Graphs as Visualization Aids

- Graphs are mathematical structures with obvious visualizations that seem often to help many stakeholder communities to visualize key relations.
- A graph’s edges visually represent the ordered pairs that compose the relation.
- If the pairs in $E$ are ordered, then $G$ is a directed graph, and its edges are depicted with arrowheads.
  - If not, the graph is called an undirected graph.
What is This Graph Specifying?

Annotations Provide Intuitions

But are they suggesting too much?
What’s wrong with this diagram

There is ambiguity and misuse of notation here:
- one circle is a test, others are functions
- some edge annotations are data, some predicates
- are multiple arrows in and out “and” or “or”?

Differences in Graphs Result from Different Choices for Nodes & Relations

- Data Flow:
  --Nodes represent set of sites where data is generated/used
  --Each edge is a (data generated, data used) node pair
- Control Flow:
  --Nodes represent units of functionality
  --(n₁, n₂) is an edge in this graph if and only if unit n₂ can execute immediately after n₁ executes (ImmFol relation)
- Hierarchy:
  --Models “consists of” or “is a part of”
  --Key to divide-and-conquer approaches to understanding
- Finite State Machines
  --Nodes represent all possible different “execution states”
  --(s₁, s₂) is an edge if and only if it is possible for state s₁ to immediately succeed s₂. Called a transition from s₁ to s₂
  --Edges annotated with transition condition
- Petri Nets
  --Multiple node and edge types in the same diagram
DATA FLOW DIAGRAMS

• Capture system functionality: What does system do? How?
• Basic components of a data flow diagram:
  --Nodes, represented by circles (boxes), are functional units
  --Edges, represented by arrows, are data flows between units
  --Both augmented by separate annotation relations
  --Boxes (sometimes circles), represent I/O data

EG.

formalizing DFDs

• DataFlow(i, j) if node i creates data that node j uses
• INPUT--set of nodes {input} such that input is a provider of input from external source
• OUTPUT--set of nodes {output} such that output is a conveyor of computed artifacts to external source
• EdgeAnnotation(e, operand) where operand is the identification of the artifact that flows along edge e
  – Preferably the data artifact is defined rigorously
• NodeAnnotation(n, text) if the string text describes the functioning of node n

Questions this helps answer:
• Why create this data? Who uses this data? What results does the end user see? What does the end user have to input?
Questions this can’t answer: What is the exact sequence of events? How does a node do its job?
Flowgraphs

Let $S = \{\text{all statements } s_i \text{ in a program, } P\}$

Let $\text{ImmFol} = \{ (s_i, s_j) | \text{the execution of } s_j \text{ immediately follows} \}
\text{the execution of } s_i \text{ for some execution of } P \}$

Then: If $FG = (S, \text{ImmFol})$, $FG$ is called the flowgraph of $P$

$FG$ is an ordered graph

Every execution sequence (i.e. the sequence in which the
statements of $P$ are executed for a given execution of $P$)
corresponds to a path in $FG$.

However---the converse is not true. A path through $FG$ may
not correspond to an execution sequence for $P$

A loop in $P$ appears as a cycle in $FG$

Callgraphs

Let $\text{PROC} = \{\text{procedures } S_i \text{ that the program } P \text{ comprises}\}$

Let $\text{CALLS} = \{ (S_i, S_j) | S_j \text{ is directly invoked from } S_i \text{ during}
some execution of } P \}$

Then $CG = (\text{PROC}, \text{CALLS})$ is called the Call Graph of $P$

$CG$ is a directed graph

If $P$ is written in a language that does not allow recursion,
then $CG$ will be acyclic

A cycle in $CG$ indicates that the nodes along the cycle
participate in a recursive calling chain

NOTE: DEPICTIONS OF THESE GRAPHS MAY BE
SUPERIMPOSED OVER EACH OTHER TO CLARIFY (?!)
THINGS
Hierarchy

- Enables incrementally adding detail
- Increased precision too
- Draws upon innate human mental capability
  - Abstraction
  - Encapsulation
- A typical solution to the problem of needing detail, but needing to avoid overload
- But creates potential problems

Consistency is a principal concern

- Are the diagrams consistent with each other?
- Top view consistent with elaborations?
  - Arrows consistent
  - Data flows consistent
  - Other semantics?
- Invitation to subtle errors
Consistency is a principal concern

• Are the diagrams consistent with each other?
• Top view consistent with elaborations?
  – Arrows consistent
  – Data flows consistent
  – Other semantics?
• Invitation to subtle errors

Simple Hierarchical Elaboration

Check args.
height
width
args OK
args not OK

Check args.
height
width
valid height
valid width
height > width
valid pair
args OK
args not OK
width > 0
no

Hierarchical Elaboration (of “Select New Floor”)

Select New Floor

Floor to Move to

New Status

Do Updates

Find Nearest Floor

Consider Other Cars

Delete Floor Selected

Floor

New Floor

Floor

Delete Request Just Satisfied

Turn off Light

Move Elevator

Update Request List

New Request

Current Floor

Current Floor

Get copy of Request List

New Floor

Floor in the right direction

Floors in the right direction

Process Request List

Request List

Process Request List

New Request

Turn on Light

Press Button

New Floor

New Request
Hierarchical Elaboration
(of “Select New Floor”)

Current Floor

Find Nearest Floor

Floor to Move to

New Status

Do Updates

Consider Other Cars

Delete Floor Selected

Floor

Rejection

Request List

Floors in the right direction

Direction

Floor

Inconsistencies with parent

Is this consistent with its parent?

Current Floor

Find Nearest Floor

New Status

Do Updates

Consider Other Cars

Delete Floor Selected

Floor

Rejection

Request List

Floors in the right direction

Direction

Floor

New Status

Inconsistencies with parent
Hierarchical Elaboration
(of “Select New Floor”)

Consistency definitions

- Let NodeAnnotation(n, somenode) for some n ∈ N, where ParentGraph = (N, E)
- Let somenode = (N’, E’)
  - This is the DFG elaborating on “somenode”
- Some consistency properties
  - If {(m_i, n)} ≠ ∅, m_i, n ∈ N, then | INPUT(somenode) | ≠ 0
  - If {(n, m_i)} ≠ ∅, m_i, n ∈ N, then | OUTPUT(somenode) | ≠ 0
- Maybe some others(?)
  - If | {(m_i, n)} | = k, m_i, n ∈ N, then | INPUT(somenode) | = k
  - If | {(n, m_i)} | = k, m_i, n ∈ N, then | OUTPUT(somenode) | = k
Data Flow Diagram

Elaboration of “Select New Floor” Suggests Artifact Issues
Data needs precise specification too

• DFD's focus on functionality, using data as a vehicle
• Data shown as unstructured atomic units--usually unrealistic
• Complex functions cannot be adequately defined without delving into the details of how they handle structured data
• Sub-DFD's can show how the high level data that high level DFD's deal with is decomposed
  --But this is implicit data definition
  --Can be hard to read/inconsistent
• Data specification is worth doing explicitly, carefully
• Usually using Disciplined Natural Language--eg. Templates
• Formal specification of data is an important future topic

IDEF0

• Commercial DFD formalism
• Some formality and rigor behind it
• Primarily pictorial
• In wide use
• Additional semantics in IDEF1, IDEF2, etc.
Note: The edges here do not comprise a set. They comprise a collection.
Broadening DFG Semantics

- Node cannot begin until data arrives along all in-edges
  - DFGNODE is set of nodes, \{dfgnode\} such that dfgnode is a function defined on the set of all artifacts \( IA_i = \{ia_i\} \), such that ia is an annotation on an edge (dfgnode, dfgnode)

- How to adapt this for
  - any semantics?
  - for output semantics?
  - exactly one Output node
  - Etc.

These sorts of constraints can support additional types of reasoning:
Eg. about parallelism
More Broadening

- Use of "open boxes" to indicate data store
  -- A different set, with different semantics
  -- Not a computation function
  -- Methods are: put, get, search(?)

Still More Broadening

- Different shapes of boxes
- Different pictures instead of boxes
- Different colors
- Different lines
- ......

Central questions: What are the semantics? Does this really help? Or confuse?
Kepler--Another DFG Technology

- Data Flow diagram notation
- Has hierarchical decomposition
- Capability for specifying DFD semantics
  - For each diagram
  - Can be different at each level of hierarchy (!)
- Based on Ptolemy II system

A Kepler Example
Kepler Elaboration

What kinds of questions are well addressed by DFGs?

- Overall structure of functional capabilities
  - What does this piece do?
- System outputs and inputs
- How might changes be made?
- What functions create what data entities
Given that Precision is essential

- What about the other three dimensions?
- Detail
  - Gained from hierarchical elaboration
- Breadth
  - Comes from different (sub)types of DFG
- Clarity
  - Seems to be reduced by increased Detail and Breadth
  - With the need for Precision

Clarity helps appeal to diverse stakeholders

- Users
- Developers
- Managers
- Evolvers
- Bystanders (?)
Final observations

• Very primitive representation
  --very limited semantics

• But actually more a family of model types
  --different sets of semantics

• The actual relation(s) are rarely made clear and precise

• Powerful aid to intuition and efficiency of communication
  --Clear advantages over natural language

• But is intuition misled by ambiguity, misinterpretation?

• Does not help explain HOW things get done

Control Flow Diagrams

• Semantics are different from DFD’s semantics
  --Arrows represent flow of control, rather than flow of data
  --Different shapes/types of boxes with different semantics

• Basic control flow graph:
  --Boxes represent functions
  --Some other shape represents control flow alternation
  --Arrows represent control flow:
    If there is an arrow from circle A to circle B, it means
    ImmFol: "the execution of B can immediately follow
    the execution of A for some execution"
  --Different semantics for arrows between different shapes
  --Still other shapes represent Start and Stop
  --Use of hierarchy for elaborating boxes

• Usual enhancements:
  --Annotate edges with predicates
  --Special symbols for branching, concurrency control....
Control Flow Diagrams

• Semantics are different from DFD's semantics
  --Arrows represent flow of control, rather than flow of data
  --Different shapes/types of boxes with different semantics

• Basic control flow graph:
  --Boxes represent functions
  --Some other shape represents control flow alternation
  --Arrows represent control flow:
    If there is an arrow from circle A to circle B, it means
    ImmFol: "the execution of B can immediately follow the execution of A for some execution"
  --Different semantics for arrows between different shapes
  --Still other shapes represent Start and Stop
  --Use of hierarchy for elaborating boxes

• Usual enhancements:
  --Annotate edges with predicates
  --Special symbols for branching, concurrency control....

Control flow graphs also address questions like “what does this do” and “how does this do it”

Example Control Flow Graph Semantics

• Square boxes: Functions
  – Any number of inedges
  – One outedge

• Edges: ImmFol relation
  – Function at head executes immediately after function at tail concludes

• Round ovals: Branches/decisions
  – Each oval represents a Boolean function
  – For each oval there are exactly two outedges, labeled True and False
Example Control Flow Graph Semantics

- FG = (FN, FE), where FN = OPS ∪ TEST and
  OPS = {ops_i} nodes where computation is done
  TEST = {test_i} nodes where tests are done
- Edges: ImmFol relation
  if (fe_i, fe_k) ∈ FE then there is an execution path for which
  the execution of fe_k immediately follows the execution of fe_i
- | {fe_i, fe_k} | where fe_i ∈ TEST = 2
- | {fe_i, fe_k} | where fe_i ∈ OPS = 1
- FG is a directed, connected graph
- Maybe(?) : FN = OPS ∪ TEST ∪ INPUT ∪ OUTPUT where
  - fn_i ∈ OUTPUT ⇒ ∄ (fn_i, fn_k) ∈ FE
  - fn_i ∈ INPUT ⇒ ∄ (fn_k, fn_i) ∈ FE
  - Other semantics?

Control Flow Graph

- Read Floor, Direction
- DB_Read: curr-length, Floors(.), Directions(.)
- Direction = Up
  - incr <-- 1
  - incr <-- -1
- NFloor <-- Floor;
- Increment NFloor
- Complement Direction (Incr <-- -Incr)
- Directions[NFloor] = incr ?
  - yes
  - no
- other car is nearer ?
  - yes
  - no
- NFloor out of bounds
- output new floor
More Detail through Hierarchical Elaboration

- The challenges and solutions here are similar to those for DFGs
- Need for consistency between levels
- Semantics can be hard to define, stick to

More Breadth from more (sub)types of CFG

- Concurrency
- Timing
- Data flow augmentations
- Etc.
Concurrency graph with “fork and join”

<p>|</p>
<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>call a</td>
<td>accept a</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

Rendezvous Graph

- Rendezvous graph: $RG = (N, E)$, where $N = \text{COMP} \cup \text{SYNC}$
- $S \in \text{SYNC} \Rightarrow S$ executes only after the execution of all $C \in \text{COMP}$ such that $(C, S) \in E$
- If $(S, C) \in E$, where $S \in \text{SYNC}$, then $C$ cannot execute until $S$ has executed
Trace Flow Graph

- Explicitly represents interleaved execution
- The green edges are the ImmFol relation
  - Double-headed arrows represent one arrow in each direction
- Sync nodes are filled in black
- Synch edges are blue

Different variations to address different stakeholders and their needs

- More semantic issues
  - Different graph types for different issues
- More detail
  - Hierarchy helps here
- More clarity
  - Does more elaboration and more graph types help clarity or impede it?
Focus on Clarity

- CFG has more procedural detail
- Does this help?
- What about combining DFG and CFG?
  - Iconography must be clear (?)

Control Flow Graph

Read Floor, Direction

DB_Read: curr-length, Floors(.), Directions(.)

Direction = Up

NFloor <-- Floor;

increment NFloor

Complement Direction (Incr <-- -Incr)

NFloor out of bounds

other car is nearer?

yes

output new floor

no
Control Flow Graph with Data Flow

Read Floor, Direction

DB_Read: curr-length, Floors(.), Directions(.)

Direction = Up

NFloor <-- Floor;

NFloor = incr ?

other car is nearer ?

yes

no

output new floor

increment NFloor

complement Direction

(Incr <-- -Incr)

yes

no
What is Control Flow good/not good for?

• Sense of what algorithms to use
• Constraints on data appearing
  – eg. Assuming that the Request List is maintained in sorted order
• Can estimate running speed
• Can reason about functionality
  – Possible to strand requesters
  – Possible to take riders in the wrong direction
• Drawbacks:
  – What about safety?
  – What about data?

What Stakeholders does this address?

• Developers
• ???
Control Flow vs. Data Flow Graphs

- Both shed light on similar questions
- One focuses on data evolution, the other on functional development
- Both are useful, neither removes the need for the other
- Control flow graphs map closely to implementation code written in procedural languages.
  - Good basis for determining consistency of code with ideas expressed as data flow
- Data flow graphs focus more on the product itself, seem better at helping understand if and how it gets evolved
  - Seem better adapted to studying earlier formulations of the problem to be solved, and ways of solving it

Finite State Machines (FSM's)

- FSM's describe behavior of a system:
  -- The sequence of stages/steps/conditions that the system goes through
  -- FSM shows how a system acts/reacts to inputs
  -- Does this by showing progress through different states
- Hypothesis:
  -- The universe in which the system being described must operate can be accurately modeled as always being in exactly one of a finite number of states (situations)
  -- There are only a finite number of possible system inputs
Finite State Machines (FSM's)

- FSM = (Q, I, \( \partial \)), where
  
  Q = \{q_i\} , the set of all possible system states

  I = \{i_j\} , the set of all events that can affect the state of the system

  \( \partial \) is a function, \( \partial: Q \times I \rightarrow Q \) such that if the system is in state \( q \) and event \( i \) occurs, then the system transitions to state \( \partial(q, i) \)

- Use of hierarchy can help add detail
  - But presents familiar consistency problems

- Other definitions:
  - Start state
  - Accepting state
  - Trap state

Why FSM's?

- Primary appeal is visualizability--clarity
  --Circles represent states
  --(Curved) arrows represent transitions
  --Arrows are annotated with inputs

- Intuitively: Can "watch" a stream of inputs "drive" the behavior of the system as a sequence of movements from state to state

Kinds of Questions FSMs seem adept at helping answer:
  "What is a good way to think about the problem to be solved?"
  "What is the solution approach?"
  "How does this program work?"
Finite State Machine

Doors Closed, Elevator Idle

Request to Move to new floor

Doors closed, Elevator in Motion

Request to Move to new floor

Request to Move to new floor

Request to Move to new floor

Queueing new floor request

Arrival at New Floor

Getting new floor request

Arrival at New Floor

Queue empty

Doors open, Elevator Stopped

No new request timeout

Doors open, Elevator in Motion

Request to Move to new floor

Finite State Machine for Digital Watch

Time display mode

Datebook mode

Alarm display mode

Phone book mode

press button A

press button A

press button A

press button A
More Finite State Machine Details

More Complex FSMs

- State also specifies activities
  - Leans towards a CFG
- Transition may involve computation
  - Considered to be “instantaneous”
- Transition may be conditional
  - Event as well as a condition
- Transition may emit events
  - To drive other FSMs
- FSMs may be hierarchical
  - What exactly are semantics?
Statecharts: More Complex FSMs

- Initialize:
  - do: Initialize course object

- Unassigned:
  - do: Assign professor to course
  - addStudent/numStudents = 0

- Open:
  - entry: Register a student
  - cancelCourse
  - registration closed
    - numStudents < 3
    - numStudents ≥ 3

- Closed:
  - numStudents = 10
  - do: Report course is full
  - registration closed
    - numStudents ≤ 10

- Canceled:
  - do: Send cancellation notices
  - cancelCourse

- RegistrationComplete:
  - do: Generate class roster

Activity to be performed in this state:
- Initialize
- Unassigned
- Open
- Closed
- Canceled
- RegistrationComplete
Statecharts: More Complex FSMs

Initialize
- do: Initialize course object

Unassigned
- do: Assign professor to course
- numStudents = 0

Open
- entry: Register a student
- cancelCourse

Canceled
- do: Send cancellation notices
- registration closed
  - numStudents < 3
  - numStudents > 3

Closed
- do: Report course is full
- numStudents = 10

Registration Complete
- do: Generate class roster

addStudent

condition on this transition

activity on this transition

CS 520/620 Spring 2012 Univ. of Massachusetts
Copyright L. Osterweil, all rights reserved
Statechart with Nested States

What is FSM good/not good for?

- Focus on specific issue: safety concern
  - Model unsafe state
  - Model state transitions
  - Can unsafe state be reached?
- Drawbacks
  - No sense of functionality
    » Unless additional semantics
  - No sense of how functionality achieved
    » Except perhaps hierarchy
  - Hard to deal with concurrency
    » Without additional semantics
  - Impossible to reason about timing
Petri Nets

- More powerful and intuitive depiction of control flow
  strong on depiction of parallelism and concurrency
- A Petri Net structurally consists of
  - A finite number of places
  - A finite number of transitions
  - A finite set of arrows that connect places to transitions
    (or vice versa)
- If an arrow goes from a place to a transition, then place is
  said to be an input place of the transition.
- If an arrow goes from a transition to a place, then place is
  said to be an output place of the transition.

Marking and Firing Petri Nets

- A Petri Net place can be marked by the presence of a
  token
  - Any collection of places can be marked.
  - Any such marking is said to define a state of the
    Petri Net
- Petri Nets proceed from one state to another by means
  of a firing
  - Occurs only when every input place of a transition is
    marked with a token.
- The effect of the firing of a transition is to
  - Remove all of the tokens from the transition's input
    places
  - Put tokens in all of the transition's output places
Unmarked Petri Net

Petri Net Marked
With Up button pressed
Petri Net Marked
With Up button pressed

Car going up

Up button Pressed

Stop at Floor

Go to Next Floor

Up button Not pressed
Petri Net--Two Elevator Cars
Car 1 Wins Race

Petri Net--Two Elevator Cars
Car 1 Wins Race
Petri Net--Two Elevator Cars
Car 2 Wins Race

Car 1 approaching Floor n, going up

Car 2 approaching Floor n, going up

Up Button Pressed at Floor n

No Press

Stop at Floor

Go to Next Floor Up

Car 1 approaching Floor n, going up

Car 2 approaching Floor n, going up

Up Button Pressed at Floor n

No Press

Stop at Floor

Go to Next Floor Up
Petri Net--Two Elevator Cars

Car 2 Wins Race

Car 1 approaching Floor n, going up

Car 2 approaching Floor n, going up

Up Button Pressed at Floor n

No Press

Stop at Floor

Go to Next Floor Up

Car 1 approaching Floor n, going up

Car 2 approaching Floor n, going up

Up Button Pressed at Floor n

No Press

Stop at Floor

Go to Next Floor Up
Petri Net--Two Elevator Cars
Car 2 Wins Race

Petri Net for a Different Elevator
Marking for moving up to pick up a passenger

- Car going up
  - Up button higher
    - No Up Button Higher
  - Move up to Nearest Up Button
    - No Down Button Lower
    - Visit floor, Turn off button
- Car going down
  - Down button lower
    - No Down Button Lower
    - Move to nearest Down Button
    - Visit floor, Turn off button
Marking for moving up to pick up a passenger

Car going up

Up button higher

Move up to Nearest Up Button

Car going down

Down button lower

Visit floor, Turn off button

Move to nearest Down Button

Visit floor, Turn off button

No Up Button Higher

No Down Button Lower

Marking for moving up to pick up a passenger

Car going up

Up button higher

Move up to Nearest Up Button

Car going down

Down button lower

Visit floor, Turn off button

Move to nearest Down Button

Visit floor, Turn off button

No Up Button Higher

No Down Button Lower
Marking for moving up to pick up a passenger

Queues at places?

Car going up
- Up button higher
  - Move up to Nearest Up Button
  - No Up Button Higher
  - No Down Button Lower
- Down button lower
  - Visit floor, Turn off button
  - Move to nearest Down Button
  - Visit floor, Turn off button

Car going down
- No Up Button Higher
- Down button lower
- Move to nearest Down Button
- Visit floor, Turn off button

Marking when no passengers higher, but passenger lower

Car going up
- Up button higher
  - Move up to Nearest Up Button
  - No Up Button Higher
- No Down Button Lower
- Down button lower
  - Visit floor, Turn off button
  - Move to nearest Down Button
  - Visit floor, Turn off button
Marking when no passengers higher, but passenger lower

Car going up
- Up button higher
  - No Up Button Higher
  - Move up to Nearest Up Button
  - Visit floor, Turn off button

Car going down
- Down button lower
  - No Down Button Lower
  - Move to nearest Down Button
  - Visit floor, Turn off button

Up button higher
- No Up Button Higher
- Move up to Nearest Up Button
- Visit floor, Turn off button

Down button lower
- No Down Button Lower
- Move to nearest Down Button
- Visit floor, Turn off button
Marking for both up and down buttons

- Car going up
  - Up button higher
  - Move up to Nearest Up Button
  - Visit floor

- Car going down
  - Down button lower
  - Move to nearest Down Button
  - Visit floor

Marking for neither up nor down buttons

- Car going up
  - Up button higher
  - Move up to Nearest Up Button
  - Visit floor

- Car going down
  - Down button lower
  - Move to nearest Down Button
  - Visit floor
Petri Nets

• More powerful and intuitive depiction of control flow strong on depiction of parallelism and concurrency
• A Petri Net structurally consists of
  – A finite number of places
  – A finite number of transitions
  – A finite set of arrows that connect places to transitions (or vice versa)
• If an arrow goes from a place to a transition, then place is said to be an input place of the transition.
• If an arrow goes from a transition to a place, then place is said to be an output place of the transition.

Marking and Firing Petri Nets

• A Petri Net place can be marked by the presence of a token
  – Any collection of places can be marked.
  – Any such marking is said to define a state of the Petri Net
• Petri Nets proceed from one state to another by means of a firing
  – Occurs only when every input place of a transition is marked with a token.
• The effect of the firing of a transition is to
  – Remove all of the tokens from the transition's input places
  – Put tokens in all of the transition's output places
Some Semantics

- PN = (Places, Transitions, Edges)
  - Places (PN) = \{place_i\}
  - Transitions (PN) = \{transition_j\}
  - Edges (PN) = (Inedges U Outedges)
    - Inedges (PN) = \{(place_i, transition_j)\} ∩ Outedges (PN) = \{(transition_j, place_i)\}.
    - where place_i ∈ Places (PN), transition_j ∈ Transitions (PN)
- Marked: PN X Places -> {True, False}
  - If Marked (PN, place_i) = True we say that place_i is marked
- A transition, t_i ∈ Transitions (PN) can fire if for all of its inedges, (place_j, t_i) Marked (place_j) = True
  - After a transition t_i ∈ Transitions (PN) fires
    - Marked (p_j) <- False for all places, p_j such that (p_j, t_i) ∈ Edges (PN)
    - Marked (place_k) <- True, for all place_k, such that (t_i, place_k) ∈ Edges (PN)

Evaluation of Petri Nets

What are Petri Net representations good for?:
- How do things get done?
- Especially parallelism and nondeterminism
- Helps spot races and deadlocks

What kinds of things are Petri Nets not good for?
- Too little focus on product (tokens represent it)
- Many extensions:
  - Hierarchical Petri Nets
  - Colored tokens
  - "Or" transitions
  - Queues at places
- Too many extensions confuse the picture?