uparrow$ in the LTS.
- For a passive node $\vdash^{\mathsf{v}} \overrightarrow{V} : \overrightarrow{U\underline{B}_i}$

$$(\overrightarrow{V}) : (j, q)(\overrightarrow{W}) \stackrel{\text{def}}{=} (\texttt{force } V_j)\ q(\overrightarrow{W})$$

Normal Form (Open) Bisimulation

In the applicative LTS, when the Opponent plays an operand list ultimate pattern, he supplies a filling of closed values.

Normal Form (Open) Bisimulation

In the applicative LTS, when the Opponent plays an operand list ultimate pattern, he supplies a filling of closed values.

But in a normal form LTS, he does not supply a filling. It gets filled with fresh identifiers.

Normal Form (Open) Bisimulation

In the applicative LTS, when the Opponent plays an operand list ultimate pattern, he supplies a filling of closed values.

But in a normal form LTS, he does not supply a filling. It gets filled with fresh identifiers.

This requires operational semantics to be defined on open computations.

Normal Form (Open) Bisimulation

In the applicative LTS, when the Opponent plays an operand list ultimate pattern, he supplies a filling of closed values.

But in a normal form LTS, he does not supply a filling. It gets filled with fresh identifiers.

This requires operational semantics to be defined on open computations.

A mode contains two contexts Γ_P and Γ_O, representing the free identifiers possessed by each player.

Normal Form (Open) Bisimulation

In the applicative LTS, when the Opponent plays an operand list ultimate pattern, he supplies a filling of closed values.

But in a normal form LTS, he does not supply a filling. It gets filled with fresh identifiers.

This requires operational semantics to be defined on open computations.

A mode contains two contexts Γ_P and Γ_O, representing the free identifiers possessed by each player.

Such transition systems are closely related to game semantics using pointers (Laird 2007, Jagadeesan, Pitcher and Riely 2007, Lassen and Levy 2007).

Renamings—recent work with Sam Staton

A renaming $\Gamma_{\mathsf{P}} \xrightarrow{\theta_{\mathsf{P}}} \Gamma'_{\mathsf{P}}$ and a renaming $\Gamma'_{\mathsf{O}} \xrightarrow{\theta_{\mathsf{O}}} \Gamma_{\mathsf{O}}$ induce a map from the nodes in mode $\Gamma_{\mathsf{P}};\Gamma_{\mathsf{O}}$ to the nodes in mode $\Gamma'_{\mathsf{P}};\Gamma'_{\mathsf{O}}$.

Renamings—recent work with Sam Staton

A renaming $\Gamma_P \xrightarrow{\theta_P} \Gamma'_P$ and a renaming $\Gamma'_O \xrightarrow{\theta_O} \Gamma_O$ induce a map from the nodes in mode $\Gamma_P; \Gamma_O$ to the nodes in mode $\Gamma'_P; \Gamma'_O$.

It is an easy result that bisimilarity is preserved by renaming.

Renamings—recent work with Sam Staton

A renaming $\Gamma_{\mathsf{P}} \xrightarrow{\theta_{\mathsf{P}}} \Gamma'_{\mathsf{P}}$ and a renaming $\Gamma'_{\mathsf{O}} \xrightarrow{\theta_{\mathsf{O}}} \Gamma_{\mathsf{O}}$ induce a map from the nodes in mode $\Gamma_{\mathsf{P}}; \Gamma_{\mathsf{O}}$ to the nodes in mode $\Gamma'_{\mathsf{P}}; \Gamma'_{\mathsf{O}}$.

It is an easy result that bisimilarity is preserved by renaming.

We make this automatic by adapting the notion of game graph and LTS to incorporate morphisms between modes.