Control Data Flow Graphs
An experiment using Design/CPN

Sue Tyerman

Outline

• Informal description of CDFGs
• Current proposed general formalism for CDFGs
• Experiment using Design/CPN
• Next phase....
Informal description of CDFGs (1)

- Scheduling is the most popular application area for CDFGs
- Used as an intermediate form to carry out optimisations
- Examples in circuit design
  - Find spare cycles to insert self test circuitry
  - Save resources (components, time)

Informal description of CDFGs (2)

- Captures data and control flow in one graph
- Composed of
  - Vertices
  - Directed edges
- Event driven
  - Data or control event on an edge
- Vertices
  - Transformative operations
  - Cause and detect events
- Edges
  - Communication, storage and precedence
Informal description of CDFGs (3)

- Dynamic behaviour of CDFGs
  - When a vertex fires
    - Consumes inputs
    - Carries out transformation
    - Produces output
  - Vertices must be enabled to be able to fire
    - Vertex has an enabling function
    - The enabling function assesses all inputs
    - Firing may only require a subset of inputs

- Type and order are important in CDFGs
  - Queue vs overwrite semantics

Informal description of CDFGs (4)

- Visualisation

Quiescent Vertex

Enabled Vertex

Firing Vertex

Vertex after firing
Current proposed general formalism for CDFGs (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.

- All vertices in the set of $V$ are uniquely identified
  $V = \{v_0, \ldots, v_n\}$ is a finite set whose elements are nodes

- Each edge captures the relationship between the nodes it connects
  $E \subseteq V \times V$ is an irreflexive flow relation whose elements are directed edges

$V \cap E = \emptyset$

Current proposed general formalism for CDFGs (2)

- The set of edges directed to a vertex $w$ is the pre-set $\bullet w$, or the set of predecessor edges, of that vertex.
- This means that an edge is in the pre-set of the destination of the edge.
- The in-degree of a vertex ($\text{in-deg}(w)$) is the number of elements in the pre-set of that vertex.

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

$\text{in-deg}(w) = |\bullet w|$
Current proposed general formalism for CDFGs (3)

- The set of edges leading from a vertex \( w \) is the post-set \( w^* \), or the set of successor edges, of that vertex.
- This means that an edge is in the post-set of the source of the edge.
- The out-degree of a vertex (out-deg(w)) is the number of elements in the post-set of that vertex.

\[
\begin{align*}
  w^* &= \{(w, v) \in E | v \in V, w \in V\} \\
  \text{out-deg}(w) &= |w^*|
\end{align*}
\]

Current proposed general formalism for CDFGs (4)

- The function I maps each vertex to an enabling function.
- The enabling function is a set of subsets of the pre-set of a vertex, excluding the empty set.
- If there is only one possible enabling for a vertex, then the single element given by the enabling function must contain all members of the pre-set.
- Where there is more than one possible enabling, each member of the pre-set must contribute to at least one of those possible enablings.
- The possible set of output edges is a subset of the post set, excluding the empty set.

\[
\begin{align*}
  I(v) &= \mathcal{P}(\mathcal{P}(v^*)) - \{\}\ 	ext{where } v \in V \\
  O(v) &= \mathcal{P}(w^*) - \{\}\ 	ext{where } v \in V
\end{align*}
\]
Current proposed general formalism for CDFGs (5)

• The state $P$ of CDFG $C$ is given by a pair $(R, F)$ where $R$ is a function giving the readable status of each edge, and $F$ is a function giving the firing status of each vertex.

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

• Readable edges in the pre-set of a vertex may contribute to a number of possible enablings. Only one such $S$ will be selected when the vertex fires.

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

Current proposed general formalism for CDFGs (6)

• We define $en(V)$, the set of enabled vertices as the vertices whose enabling function returns true.

• The enabling function requires that a vertex not be firing and that there is a set of edges that are readable and satisfy the enabling function for that vertex.

\[ en(V) = \{ v \in V \mid \neg F(v) \land \exists S \in I(v) : \forall e \in S : R(e) \} \]
Current proposed general formalism for CDFGs (7)

- A vertex can start firing if it is enabled.
- In doing so, those edges that contribute to the enabling are read and are set to not readable.

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

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

and

\[
R'(e) = R(e) \quad \text{if } e \notin S \\
R'(e) = \text{false} \quad \text{if } e \in S
\]

Current proposed general formalism for CDFGs (7)

- A vertex may finish firing and produce outputs if a possible subset \( T \) of the post-set is writeable.
- Those edges that are affected by the completion of firing are written to and are set to readable.

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

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

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

and

\[
R'(e) = R(e) \quad \text{if } e \notin T \\
R'(e) = \text{true} \quad \text{if } e \in T
\]
Experiment using Design/CPN (1)

- Use Petri Nets as a denotational semantics
  - One safe nets have limited capacity to model CDFGs
    - Single type of token
    - Modeling queues is difficult
  - Use colored Petri Nets
    - Support types and queues (lists)
  - Use Design/CPN to model and simulate CDFGs

Experiment using Design/CPN (2)

- When a CDFG executes, the order of events must be maintained
  - Cannot use a bag as this is not ordered
- A FIFO queue is a useful representation of a CDFG edge
- A CDFG node behaves atomically
  - This is problematic once we start looking at hierarchy and efficiency but will leave this as is for the moment
- A CDFG node takes time to execute
  - While not looking at time in the formalism it is still useful to consider
Experiment using Design/CPN (3)

- Simple Petri Net model of a CDFG
- Each edge maps to a single place
- Each node maps to a net comprising 2 transitions and two places
  - Enabling transition
  - Place invariant
  - Data place
  - Completion of firing transition
- Place invariant ensures only one set of inputs are processed per CDFG node execution cycle
- Does not prevent multiple tokens building up in CDFG edges
- Allows us to think about duration (between start and end of firing)

Experiment using Design/CPN (4)

- Alternative CPN model of a CDFG
  - Sequence
  - Uses an edge invariant to limit storage on the edge to one element
  - Limitations and difficulties using this model
**Experiment using Design/CPN (5)**

- Alternative CPN model of a CDFG
  - Sequence
  - Node modeled as a single transition
  - Edge modeled as a FIFO queue

- Input and output to the environment
  - Edges have elements added and removed in order
Experiment using Design/CPN (6)

- **Basic node**
  - All nodes based on this model
  - Interacts with connecting edges for input and output
  - Only wait if input not available
  - Note the guard on StartFiring
  - Initialisation of Invariant limits number of inputs
  - Set to one in all cases
  - Need to implement queues

- **Null Node**

- **Simple Branch**
Experiment using Design/CPN (8)

- **Simple Join**
  - Note altered guard
  - In this example, the function `hd(plist01)` is used. Other functions can replace it to achieve different results.

Experiment using Design/CPN (9)

- **Simple Merge**
  - Non-deterministic
Experiment using Design/CPN (10)

- Top level model of a trivial example

Experiment using Design/CPN (11)

- Top level model of a trivial example
Experiment using Design/CPN (12)

- Top level model of a trivial example
Experiment using Design/CPN (14)

- Top level model of a trivial example

Experiment using Design/CPN (15)

- Model of the GCD example
  - Edges behave as FIFO queues
  - Input and Output ports behave as interface with the environment so GCD has a consistent interface with the edges
Experiment using Design/CPN (16)

- GCD node

Experiment using Design/CPN (17)

- Original GCD model
  - Looks less cluttered but is a less suitable/accurate model for CDFG

```.scheme
(* OUTSIDE WORLD *)
val Edgelimit = 1;
color AI = int;
color Token = with present;
var a, b, c, plain, is01 : AI;
var token : Token;
```
Experiment using Design/CPN (18)

- Original GCD node

Nexy phase....

- Some interesting observations have been made during this experiment
- If a queue is used the definition of a readable or writeable edge has to be modified
  - An edge is readable if the queue is not empty
  - An edge is writeable if the queue is not full
  - This also has an effect on whether a node can fire
    - Removes the potential for a graph to stall because a node cannot output a result
Next phase....

- The enabling function can be very complex when lists are used in CPN to model the queue
  - CPN permits an empty queue to participate in a transition
  - CDFG nodes only become enabled and fire if there is input ie an empty input cannot participate in an enabling or firing
  - Guards and inscriptions on arcs are all part of the enabling and firing conditions

Next phase....

- CPN does not make it easy if we wish to build nodes with n-ary inputs or outputs out of simpler components while retaining semantics (possible opportunity?)
Next phase....

- **Control and data information on edges is fundamentally the same**
  - data
    - The difference is how it is used by the nodes
    - The difference between control-flow nodes and purely data-flow nodes is the element of choice
      - Purely data-flow nodes are deterministic
      - No cycles, no branching based on choice
    - Purely data-flow graphs behave deterministically, with all nodes firing and all arcs utilised in any execution cycle.

Next phase....

- **If only one set of inputs is processed by a node at any given time is it possible to simplify the general graph?**
  - Change lists to single elements of the list
  - Remove feedback arcs
  - Certainly if the graph is a data-flow graph ie deterministic
  - Not sure if the graph contains some element of control
    - Might be possible if the control is restricted to a sub-graph
Next phase....

• Current CDFG model does not allow
  — Nodes that do not produce output
  — A node to have an arc to itself