Part II

A. Basic theory of contracts

B. Application: UML use cases

C. Extension: temporal reasoning with contracts

Presentation is based on article:

R. J. R. Back and J. von Wright, *Enforcing Temporal Properties in Contracts*. In C.Morgan and A. McIver (eds.): *Programming Methodology*, forthcoming.

Enforcing behavioral properties

Example contract

Consider the contract:

$$\begin{array}{rcl} S & = & (x := x + 1 \ \sqcup_a \ x := x + 2)\ ; \\ & & (x := x - 1 \ \sqcup_b \ x := x - 2) \end{array}$$

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:

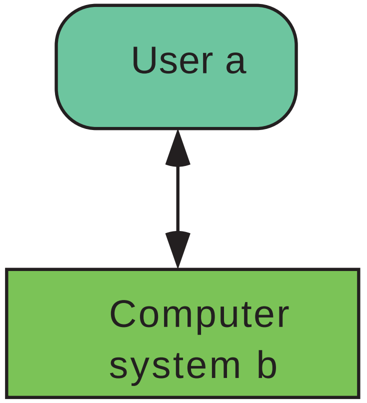
$$x = 0 \ \{\!|\ S\ |\!\}_a \ \Box(0 \leq x \leq 2)$$

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, and then determine whether the property holds or not.

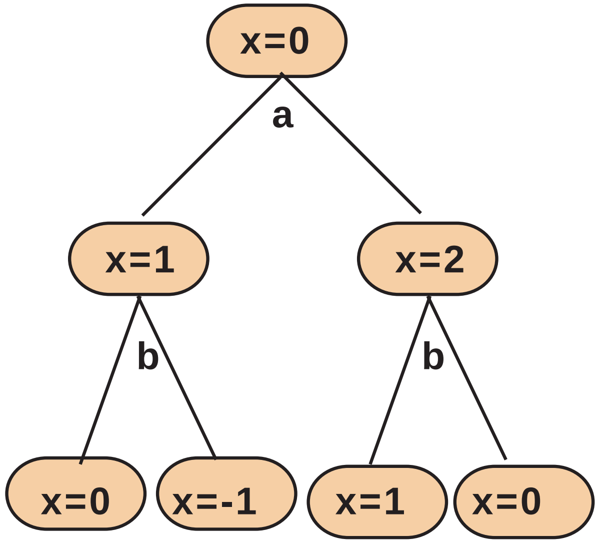
Enforcing temporal properties



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)



Implicit agents

We postulate that the the set Ω of agents always contains two distinguished agents, *angel* and *demon*.

Any coalition of agents A from Ω must be such that $\mathit{angel} \in A$ and $\mathit{demon} \notin A$. Then, define

$$\begin{array}{rcl}
\mathsf{abort} & = & \mathsf{abort}_{\mathit{angel}} \\
\mathsf{magic} & = & \mathsf{abort}_{\mathit{demon}} \\
\{R\} & = & \{R\}_{\mathit{angel}} \\
[R] & = & \{R\}_{\mathit{demon}} \\
\sqcup & = & \sqcup_{\mathit{angel}} \\
\sqcap & = & \sqcap_{\mathit{demon}} \\
\mu & = & \mathsf{rec}_{\mathit{angel}} \\
\nu & = & \mathsf{rec}_{\mathit{demon}}
\end{array}$$

In addition, with $\sigma \; \{\!|\; S \;|\!\} \; q$, we will mean $\sigma \; \{\!|\; S \;|\!\}_{\mathit{angel}} \; q$.
This convention means that we can use predicate transformer notation directly in contracts.

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 $A \in \Omega$.

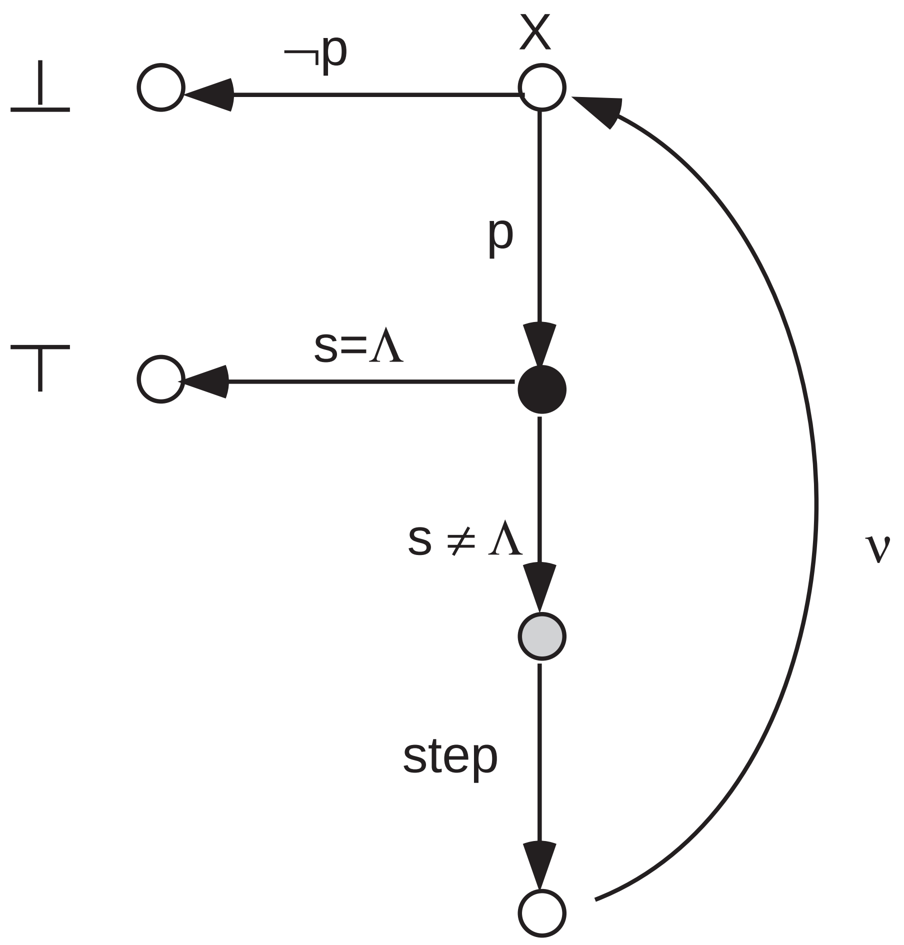
Define the the contract $\mathsf{Always}.\,p$, called a *tester for $\Box p$*, by

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

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

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

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

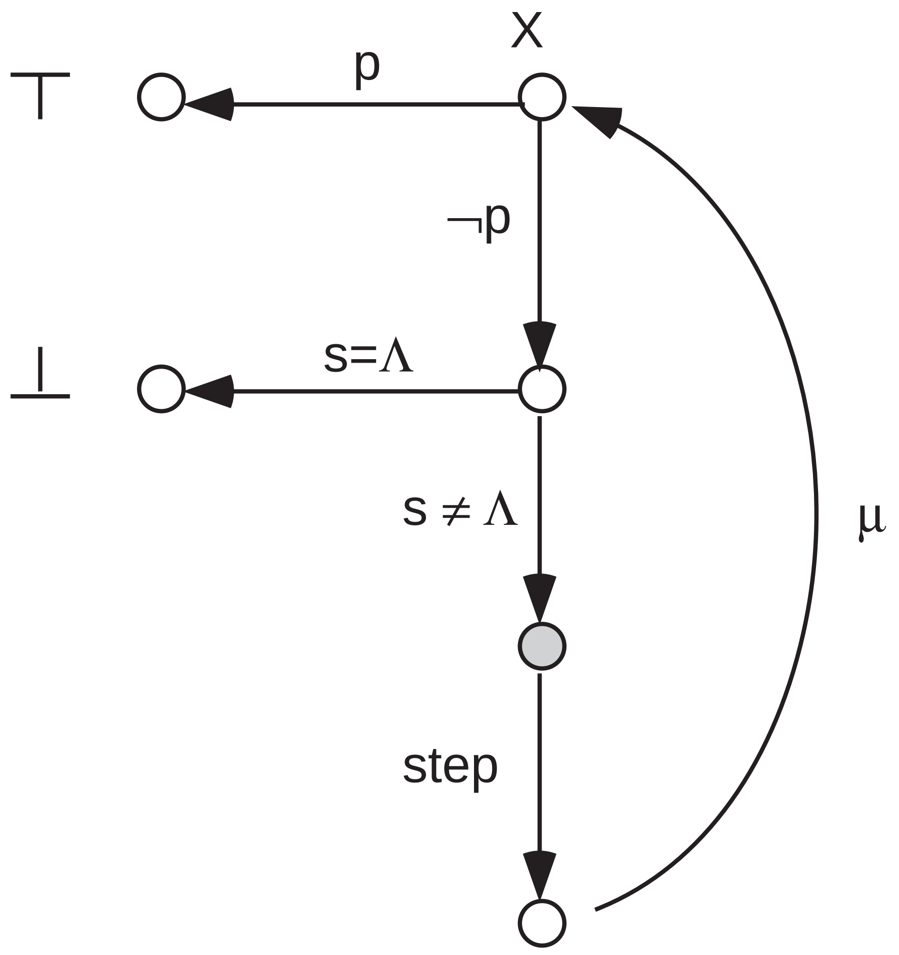
In a similar way, we can define a tester $\mathsf{Eventually}.\,p$ for the property $\Diamond p$, by

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

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

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

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

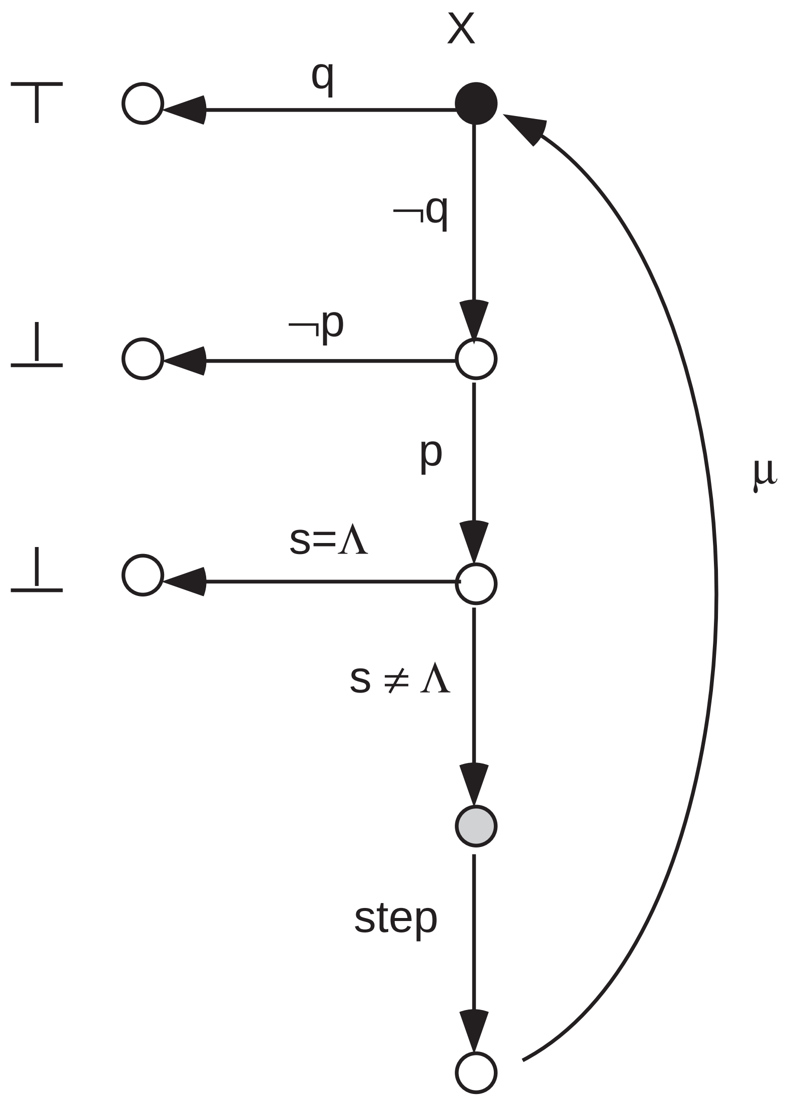
$$\tau\ \{\!|\ (\mu X \bullet [\neg q]\ ; \{p\}\ ; \{s \neq \Lambda\}\ ; \mathrm{step}\ ; X)\ |\!\}\ \mathsf{false}$$

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

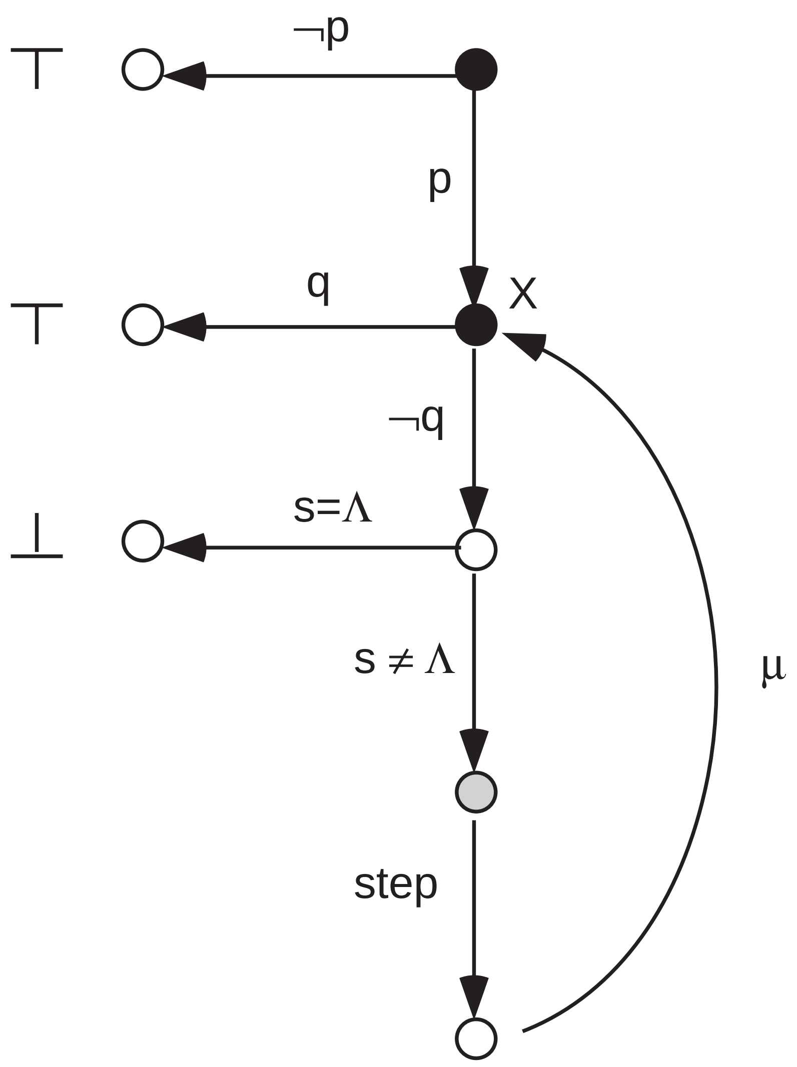
$$\sigma_0 \ \{\!| \, S \, |\!\} \ p \mapsto q$$

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

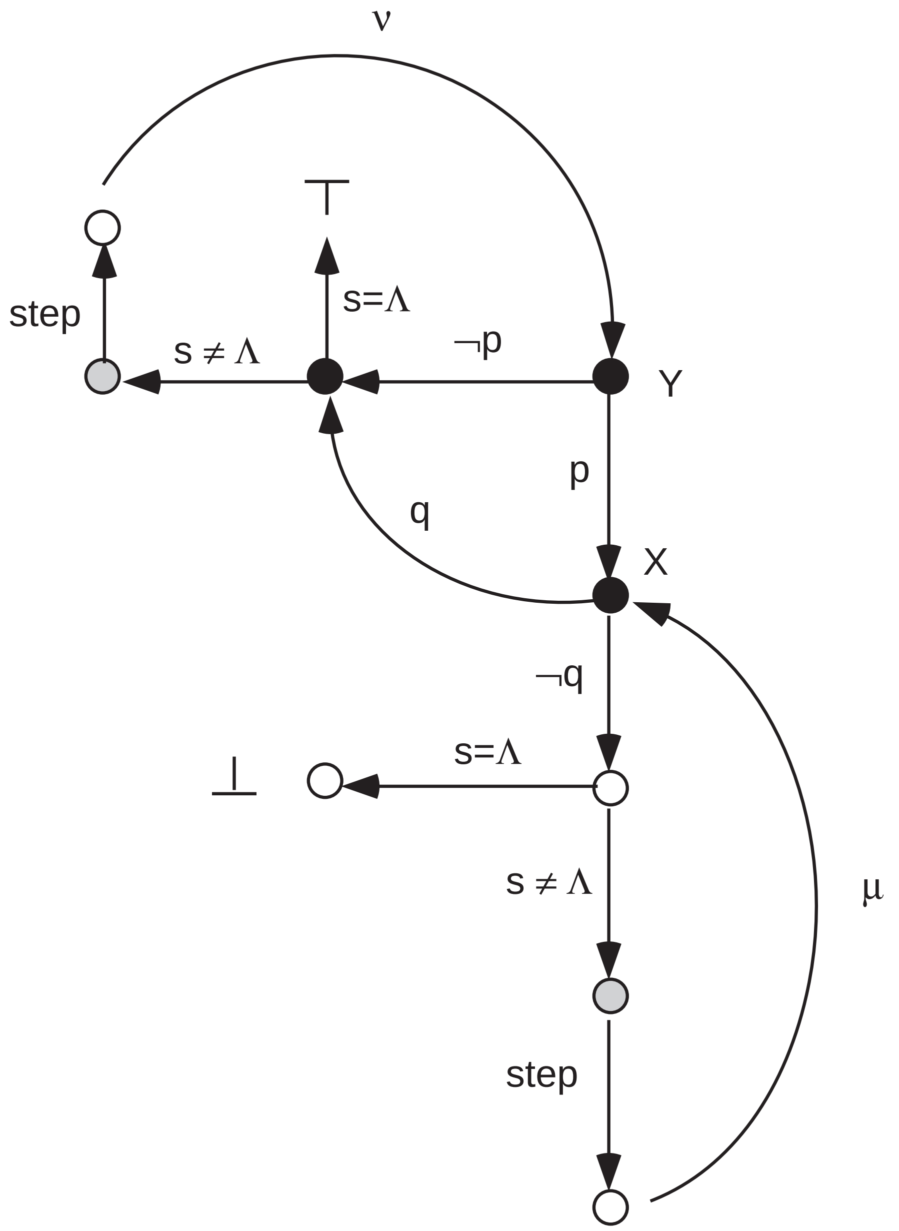
$$\sigma_0 \; \{\!| \; S \; |\!\} \; \Box(p \mapsto q)$$

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.

Modeling concurrent and interactive systems

Action loop

An *action loop* is of the form

$$(\mathsf{rec}_c\ X \bullet \langle S \rangle \,;\, X \ \sqcup_a \ \{g\}_a)$$

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,

$$S \quad = \quad S_1 \ \sqcup_b \ S_2 \ \sqcup_b \ \ldots \ \sqcup_b \ S_n$$

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

Thus, an action system is in general of the form

$$B \,;\, (\mathsf{rec}_c \ X \bullet (S_1 \ \sqcup_b \ S_2 \ \sqcup_b \ \ldots \ \sqcup_b \ S_n)\,;\, X \ \sqcup_a \ \{g\}_a)\,;\, E$$

Action system

B ;

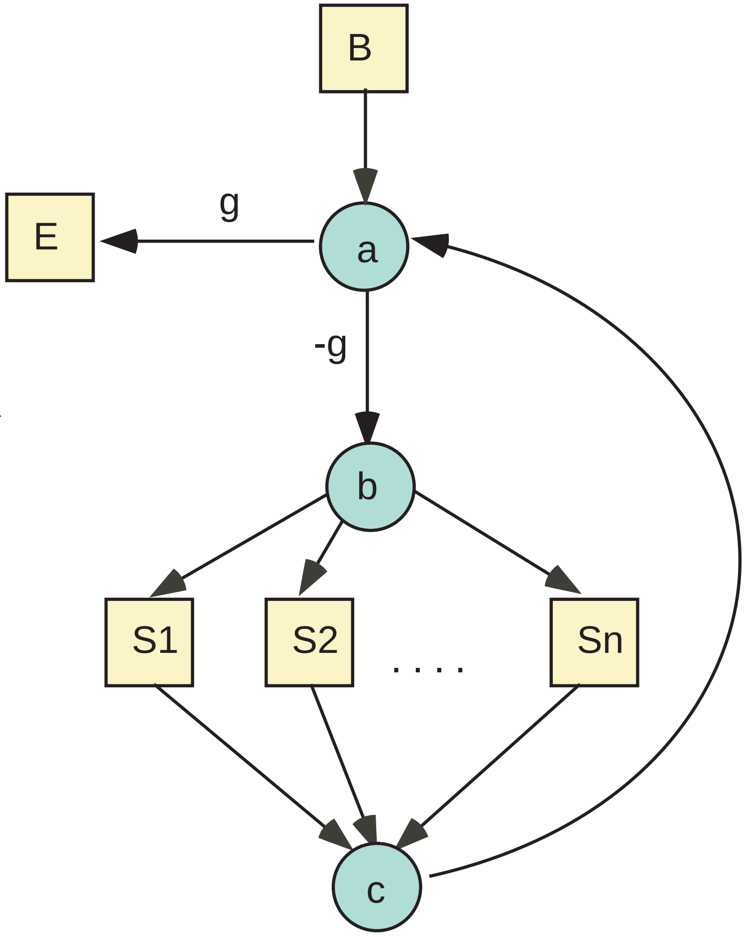
$(\mathsf{rec}_c\ X \bullet$

$\quad (S_1\ \sqcup_b\ S_2\ \sqcup_b\ \ldots\ \sqcup_b\ S_n)\ ;\ X$

$\ \sqcup_a\ \{g\}_a$

$)$;

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 .
- 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.

Extension of traditional models for temporal behavior

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 either a *failed* or a *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 systems*, 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 system 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.

Atomic contracts

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.

An *atomic* contract statement is denoted $\langle S \rangle$:

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

Modifying the tester

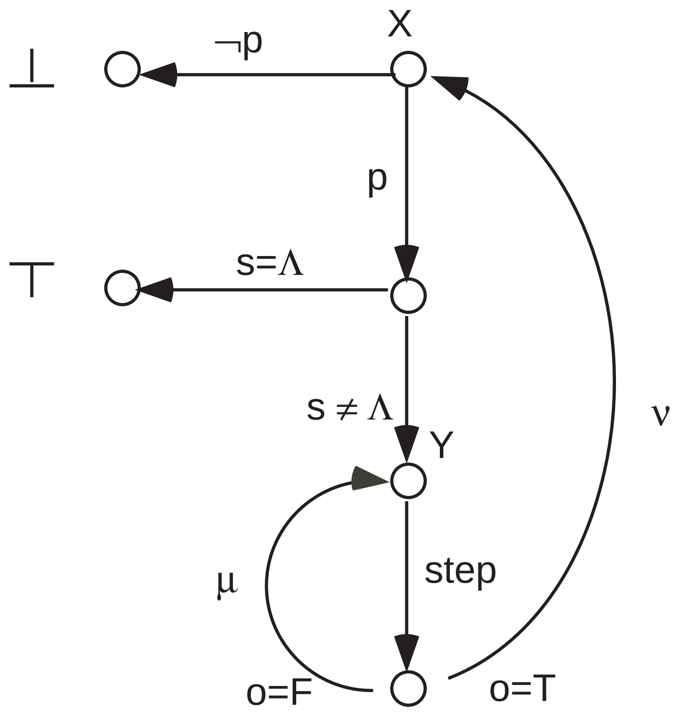
We need to modify the tester, to take the hidden states into account. The modified tester for the property $\Box p$ is as follows:

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

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

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

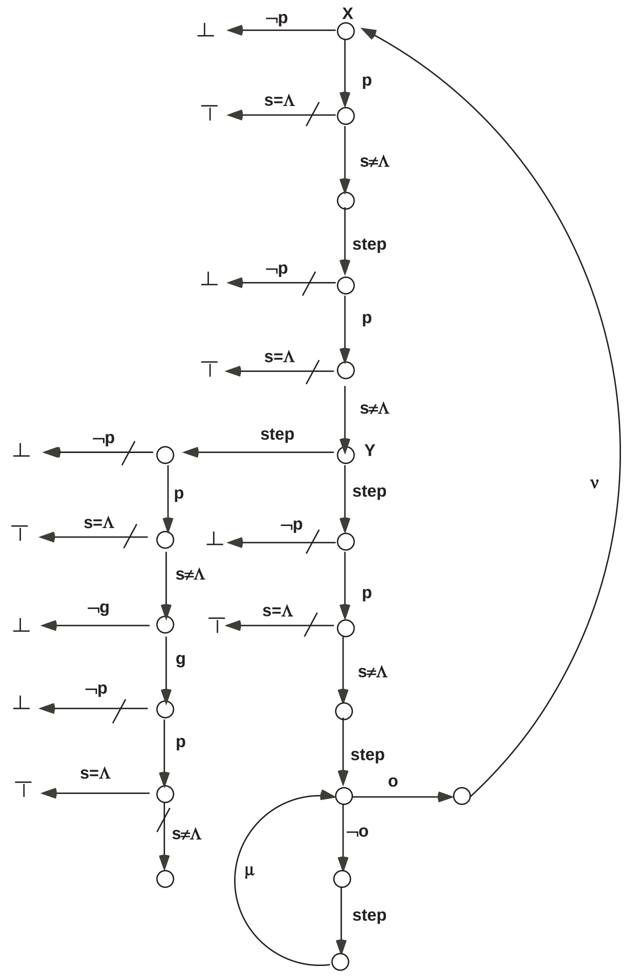
assuming that $a \in A$ and that $c \notin A$. We have that

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

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



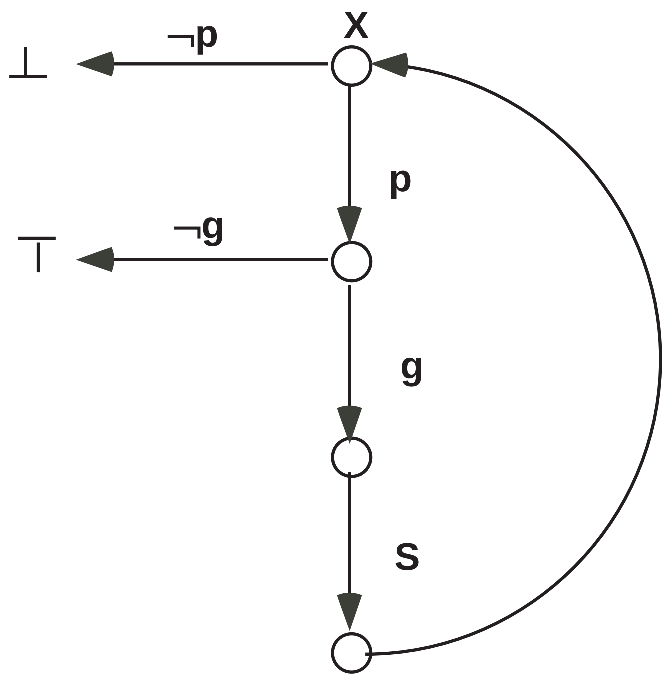
Explanation

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

Wolf, 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 he cannot leave the goat and the cabbage on the same side (because the goat eats the cabbage).

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:

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

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 goat 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 enforce 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

true

$\{\!|\ m, w, g, c := \mathsf{F}, \mathsf{F}, \mathsf{F}, \mathsf{F}\ |\!\}$

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

$\{\!|\ A\ |\!\}$ (goat over to other side)

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

$\{\!|\ A\ |\!\}$ (man back to this side)

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

$\{\!|\ A\ |\!\}$ (wolf to other side)

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

$\{\!|\ A\ |\!\}$ (goat back to this side)

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

$\{\!|\ A\ |\!\}$ (cabbage over to other side)

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

$\{\!|\ A\ |\!\}$ (man back to this side)

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

$\{\!|\ A\ |\!\}$ (goat over to other side)

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

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

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 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 can decide to close the restaurant at any time (and must close it when there are no items left).

$$
\begin{array}{rcl}
DS & = & x_0, x_1, x_2, f := 0, 0, 0, 3 \,;\, X \\
X & =_z & (\{r > 0\}_e \,;\, (\ \{f \neq 0\}_d \,;\, (\{x_0, r := x_0 + 1, r - 1\} \sqcup_a \mathsf{skip}) \,;\, \langle f := 0\rangle \,;\, X \\
 & & \qquad\qquad \sqcup_d \ \{f \neq 1\}_d \,;\, (\{x_1, r := x_1 + 1, r - 1\} \sqcup_b \mathsf{skip}) \,;\, \langle f := 1\rangle \,;\, X \\
 & & \qquad\qquad \sqcup_d \ \{f \neq 2\}_d \,;\, (\{x_2, r := x_2 + 1, r - 1\} \sqcup_c \mathsf{skip}) \,;\, \langle f := 2\rangle \,;\, X\ ) \\
 & & \sqcup_e \ \mathsf{skip}\ )
\end{array}
$$

Analyzing the restaurant

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

$$
\begin{aligned}
&x_0 = 0 \wedge x_1 = 0 \wedge x_2 = 0 \wedge f = 3\\
&\{\!|qA|\!\}_{\{a,d\}}\\
&\Box(x_0 \geq x_1 + x_2
\end{aligned}
$$

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

$$
\begin{aligned}
&x_0 = 0 \wedge x_1 = 0 \wedge x_2 = 0 \wedge f = 3\\
&\{\!|qA|\!\}_{\{a,b,e\}}\\
&\Delta(r = 0 \wedge x_0 + x_1 \geq x_2 - 1)
\end{aligned}
$$

Here Δp says that p holds in the end.