Control Data Flow Graphs
Further work on the definitions...

Sue Tyerman

Quick refresh (1)

A CDFG is a tuple \( C = (V, E, I, O) \), where:
1. \( (V, E) \) is a connected, directed graph,
2. \( I \) is a function mapping each vertex to an (input) enabling function,
3. \( O \) is a function mapping each vertex to a possible set of outputs.

\[ V = \{v_o, \ldots, v_n\} \text{ is a finite set whose elements are nodes} \]

\[ E \subseteq V \times V \text{ is an irreflexive flow relation whose elements are directed edges} \]

\[ V \cap E = \emptyset \]
Quick refresh (2)

\*w = \{(v, w) \in E \mid v \in V, w \in V\}

\text{in-deg}(w) = |\*w|

w* = \{(w, v) \in E \mid v \in V, w \in V\}

\text{out-deg}(w) = |w*|

I(v) \subset \mathcal{F}(\*v) - \{\{}\} \text{ where } v \in V

O(v) \subset \mathcal{F}(v*) - \{\{}\} \text{ where } v \in V

\forall e \in E : R(e) \in \text{Boolean}

\forall v \in V : F(v) \in \text{Boolean}

P = (R, F)

S \in I(v) : \forall e \in S : R(e)

en(V) = \{v \in V \mid \neg F(v) \land e \in S \in I(v) : \forall e \in S : R(e)\}

Quick refresh (3)

A node v which is enabled on the basis of \( S \in I(v) \)

can start firing which causes a change of state

from \( (R, F) \) to \( (R', F') \) where

\( F'(w) = F(w) \) if \( w \neq v \)

\( F'(v) = \text{true} \) if \( w = v \)

and

\( R'(e) = R(e) \) if \( e \notin S \)

\( R'(e) = \text{false} \) if \( e \in S \)
Quick refresh (4)

\[ T \in O(v) : \forall e \in T : \neg R(e) \]

A node \( v \) may complete firing on the basis of \( T \in O(v) \) and cause a change of state from \( (R, F) \) to \( (R', F') \) where

\[ F'(w) = F(w) \quad \text{if } w \neq v \]
\[ F'(v) = \text{false} \quad \text{if } w = v \]

and

\[ R'(e) = R(e) \quad \text{if } e \notin T \]
\[ R'(e) = \text{true} \quad \text{if } e \in T \]

Some new work (1)

- Original
  \[ I(v) \supseteq \{v\} - \{\} \text{ where } v \in V \]
- Add a requirement that all inputs to a vertex must belong to at least one of the possible vertex enablings
  \[ \forall e \in \cdot V : \exists S \in I(v) : e \in S \]
Some new work (2)

- Original

\[ O(v) \subseteq GF(v^*) \setminus \{\emptyset\} \text{ where } v \in V \]

- Add a requirement that all outputs of a vertex must belong to at least one of the possible vertex output sets

\[ \forall e \in v^* \text{ : } T \in O(v) : e \in T \]

Some new work (3)

- Original

\[ \forall e \in E : R(e) \in \text{ Boolean} \]
\[ \forall v \in V : F(v) \in \text{ Boolean} \]
\[ P = (R, F) \]

- Edges are queues so the link between readability and writeability has altered, therefore altering what must be considered when describing the state of the CDFG

\[ \forall e \in E : R(e) \in \text{ Boolean, } W(e) \in \text{ Boolean} \]
\[ \forall v \in V : F(v) \in \text{ Boolean} \]
\[ P = (R, W, F) \]
Some new work (4)

• A pure data flow graph has no element of choice.
• All vertices execute and all paths are traversed in exactly the same manner each time a CDFG is executed.
• This requires a couple of new definitions
  \[ I(v) = \{ *v \} \text{ where } v \in V \]
  \[ O(v) = \{ v^* \} \text{ where } v \in V \]

Some new work (4)

• Another way of looking at it is that the degree of the inputs and outputs is one
  For any node \( v \) that is a pure data flow node
  \[ |I(v)| = 1 \]
  \[ |O(v)| = 1 \]
• A node that is not a pure data flow node will have some element of choice
  \[ |I(v)| > 1 \text{ or } |O(v)| > 1 \]
Some new work (5)

- The maximum number of elements an edge can hold is set by the designer and is called Capacity, while the current number of elements on the edge is called Size.
- So far we have only defined $R(e)$ and $W(e)$ to be Boolean. We can use Size and Capacity to determine the readability and writeability of an edge.

An edge may hold multiple values and so we define $Size(e)$ and $Capacity(e)$. Then

\[ R(e) = Size(e) > 0 \quad \text{and} \quad W(e) = Size(e) < Capacity(e) \]

If $Capacity(e) = 1$, then $R(e) = \neg W(e)$

Some new work (6)

- We have a definition that was useful when only one value could be on an edge
  \[ T \in O(v) : \forall e \in T : \neg R(e) \]
- Add a definition that copes with queues
  \[ T \in O(v) : \forall e \in T : W(e) \]
- If we model edges as queues and ensure in our designs that $Capacity(e)$ will always be greater than or equal to the maximum $Size(e)$, then this removes the problem of a full edge preventing a vertex from finishing firing
Some new work (7)

- **Original**
  A node $v$ which is enabled on the basis of $S \in I(v)$ can start firing which causes a change of state from $(R, F)$ to $(R', F')$ where
  \[
  F'(w) = F(w) \quad \text{if } w \neq v
  \]
  \[
  F'(v) = \text{true} \quad \text{if } w = v
  \]
  and
  \[
  R'(e) = R(e) \quad \text{if } e \notin S
  \]
  \[
  R'(e) = \text{false} \quad \text{if } e \in S
  \]

- **Modify this to**
  A node $v$ which is enabled on the basis of $S \notin I(v)$ can start firing which causes a change of state from $(R, W, F)$ to $(R', W', F')$ where
  \[
  F'(w) = F(w) \quad \text{if } w \neq v
  \]
  \[
  F'(v) = \text{true} \quad \text{if } w = v
  \]
  and
  \[
  R'(e) = R(e), W'(e) = W(e) \quad \text{if } e \notin S
  \]
  \[
  \text{Size}'(e) = \text{Size}(e) - 1, \quad \text{if } e \notin S
  \]
  \[
  W'(e) = \text{Size}'(e) < \text{Capacity}(e), \quad \text{if } e \notin S
  \]
  \[
  R'(e) = \text{Size}'(e) > 0 \quad \text{if } e \in S
  \]

- **We can replace** $W'(e) = \text{Size}'(e) < \text{Capacity}(e)$ with $W'(e) = \text{true}$

---

Some new work (8)

- **Original**
  A node $v$ may complete firing on the basis of $T \in O(v)$ and cause a change of state from $(R, F)$ to $(R', F')$ where
  \[
  F'(w) = F(w) \quad \text{if } w \neq v
  \]
  \[
  F'(v) = \text{false} \quad \text{if } w = v
  \]
  and
  \[
  R'(e) = R(e) \quad \text{if } e \notin T
  \]
  \[
  R'(e) = \text{true} \quad \text{if } e \in T
  \]

- **Modify this to**
  A node $v$ may complete firing on the basis of $T \notin O(v)$ and cause a change of state from $(R, W, F)$ to $(R', W', F')$ where
  \[
  F'(w) = F(w) \quad \text{if } w \neq v
  \]
  \[
  F'(v) = \text{false} \quad \text{if } w = v
  \]
  and
  \[
  R'(e) = R(e), W'(e) = W(e) \quad \text{if } e \notin T
  \]
  \[
  \text{Size}'(e) = \text{Size}(e) + 1, \quad \text{if } e \notin T
  \]
  \[
  W'(e) = \text{Size}'(e) < \text{Capacity}(e), \quad \text{if } e \notin T
  \]
  \[
  R'(e) = \text{Size}'(e) > 0 \quad \text{if } e \notin T
  \]

- **We can replace** $R'(e) = \text{Size}'(e) > 0$ with $R'(e) = \text{true}$
Some new work (9)

• Bringing data into the picture
• Types start to become important
• Till now we have only had “colourless” tokens
  \[ \forall e \in E : \text{Type}(e) \in Q \text{ where } Q \text{ is an infinite set of types} \]
• The queue also has to be defined
  \[
  \text{Queue}(e) = [a_1, a_2, ..., a_n]
  \]
  \[
  \text{Size}(e) = n \text{ where } n \leq \text{Capacity}(e)
  \]

Some new work (10)

• And read and write operations on the queue
• Read(e) reads from a non-empty edge
• Write(e, x) writes to a non-full edge
  If \( \text{Size}(e) > 0 \) then \( \text{Read}(e) \) transforms \( \text{queue}(e) \) to \( \text{queue}'(e) \) where \( \text{queue}'(e) = [a_2, ..., a_n] \)
  If \( \text{Size}(e) < \text{Capacity}(e) \) then \( \text{Write}(e, x) \) transforms \( \text{queue}(e) \) to \( \text{queue}'(e) \) where \( \text{queue}'(e) = [a_1, a_2, ..., a_n, x] \)
Some new work (11)

- Now we can say something about the actual edge values that are read and written.

When a vertex $v$ begins firing, the values on the edges in $S$ are read and initialise the value on the vertex ($\text{Val}(v)$) where

$$\text{Val}(v) = \text{init}_s(\text{Read}_s(e_1), \text{Read}_s(e_2)...\text{Read}_s(e_n))$$

where $S = \{e_1, e_2...e_n\}$

When a vertex $v$ completes firing, the initial value on the vertex ($\text{Val}(v)$) is transformed by the operation $X$ to $\text{Val}'(v)$ and this is used to produce the outputs on the edges of $T$ where $\text{Val}'(v) = (\text{Write}_t(e_1 \text{Val}'(v)), \text{Write}_t(e_2 \text{Val}'(v))...\text{Write}_t(e_n \text{Val}'(v)))$

where $T = \{e_1, e_2...e_n\}$

Some new work (12)

- We have

$$\mathcal{O}(v) \subseteq \mathcal{F}(v^*) \setminus \{\{\}\} \text{ where } v \in V$$

- If we consider that the vertex values $\text{Val}'(v)$ are ready to be output

$$\mathcal{O}(v, \text{Val}'(v)) \subseteq \mathcal{F}(v^*) \setminus \{\{\}\} \text{ where } v \in V$$

- For all combinations of valid enabling and the vertex operation, there must be at least one possible output set

$$|\mathcal{O}(v, \text{Val}'(v))| \geq 1$$
Inputs/outputs/data

- The vertex is where choices are made.
- Edges store and transfer data
  - Both “data” and “control”
  - It is all data
- What things can influence choice?
  - Which inputs?
  - What data on the inputs?
  - What transformations by the vertex?
  - What outputs?

What things can influence choice?

- Input edge
  - Connectors: copy, branch, merge, join – no choice
  - Priority: one edge has priority over others
- Data
  - Data affects choice of edge: inspection of values on the edge
- Transformation
  - Vertex computation determines behaviour
- Output edge
  - Connectors – no choice
  - Priority: one edge has priority over others
  - Extreme example is the “hot potato”
    - get rid of the result ASAP
    - so any edge will do