Enforcing behavior with contracts

Ralph-Johan Back Joakim von Wright

Åbo Akademi University and TUCS

September 5, 2000

Contracts

Contracts

Consider a collection of *agents* Ω, each with the capability to change the state by choosing between different *actions*. We let A range over sets of agents and a, b, c over individual agents.

The behavior of agents is regulated by *contracts* S:

$$\begin{array}{lll} S & ::= & \langle f \rangle \mid \langle p \rangle_a \mid \langle R \rangle_a \mid \\ & & \mathsf{if}\ p\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \mid S_1 \,;\, S_2 \mid \\ & & S_1\ []_a\ S_2 \mid \\ & & (\mathsf{rec}_a\ X \bullet S) \mid X \end{array}$$

Here a stands for an agent while f stands for a state transformer, p for a state predicate, and R for a state relation, all expressed using higher-order logic. X is a variable that ranges over (the meaning of) contracts.

Executing a contract

- The *functional update* $\langle f \rangle$ changes the state by applying the state transformer f, i.e., if the initial state is σ_0 then the final state is $f.\sigma_0$. An assignment statement is a special kind of update: $\langle x := x + 1 \rangle$

- In the *conditional composition* $\mathsf{if}\ p\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$, S_1 is carried out if p holds in the initial state, and S_2 otherwise.

- The *assertion* $\langle p \rangle_a$ checks whether condition p holds. An example: $\langle x + y = 0 \rangle_a$. If the assertion holds at the indicated place, then the state is unchanged, and the rest of the contract is carried out. If, on the other hand, the assertion does not hold, then agent a has breached the contract.

- The *relational update* $\langle R \rangle_a$ requires the agent a to choose a final state σ' so that $R.\sigma.\sigma'$ is satisfied, where σ is the initial state. If it is impossible for the agent to satisfy this, then the agent has *breached* the contract. An example: $\langle x := x' \mid x' < x \rangle_a$.

- A *choice* $S_1\ [\,]_a\ S_2$ allows agent a to choose which subcontract is to be carried out, S_1 or S_2.

- In the *sequential composition* $S_1\ ;\ S_2$, subcontract S_1 is first carried out, followed by S_2, provided that there is no breach of contract when executing S_1.

Recursion

A *recursive contract* is of the form

$$X \quad =_a \quad S$$

where S may contain occurrences of the contract variable X.

We also use the syntax $(\mathsf{rec}_a\ X \bullet S)$ for the contract X defined by the equation $X = S$.

The index a for the recursion construct indicates that agent a is responsible for termination of the recursion. If the recursion does not terminate, then a will be considered as having breached the contract.

Example: The while loop is defined by

$$\mathsf{while}_a\ p\ \mathsf{do}\ S\ \mathsf{od} \quad \triangleq \quad (\mathsf{rec}_a\ X \bullet \mathsf{if}\ p\ \mathsf{then}\ S\ ;X\ \mathsf{else\ skip\ fi})$$

Role of agents

In general, agents have two roles in contracts:

- they *choose* between different alternatives that are offered to them, and
- they take the *blame* when things go wrong.

These two roles are interlinked, in the sense that things go wrong when an agent has to make a choice, and there is no acceptable choice available.

Operational semantics

The rules of the operational semantics are given in terms of a transition relation between configurations. A *configuration* is a pair (S, σ), where

- S is either an ordinary contract statement or the empty statement symbol Λ, and
- σ is either an ordinary state, or the symbol $\bot_a$ (indicating that agent a has breached the contract).

The transition relation $\rightarrow$ shows what moves are permitted. It is inductively defined by a collection of axioms and inference rules.

We assume that σ stands for a proper state while γ stands for either a state or the symbol $\bot_x$ for some agent x

Transition axioms

- *Functional update*

$$\frac{}{(\langle f\rangle, \sigma) \to (\Lambda, f.\,\sigma)} \qquad \frac{}{(\langle f\rangle, \bot_a) \to (\Lambda, \bot_a)}$$

- *Assertion*

$$\frac{p.\,\sigma}{(\langle p\rangle_a, \sigma) \to (\Lambda, \sigma)} \qquad \frac{\neg p.\,\sigma}{(\langle p\rangle_q, \sigma) \to (\bot_a, \sigma)}$$

- *Conditional composition*

$$\frac{p.\,\sigma}{(\textsf{if } p \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}, \sigma) \to (S_1, \sigma)} \qquad \frac{\neg p.\,\sigma}{(\textsf{if } p \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}, \sigma) \to (S_2, \sigma)}$$

$$\frac{}{(\textsf{if } p \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}, \bot_a) \to (\Lambda, \bot_a)}$$

- *Sequential composition*

$$\frac{(S_1,\gamma)\to(S_1',\gamma'),\quad S_1'\neq\Lambda}{(S_1\,;S_2,\gamma)\to(S_1'\,;S_2,\gamma')}\qquad\frac{(S_1,\gamma)\to(\Lambda,\gamma')}{(S_1\,;S_2,\gamma)\to(S_2,\gamma')}$$

- *Relational update*

$$\frac{R.\,\sigma.\,\sigma'}{(\langle R\rangle_a,\sigma)\to(\Lambda,\sigma')}\qquad\frac{R.\,\sigma=\emptyset}{(\langle R\rangle_a,\sigma)\to(\Lambda,\bot_a)}\qquad\frac{}{(\langle R\rangle_a,\bot_b)\to(\Lambda,\bot_b)}$$

- *Choice*

$$\frac{}{(S_1\ []_a\ S_2,\gamma)\to(S_1,\gamma)}\qquad\frac{}{(S_1\ []_a\ S_2,\gamma)\to(S_2,\gamma)}$$

- *Recursion*

$$\frac{}{((\mathsf{rec}_a\ X\bullet S),\gamma)\to(S[X:=(\mathsf{rec}_a\ X\bullet S)],\gamma)}$$

Scenarios and behaviors

A *scenario* for the contract S in initial state σ is a sequence of configurations

$$C_0 \to C_1 \to C_2 \to \cdots$$

where

1. $C_0 = (S, \sigma)$,
2. each transition $C_i \to C_{i+1}$ is permitted by the axiomatization above, and
3. if the sequence is finite with last configuration C_n, then $C_n = (\Lambda, \gamma)$, for some γ.

Intuitively, a scenario shows us, step by step, what choices the different agents have made and how the state is changed when the contract is being carried out.

A finite scenario cannot be extended, since no transitions are possible from an empty configuration.

Note that the statement component of a configuration shows which agent is to choose the next step.

Example

Consider the contract (a is user, b is computer).

$$\begin{aligned} S \quad = \quad & (x := x + 1 \; []_a \; x := x + 2) \, ; \\ & (x := x - 1 \; []_b \; x := x - 2) \end{aligned}$$

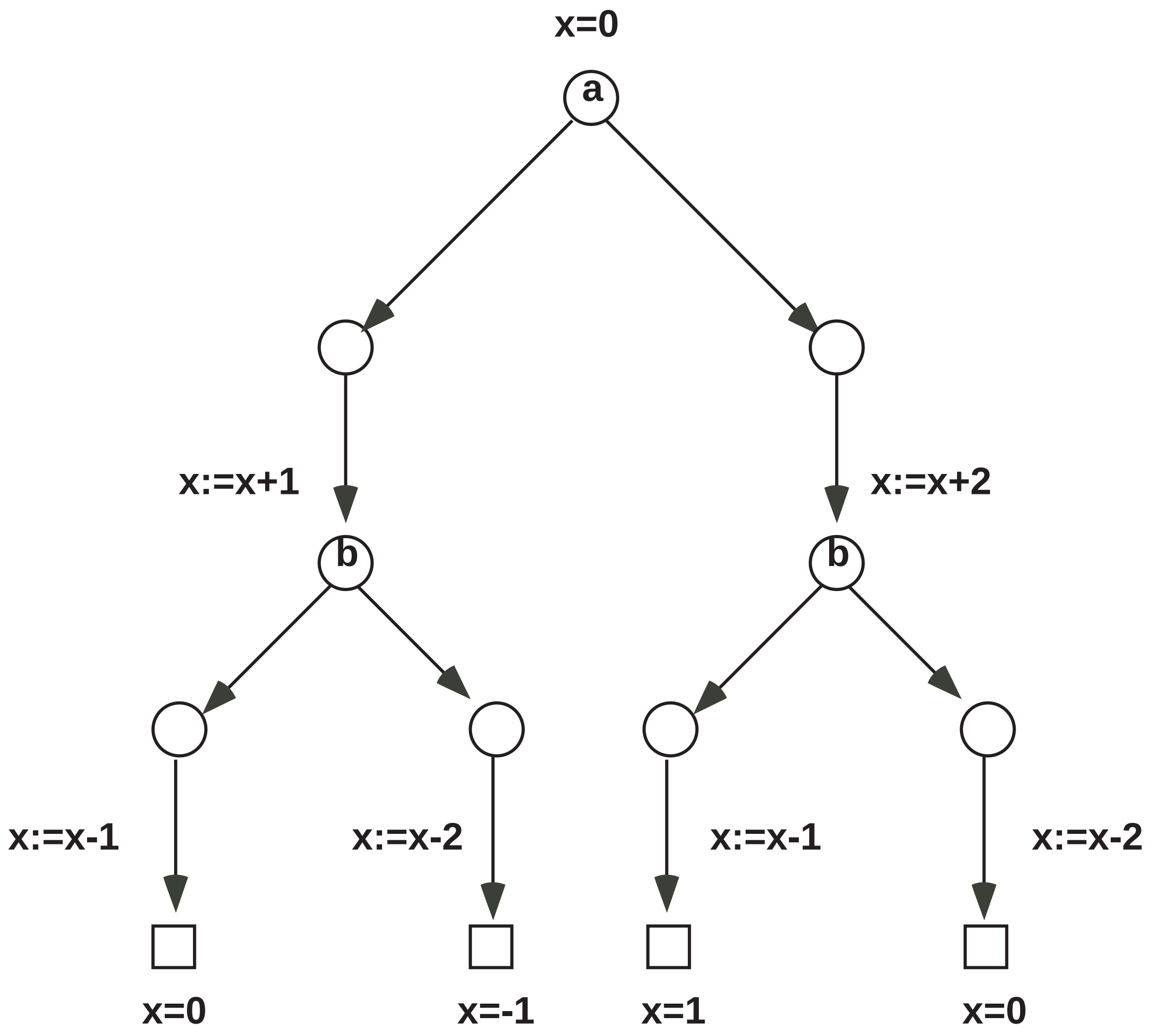
After initialization, the user a chooses to increase the value of x by either one or two. After this, the computer b decides to decrease x by either one or two.

The choice of the user depends on what she wants to achieve. If, e.g., she is determined that x should not become negative, she should choose the second alternative.

If, again, she is determined that x should not become positive, she should choose the first alternative.

We can imagine this user interaction as a *menu choice* that is presented to the user after the initialization, where the user is requested to choose one of the two alternatives.

Execution tree



Alternative view

We could also consider b to be the user and a to be the computer.

In this case, the system starts by either setting x to one or to two. The user can then inspect the new value of x and choose to reduce it by either 1 or 2, depending on what she tries to achieve.

Agent b can achieve that $x = 0$. Agent b can also achieve that $x \neq 0$.

Winning strategy and correctness

We say that *agents A can use contract S in initial state σ to establish postcondition q*,

$$\sigma \;\{\!|\, S \,|\!\}_A\; q$$

if there is a *winning strategy* for the agents in A which guarantees that initial configuration (S, σ) will

- either lead to termination in such a way that the final configuration is some (Λ, γ) where γ is either a final state in q or $\bot_b$ for some $b \notin A$, or
- prevent termination, when another agent is responsible for the termination.

Thus $\sigma \;\{\!|\, S \,|\!\}_A\; q$ means that, no matter what the other agents do, the agents in A can (by making suitable choices) either achieve postcondition q or make sure that som agent outside A breaches the contract.

We define

$$p \;\{\!|\, S \,|\!\}_A\; q \quad \triangleq \quad (\forall \sigma \in p \bullet \sigma \;\{\!|\, S \,|\!\}_A\; q)$$

The traditional notion of correctness for deterministic and (demonically) nondeterministic programs is a special case of this defition.

Weakest preconditions

Assume that S is a contract statement and A a *coalition*, i.e., a set of agents.

We want to define the *predicate transformer*

$$\mathsf{wp}.\, S.\, A$$

so that it maps postcondition q to the set of all initial states σ from which the agents in A have awinning strategy to reach the goal q if they co-operate.

Thus, $\mathsf{wp}.\, S.\, A.\, q$ would be the *weakest precondition* that guarantees that the agents in A can cooperate to *achieve* postcondition q.

Weakest preconditions for contracts

$$
\begin{array}{rcl}
\mathsf{wp}.\,\langle f\rangle.\,A.\,q & = & (\lambda\,\sigma \bullet q.\,(f.\,\sigma)) \\
\mathsf{wp}.\,(\mathsf{if}\ p\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}).\,A.\,q & = & (p \cap \mathsf{wp}.\,S_1.\,A.\,q) \cup (\neg p \cap \mathsf{wp}.\,S_2.\,A.\,q) \\
\mathsf{wp}.\,(S_1\ ;\ S_2).\,A.\,q & = & \mathsf{wp}.\,S_1.\,A.\,(\mathsf{wp}.\,S_2.\,A.\,q) \\
\mathsf{wp}.\,\langle R\rangle_a.\,A.\,q & = & \begin{cases} (\lambda\,\sigma \bullet \exists\,\sigma' \bullet R.\,\sigma.\,\sigma' \wedge q.\,\sigma') & \text{if } a \in A \\ (\lambda\,\sigma \bullet \forall\,\sigma' \bullet R.\,\sigma.\,\sigma' \Rightarrow q.\,\sigma') & \text{if } a \notin A \end{cases} \\
\mathsf{wp}.\,(S_1\ []_a\ S_2).\,A.\,q & = & \begin{cases} \mathsf{wp}.\,S_1.\,A.\,q \cup \mathsf{wp}.\,S_2.\,A.\,q & \text{if } a \in A \\ \mathsf{wp}.\,S_1.\,A.\,q \cap \mathsf{wp}.\,S_2.\,A.\,q & \text{if } a \notin A \end{cases}
\end{array}
$$

Wp for recursive contracts

The semantics of a recursive contract is given in a standard way, using least fixpoints.

$$\mathsf{wp}.\,(\mathsf{rec}_a\ X \bullet S).\,A \quad = \quad \begin{cases} \mu.\,f_{S,A} & \text{if } a \in A \\ \nu.\,f_{S,A} & \text{if } a \notin A \end{cases}$$

We take the least fixed point μ when non-termination is considered bad, as is the case when agent $a \in A$ is responsible for termination.

We take the greatest fixpoint ν when termination is considered good, i.e., when an agent not in A is responsible for termiation.

Correctness and weakest preconditions

The connection between winning strategies and weakest preconditions is the desired one:

Theorem 1 *Assume that contract statement S, coalition A, precondition p and postcondition q are given. Then $p \subseteq \mathsf{wp}.\,S.\,A.\,q$ if and only if $p \;\{\!|\, S \,|\!\}_A\; q$.*

Enforcing behavioral properties

Example contract

Consider the contract:

$$\begin{aligned} S &= (x := x + 1 \;[]_a\; x := x + 2)\;; \\ & \quad\; (x := x - 1 \;[]_b\; x := x - 2) \end{aligned}$$

A temporal property is, e.g., that a can *enforce* the propery $0 \leq x \leq 2$ to hold during the execution of the contract:

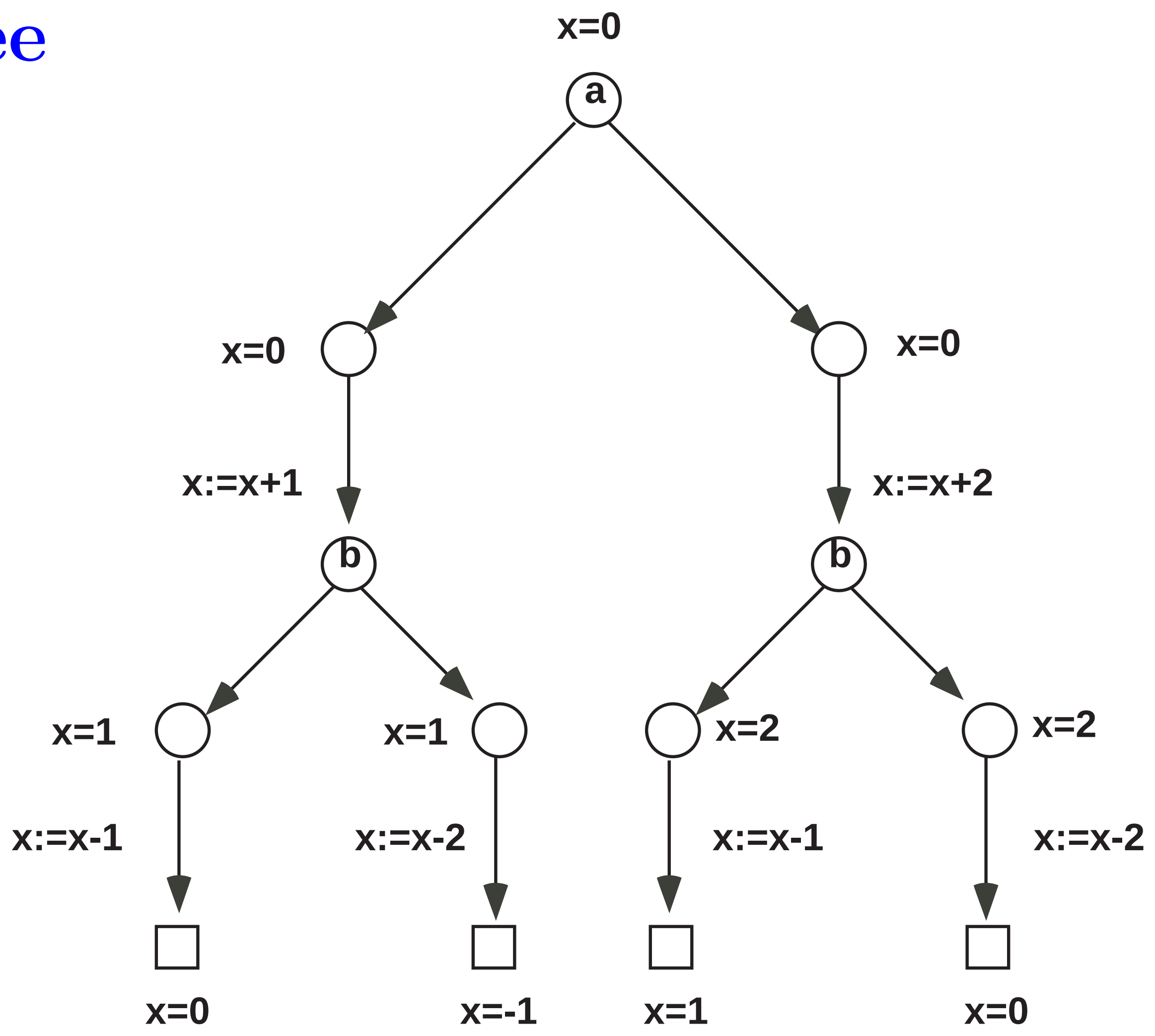
$$x = 0 \;\{\!|\; S \;|\!\}_a\; \Box(0 \leq x \leq 2)$$

Here $\Box(0 \leq x \leq 2)$ says that $0 \leq x \leq 2$ is *always* true.

This property need not hold for every possible execution, it is sufficient that there is a way for a to *enforce* the property by making suitable choices during the execution.

Using the operational semantics of S, we can determine all the possible execution sequences of this contract.

Execution tree



Properties

Agent a *can* enforce the condition $\Box(0 \leq x \leq 2)$, irrespectively of which alternative b chooses.

Agent a *can not* enforce the condition $\Box(1 \leq x \leq 2)$.

Agent a *can* also enforce the condition $\Diamond(x = 1)$ (eventually $x = 1$ holds)

Interpreting a contract

Consider a contract statement S that operates on a state space Σ, and includes agents Ω.

Question: When can the temporal property $\Box p$ be enforced by a coalition of agents $A \in \Omega$.

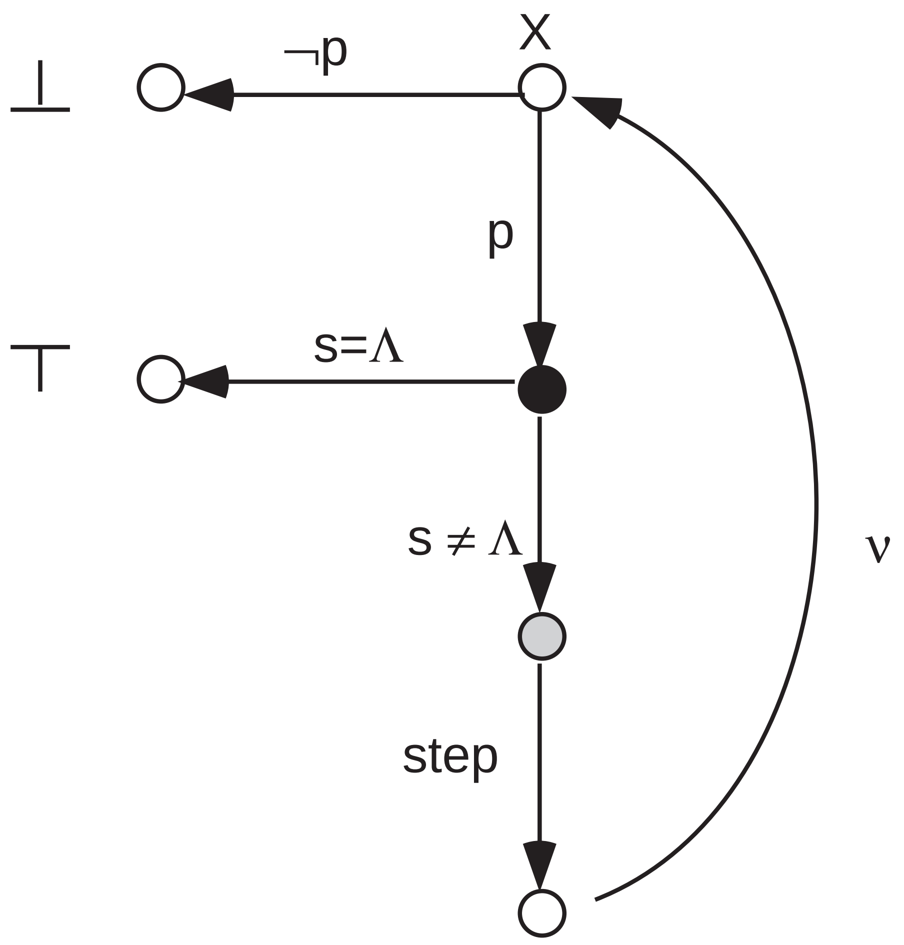
Define the the contract $\mathsf{Always}.\, p$, called a *tester for $\Box p$*, by

$$\begin{array}{rcl} \mathsf{Always}.\, p & = & (\nu\ X \bullet \{p\}\,;[s \neq \Lambda]\,;\mathit{step}\,;X) \\ \mathit{step} & = & \langle s, \sigma := s', \sigma' | (s, \sigma) \to (s', \sigma') \rangle_{ch.\, s} \end{array}$$

This contract operates on a state space consisting of tuples (s, σ), where s is a contract statement and σ a state in Σ. The function $ch.\, s$ gives the agent that is making the choice in s, if any.

The tester is a *special interpreter* for contract statements, which executes them in order to determine whether a specific temporal property is valid.

Diagram for always tester



Explanation

The diagram shows the angelic choices as hollow circles, and the demonic choices as filled circles.

A grey circle indicates that we do not know whether the choice is angelic or demonic, or maybe both.

The X labels the node at which the iteration starts.

The arrow labeled ν indicates that we have ν-iteration, i.e., the arrow can be traversed any number of times.

An arrow labeled μ would indicate μ-iteration, where the arrow only can be traversed a finite number of times during each iteration.

Testing lemma

Lemma 2 *Let S, p, and C be as above. Let A be a coalition of agents in Ω. Then*

$$\sigma \ \{\!|\ S\ |\!\}_A\ \Box p \quad \equiv \quad \mathsf{wp}.\,(\mathsf{Always}.\,p).\,A.\,\mathsf{false}.\,(S,\sigma)$$

Enforcement can thus be reduced to *achievement*: whether a temporal property can be enforced for a contract can be reduced to the question of whether a certain goal can be achieved.

In this case, the goal itself is false and cannot really be achieved, so success can only be achieved by miraculous termination, or by nontermination of another agent, not belonging to A.

Tester for eventuality

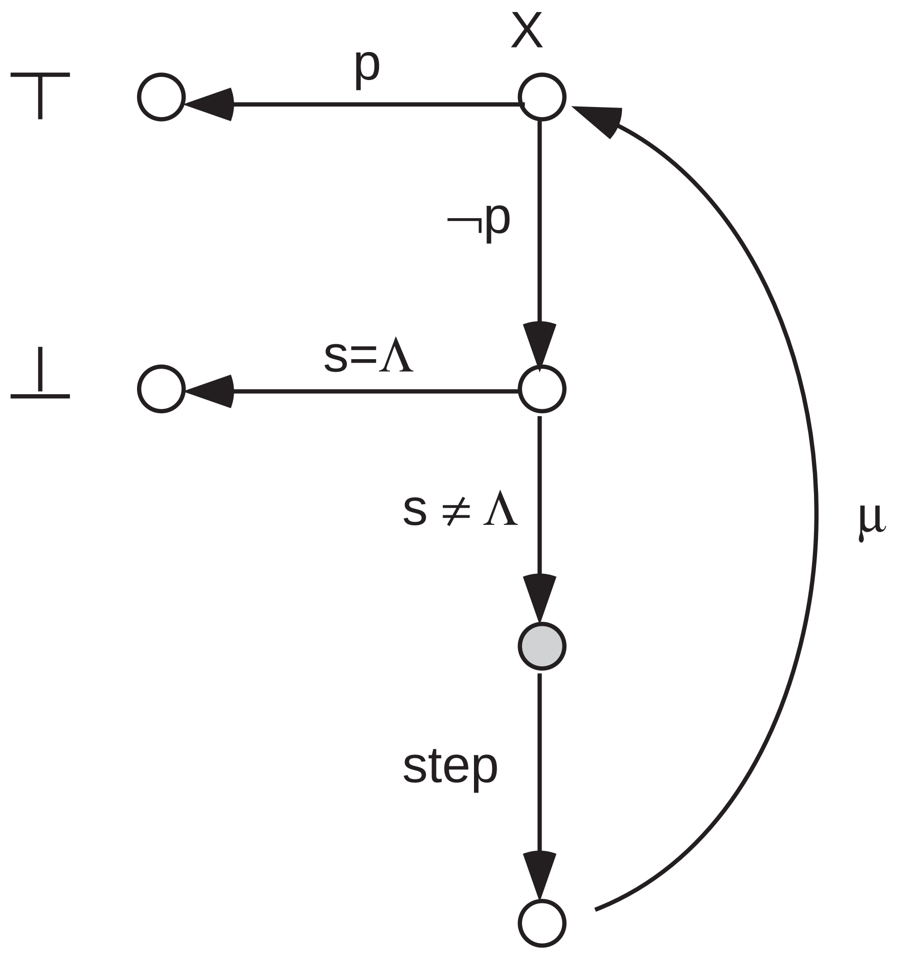
In a similar way, we can define a tester $\mathsf{Eventually}.\,p$ for the property $\Diamond p$, by

$$\mathsf{Eventually}.\,p \quad = \quad (\mu\ X \bullet [\neg p]\,;\{s \neq \Lambda\}\,;\mathit{step}\,; X)$$

Lemma 3 *Let S, p, and C be as above. Let A be a coalition of agents in Ω. Then*

$$\sigma\ \{\!|\ S\ |\!\}_A\ \Diamond p \quad \equiv \quad \mathsf{wp}.\,(\mathsf{Eventually}.\,p).\,A.\,\mathsf{false}.\,(S,\sigma)$$

Diagram for eventuality tester



Until

We say that a property p holds until property q, denoted $p \mathcal{U} q$, if q will hold eventually, and until then p holds. We have that

$$\sigma_0 \ \{\!|\ S\ |\!\}\ p\, \mathcal{U}\ q$$

holds if and only if

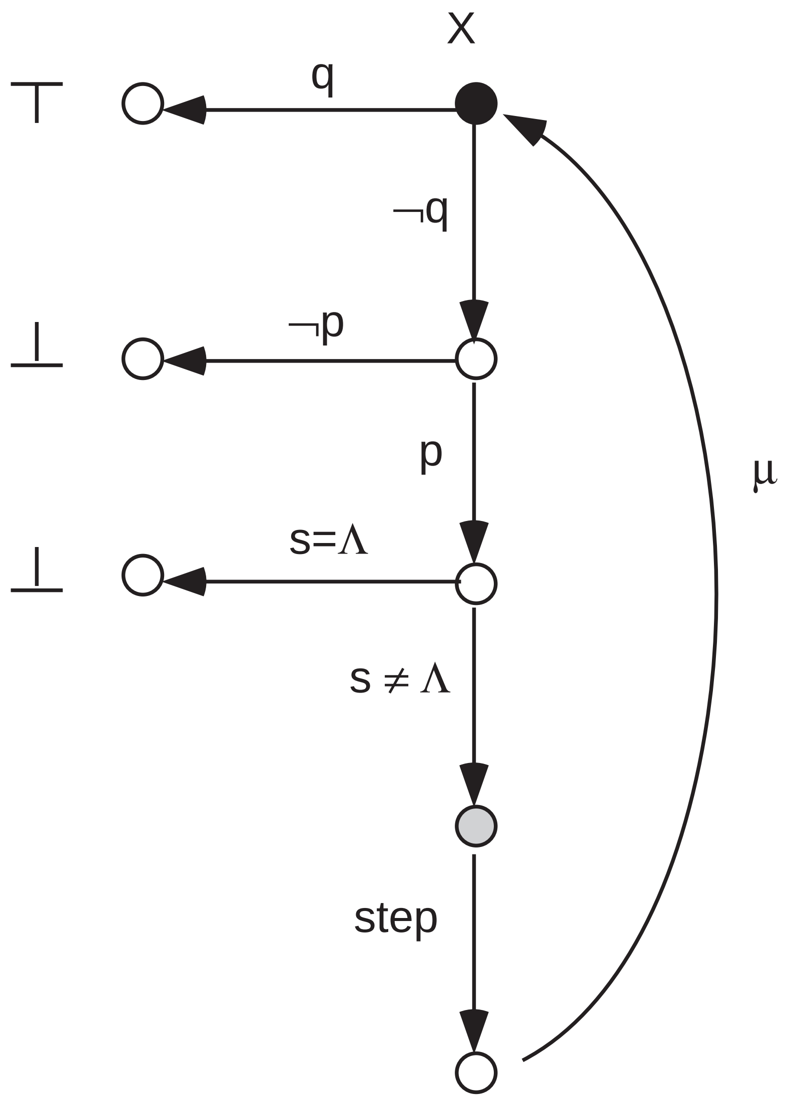
$$\tau\ \{\!|\ (\mu X \bullet [\neg q]\,;\{p\}\,;\{s \neq \Lambda\}\,;\mathrm{step}\,;X)\ |\!\}\ \mathsf{false}$$

where $\sigma.\tau = \sigma_0$ and $s.\tau = S$.

The *weak until*, denoted $p \mathcal{W} q$, can be defined in a similar matter. It differs from the previous operator in that it is also satisfied if q is never satisfied, provided p is always satisfied.

The always and sometimes operators arise as special cases of these operators: $\Box p = p\ \mathcal{W}\ \mathsf{false}$ and $\Diamond p = \mathsf{true}\ \mathcal{U}\ p$.

Diagram for until



Leads to

Another interesting property is that condition p *leads to* condition q, denoted $p \mapsto q$. We have that

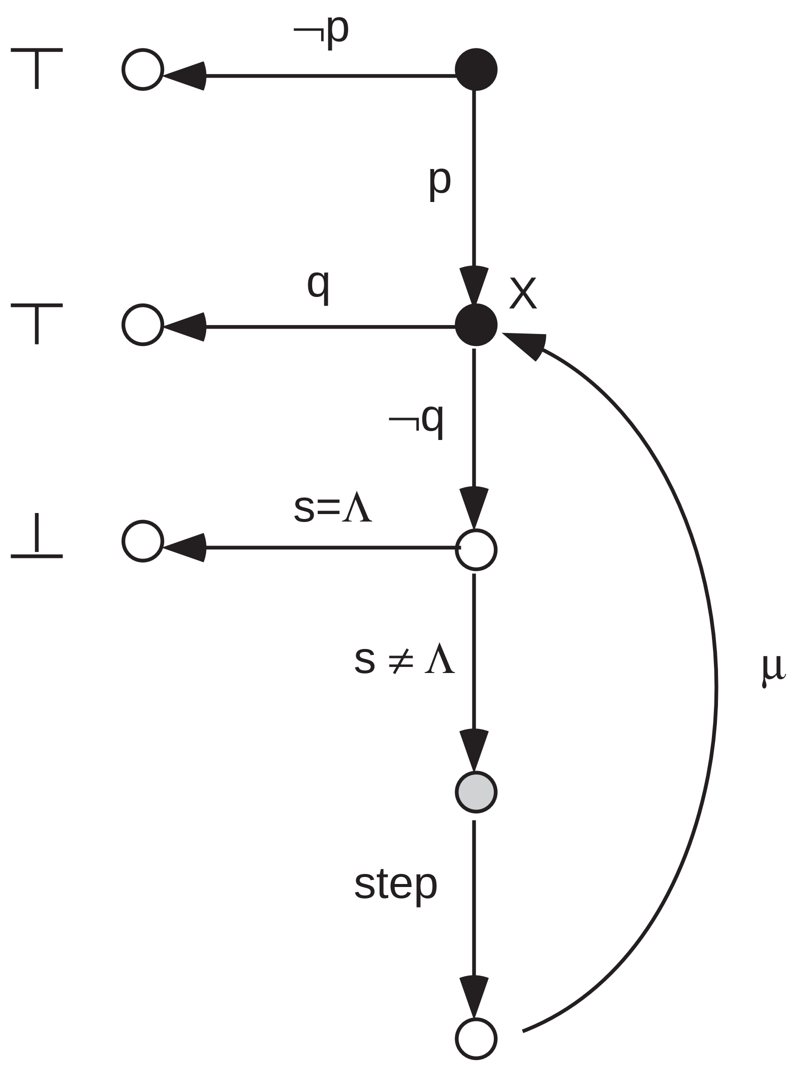
$$\sigma_0 \;\{\!|\; S \;|\!\}\; p \mapsto q$$

holds if and only if

$$\tau \;\{\!|\; [p]\,;(\mu X \bullet [\neg q]\,;\{s \neq \Lambda\}\,;\mathrm{step}\,;X) \;|\!\}\; \mathsf{false}$$

where $\sigma.\tau = \sigma_0$ and $s.\tau = S$.

Diagram for leads to



Always leads to

An even more useful property is to say that property p always leads to property q, denoted by $\Box(p \mapsto q)$. This requires that we use two loops, a ν- loop and a μ-loop. We have that

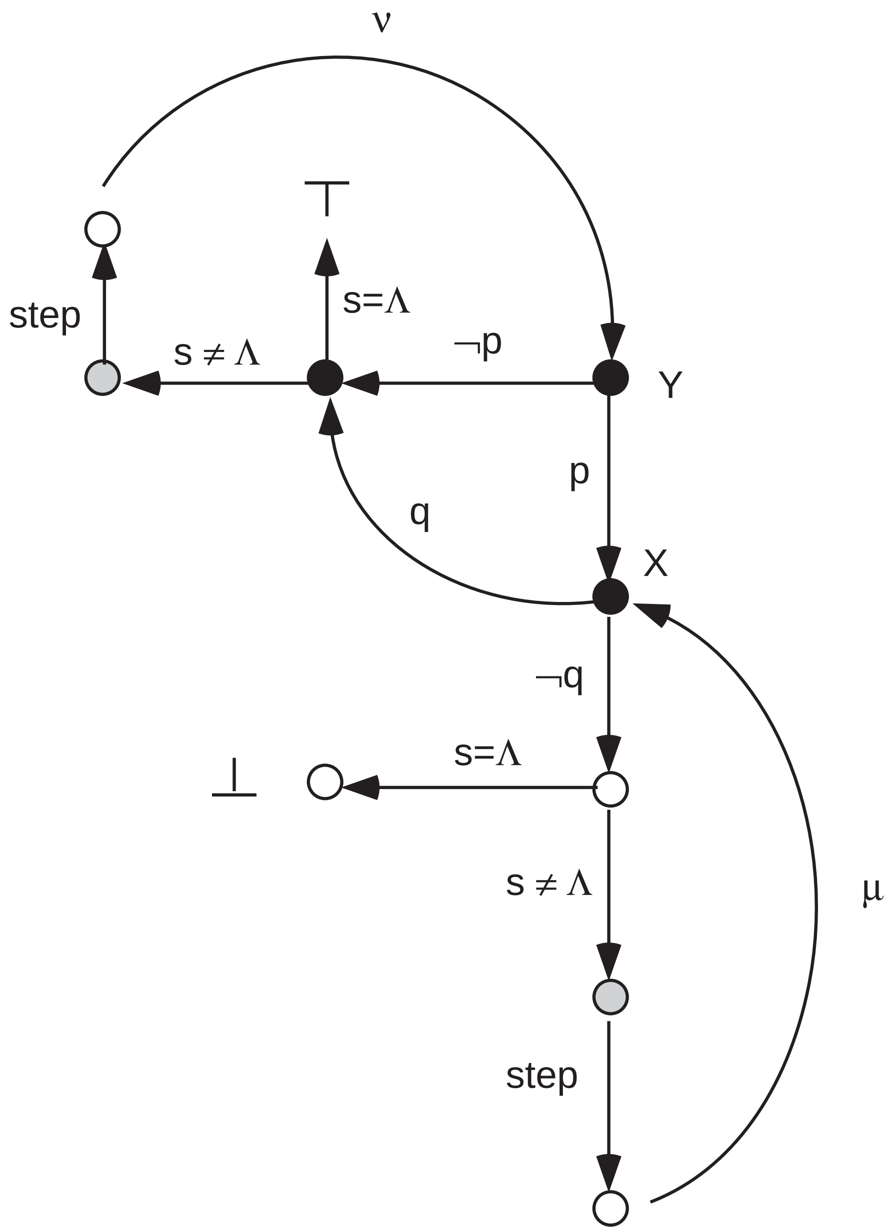
$$\sigma_0 \; \{\!|\; S \;|\!\} \; \Box(p \mapsto q)$$

holds if and only if $\tau \; \{\!|\; H \;|\!\} \; \mathsf{false}$ holds, where

$$\begin{aligned} H \quad = \quad & (\nu Y \bullet [\neg p] \,;\, [s \neq \Lambda] \,;\, \mathrm{step} \,;\, Y \\ & \qquad \sqcap [p] \,;\, (\mu X \bullet [\neg q] \,;\, \{s \neq \Lambda\} \,;\, \mathrm{step} \,;\, X \sqcap [q]) \,;\, [s \neq \Lambda] \,;\, \mathrm{step} \,;\, Y) \end{aligned}$$

where $\sigma.\tau = \sigma_0$ and $s.\tau = S$.

Diagram for always leads to



Insensitive to stuttering

The above behavioral properties are all *insensitive to finite stuttering*: if a step of the computation does not change the state, then the effect is the same as if that step had been omitted.

In the diagram, this means that a stuttering step will lead back to the same state in the diagram.

Being insensitive to stuttering means that the number of steps that are taken is not important for the behavioral property, only the sequence of properties that arise during the execution.

Note that a computation should not be insensitive to infinite stuttering, as this is amounts to a form of nontermination.

Concurrent and interactive systems

Action loop

An *action loop* is of the form

$$(\mathsf{rec}_c\ X \bullet \langle S \rangle\ ;\ X\ []_a\ \langle g \rangle_a)$$

The contract S is in this contract iterated as long as agent a wants.

Termination is normal if the *exit condition* g holds when a decides to stop the iteration, otherwise a will fail.

Agent c gets the blame if the execution does not terminate.

Action systems

In general, we will also allow an *initialization (begin)* statement B and a *finalization (end)* statement E, in addition to the *action* statement S.

The initialization statement would typically introduce some local variables for the action system, and initialize these. The finalization statement would then remove these local variables.

The action statement S can in turn be a choice statement,

$$S \quad = \quad S_1 \; []_b \; S_2 \; []_b \; \ldots \; []_b \; S_n$$

We refer to each S_i here as an *action* of the system.

Thus, an action system is in general of the form

$$B \; ; (\mathsf{rec}_c \; X \bullet (S_1 \; []_b \; S_2 \; []_b \; \ldots \; []_b \; S_n) \; ; X \; []_a \; \langle g \rangle_a) \; ; E$$

8 kinds of action systems

There are 8 different possibilities to consider for (a, b, c).

- *Angelic iteration* $(a, b \in A)$: models an *event loop*, where the user can choose what action or event to execute next, and also may choose to exit the loop whenever this is allowed by the exit constraint.
- *Angelic iteration with demonic exit* $(a \notin A, b \in A)$: like an event loop, where termination may happen at any time when termination is enabled, the choice of when to terminate being outside the control of the user.
- *Demonic iteration* $(a \notin A, b \notin A)$: models a *concurrent system*, where the nondeterminism in the choice of the next iteration action expresses the arbitrary interleaving of enabled actions.
- *Demonic iteration with angelic exit* $(a \in A, b \notin A)$: a concurrent system, where termination is decided by the user. At each step, the user can decide whether to terminate or continue (provided the exit condition is satisfied), but the user cannot influence the choice of the next action, if he decides to continue.

In each case, $c \in A$ means that the computation must terminate eventually for A not to breach the contract, $c \notin A$ means that the loop can go on forever.

Termination of loop

Termination is more complex here than is usully considered:

- We can have a situation where neither the exit condition nor any of the actions is enabled (can be taken without immediate failure). In this case, the agent who is to choose between termination of continuation looses (the exit agent).
- We can have a situation where both termination and some action is enabled. In this case, termination is nondeterministic, and is determined by the exit agent.
- It is possible that termination is enabled if and only if no action is enabled. In this case, termination is deterministic. This is the traditional semantics for, e.g., Dijkstra's guarded iteration statement.
- It is possible that the exit guard is never enabled (i.e., $g = \mathsf{false}$), in which case the iteration never terminates. This is the traditional choice in termporal logic models.

Traditional systems

Dijkstra's *guarded iteration statement* is a special kind of demonic iteration, where the exit condition is the negation of the conditions when the actions are enabled. Thus, there is no nondeterminism regarding termination. The actions can only have demonic iteration and angelic assert statements (abort).

A traditional *temporal logic model* is essentially a demonic iteration where the exit condition is always false, i.e., the system never terminates, and no abortions are permitted.

Our formalization introduce three main extensions to the traditional temporal logic model:

- the possibility that an execution may *terminate*,
- the possibility of *angelic choice* during the execution, and
- the possibility of a *failed* or *miraculousy succesful* execution.

Duality of concurrency and interactivity

Action systems can be used to model both

- *interactive systems*, where the choice of actions is under the control of the user, and
- *concurrent system*s, where the choice of actions is outside the control of the urser.

These two can be seen as duals of each other.

In fact, the action contract formalism is much more expressive than either one of these two formalisms, because the actions themselves may also be either angelic or demonic.

An angelic choices made inside an action can be understood as an *input statement* in the action.

A demonic choice inside an action can be understood as a *specification statement*, where only partial information about the result is known.

Analyzing behavior of action systems

Observing action system behavior

We consider the actions S_i as a *specification* of what kind of state change is taking place, rather than as an actual implementation.

We will therefore assume that the execution of S is *atomic*, in the sense that the state can not be observed inside the execution of S.

This means that a state may violate the property p inside the execution of S, without violating the property $\Box p$ and it may satisfy the property p inside the execution of S, without satisfying the property $\Diamond p$.

If the internal working of S needs to be taken into account, then each internal step has to be modelled as a separate action.

Internal steps

We augment the syntax and operational semantics of contracts with a feature that indicates that a sequence of execution steps are internal, thus resulting in unobservable internal states:

- hide will make the subsequent steps non-observable, and
- unhide will make them observable again.

Rules for operational semantics:

$$\frac{}{(\mathsf{hide}, (\sigma, o)) \rightarrow (\Lambda, (\sigma, \mathsf{F}))} \qquad \frac{}{(\mathsf{unhide}, (\sigma, o)) \rightarrow (\Lambda, (\sigma, \mathsf{T}))}$$

The hide and unhide operations thus just toggle the flag o, indicating whether the state is considered observable or not. No nesting of hide and unhide is permitted.

A contract statement whose internal computation is hidden is denoted $\langle S \rangle$:

$$\langle S \rangle \quad = \quad \mathsf{hide}\ ; S\ ; \mathsf{unhide}$$

Modifying the tester

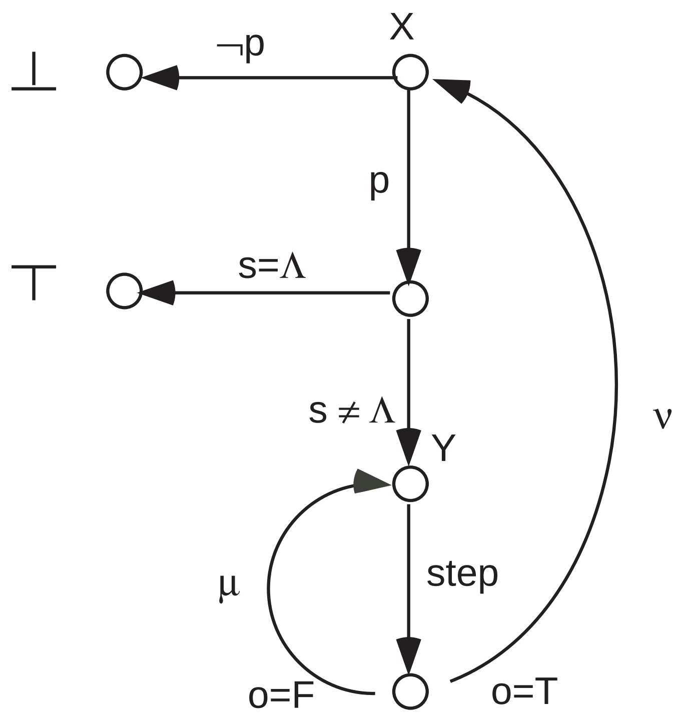
We need to modify the tester, to take the hidden states into account. The modified tester for the property $\Box p$ is as follows:

$$\mathsf{Always}.\,p \quad = \quad (\nu\ X \bullet \{p\}\,;\,[s \neq \Lambda]\,;\,(\mu\ Y \bullet \mathit{step}\,;\,\mathsf{if}\ o\ \mathsf{then}\ X\ \ \mathsf{else}\ \ Y\ \mathsf{fi}))$$

We consider internal non-termination in evaluating the action as bad, hence the μ label on the arrow in the inner loop. This means that we do not permit *internal divergence*.

The alternative that nontermination here is good, is also reasonable.

Tester respecting internal steps



Derivation of always-tester for action systems

We can compute the precondition for agents A to enforce the property $\Box p$ in the action system

$$\mathcal{A} \quad = \quad (\mathsf{rec}_c\ X \bullet \langle S \rangle \,;\, X \,[]_a\, \langle g \rangle_a)$$

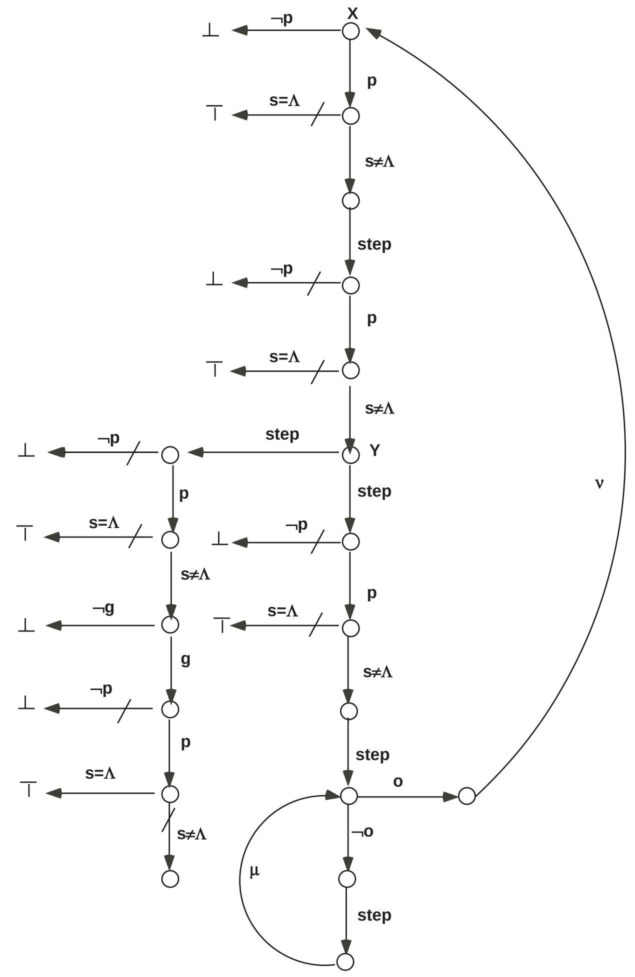
assuming that $a \in A$ and that $c \notin A$. We have that

$$\sigma \ \{\!|\, \mathcal{A} \,|\!\}_A \ \Box p \quad \equiv \quad (\nu X \bullet \{p\} \,;\, [\neg g] \,;\, \mathsf{wp}.\,S \,;\, X).\,\mathsf{false}$$

We can show that this is indeed the case, by considering how the coalition A would execute the contract $\mathsf{Always}.\,p$ from initial state $(\mathcal{A}, (\sigma, T))$.

The main advantage here is that we can argue directly about the weakest preconditions of the actions, without having to go the indirect route of an interpreter for the system.

Derivation of tester for action system



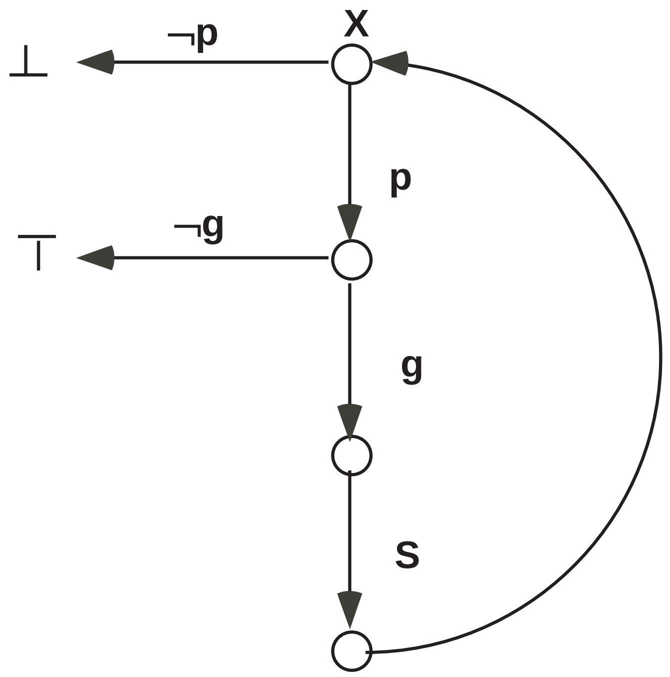
Explanation

We have crossed out all those branches in the figure that cannot be taken, because the condition is known to be false.

By eliminating these branches, as well as branches that the coalition would avoid because they would lead to certain failure, we derive the simpler diagram.

This proves that our characterization of the always tester for the action system $\mathcal{A}$ given above is correct.

Simpler diagram for action system



Examples

The game of Nim

Two players take turns removing sticks from a pile.

A player can remove one or two sticks at a time.

The player who removes the last stick loses the game.

We model the players as two agents a and b. Agent c is the scheduler, who decides which player makes the first move. Agent z is responsible for termination.

Nim contract

The game of Nim is described by the following contract:

$$\begin{array}{lll} Nim & = & (f := \mathsf{T} \ []_c \ f := \mathsf{F}) \ ; Play \\ Play & =_z & \quad f \wedge x > 0 \rightarrow \langle x := x' \mid x' < x \leq x' + 2 \rangle_a \ ; f := \neg f \ ; Play \\ & & []_c \ \neg f \wedge x > 0 \rightarrow \langle x := x' \mid x' < x \leq x' + 2 \rangle_b \ ; f := \neg f \ ; Play \\ & & []_c \ x = 0 \rightarrow \mathsf{skip} \end{array}$$

This only describes the moves and the scheduling. Notions like winning and losing are modeled using postconditions to be established and are part of the analysis of the system.

Note that the initial number of sticks is left unspecified (x is not initialised).

Questions

Under what initial conditions can agent a (or b) win the game?:

- agent a wants to achieve $f \wedge x = 0$
- agent b wants to achieve $\neg f \wedge x = 0$.

We can also ask questions that are not directly related to winning or losing the game, such as "Will the game always terminate"?

Does the game always terminate?

Before considering winning or losing the game, we show that the game must always terminate, regardless of the initial number of sticks and of how the agents make their choices.

In this analysis, the angelic set is empty and infinite executions are bad, so the predicate transformer to be analysed is

$$\begin{aligned}
&(\langle f := \mathsf{T}\rangle \sqcap \langle f := \mathsf{F}\rangle)\ ;\\
&(\mu X \bullet \{f \land x > 0\}\ ;\ [x := x' \mid x' < x \le x' + 2]\ ;\ \langle f := \neg f\rangle\ ;\ X\\
&\qquad \sqcap \{\neg f \land x > 0\}\ ;\ [x := x' \mid x' < x \le x' + 2]\ ;\ \langle f := \neg f\rangle\ ;\ X\\
&\qquad \sqcap \{x = 0\})
\end{aligned}$$

This is equivalent to the always terminating loop program

$$\begin{aligned}
&(\langle f := \mathsf{T}\rangle \sqcap \langle f := \mathsf{F}\rangle)\ ;\\
&\mathsf{do}\ x > 0 \rightarrow \mathsf{if}\ f\ \mathsf{then}\ [x := x' \mid x' < x \le x' + 2]\ ;\ \langle f := \neg f\rangle\\
&\qquad\qquad\qquad\quad \mathsf{else}\ \ [x := x' \mid x' < x \le x' + 2]\ ;\ \langle f := \neg f\rangle\ \mathsf{fi}\ \ \mathsf{od}
\end{aligned}$$

Can agent a win the game?

We first show that in $Play$, agent a can win under the precondition

$$(f \wedge x \text{ mod } 3 \neq 1) \vee (\neg f \wedge x \text{ mod } 3 = 1)$$

regardless of how the scheduler works.

In this case, the scheduler is deterministic and the correctness property that we verify is

$$\begin{array}{l} (f \wedge x \text{ mod } 3 \neq 1) \vee (\neg f \wedge x \text{ mod } 3 = 1) \\ \{\!|\text{do } x > 0 \rightarrow \text{if } f \text{ then } [x := x' \mid x' < x \leq x' + 2]\,;\langle f := \neg f\rangle \\ \qquad\qquad\qquad\qquad \text{else } [x := x' \mid x' < x \leq x' + 2]\,;\langle f := \neg f\rangle \text{ fi od}|\!\} \\ f \wedge x = 0 \end{array}$$

Proof

The invariant used in the reasoning is $(f \wedge x \textsf{ mod } 3 \neq 1) \vee (\neg f \wedge x \textsf{ mod } 3 = 1)$, i.e., the same as the precondition.

The idea is that a can always make the state change from a situation where $x \textsf{ mod } 3 \neq 1$ to a situation where $x \textsf{ mod } 3 = 1$ while b must then re-establish $x \textsf{ mod } 3 \neq 1$.

Now it is easy to show that the initialisation always establishes the precondition if the scheduler is angelic but never if the scheduler is demonic.

The conclusion of this is that in the original game, we can always win if we are allowed to decide who should start (after we know how many sticks are in the pile).

Wold, goat and cabbage

A man comes to a river with a wolf, a goat and a sack of cabbages. He wants to get to the other side, using a boat that can only fit one of the three items (in addition to the man himself). He cannot leave the wolf and the goat on the same side (because the wolf eats the goat) and similarly, he cannot leave the goat and the cabbage on the same side.

The question: can the man get the wolf, the goat, and the cabbages safely to the other side.

We model the situation with one boolean variable for each participant, indicating whether they are on the right side of the river or not: m for the man, w for the wolf, g for the goat, and c for the cabbages.

The boat does not need a separate variable, because it is always on the same side as the man.

There is only one agent involved (the scheduler, who is in practice the man).

Contract

The contract that describes the situation is the following:

$$\begin{array}{rcl} \mathit{CrossRiver} & = & m, w, g, c := \mathsf{F}, \mathsf{F}, \mathsf{F}, \mathsf{F} \,;\, \mathit{Transport} \\ \mathit{Transport} & =_a & \quad m = w \rightarrow m, w := \neg m, \neg w \,;\, \mathit{Transport} \\ & & []_a \; m = g \rightarrow m, g := \neg m, \neg g \,;\, \mathit{Transport} \\ & & []_a \; m = c \rightarrow m, c := \neg m, \neg c \,;\, \mathit{Transport} \\ & & []_a \; m := \neg m \,;\, \mathit{Transport} \end{array}$$

The initialisation says that all four are on the wrong side, and each action corresponds to the man moving from one side of the river to the other, either alone or together with an item that was on the same side.

We do not model termination; even if the man gets all items safely to the other side of the river, he could then continue by taking things back again.

The fact that we want to achieve a situation where $m \wedge w \wedge g \wedge c$ holds will be part of the analysis, rather than the description.

Temporal property

We want to reach a situation where all four items are on the right side of the river, i.e., we want

$$m \wedge w \wedge g \wedge c$$

to be true at some point in the execution.

Furthermore, if the wolf and the goal are on the same side of the river, then the man must also be on that side. Thus, we want the property

$$(w = g \Rightarrow m = w) \wedge (g = c \Rightarrow m = g)$$

to be true at every point of the execution We thus want to prove that the agent (the man) can satisfy the following temporal property using the contract:

$$(w = g \Rightarrow m = w) \wedge (g = c \Rightarrow m = g) \;\; \mathcal{U} \;\; m \wedge w \wedge g \wedge c$$

Proof of enforcement

The simplest way to show this is to verify the following sequence of correctness steps

$$\begin{array}{l}
\mathsf{true} \\
\{\!|\ m, w, g, c := \mathsf{F}, \mathsf{F}, \mathsf{F}, \mathsf{F}\ |\!\} \\
\neg m \wedge \neg w \wedge \neg g \wedge \neg c \\
\{\!|\ A\ |\!\} \\
m \wedge \neg w \wedge g \wedge \neg c \\
\{\!|\ A\ |\!\} \\
\neg m \wedge \neg w \wedge g \wedge \neg c \\
\{\!|\ A\ |\!\} \\
m \wedge w \wedge g \wedge \neg c \\
\{\!|\ A\ |\!\} \\
\neg m \wedge w \wedge \neg g \wedge \neg c \\
\{\!|\ A\ |\!\} \\
m \wedge w \wedge \neg g \wedge c
\end{array}$$

$$\{\!|\ A\ |\!\}$$

$$\neg m \wedge w \wedge \neg g \wedge c$$

$$\{\!|\ A\ |\!\}$$

$$m \wedge w \wedge g \wedge c$$

and that each of the intermediate conditions imply $(w = g \Rightarrow m = w) \wedge (g = c \Rightarrow m = g)$.

Action definition

Here A stands for the action of the system, i.e.,

$$\begin{aligned}
&\{m = w\}\,;\, m, w := \neg m, \neg w \\
\sqcup\ &\{m = g\}\,;\, m, g := \neg m, \neg g \\
\sqcup\ &\{m = c\}\,;\, m, c := \neg m, \neg c \\
\sqcup\ &m := \neg m
\end{aligned}$$

The Dim Sum restaurant

Customers a, b, and c with x_0, x_1 and x_2 items, respectively. Servant d who can offer customer an item, but not to the same customer twice in a row (f indicates who got the last offer and r is the remaining number of items). The manager e who can decide to close the restaurant at any time (and who must close it when there are no items left).

$$\begin{aligned}
\mathit{Dim\ Sun} \quad &= \quad x_0, x_1, x_2, f := 0, 0, 0, 3 \,;\, \mathit{Serve} \\
\mathit{Serve} =_z \quad & \quad (\langle r > 0\rangle_e \,;\, (\ \langle f \neq 0\rangle_d \,;\, (\langle x_0, r := x_0 + 1, r - 1\rangle [\,]_a \mathsf{skip}) \,;\, \langle f := 0\rangle \,;\, \mathit{Serv} \\
& \qquad\qquad [\,]_d\ \langle f \neq 1\rangle_d \,;\, (\langle x_1, r := x_1 + 1, r - 1\rangle [\,]_b \mathsf{skip}) \,;\, \langle f := 1\rangle \,;\, \mathit{Serv} \\
& \qquad\qquad [\,]_d\ \langle f \neq 2\rangle_d \,;\, (\langle x_2, r := x_2 + 1, r - 1\rangle [\,]_c \mathsf{skip}) \,;\, \langle f := 2\rangle \,;\, \mathit{Serv} \\
& \quad [\,]_e\ \mathsf{skip}\ )
\end{aligned}$$

With this setup we can prove that with the help of the servant, a customer can get at least half of the items that have been taken:

$$x_0 = 0 \wedge x_1 = 0 \wedge x_2 = 0 \wedge f = 3 \ \{\!|\, \mathcal{A} \,|\!\}_{\{a,d\}}\ \Box(x_0 \geq x_1 + x_2)$$

This proof goes through with invariant $(f = 0 \wedge x_0 > x_1 + x_2) \vee (f \neq 0 \wedge x_0 \geq x_1 + x_2)$.

Similarly, we can prove that two cooperating customers can get almost half of the items, provided that the manager helps by not closing too early:

$$x_0 = 0 \land x_1 = 0 \land x_2 = 0 \land f = 3 \;\{\!|\, \mathcal{A} \,|\!\}_{\{a,b,e\}}\; \Delta(r = 0 \land x_0 + x_1 \geq x_2 - 1)$$