A Game Logic for Workflows of Non-cooperative Services

— *SUNYSB Summer'02* —

Hasan Davulcu

Dept of CS
University at Stony Brook
N.Y. 11794, U.S.A.

Workflows

Workflow: A collection of inter-related tasks and transactions designed to carry out a well-defined business process.

Workflow Management: Automated *coordination* of work, among processing entities, to achieve *an overall business goal.*

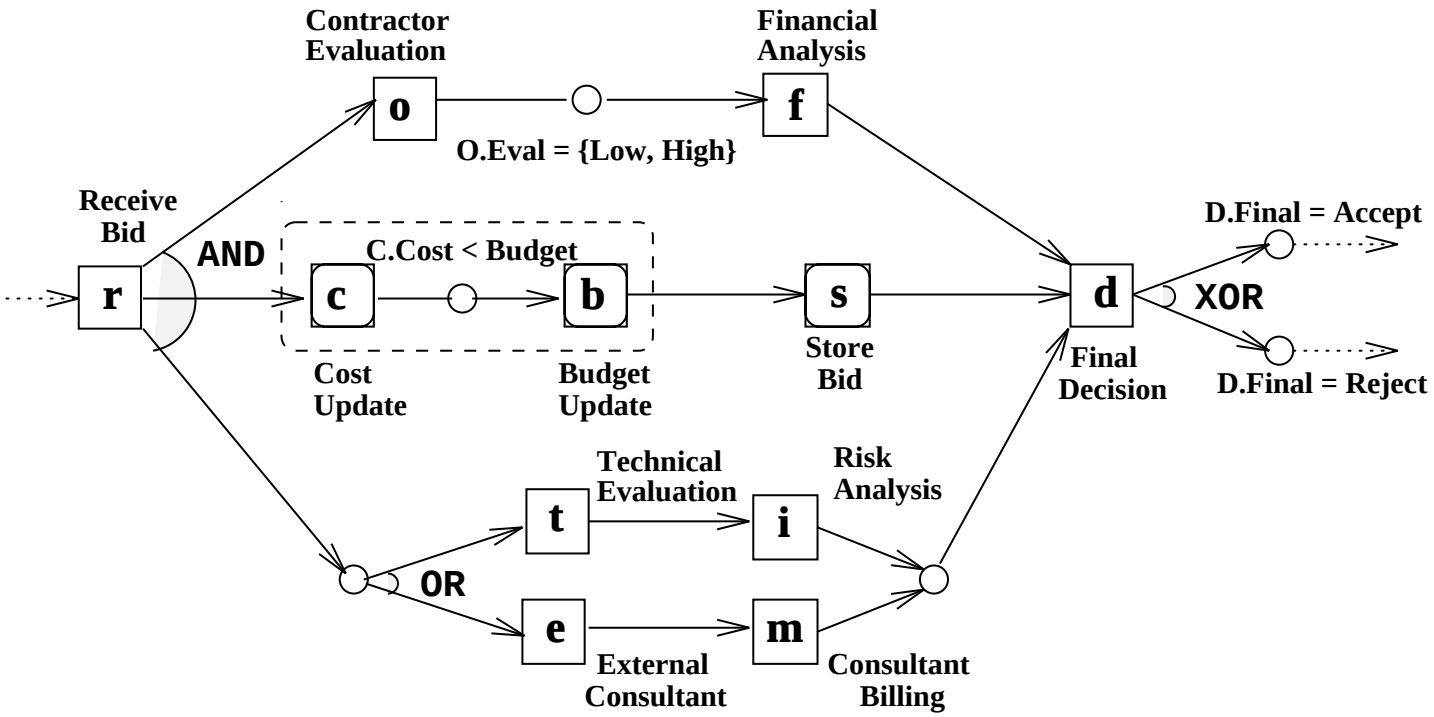
Workflow Management System (WfMS): System for automation of workflow processes (like **DBMS** facilitates creation and maintenance of large data sets).

What Kind of Computations are Workflows ?

- **Long-running:** Hours, days, months;
- **Autonomous, distributed processing entities**;
- **Complex Inter-Task Dependencies**

Bid Evaluation Example

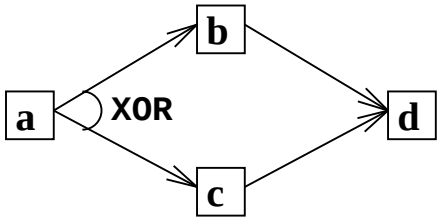
❑ **Control Flow Graph:**



❑ **Global Coordination Dependencies:**

1. IF o.eval = low THEN not e
2. IF occurs (e) THEN o before e
3. IF occurs (t) AND occurs (e) THEN e before i
4. c before f

Central Workflow Problems



1. d before c
2. b AND IF occurs (b) THEN c

Consistency: Is control flow graph and coordination dependencies *consistent* ?

Correctness: Does the specification satisfy certain key properties ? *e.g.*, Every bid is either accepted or rejected by the bid-evaluation workflow.

Coordination: How to schedule tasks automatically ?

What is Needed to Enable *Workflow Automation* ?

Creation: A *formal language* to specify the structure of processes and their interactions at a high-level;

Verification: *Reasoning methods* to ascertain that a workflow is correct;

Execution: Techniques for *automatically deriving* correct executions from high-level specifications.

- Ideally, all should fit in a single, unifying framework.

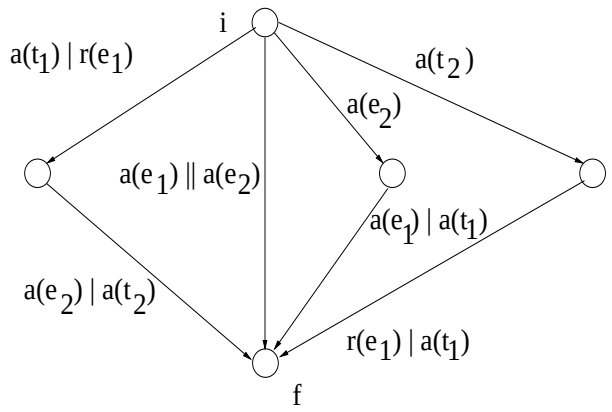
Related Work on Workflows

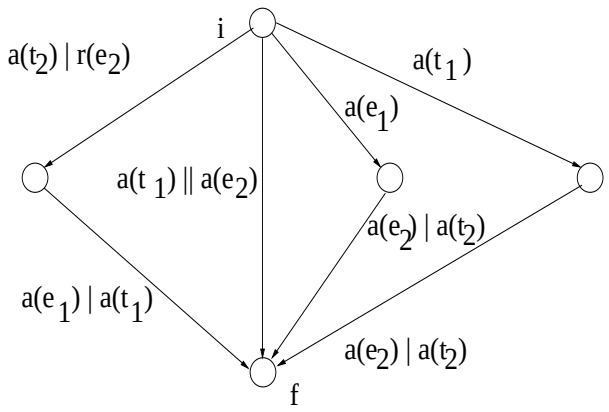
- **Temporal Logic* [Sheth et.al.93]**
- **Event Algebra* [Singh-96]**
- **Petri Nets* [Nabil et.al.-98]**
- **ECA Rules and Vortex [Hull-99]**
- **Dynamic Restructuring of Workflows [Liu-98]**
- **Nested Triggers [Dayal-90]**
- **Extended Transaction Models and ACTA [Ramarithram-92]**

Temporal Logic Based Modeling: [Sheth et.al.-93]

- Workflows are a set of inter-task dependencies
- Dependencies are specified in CTL
- CTL formulas are converted into automata
- Scheduler ensures that a sequence of events are accepted by *all* automaton
- Worst-case complexity of scheduling is exponential: $O(N^m)$

An Example: [Sheth et.al.93]





(a) $e_1 \rightarrow e_2 \equiv \neg E(\neg e_2\ \mathcal{U}(e_1 \wedge E\mathcal{G}\neg e_2))$ (b) $e_1 < e_2 \equiv \mathcal{AG}(e_2 \rightarrow \mathcal{AG}\neg e_1)$

- Every received event is checked against every dependency
- IF a legal execution path satisfying *all* automaton exists
- THEN accept event, ELSE delay or reject events

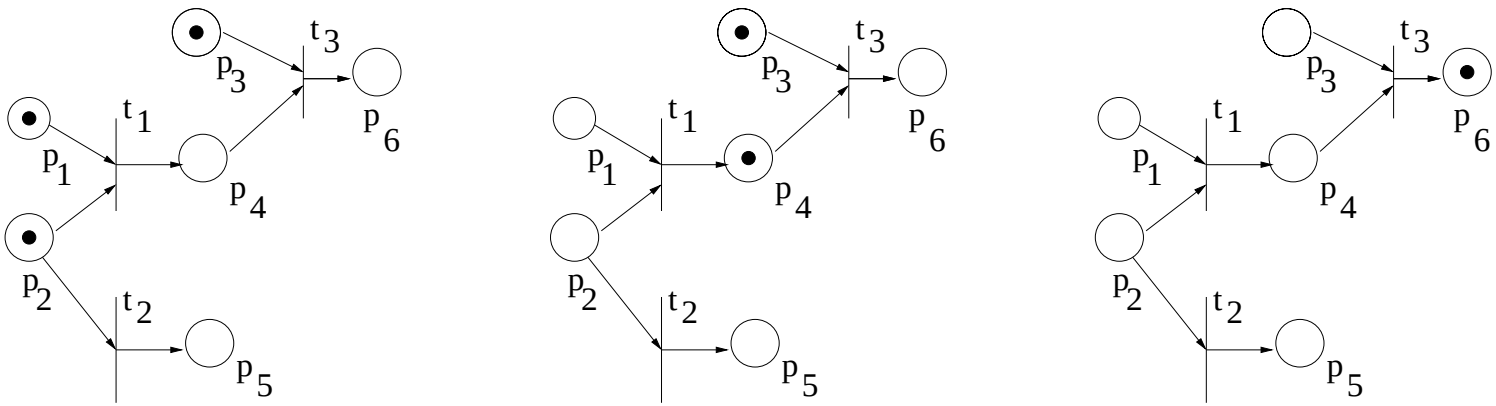
Event Algebra Based Approach [Singh-96]

- Workflows are a set of inter-task dependencies
- Dependencies are specified in an expressive algebra with denotational semantics based on traces $(e \cdot \bar{f})$
- **Residuation Operation of** D/e: If the state of the scheduler is D and event e arrives, the new state is D/e, which reflects *what remains to be statisfied by the incoming events.*
- It is possible to characterize residuation by a set of equations:
  - $0/e = 0$
  - $\top/e = \top$
  - $(E_1|E_2)/e = (E_1/e)|(E_2/e)$
  - $(E_1 + E_2)/e = (E_1/e) + (E_2/e)$
  - $(e \cdot E)/e = E$, if $e, \bar{e}$ do not appear in the event expression E.
  - $(e' \cdot E)/e = 0$, if either e or $\bar{e}$ occurs in the event expression E
  - $E/e = E$, if e or $\bar{e}$ does not appear in the event expression E.

An Example: [Singh-96]

- $D_1 : \bar{start}_{buy} + start_{book}$: If *buy* starts then *book* must also start.
- $D_2 : \bar{commit}_{book} + \bar{commit}_{buy} + commit_{book} \cdot commit_{buy}$: If both *book* and *buy* commit then *book* commits before *buy*.
- This workflow is satisfied by several traces out of which two are $start_{buy} start_{book} commit_{book} commit_{buy}$ and $start_{book} start_{buy} commit_{book} commit_{buy}$.
- When an event, e, arrives, we compute $E' = E/e$. If $E' \neq 0$, the event is scheduled and E' becomes the new constraint that needs to be satisfied.
- Checking that $E' \neq 0$ requires a *satisfiability* check which requires converting E into DNF, an operation that is worst case exponential in $|E|$.

Petri Nets Based Approach

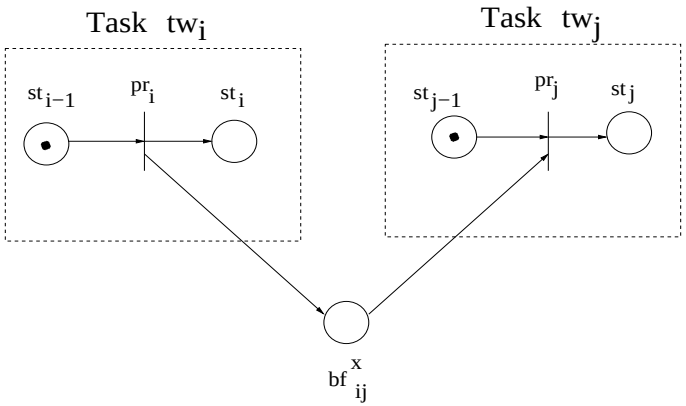


(a) Initial marking (b) Marking after t_1 fires (c) Final marking

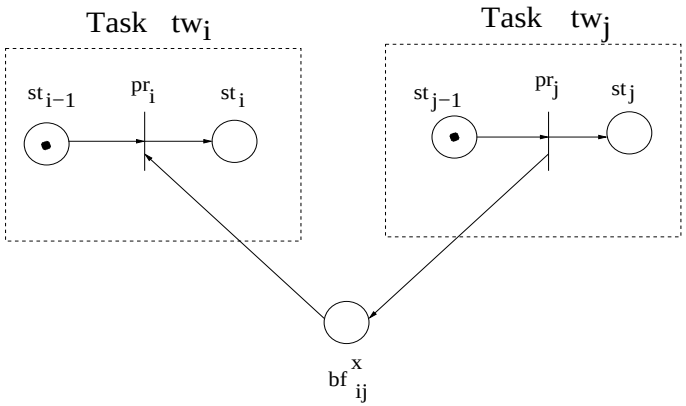
Figure 1: An example Petri Net

A state M' is reachable from a state M, denoted by $M \xrightarrow{*} M'$, if and only if there exists a sequence of firing transitions $\sigma = t_1 t_2 ... t_n$ such that $M \xrightarrow{\sigma} M'$ i.e. $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 ... \xrightarrow{t_n} M'$.

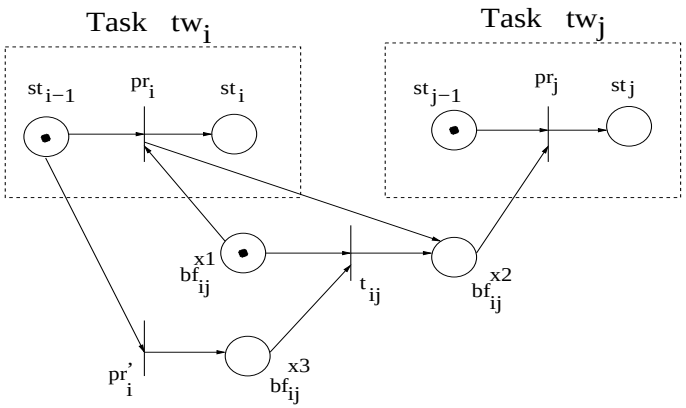
Petri Nets Based Dependencies [Adam-Atluri-98]



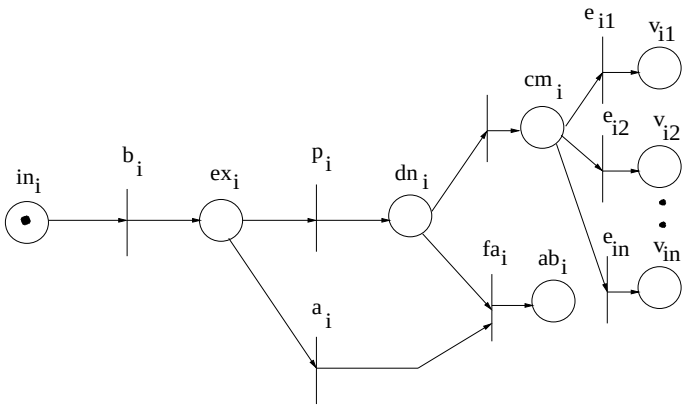
(a) Strong-causal dependency



(b) Weak-causal dependency

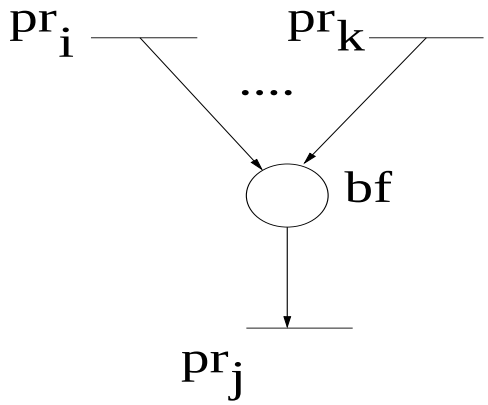


(c) Precedence dependency

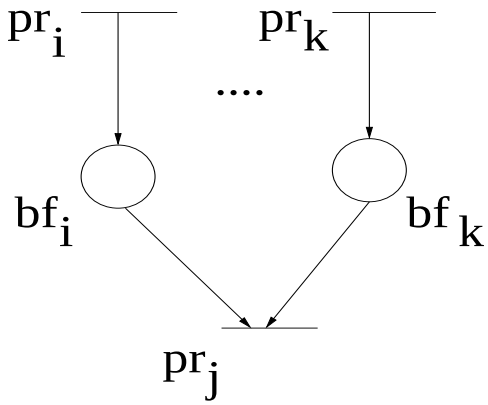


(d) Task with value states

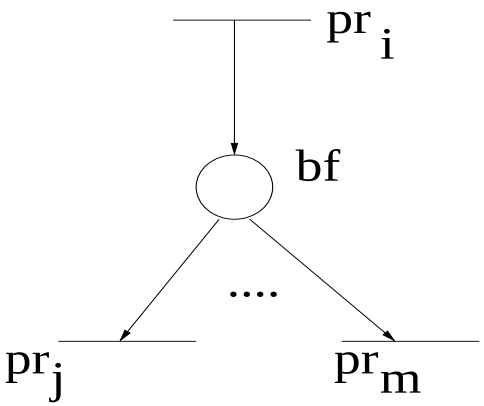
Petri Nets Based Workflow Composition [Adam-Atluri-98]



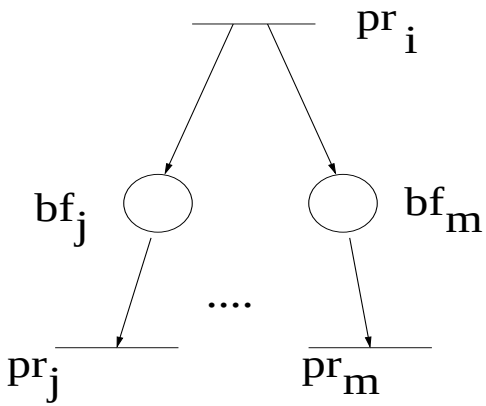
(a) OR-Join



(b) AND-Join



(c) OR-Split



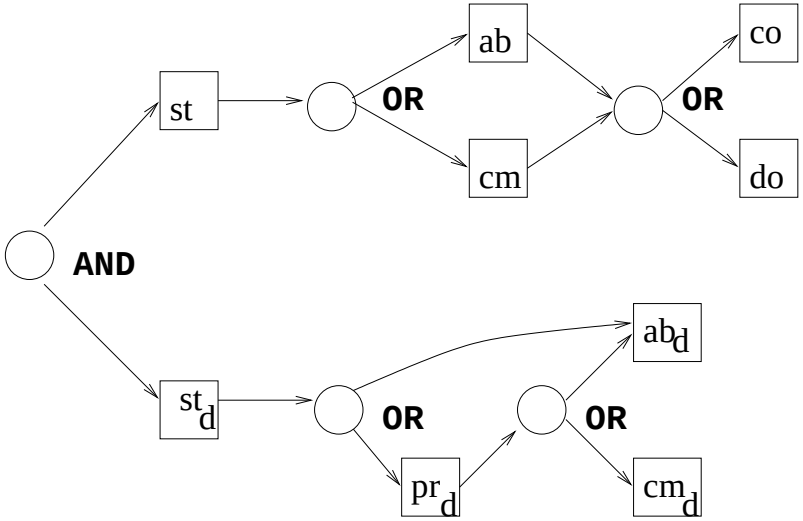
(d) AND-Split

Petri Nets Based Workflow Analysis [Adam-Atluri-98]

- *Consistency*: A workflow specification is inconsistent if there are:
  - inconsistent *precedence* relationships between tasks (e.g. loops).
  - inconsistent *logical* relationships between tasks.
  - reduces to **siphon** detection on the Petri Net. Polynomial with respect to the number of tasks and dependencies.
- *Safety*: A workflow is said to be safe if it terminates in any acceptable state. This reduces to *reachability* problem of Petri Nets. This problem is known to be DSPACE(exp)-hard for Petri Nets.
- *Schedulable*: A workflow is schedulable if there are no inconsistent temporal constraints. The *earliest and latest* firing times of each transition are calculated and then propagated thru the Petri-Net. The complexity is polynomial in the number of transitions in the Petri Net.

A Simple Workflow of Non-cooperative Tasks

- **Control Flow Graph**

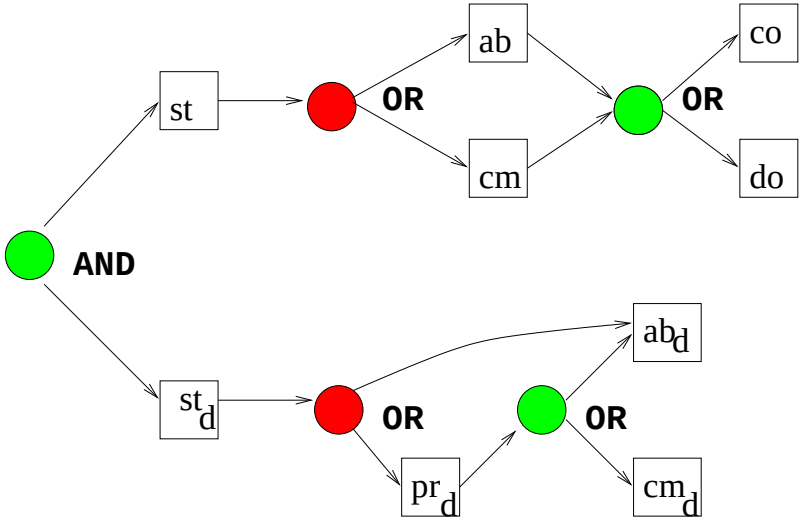


- **Dependencies**

  1. ab -> ab_d
  2. (ab_d /\ cm) -> co
  3. (cm_d /\ cm) -> do

A Simple Workflow of Non-cooperative Tasks: cont

- **Game Flow Graph**



- **Winning Conditions**

  1. ab -> ab_d
  2. (ab_d /\ cm) -> co
  3. (cm_d /\ cm) -> do

A few Remarks about Game Logics

- An expressive logic of games should help us reason about the following:
  - **Equivalence of games:** when are two games equivalent ?
  - **A player's capabilities:** which winning conditions are enforceable ?
  - **Strategy synthesis:** what is the complexity of computing the winning strategies ?
  - **Extracting and executing strategies:** how can we execute winning strategies ?
- Ideally, all should fit in a single, unifying framework.

Our Framework: Game - Concurrent Transaction Logic (Game-$\mathcal{CTR}$)

- Game-$\mathcal{CTR}$: Conservative extension of first order logic for *programming*, *executing* and *reasoning* with state-changing concurrent games.
- Game-$\mathcal{CTR}$ uniformly models:
  - Concurrent and sequential composition of games
  - `Opponent`'s and `Player`'s choices
  - Temporal Constraints to express winning conditions

– Model Theory:

• A model, M, assigns game formulas *truth values* over *sets of paths*. (i.e., sets of finite sequences of states)

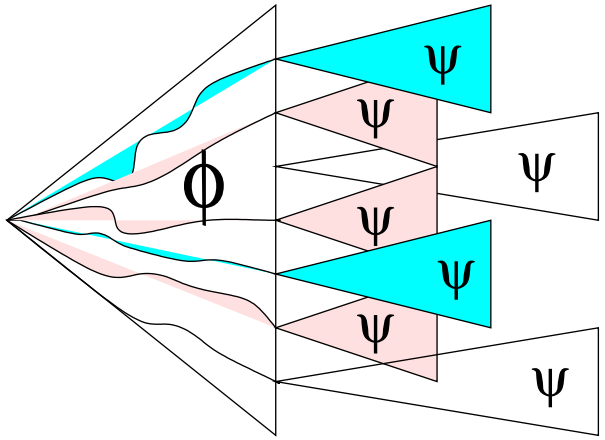
• Informally, $M, \mathcal{O} \models \alpha$, or α being *true* over a set $\mathcal{O} = \{\pi_1, ..., \pi_n\}$ means "the `player` has a strategy to enforce the game to the outcome denoted by $\mathcal{O}$".

– Proof Theory:

• Constructs *enforcible outcomes* as it proves statements.

Game-$\mathcal{CTR}$: Logical Connectives

$\phi \otimes \psi$ means, "Play a strategy for ϕ and then play a strategy for ψ".



- $(\alpha \otimes \beta) \mid (\gamma \otimes \delta)$ means, "Play a strategies for $(\alpha \otimes \beta)$ and $(\gamma \otimes \delta)$ in an *interleaved* fashion".

Game-$\mathcal{CTR}$: Logical Connectives

- $\alpha \wedge \mathbf{\Phi}$ means, "Play a strategy for α such that Φ is also satisfied". (i.e. $\wedge$ can be used to express winning conditions on games.)

- $\alpha \vee \beta$ means, "`Player` chooses a strategy from either α or β".

- $\alpha \sqcap \beta$ means, "`Opponent` chooses a strategy from either α or β".

- $\neg\ \alpha$ means, "Play anything but a strategy of α".

- $\alpha \leftarrow \beta$ means, "A strategy of β is also a strategy of α".
($\alpha \leftarrow \beta$ is defined as $\alpha \vee \neg\beta$).

- $\odot\, \alpha$ means, "Play a strtegy of α in isolation, that is without any interleavings from another strategy".

A Simple Workflow

Consider the formula

$$\phi \;=\; ins(st) \;\otimes\; (ins(ab) \;\sqcap\; ins(cm)) \;\otimes\; (ins(cp) \vee ins(no))$$

Then the possible outcomes for ϕ can be computed as follows:

1. Lets assume : $\{\langle d,\ d \cup \{\alpha\}\rangle\} \models ins(\alpha)$, for all $\alpha \in \mathcal{E}\mathcal{V}\mathcal{E}\mathcal{N}\mathcal{T}$
2. $\{\langle d,\ d \cup \{cp\}\rangle\} \models (ins(cp) \vee ins(no))$, and
   $\{\langle d,\ d \cup \{no\}\rangle\} \models (ins(cp) \vee ins(no))$
3. $\{\langle d,\ d \cup \{ab\}\rangle,\ \langle d,\ d \cup \{cm\}\rangle\} \models (ins(ab) \;\sqcap\; ins(cm))$
4. $\{\langle \emptyset, \{st\}, \{st, ab\}\rangle, \langle \emptyset, \{st\}, \{st, cm\}\rangle\} \models\ ins(st) \otimes (ins(ab) \sqcap ins(cm))$
5. $\{\langle \emptyset,\ \{st\},\ \{st, ab\}, \{st, ab, cp\}\rangle,\ \langle \emptyset, \{st\}, \{st, cm\}, \{st, cm, cp\}\rangle\} \models \phi$,
   $\{\langle \emptyset, \{st\}, \{st, ab\}, \{st, ab, cp\}\rangle,\ \langle \emptyset, \{st\}, \{st, cm\}, \{st, cm, no\}\rangle\} \models \phi$,
   $\{\langle \emptyset, \{st\}, \{st, ab\}, \{st, ab, no\}\rangle,\ \langle \emptyset, \{st\}, \{st, cm\}, \{st, cm, cp\}\rangle\} \models \phi$,
   $\{\langle \emptyset, \{st\}, \{st, ab\}, \{st, ab, no\}\rangle,\ \langle \emptyset, \{st\}, \{st, cm\}, \{st, cm, no\}\rangle\} \models \phi$

A Complex Procurement Workflow

The following is the Game-$\mathcal{CTR}$ specification of a procurement workflow among a *buyer*, *seller*, and two services, *financing* and *delivery*.

$$buyer \leftarrow pay\_escrow \otimes (financing \mid seller)$$
$$financing \leftarrow (approve \otimes (make\_payment \vee cancel)) \sqcap (reject \otimes cancel)$$
$$seller \leftarrow reserve\_item \otimes (delivery \vee recv\_escrow)$$
$$delivery \leftarrow insured \vee non\_insured$$
$$insured \leftarrow (delivered \otimes pay\_back) \sqcap (lost \otimes pay\_back)$$
$$non\_insured \leftarrow (delivered \otimes pay\_back) \sqcap lost$$

The terms of the contract between these actors can be expressed using the following constraints:

$*(approve \wedge \neg cancel \rightarrow pay\_back)$ – if *financing* is approved *buyer* should pay back

$*(cancel \rightarrow recv\_escrow)$ – if *buyer* cancels *seller* keeps the escrow

$*(make\_payment \rightarrow delivery)$ – if *buyer* pays, *seller* must deliver

Temporal Dependencies as Winning Conditions

1. **Elementary Primitive constraints**: If $e \in \mathcal{EVENT}$ is an event, then $*e$ and $*(\neg e)$ are primitive constraints.
2. **Disjunctive and Conjunctive Primitive constraints**: Any $\vee$ or $\wedge$ combination of propositions from $\mathcal{EVENT}$ is allowed under the scope of $*$.
3. **Serial Primitive constraints**: If $e_1, ..., e_n \in \mathcal{EVENT}$ then $*(e_1 \otimes ... \otimes e_n)$ is a primitive constraint.
4. **Complex constraints**: The set of all constraints, $\mathcal{CONSTR}$, consists of all Boolean combinations of primitive constraints using the connectives $\vee$ and $\wedge$.

Some Examples of Winning Conditions

$*e$ – event e should always eventually happen;

$*e \wedge *f$ – events e and f must always both occur (in some order);

$*(e \vee f)$ – always either event e or event f or both must occur;

$*e \vee *f$ – either always event e must occur or always event f must occur;

$*(\neg e \vee \neg f)$ – it is not possible for e and f to happen together;

$*(e \rightarrow f)$ – if event e occurs, then f must also occur (before or after e).

Game Solver Algorithm for Workflows

- Executing a workflow $\mathcal{G} \wedge \mathcal{C}$ by the general proof theory has *exponential* run-time complexity.
- We develop an algorithm which transforms $\mathcal{G} \wedge \mathcal{C}$ into an equivalent $\wedge$-free formula $\mathcal{G}_{\mathcal{C}}$ such that:
  - $\mathcal{G}_{\mathcal{C}}$ represents *all executable* strategies of the `player` in $\mathcal{G}$ for enforcing $\mathcal{C}$
  - *Scheduling* becomes more efficient on $\mathcal{G}_{\mathcal{C}}$.

Solver Algorithm: Contd.

Enforcing complex constraints. Let $*C_1, *C_2 \in \mathcal{CONSTR}$, $\mathcal{W}$ be a Game-$\mathcal{CTR}$ workflow, then

$$(*C_1 \vee *C_2) \ \wedge \ \mathcal{W} \quad \equiv (*C_1 \ \wedge \ \mathcal{W}) \ \vee \ (*C_2 \ \wedge \ \mathcal{W})$$
$$(*C_1 \wedge *C_2) \ \wedge \ \mathcal{W} \quad \equiv (*C_1 \ \wedge \ (*C_2 \ \wedge \ \mathcal{W}))$$

Enforcing elementary constraints. Let $\alpha, \beta \in \mathcal{EVENT}$ and $\mathcal{W}_1, \mathcal{W}_2$ be Game-$\mathcal{CTR}$ workflows. Then:

$$*\alpha \ \wedge \ \alpha \ = \ \alpha \qquad *\neg\alpha \ \wedge \ \alpha \ = \ \neg\texttt{Playset}$$
$$*\alpha \ \wedge \ \beta \ = \ \neg\texttt{Playset} \qquad *\neg\alpha \ \wedge \ \beta \ = \ \beta \qquad \text{if } \alpha \neq \beta$$

$$*\alpha \ \wedge \ (\mathcal{W}_1 \otimes \mathcal{W}_2) \ = \ (*\alpha \ \wedge \ \mathcal{W}_1) \otimes \mathcal{W}_2 \ \vee \ \mathcal{W}_1 \otimes (*\alpha \ \wedge \ \mathcal{W}_2)$$
$$*\neg\alpha \ \wedge \ (\mathcal{W}_1 \otimes \mathcal{W}_2) \ = \ (*\neg\alpha \ \wedge \ \mathcal{W}_1) \otimes (*\neg\alpha \ \wedge \ \mathcal{W}_2)$$
$$*\alpha \ \wedge \ (\mathcal{W}_1 \mid \mathcal{W}_2) \ = \ (*\alpha \ \wedge \ \mathcal{W}_1) \mid \mathcal{W}_2 \ \vee \ \mathcal{W}_1 \mid (*\alpha \ \wedge \ \mathcal{W}_2)$$
$$*\neg\alpha \ \wedge \ (\mathcal{W}_1 \mid \mathcal{W}_2) \ = \ (*\neg\alpha \ \wedge \ \mathcal{W}_1) \mid (*\neg\alpha \ \wedge \ \mathcal{W}_2)$$

Solver Algorithm: Contd.

Here we use σ to denote $*\alpha$ or $*\neg\alpha$:

$$\sigma \ \wedge \ (\mathcal{W}_1 \sqcap \mathcal{W}_2) \quad = \quad (\sigma \ \wedge \ \mathcal{W}_1) \ \sqcap \ (\sigma \ \wedge \ \mathcal{W}_2)$$

$$\sigma \ \wedge \ (\mathcal{W}_1 \vee \mathcal{W}_2) \quad = \quad (\sigma \ \wedge \ \mathcal{W}_1) \vee (\sigma \ \wedge \ \mathcal{W}_2)$$

Enforcing serial constraints. Let $\alpha, \beta \in \mathcal{EVENT}$ and let $\mathcal{W}$ be a Game-$\mathcal{CTR}$ workflow. Then:

$$*(\alpha \otimes \beta) \ \wedge \ \mathcal{W} = \texttt{sync}(\alpha < \beta, (*\alpha \ \wedge \ (*\beta \ \wedge \ \mathcal{W})))$$

Enforcing conjunctive primitive constraints. Use $*(\sigma_1 \wedge ... \wedge \sigma_m) \equiv *\sigma_1 \wedge ... \wedge *\sigma_m$, where all σ_i are elementary.

Solver Algorithm: Contd.

Enforcing disjunctive primitive constraints. Let σ_i be elementary constraints. Then

$$
\begin{aligned}
(\sigma_1 \vee \ldots \vee \sigma_n) \;\equiv\; & (\neg\sigma_2 \wedge \ldots \wedge *\neg\sigma_n \rightarrow *\sigma_1) \;\sqcap \ldots \\
& \sqcap\; (*\neg\sigma_1 \wedge \ldots \wedge *\neg\sigma_{i-1} \wedge *\neg\sigma_{i+1} \wedge \ldots \wedge *\neg\sigma_n \rightarrow *\sigma_i) \;\sqcap \ldots \\
& \sqcap\; (*\neg\sigma_1 \wedge \ldots \wedge *\neg\sigma_{n-1} \rightarrow *\sigma_n)
\end{aligned}
$$

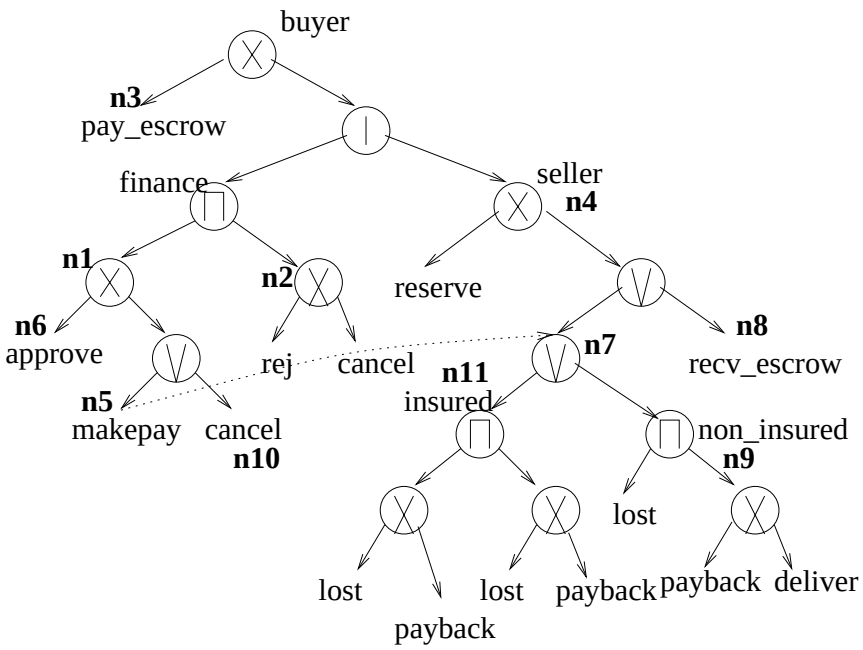
Maximal guarantee for an event. Let $*\sigma$ be an elementary constraint (*i.e.*, σ is e or $\neg e$), $\mathcal{W}$ be a workflow, and φ be a subformula of $\mathcal{W}$. $GS_{*\sigma}(\mathcal{W})$ iff φ is a max subformula of $\mathcal{W}$ and $(\mathcal{W} \;\wedge\; (\varphi \mid \texttt{Playset})) \rightarrow *\sigma$

Co-occurrence sub-games of a subformula. Let $\psi, \mathcal{W}, \varphi$ same as above then $\psi \in coExec(\mathcal{W}, \varphi)$ iff $(\mathcal{W} \;\wedge\; (\varphi \mid \texttt{Playset})) \rightarrow (\psi \mid \texttt{Playset})$, and, ϕ and ψ are disjoint subformulas in $\mathcal{W}$, and ψ is a maximal subformula in $\mathcal{W}$ satisfying (1) and (2)

$GS_{*\sigma}(\mathcal{W})$ and $coExec(\mathcal{W}, \varphi)$ can be computed in linear time in $|W|$.

Enforcing $(*\sigma_1 \wedge ... \wedge *\sigma_n \rightarrow *\sigma) \ \wedge \ \mathcal{W}$: An Example

To enforce $(*cancel \rightarrow *recv\_escrow)$



$GS_{*\neg cancel}$(buyer) = {n5}

CoExec(buyer, n5) = {n3, n4, n6}

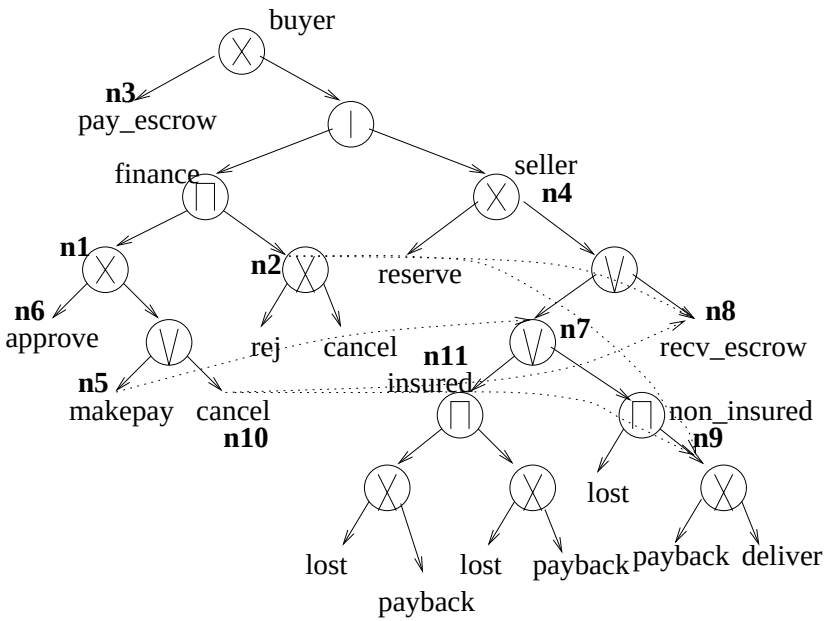
n4 can enforce *recv_escrow

$GS_{*\neg recv\_escrow}$(n4) = {n7}

Hence, delay(n5 < n7) !

Enforcing $(*\sigma_1 \wedge ... \wedge *\sigma_n \rightarrow *\sigma) \ \wedge \ \mathcal{W}$: An Example

To enforce $(*approve \ \wedge \ * \neg cancel \rightarrow *pay\_back)$



Case 1:

$GS_{*\neg approve}$(buyer) = {n2}

n4 $\in$ coExec(buyer, n2)

*payback is enforcible in n4

$GS_{*\neg payback}$(n4) = {n8, n9}

Hence, delay(n2 < n8, n2 < n9) !

Case 2:

$GS_{*\ cancel}$(buyer) = {n10, n2}

n4 $\in$ coExec(buyer, n2)

n4 $\in$ coExec(buyer, n10)

*payback is enforcible in n4

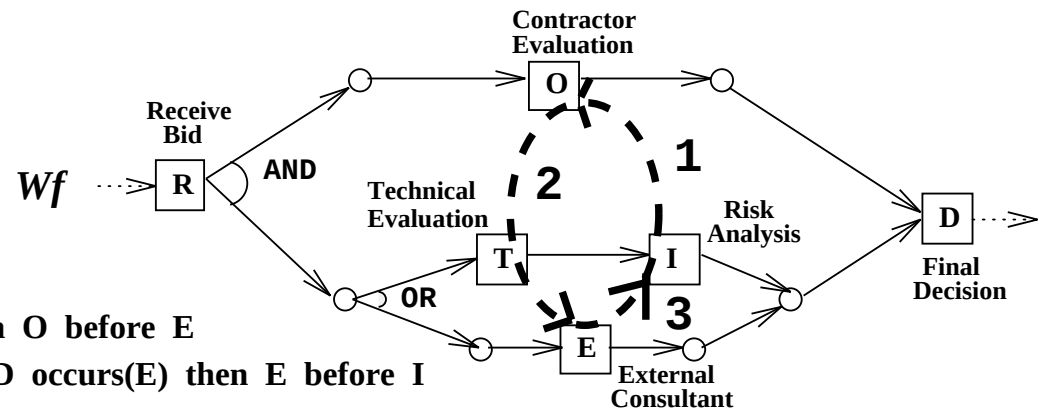
$GS_{*\neg payback}$(n4) = {n8, n9}

Hence, delay(n2 < n8, n2 < n9) !

Hence, delay(n10 < n8, n10 < n9) !

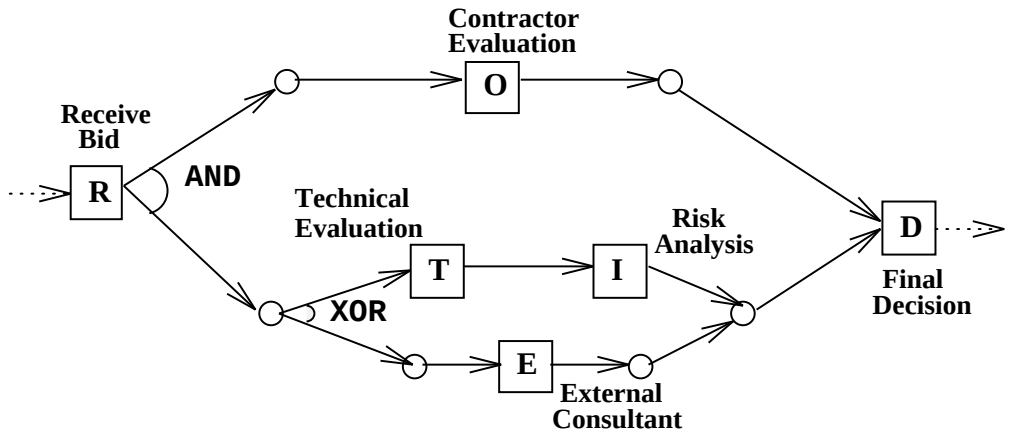
Cycle Detection and Elimination

Insertion of *send/receive* may cause a cyclic wait, which we call a *knot*.



- **Coord. Dependencies :**
  1. **I before O**
  2. **If occurs (E) then O before E**
  3. **If occurs (T) AND occurs(E) then E before I**

Unclog Transformation rewrites the above workflow into a knot-free workflow in $O(|W|^3)$ time:



Results for $\mathcal{CTR}$ workflows

Theorem (*Consistency*)
A workflow specification $\mathcal{G} \wedge \mathcal{C}$ is inconsistent iff $\texttt{Unclog}(\texttt{Apply}(\mathcal{C}, \mathcal{G})) \equiv false$.

Theorem (*Correctness*)
There is a constructive way of verifying workflow properties with `Apply`.

Theorem (*Coordination*)
Let $|\mathcal{G}|$ denote the size of a control-flow graph, then we can pick a legal execution from $\texttt{Unclog}(\texttt{Apply}(\mathcal{C}, \mathcal{G}))$ in time *linear* in $|\mathcal{G}|$.

Theorem (*Complexity*) Let $|\mathcal{G}|$ denote the size of a control-flow graph, N be the number of constraints in $\mathcal{C}$, and d be the largest number of disjuncts in a constraint.

- The worst-case size of $\texttt{Apply}(\mathcal{C}, \mathcal{G})$ is $O(d^N \times |\mathcal{G}|)$. (where as standard model-checking algorithms for this problem are exponential in $|\mathcal{G}|$)
- For certain classes of constraints $\texttt{Apply}(\mathcal{C}, \mathcal{G})$ is linear in $|\mathcal{G}|$.
- $\texttt{Unclog}(\mathcal{G})$ is linear in $|\mathcal{G}|$.

Results for Game-$\mathcal{CTR}$ workflows

Theorem 0.1 (Complexity for enforcing disjunctive primitive constraints). *Let $\mathcal{W}$ be a control flow graph and $*\Phi \in \mathcal{PRIMITIVE}$ be a disjunctive primitive constraint. Let $|\mathcal{W}|$ denote the size of $\mathcal{W}$, and d be the number of elementary disjuncts in $*\Phi$. Then the worst-case size of $*\Phi \wedge \mathcal{W}$ is $O(d \times |\mathcal{W}|)$, and the time complexity is $O(d \times |\mathcal{W}|^2)$*

Theorem 0.2 (Complexity of solving games). *Let $\mathcal{W}$ be a control flow graph $\mathcal{W}$ and $\Phi \subset \mathcal{CONSTR}$ be a set of global constraints in the conjunctive normal form $\bigwedge_N (\vee_j Prim)$ where $Prim \in \mathcal{PRIMITIVE}$. Let $|\mathcal{W}|$ denote the size of $\mathcal{W}$, N be the number of constraints in Φ, and d be the largest number of disjuncts in a primitive constraint in Φ. Then the worst-case size of $solve(\Phi, \mathcal{W}) \doteq \Phi \wedge \mathcal{W}$ is $O(d^N \times |\mathcal{W}|)$, and the time complexity is $O(d^N \times |\mathcal{W}|^2)$.*

Contributions of the Thesis

Efficient Workflow Verification Algorithm: *Polynomial* in the size of the workflow graph, exponential *only* in the size of the dependencies.

Efficient Workflow Scheduling Algorithm: Our scheduler is *linear* in the size of the workflow graph as opposed to *exponential schedulers* for temporal logic and event algebra schedulers.

More expressive framework for Non-Cooperative Service Workflows: Game-$\mathcal{CTR}$ is the first framework enabling programming, executing and reasoning with workflows of non-cooperative service compositions.

A Novel Game Logic: Game-$\mathcal{CTR}$ is interesting in its own right. Related game logics only deal with assertions about games and their winning strategies. Games are modalities rather than executable specifications, so they are only useful for *reasoning*, not for *programming* and *solving games and scheduling* them.

Future Work

Failure Management: Facilitate failure detection and handling with advanced features like contingency and compensation;

Workflows with Loops: Develop new algorithms for reasoning with workflows with cycles.

Dynamic Workflow Composition: Automatic identification of component services that match client's business requirements and dynamic reconfiguration upon failure.

Reasoning with Utility and Optimizations: Include a cost and gain functions for service invocations and compute Nash-like strategies.

Formal Soundness and Completeness Proof for the Game-$\mathcal{CTR}$