Refinement Calculus Foundations and Applications

Ralph-Johan Back Joakim von Wright

Turku Centre for Computer Science and
Åbo Akademi University

Summer School on Specification, Refinement and Verification, Turku 2002

Part II

A. Basic theory of contracts

B. Application: UML use cases

C. Extension: temporal reasoning with contracts

Presentation is based on the book

> R. J. R. Back and J. von Wright, *Refinement Calculus:A Systematic Introduction*. Graduate Texts in Computer Science, Springer-Verlag, New York 1998 (519 pages), ISBN 0-387-98417-8.

and on the article

> R. J. R. Back and J. von Wright, Contracts as Mathematical Entities in Programming Logic. In *Proc. Workshop on Abstraction and Refinement*, Osaka 1999, Elsevier. (Also available as TUCS Tech report 372).

States, agents and contracts

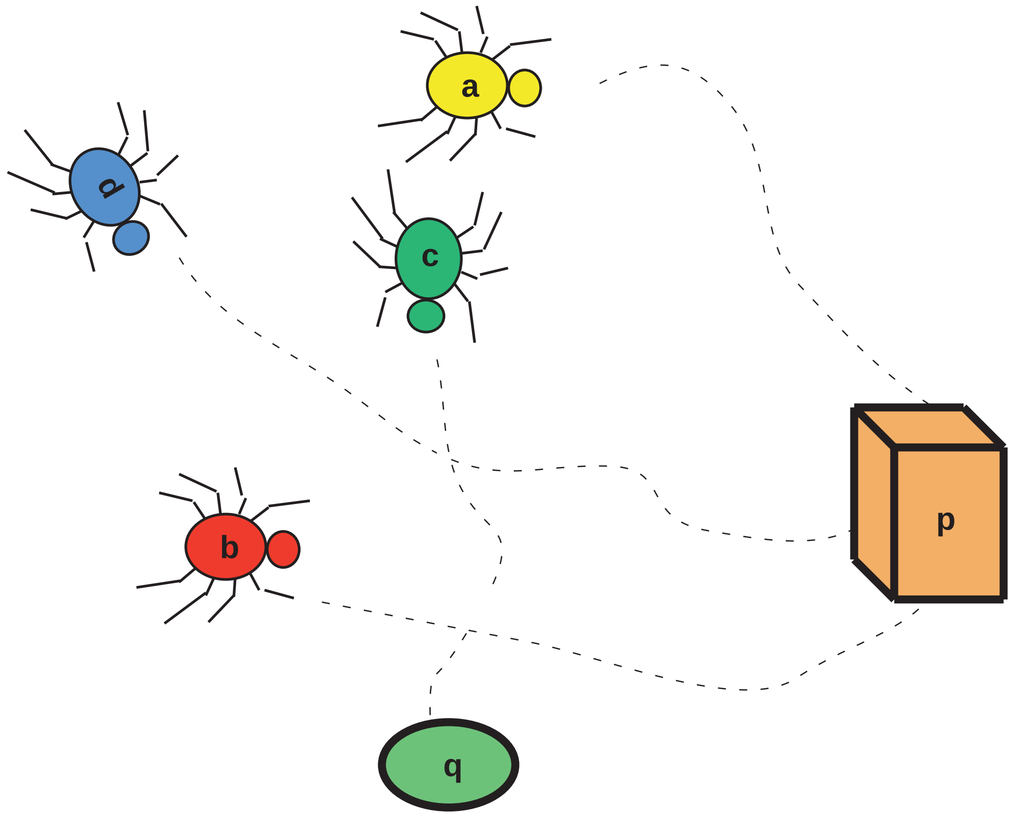
Consider

- a collection of *agents* $a, b, c, d, \ldots$
- that operate in a *world* Σ
- in order to achieve their respective *goals* $p, q, r, \ldots$.

The interaction between agents is regulated by *contracts*.

A contract stipulates what agents are permitted and expected to do.

We want to analyze what an agent can *achieve* with a given contract.



Contracts (I)

Contract statements

The behavior of agents is regulated by a *contract (statement)* S. A contract is of the form

$$S \quad ::= \quad \langle f \rangle \mid \{p\}_a \mid S_1; S_2 \mid S_1 \sqcup_a S_2$$

where p is state predicate and f state transformer.

- The *update* $\langle f \rangle$ changes the state by applying the state transformer f. If the initial state is σ_0 then the final state is $f.\sigma_0$.

  An *assignment statement* $\langle x := e \rangle$ is a special kind of update where the state transformer is an assignment. The *identity statement* $\mathsf{skip} = \langle \mathsf{id} \rangle$ does not change the state at all.

- In the *sequential contract* $S_1; S_2$ the contract S_1 is first carried out, followed by S_2.

- In a *choice* $S_1 \sqcup_a S_2$, either contract S_1 or S_2 is carried out, depending on which one agent a chooses.

Sequential composition binds stronger than choice.

Assertions

The *assertion* $\{p\}_a$ is a requirement that the agent must satisfy in a given state.

Assertions can be expressed using boolean expressions. Assertion

$$\{x + y = 0\}_a$$

states that the sum of x and y in the state must be zero.

- If the assertion holds then the state is unchanged, and the agent carries on with the rest of the contract.
- If the assertion does not hold, then agent a has *breached* the contract.

The assertion $\{\mathsf{true}\}_a$ is always satisfied.

The assertion $\{\mathsf{false}\}_a$ is an *impossible assertion*. It is never satisfied, and always forces the agent to breach the contract.

Example contract

Consider the contract

$$\begin{aligned} \text{Contract1} \quad = \quad & \{1 \le y \le 4\}_a ; \langle x := 0 \rangle ; \\ & (\langle x := x + 1 \rangle \sqcup_a \langle x := x + 2 \rangle) ; \\ & \{y = x\}_a \end{aligned}$$

- If $y < 1$ or $y > 4$, then agent a must breach the contract.
- If $y = 1$, then agent a can avoid breaching the contract, by choosing the left alternative
- If $y = 2$, then agent a can avoid breaching the contract by choosing the right alternative
- If $y = 3, 4$, then agent a cannot avoid breaching the contract.

We write just $x := x + 1$ for assignment statements in contracts, rather than $\langle x := x + 1 \rangle$, when no confusion can occur.

Contract with two agents

The contract of agent a invokes subcontract for agent b.

Agent a is to carry out the contract S:

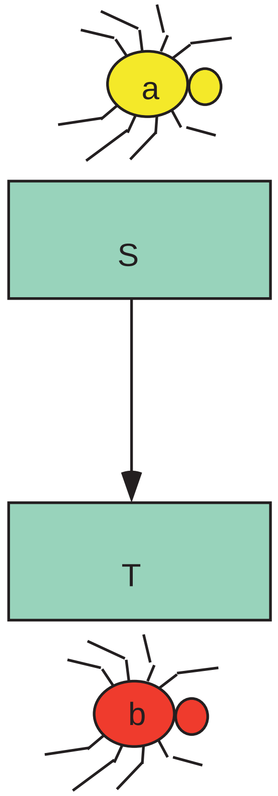
$$S \quad = \quad x := 0; (T \sqcup_a x := x + 1); \{y = x\}_a$$

Contract T is to be carried out by another agent b:

$$T \quad = \quad y := 0 \sqcup_b y := 1$$

The overall contract is

$$\begin{aligned} \text{Contract2} \quad = \quad & x := 0; \\ & ((y := 0 \sqcup_b y := 1) \sqcup_a x := x + 1); \\ & \{y = x\}_a \end{aligned}$$



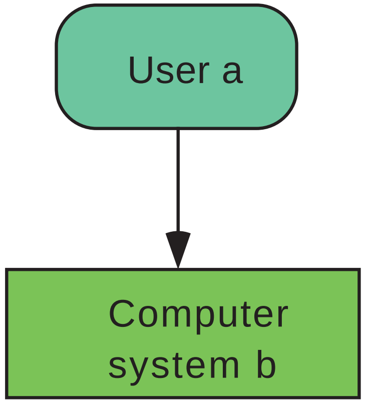
Programs

Traditional programs can be seen as special kinds of contracts, where exactly two agents are involved:

- the *user* a, and
- the *computer system* b.

Example program:

$$x := x + 1;$$
$$\{x \neq 0\}_a; y := y/x;$$
$$y := y + 1$$



User breaks contract if she attempts to do division by zero, releasing system from its obligations to satisfy the contract.

Abort statement is a total breach of contract:

$$\mathsf{abort} = \{\mathsf{false}\}_a$$

Concurrent system

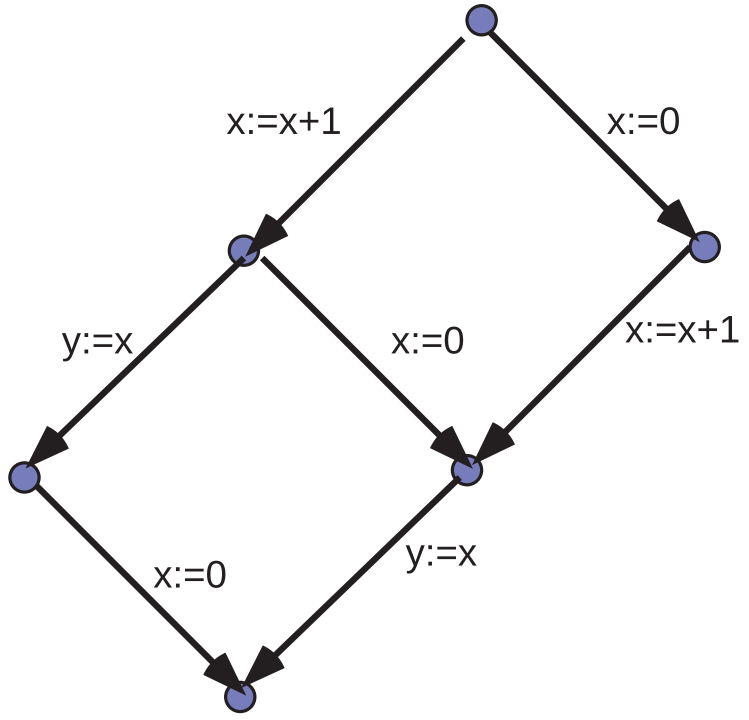
Consider the parallel composition

$$(x := x + 1; y := x) \mid\mid x := 0$$

Can be interpreted as the contract

$$\begin{aligned} &x := x + 1; y := x; x := 0 \quad \sqcup_b \\ &x := x + 1; x := 0; y := x \quad \sqcup_b \\ &x := 0; x := x + 1; y := x \end{aligned}$$

Scheduling is determined by the computer system b. It resolves internal choices in a manner that the user cannot influence or know (*demonic nondetermism*).



Interactive system

User computes a value for x by a sequence of function applications:

$$\text{Compute} \quad = \quad \text{Apply}; \text{Apply}; \ldots; \text{Apply}$$

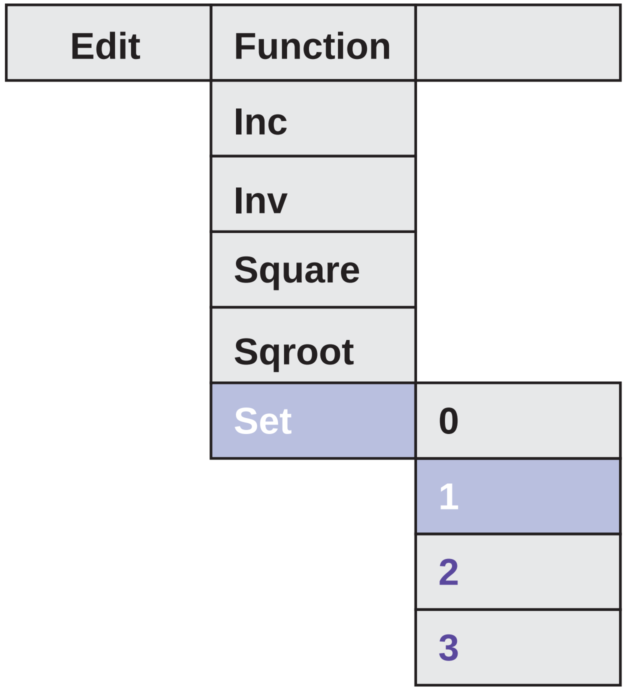
Each function applies one possible change to x:

$$\begin{aligned} \text{Apply} \quad = \quad & \mathsf{skip} \sqcup_a \text{Inc} \sqcup_a \text{Inv} \sqcup_a \\ & \text{Square} \sqcup_a \text{Sqroot} \sqcup_a \text{Set} \end{aligned}$$

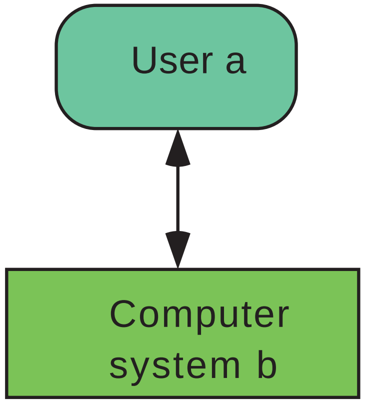
where

$$\begin{aligned} \text{Inc} \quad &= \quad x := x + 1 \\ \text{Inv} \quad &= \quad \{x \neq 0\}_a; x := 1/x \\ \text{Square} \quad &= \quad x := x * x \\ \text{Sqroot} \quad &= \quad \{x \geq 0\}_a; x := \sqrt{x} \\ \text{Set} \quad &= \quad x := 0 \sqcup_a x := 1 \sqcup_a \ldots \sqcup_a x := 9 \end{aligned}$$

User alternative can be seen as *menu choices*. Alternative skip is chosen if no menu item is selected.

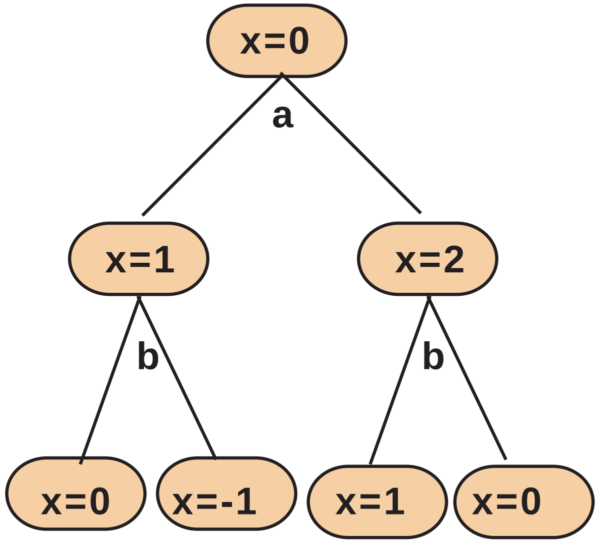


Both user and system choices



Can allow both user choices (*angelic choices*) and system choices (*demonic choices*) in the same contract.

$$\begin{array}{ll} & \text{Interaction} \\ = & x := 0; \\ & (x := x + 1 \sqcup_a x := x + 2); \\ & (x := x - 1 \sqcup_b x := x - 2) \end{array}$$



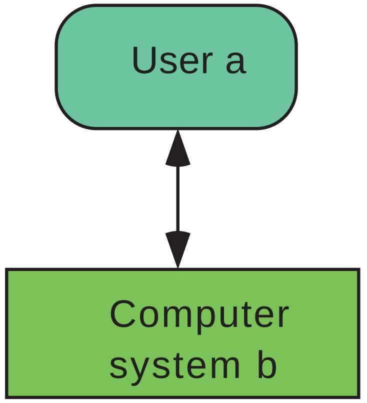
The user a chooses between alternatives in order to influence the computation. User should choose

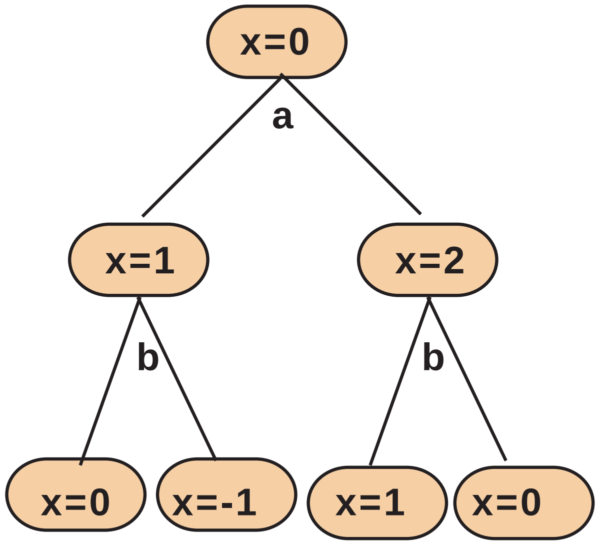
- first alternative, if she wants to establis $x \leq 0$.
- second alternative, if she wants to establish $x \geq 0$.

Interchanging user and system

Can also regard a to be the system and b the user. Then the user choice is done after the system has made its choice.

- User can choose to establish $x = 0$, no matter what system does.
- User can also choose to establish $x \neq 0$, no matter what system does.





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 of contracts

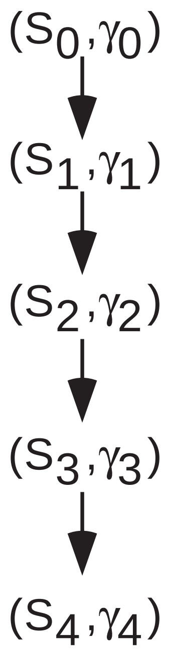
We give a formal meaning to contract statements using *structured operational semantics*. This describes step by step how a contract is carried out, starting from a given initial state.

A *configuration* is a pair (S, γ), where

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

Intuitively: S denotes what remains to be done, γ is present state.

The *transition relations* $\rightarrow$ show what moves are permitted. It is the smallest relation which satisfies the given axioms and inference rules.



Transition rules

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

Assertion
$$\frac{p.\,\sigma}{(\{p\}_a,\sigma)\to(\Lambda,\sigma)}\qquad \frac{\neg p.\,\sigma}{(\{p\}_a,\sigma)\to(\Lambda,\perp_a)}$$

$$\frac{}{(\{p\}_a,\perp_b)\to(\Lambda,\perp_b)}$$

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

Choice
$$\frac{}{(S_1\sqcup_a S_2,\gamma)\to(S_1,\gamma)}\qquad \frac{}{(S_1\sqcup_a S_2,\gamma)\to(S_2,\gamma)}$$

σ stands for a proper state
γ stands for a proper state or $\perp_a$.

Example derivation

$$
\begin{array}{ll}
 & (x := 0; ((y := 1 \sqcup_b y := 2) \sqcup_a x := x + 1); \{y = x\}_a,\ \ (x = 1, y = 1)) \\
\longrightarrow & \{\text{sequential composition rule}\} \\
 & \qquad\qquad \begin{array}{ll} & (x := 0,\ \ (x = 1, y = 1)) \\ \longrightarrow & \{\text{update rule}\} \\ & (\Lambda,\ \ (x = 0, y = 1)) \end{array} \\
 & (((y := 1 \sqcup_b y := 2) \sqcup_a x := x + 1); \{y = x\}_a,\ \ (x = 0, y = 1)) \\
\longrightarrow & \{\text{sequential composition rule}\} \\
 & \qquad\qquad \begin{array}{ll} & (((y := 1 \sqcup_b y := 2) \sqcup_a x := x + 1),\ \ (x = 0, y = 1)) \\ \longrightarrow & \{\text{choice rule}\} \\ & (x := x + 1,\ \ (x = 0, y = 1)) \end{array} \\
 & (x := x + 1; \{y = x\}_a,\ \ (x = 0, y = 1)) \\
\longrightarrow & \{\text{sequential composition rule}\} \\
 & \qquad\qquad \begin{array}{ll} & (x := x + 1,\ \ (x = 0, y = 1)) \\ \longrightarrow & \{\text{update rule}\} \\ & (\Lambda,\ \ (x = 1, y = 1)) \end{array} \\
 & (\{y = x\}_a,\ \ (x = 1, y = 1)) \\
\longrightarrow & \{\text{assertion rule}\} \\
 & (\Lambda,\ \ (x = 1, y = 1))
\end{array}
$$

All possible derivations

$$\text{Contract2} = A; ((B1 \sqcup_b B2) \sqcup_a C); D$$

where

$$\begin{array}{rcl} A & = & x; = 0 \\ B1 & = & y := 0 \\ B2 & = & y := 1 \\ C & = & x := x + 1 \\ D & = & \{y = x\}_a \end{array}$$

$$\begin{tikzcd}
 & & & \text{(Contract2, (x=1,y=1))} \arrow[d, "\text{x:= 0}"] & \\
 & & & \text{(((B1} \sqcup_b \text{B2)} \sqcup_a \text{C);D, (x=0,y=1))} \arrow[dll, "\text{a}"] \arrow[dr] & \\
 & \text{((B1} \sqcup_b \text{B2);D, (x=0,y=1))} \arrow[dl, "\text{b}"] \arrow[dr] & & & \text{(C;D, (x=0,y=1))} \arrow[d, "\text{x:= x+1}"] \\
\text{(B1;D, (x=0,y=1))} \arrow[d, "\text{y:=0}"] & & \text{(B2;D, (x=0,y=1))} \arrow[d, "\text{y:=1}"] & & \text{(D, (x=1,y=1))} \arrow[d, "\{x=y\}_a"] \\
\text{(D, (x=0,y=0))} \arrow[d, "\{x=y\}_a"] & & \text{(D, (x=0,y=1))} \arrow[d, "\{x=y\}_a"] & & (\Lambda, \text{(x=1,y=1))} \\
(\Lambda, \text{(x=0,y=0))} & & \perp_a & &
\end{tikzcd}$$

Operational semantics

A *behavior* of contract S from initial state σ is a maximal sequence of configurations ($S = S_0, \sigma = \sigma_0$):

$$(S_0, \sigma_0) \to (S_1, \sigma_1) \to \ldots \to (S_n, \sigma_n)$$

where each transition $(S_i, \sigma_i) \to (S_{i+1}, \sigma_{i+1})$ is permitted by the axiomatization.

The *operational semantics* of a contract S is a function

$$\mathsf{op} : \text{Contracts} \to \Sigma \to \mathcal{P}(\text{Behaviors})$$

where

$$\mathsf{op}.\, S.\, \sigma \quad \triangleq \quad \text{set of behaviors of } S \text{ from } \sigma$$

All behaviors of contracts are (here) finite.

Statement part of configuration always indicates which agent should choose next, if any.

Contracts (II)

Event loop with input statement

Define computation with *recursion*:

$$\text{Compute} \quad = \quad (\text{rec}_a\ X \bullet \text{Apply}; X \sqcup_a \mathsf{skip})$$

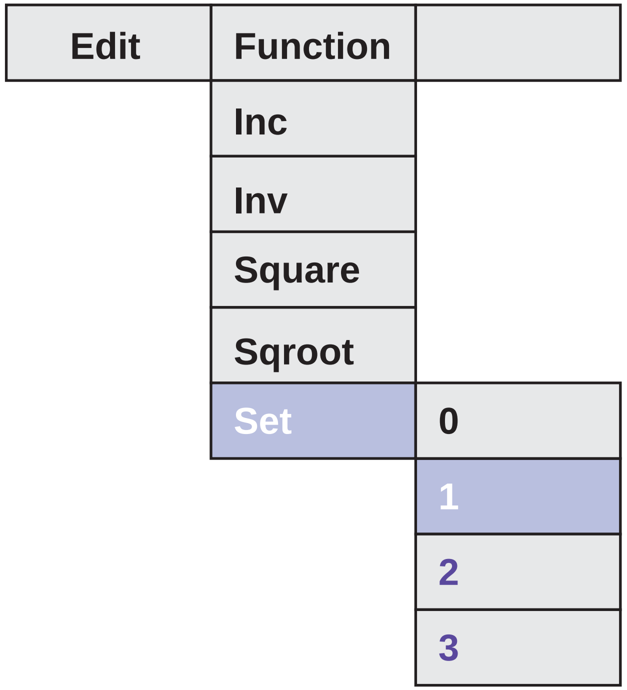
User a may at each stage choose between applying a new function or terminating.

Define setting of a value for x as a *relational update*:

$$\text{Set} \quad = \quad \{x := x' : \mathsf{Nat} \mid 0 \le x' \le 9\}_a$$

Attribute x is assigned a new value x' that satifies condition $0 \le x' \le 9$. Effect is same as in

$$\text{Set} \quad = \quad x := 0 \sqcup_a x := 1 \sqcup_a \ldots \sqcup_a x := 9$$



Relational assignment

We generalize ordinary (functional) assignment to *relational assignment*. The relation

$$(x := x' \mid x' > x + y)$$

relates state σ to state σ' if

> the value of x in σ' is greater than the sum of the values of x and y in σ *and all other attributes are unchanged*:

Thus

$$\begin{array}{ll} & (x := x' \mid x' > x + y).\,\sigma.\,\sigma' \\ \equiv & \\ & (\exists\, x' \bullet \sigma' = \mathit{set}x.\,x'.\,\sigma \;\wedge\; x' > \mathit{val}x.\,\sigma + \mathit{val}y.\,\sigma) \end{array}$$

Define in general

$$(x := x' \mid b).\,\sigma.\,\sigma' \quad \triangleq \quad (\exists\, x' \bullet \sigma' = \mathit{set}x.\,x'.\,\sigma \;\wedge b.\,\sigma)$$

Relational update

Let R be a state relation. The *relational update*

$$\{R\}_a$$

permits an agent to choose any final state related by R to the initial state.

If no such final state exists, then the agent breaches the contract.

Example:

$$\{x := x' \mid 0 \leq x' < x\}_a =$$

"change the state so that the new value x' satisfies $0 \leq x' < x$, without changing the values of the other attributes."

x	effect
0	abort
1	$x := 0$
2	$x := 0 \sqcup_a x := 1$

Arbitrary choice

Finite choice is generalized to *arbitrary choice*:

$$(\sqcup_a\ i \in I \bullet S_i)$$

Agent a chooses a statement from the set $\{S_i \mid i \in I\}$.

Index set I may be infinite.

If I is empty, then the agent breaches the contract.

Example:

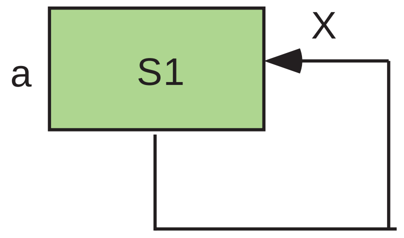
$$(\sqcup_a i \in \mathsf{Nat} \bullet x := x + i) \quad = \quad \{x := x' \mid x' \geq x\}_a$$

Recursion

Permit also *recursive* contract statements:

$$S \quad ::= \quad \ldots \mid X \mid (\mathsf{rec}_a X \bullet S_1)$$

- X is a variable that ranges over contract statements
- $(\mathsf{rec}_a X \bullet S_1)$ is the contract statement S_1 where each occurrence of X in S_1 is interpreted as a recursive invocation of the contract $(\mathsf{rec}_a X \bullet S_1)$
- Agent a breaches the contract if the recursion does not terminate
- Operational semantics has to be extended with infinite behavior



Transition axioms

- *Relational update*

$$\frac{R.\,\sigma.\,\sigma'}{(\{R\}_a, \sigma) \longrightarrow (\Lambda, \sigma')} \qquad \frac{R.\,\sigma = \emptyset}{(\{R\}_a, \sigma) \longrightarrow (\Lambda, \perp_a)} \qquad \frac{}{(\{R\}_a, \perp_b) \longrightarrow (\Lambda, \perp_b)}$$

- *Arbitrary choice*

$$\frac{k \in I}{((\sqcup_a\ i \in I \bullet S_i), \gamma) \longrightarrow (S_k, \gamma)} \qquad \frac{I = \emptyset}{((\sqcup_a\ i \in I \bullet S_i), \gamma) \longrightarrow (\Lambda, \perp_a)}$$

- *Recursion*

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

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

- $C_0 = (S, \sigma)$,
- each transition $C_i \to C_{i+1}$ is permitted by the axiomatization above, and
- 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.

Iteration

While statement:

$$\mathsf{while}\ {}_a\ g\ \mathsf{do}\ \ S\ \ \mathsf{od}$$

$$=$$

$$(\mathsf{rec}_a X \bullet\ \ \{g\}_a; S; X \sqcup_a \{\neg g\}_a)$$

Iteration with implicit exit (a and b can be same agent)

$$\mathsf{do}_a\ g_1 \rightarrow S_1 \sqcup_b \ldots \sqcup_b g_m \rightarrow S_m\ \ \mathsf{od}$$

$$=$$

$$(\mathsf{rec}_a X \bullet \{g_1\}_b; S_1; X \sqcup_b \ldots \sqcup_b \{g_m\}_b; S_m; X \sqcup_b \{\neg g_1 \cap \ldots \cap \neg g_m\}_b)$$

Iteration with explicit exit

$$\mathsf{do}_a\ g_1 \rightarrow S_1 \sqcup_b \ldots \sqcup_b g_m \rightarrow S_m \sqcup_b g_{m+1} \rightarrow \mathsf{exit}\ \ \mathsf{od}$$

$$=$$

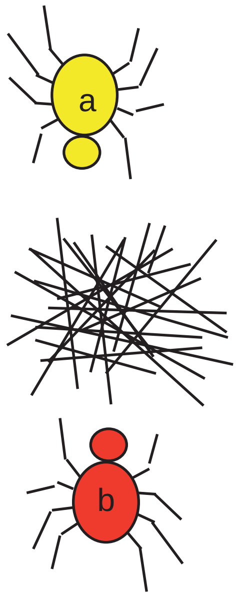
$$(\mathsf{rec}_a X \bullet \{g_1\}_b; S_1; X \sqcup_b \ldots \sqcup_b \{g_m\}_b; S_m; X \sqcup_b \{g_{m+1}\}_b)$$

Playing games: Nim

Players a and b take turns to remove either one or two sticks from a pile. The player who takes the last stick has lost. Player a starts.

1. First check whether b already has lost (if no matches in pile, then b must breach the contract).
2. Otherwise, player a removes one or two sticks from the pile.
3. Then check whether player a has lost.
4. Otherwise, player b removes one or two sticks from the pile.
5. Repeat until either player breaches the contract.

$$
\begin{array}{rcl}
 & & \mathrm{Nim} \\
 & = & (\mathsf{rec}_a X \bullet \\
1: & & \{x \neq 0\}_b; \\
2: & & (x := x-1 \sqcup_a x := x-2); \\
3: & & \{x \neq 0\}_a; \\
4: & & (x := x-1 \sqcup_b x := x-2); \\
5: & & X)
\end{array}
$$



Contracts vs. programs

- Contracts generalize the traditional notion of a program to allow for any number of *agents* or *actors*. Different choices can be made by different agents.
- Batch oriented programs, concurrent programs, interactive programs and games are special cases of contracts.
- Contracts also introduce the new notion of *breaching* a contract (and dually, of being *released* from a contract).
- Contracts are more expressive than traditional program/specification notation.

Analyzing contracts

Achieving goals

Can one agent (or a coalition of agents) achieve a specific goal with a contract?

Operational semantics describes all possible ways of carrying out a contract.

In reality, one specific execution is selected, which may or may not lead to a desired final state for a specific agent.

The different agents are unlikely to have the same goals, and the way one agent makes its choices need not be suitable for another agent.

Establishing goals

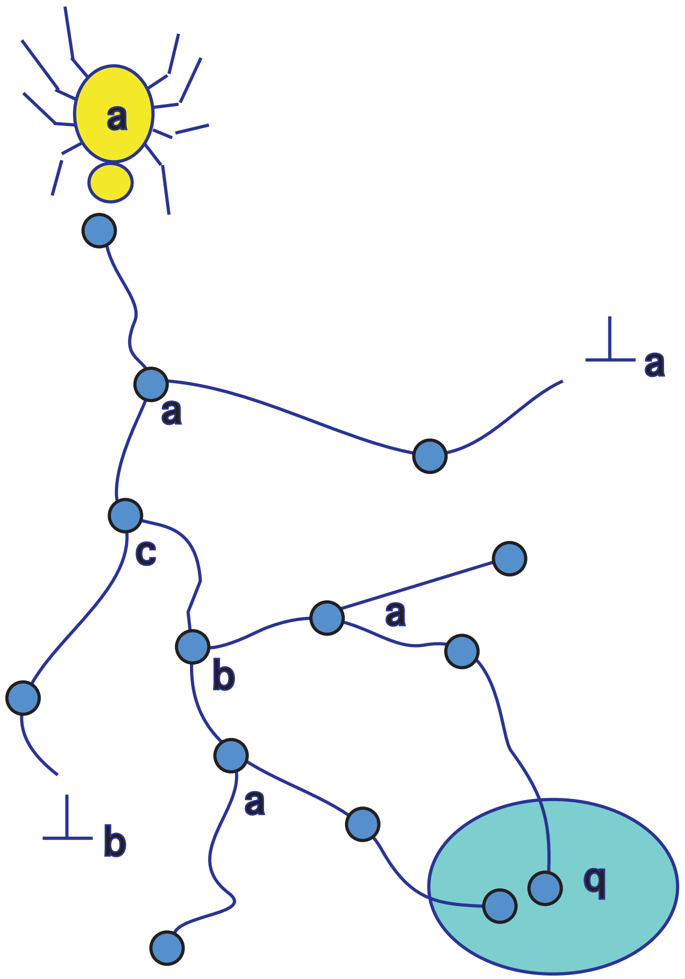
Agent a can *establish* condition q with contract S in initial state σ, denoted

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

if, assuming none of the other agents breach the contract, a can establish postcondition q no matter what the other agents do.

Thus $\sigma \; \{\!|\, S \,|\!\}_a \; q$ holds if the agent can make its own choices in such a way that

- either a final state is reached where q holds, or
- some other agent is forced to breach the contract.



Example: Contract1

$$\begin{aligned} \text{Contract1} \quad = \quad & \{1 \le y \le 4\}_a ; \langle x := 0 \rangle; \\ & (\langle x := x + 1 \rangle \sqcup_a \langle x := x + 2 \rangle); \\ & \{y = x\}_a \end{aligned}$$

$$\begin{array}{rl} & (x = 3, y = 1)\ \{\!|\ \text{Contract1}\ |\!\}_a\ x = 1 \\ & (x = 3, y = 1)\ \{\!|\ \text{Contract1}\ |\!\}_a\ x = y \\ & (x = 3, y = 2)\ \{\!|\ \text{Contract1}\ |\!\}_a\ x = y \\ \text{not} & (x = 3, y = 2)\ \{\!|\ \text{Contract1}\ |\!\}_a\ x = 1 \end{array}$$

Example:Interaction

$$\begin{aligned}\text{Interaction} \quad &= \quad x := 0;\\ &\quad\quad (x := x+1 \sqcup_a x := x+2);\\ &\quad\quad (x := x-1 \sqcup_b x := x-2)\end{aligned}$$

$$(x = 0)\ \{\!|\ \text{Interaction}\ |\!\}_a\ x \leq 0$$
$$(x = 0)\ \{\!|\ \text{Interaction}\ |\!\}_b\ x = 0$$

Coalitions

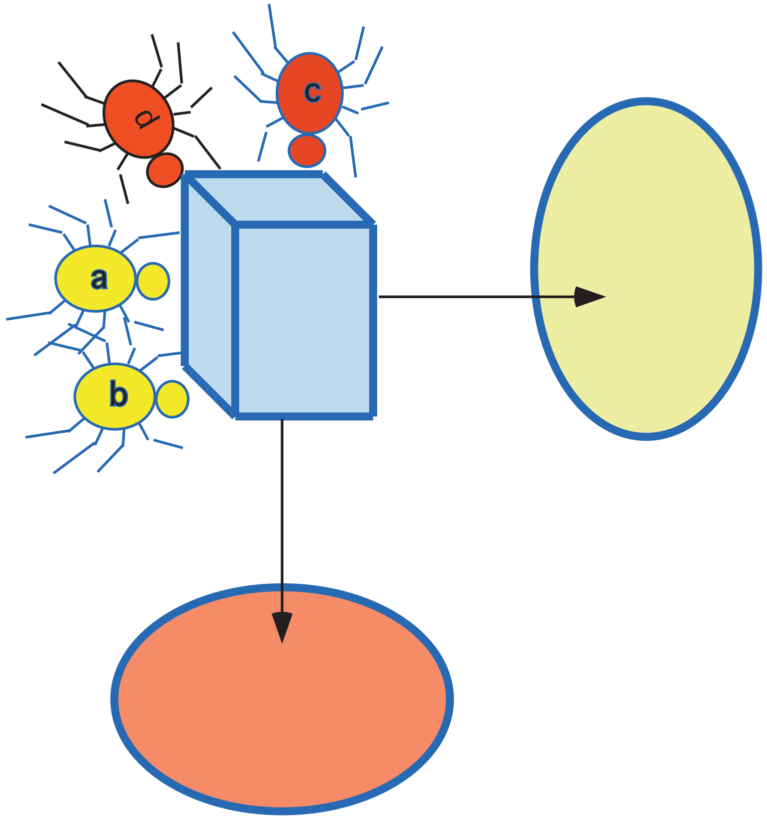
In general, we pick out a *coalition* A of agents whose side we are taking, and assume that these agents will co-operate in order to achieve a common goal.

The other agents may have other goals.

Then

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

holds if the agents in A can (by co-ordinating their choices) establish postcondition q no matter what the other agents do, whenever the other agents do not breach the contract.



Game interpretation

To prepare for the worst, we assume that the agents outside the coalition are hostile to the goal of our agents and try to prevent them from reaching it.

We call the coalition of agents collectively the *angel* and the other agents collectively the *demon*.

An *angelic choice* is made by a coalition agents, and a *demonic choice* by the other agents. We can then consider execution of a contract as a *game* between the angel and the demon.

- The game is started in a given *initial state* σ.
- The contract S gives the *rules* of the game.
- The goal of the game is some final state in q.

The angel tries to reach a final state in q. The angel wins if such a state is reached or if demon breaches the contract.

The angel looses if it breaches the contract or a state in $\neg q$ is reached.

Winning strategies

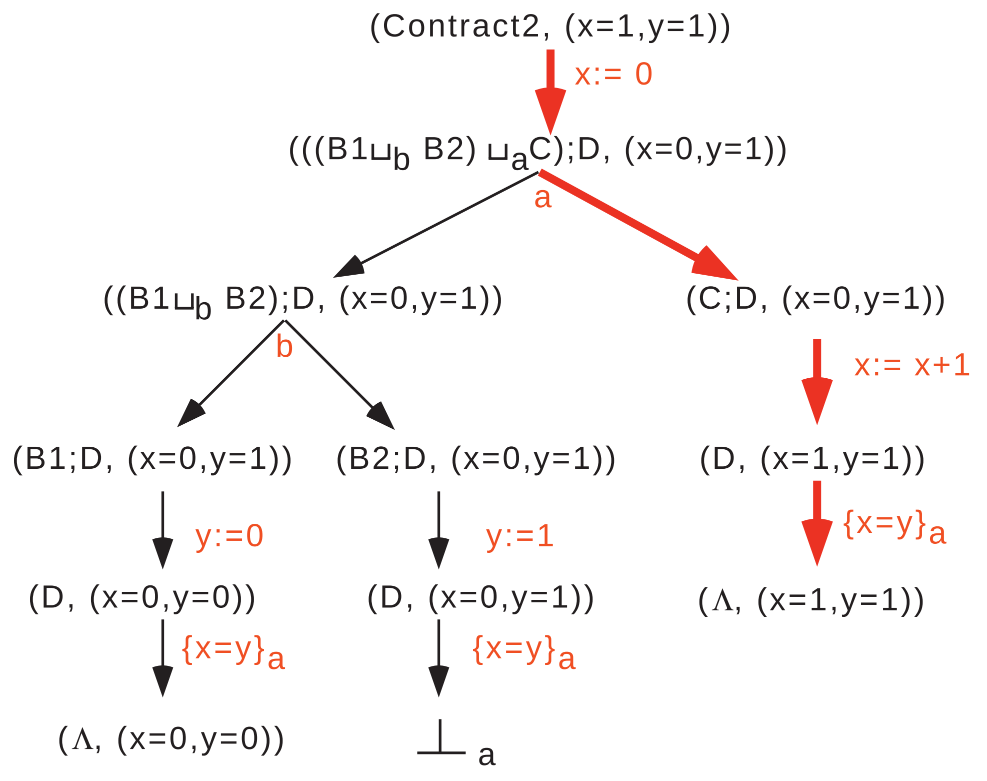
Agent a (or angel, or coalition A of agents) makes its choices according to a *strategy*: a function that for every configuration of the form $(S_1 \sqcup_a S_2, \gamma)$ returns either (S_1, γ) or (S_2, γ).

A strategy tells the agent what to do in every possible choice situation.

Thus $\sigma \ \{\!|\, S \,|\!\}_a \ q$ holds if and only if there exists a *winning strategy* for a to establish q with contract S in initial state σ. This can be determined from $\mathsf{op}.S.\sigma$ and q.

Example shows winning strategy for a to reach $x = y$ with Contract2 from initial state $(x = 1, y = 0)$.

$$(x = 1, y = 1) \ \{\!|\, S \,|\!\}_a \ x = y$$



Winning strategy for Nim

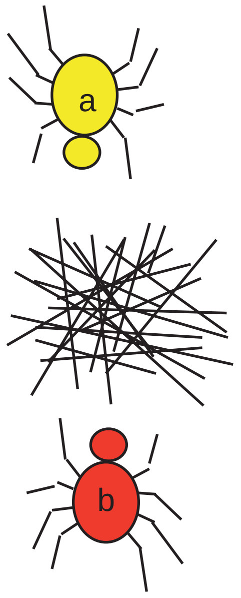
$$
\begin{array}{rcl}
 & & \mathrm{Nim} \\
 & = & (\mathsf{rec}_a X \bullet \\
1: & & \{x \neq 0\}_b; \\
2: & & (x := x - 1 \sqcup_a x := x - 2); \\
3: & & \{x \neq 0\}_a; \\
4: & & (x := x - 1 \sqcup_b x := x - 2); \\
5: & & X)
\end{array}
$$

Agent a has a winning strategy for Nim whenever

$$x.\sigma \ \mathsf{mod}\ 3 \neq 1$$

holds in the initial state σ. Thus,

$$x \ \mathsf{mod}\ 3 \neq 1 \ \{\!|\, \mathrm{Nim} \,|\!\}_a \ \mathsf{false}$$

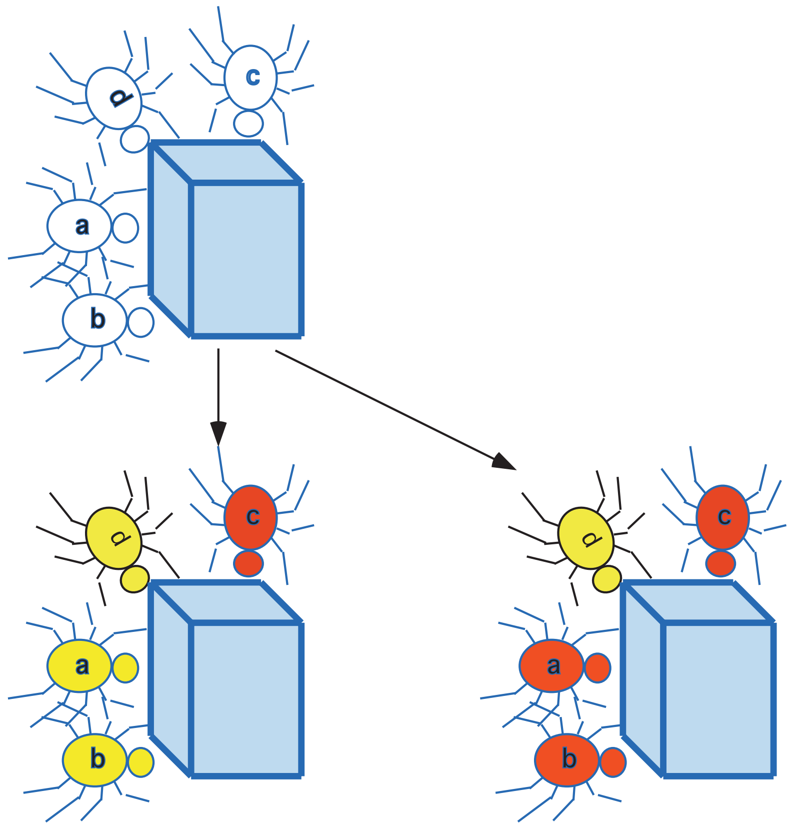


Symmetric and asymmetric system description

We can consider different coalitions of agents within the same contract, and analyze what different coalitions can achieve with the contract.

Contract provides a *symmetric* way of describing a system.

After we have determined a coalition of agents, we have an *asymmetric* (*game*) view of the system.



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 (by cooperation)can *achieve* postcondition q.

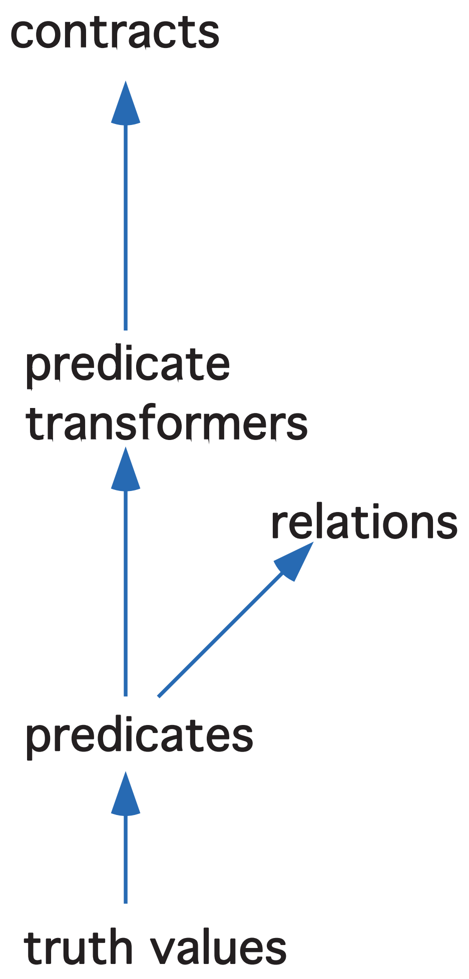
This provides us with an alternative semantic interpretation of contracts, of the type:

$$\mathsf{wp}.\,S : \mathcal{P}\Omega \to \mathcal{P}\Sigma \to \mathcal{P}\Sigma$$

where Ω is the set of all agents.

Predicate transformer hierarchy extended

Contracts are thus introduced by pointwise extension of predicate transformers:



Weakest preconditions for contracts

Using the predicate transformer lattice, we can then define

$$
\begin{array}{rcl}
\mathsf{wp}.\,\langle f\rangle.\,A & = & \langle f\rangle \\
\mathsf{wp}.\,\{p\}_a.\,A.\,q & = & \begin{cases} \{p\} & \text{if } a \in A \\ [p] & \text{if } a \notin A \end{cases} \\
\mathsf{wp}.\,(S_1; S_2).\,A & = & \mathsf{wp}.\,S_1.\,A; \mathsf{wp}.\,S_2.\,A \\
\mathsf{wp}.\,\{R\}_a.\,A & = & \begin{cases} \{R\} & \text{if } a \in A \\ [R] & \text{if } a \notin A \end{cases} \\
\mathsf{wp}.\,(S_1 \sqcup_a \; S_2).\,A & = & \begin{cases} \mathsf{wp}.\,S_1.\,A \sqcup \mathsf{wp}.\,S_2.\,A & \text{if } a \in A \\ \mathsf{wp}.\,S_1.\,A \sqcap \mathsf{wp}.\,S_2.\,A & \text{if } a \notin A \end{cases}
\end{array}
$$

Weakest preconditions for contracts (alt.)

Definitions in terms of sets:

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

Wp for recursive contracts

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

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

Here we assume that $\mathsf{wp}.\,X.\,A = X$, so that the fixpoint is taken over a predicate transformer.

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.

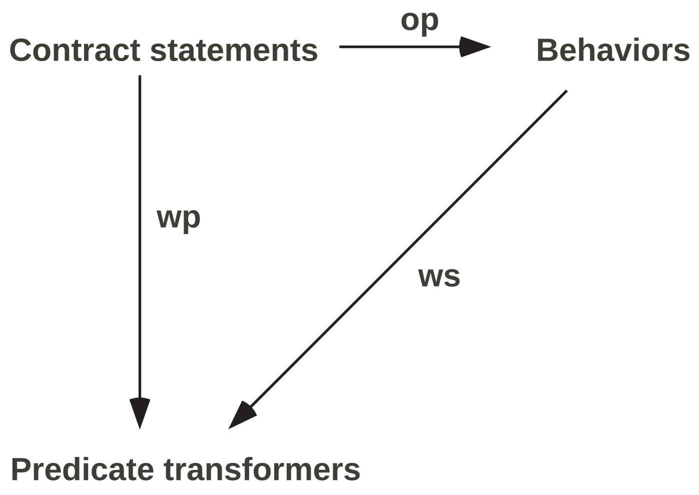
Winning strategy theorem

We can prove that $wp.\, S.\, A$ has the required property for any contract statement S, collection of agents A, initial state σ and postcondition q :

$$\sigma \in wp.\, S.\, A.\, q \quad \equiv \quad \sigma \; \{\!|\; S \;|\!\}_A \; q$$

Thus, $wp.\, S.\, A.\, q$ computes the set of initial states for which the coalition A has a winning strategy for using S to establish postcondition q from initial state σ.

In figure, $\mathsf{ws}.\, B.\, A.\, q = \{\sigma \mid R\}$, where R says that coalition A has winning strategy in $B.\, \sigma$ to reach q.


$$\begin{tikzcd}
\text{Contract statements} \arrow[r, "op"] \arrow[d, "wp"] & \text{Behaviors} \arrow[dl, "ws"] \\
\text{Predicate transformers} &
\end{tikzcd}$$

Example

We compute the set of initial states for which the agent a has a winning strategy to reach $x \geq 0$ with contract

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

We have that

$$\begin{array}{rl} & wp.\,(x := x - 1 \sqcup_b x := x - 2).\,\{a\}.\,(x \geq 0) \\ = & \{\text{demonic choice}\} \\ & wp.\,(x := x - 1).\,\{a\}.\,(x \geq 0) \cap wp.\,(x := x - 2).\,\{a\}.\,(x \geq 0) \\ = & \{\text{assignment}\} \\ & (x - 1 \geq 0) \cap (x - 2 \geq 0) \\ = & \{\text{set theory}\} \\ & (x \geq 1) \cap (x \geq 2) \\ = & \{\text{set theory}\} \\ & (x \geq 2) \end{array}$$

Similarly, we compute that

$$wp.\,(x := x + 1 \sqcup_a x := x + 2).\,\{a\}.\,(x \geq 2) \quad = \quad x \geq 0$$

Hence,

$$wp.\,S.\,\{a\}.\,(x \geq 0) \quad = \quad x \geq 0$$

Correctnesss and Refinement

Correctness

Coalition A can *establish* condition q with contract S from any initial state in p:

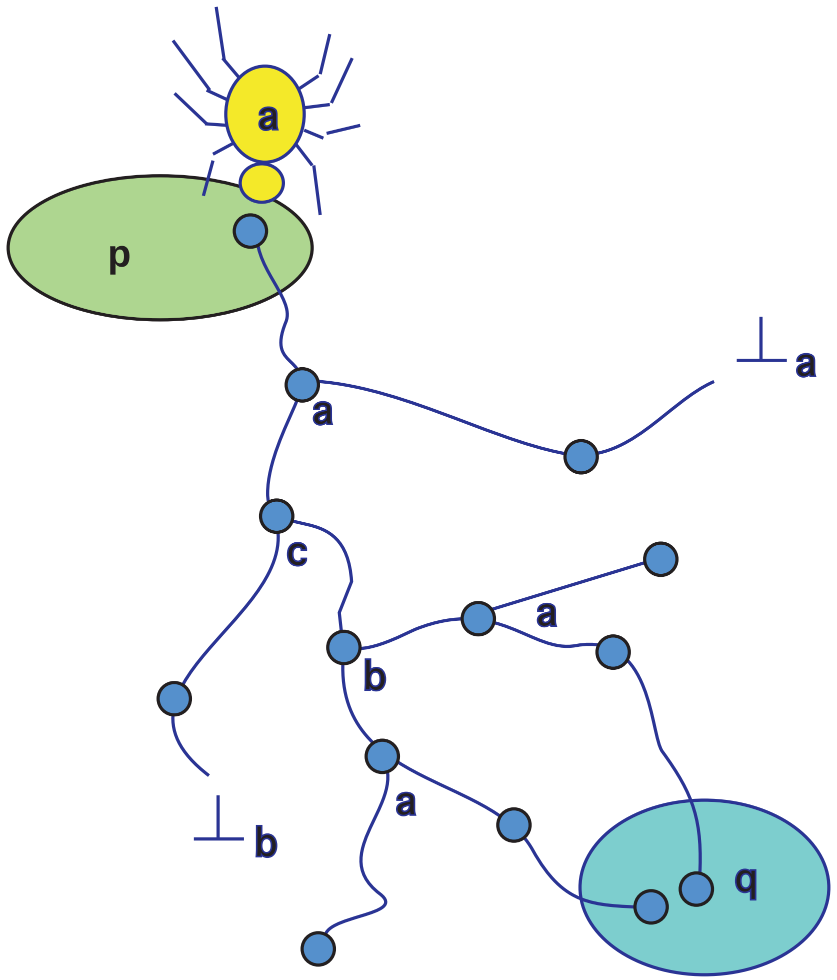
$$\begin{array}{rl} & p \;\{\!|\, S \,|\!\}_A\; q \\ \triangleq & (\forall \sigma \in p \bullet \sigma \;\{\!|\, S \,|\!\}_A\; q) \end{array}$$

A contract can be suitable for some goals and unsuitable for others.

We will say that contract S is *correct* for the goal q in initial states p for coalition A, when $p \;\{\!|\, S \,|\!\}_A\; q$. This is generalization of usual notion of correctness.

Example:

$$\begin{array}{r} 1 \le y \le 2 \;\{\!|\, \text{Contract1} \,|\!\}_b\; x = y \\ y = 1 \;\{\!|\, \text{Contract2} \,|\!\}_a\; x = y \end{array}$$



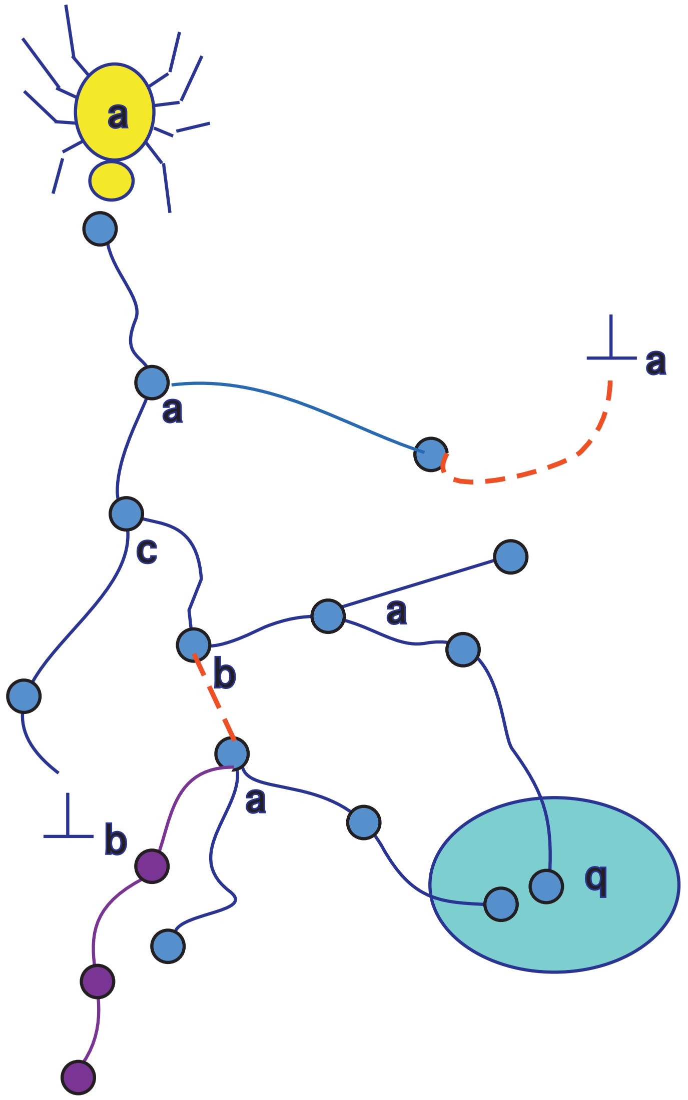
Refining contracts

Contract S' is a *refinement* of contract S for coalition A if any condition that A can establish with S can also be established with S':

$$\begin{array}{ll} & S \sqsubseteq_A S' \\ \triangleq & (\forall \sigma \forall q \bullet \sigma \{\!| S |\!\}_A\ q \Rightarrow \sigma \{\!| S' |\!\}_A\ q) \end{array}$$

Intuitively, $S \sqsubseteq_A S'$ means that contract S' is at least as good as contract S for coalition A.

Contracts S and S' are *refinement equivalent* from the point of view of coalition A, $S =_A S'$, if $S \sqsubseteq_A S'$ and $S' \sqsubseteq_A S$.



Refinement example

Refinement for agent a (and similarly for coalition of agents A) means

- adding new choices for a,
- removing choices for the other agents,
- decreasing set of states where contract can be breched by a, or
- increasing set of states where other agents can breach contract.

We have that Contract2 $\sqsubseteq_a$ Contract3, where

Contract2 $=$	Contract3 $=$
$x := 0;$	$x := 0;$
$\{\mathsf{true}\}_b;$	$\{y > 0\}_b;$
$((y := 0 \sqcup_b y := 1)$	$(y := 1$
$\sqcup_a x := x + 1);$	$\sqcup_a x := x + 1 \sqcup_a x := x + 2);$
$\{y = x\}_a$	$\{y \leq x\}_a$

Computing correctness and refinement

The winning strategy theorem gives us immediately the following corollary for *correctness*:

$$p \subseteq wp.\,S.\,A.\,q \quad \equiv \quad p \;\{\!|\, S \,|\!\}_A\; q$$

The following corollary holds for *refinement*:

$$S \sqsubseteq_A S' \quad \equiv \quad (\forall\, q \bullet wp.\,S.\,A.\,q \subseteq wp.\,S'.\,A.\,q)$$

This result allows us to prove correctness and refinement without having to rely on the operational semantics of contracts. Sufficient to analyze properties of the predicate transformer $wp.\,S.\,A$.

Algebra of contracts

Let contract statement S and predicate q be arbitrary.

- If new agents join a coalition, then the capabilities of the coalition increase (i.e., it becomes easier to reach goals):

$$A \subseteq A' \;\Rightarrow\; \mathsf{wp}.\,S.\,A \sqsubseteq \mathsf{wp}.\,S.\,A', \text{provided that } S \text{ is strict}$$

- If a coalition can establish a goal q then the remaining agents cannot establish $\neg q$:

$$\mathsf{wp}.\,S.\,A.\,q \;\subseteq\; \neg\mathsf{wp}.\,S.\,\overline{A}(\neg q)$$

  The reason that we do not have an equality in here is that in an infinite scenario neither A nor $\overline{A}$ establishes any postcondition.

- The power of two combined coalitions is greater (or at least as great) as the sum of their powers in isolation. The dual property also holds:

$$\mathsf{wp}.\,S.\,A \sqcup \mathsf{wp}.\,S.\,A' \quad\sqsubseteq\quad \mathsf{wp}.\,S.\,(A \cup A')$$

Determining achievability and refinement

- Computing *wp* for a contract allows us to determine the intial states in which an agent can reach a given goal, without having to rely on the operational semantics.
- The notion of correctness and refinement generalizes the traditional correctness and refinement notion with demonic nondeterminism, where only the system (the demon) can make choices.
- The notion used here also defines correctness and refinement for interactive systems. There it is sufficient that there is some way for the user to make choices so that the final condition is established.
- In general, we have defined correctness and refinement when both user and system can make choices, in terms of the existence of a winning strategy for the user to reach its goal.
- The general case can, e.g., be used to define correctness and refinement of an interactive system with concurrency (e.g., background processes)

Proving existence of winning strategies

Correctness of loops

Theorem Assume that $g_1, \ldots, g_n$, p, and q are predicates and S_i are monotonic statements for $i = 1, \ldots, n$. Assume that $\{r_w \mid w \in W\}$ is a ranked collection of predicates, with $r = (\cup w \in W \bullet r_w)$ and $g = g_1 \vee \cdots \vee g_n$. Then

$$p \;\{\!|\; \mathsf{do}\ g_1 \rightarrow S_1 \;[\!]\; \ldots \;[\!]\; g_n \rightarrow S_n\ \mathsf{od} \;|\!\}\; q$$

provided that

- $p \subseteq r$,
- $(\forall w \in W \bullet r_w \cap g_i \;\{\!|\; S_i \;|\!\}\; r_{<w})$, for $i = 1, \ldots, n$, and
- $r \cap \neg g \subseteq q$.

The ranked predicate r_w is usually described as a conjunction of an invariant I and a variant t, so that r_w is $I \;\wedge\; t = w$. Here

- first condition states that the *loop invariant* r holds initially;
- second condition asserts that each iteration of the loop preserves the loop invariant while decreasing the rank of the particular predicate r_w; and
- third condition states that the postcondition q is established upon termination of the loop.

Nim game

The state has only one attribute, x, the number of matches, ranging over the natural numbers. The state space is thus the type Nat.

We define the state relation

$$\mathsf{Move} \quad \triangleq \quad (x := x' \mid x - 2 \leq x' < x)$$

(since we are talking about natural numbers, we follow the convention that $m - n = 0$ when $m \leq n$).

Then the Nim-game can be expressed in the following simple form:

$$\mathrm{Nim} \quad \triangleq \quad \mathsf{do\ true} \rightarrow [x \neq 0]; \{\mathsf{Move}\}; \{x \neq 0\}; [\mathsf{Move}]\ \mathsf{od}$$

The moves

- First, the guard statement $[x \neq 0]$ is a check to see whether the angel has already won (The demon has lost if the pile of matches is empty when it is the angels move).
- If there are matches left ($x \neq 0$), then the angle moves according to the relation Move, i.e., removes one or two matches (decreases x by 1 or 2).
- Now the dual situation arises. The assert statement $\{x \neq 0\}$ is a check to see whether the angel has lost.
- If there are matches left, then the demon moves according to the relation Move, i.e., removes one or two matches.

Proving existence of a winning strategy

By instantiating in the correctness rule for loops, we get the following rule for proving the existence of winning strategies in a two-person game:

Theorem

$$p \; \{\!|\; \mathsf{do\ true} \rightarrow S \;\mathsf{od} \;|\!\} \; \mathsf{false}$$

if

- $p \subseteq r$, and
- $(\forall\, w \in W \bullet r_w \; \{\!|\; S \;|\!\} \; r_{<w})$

holds for some ranked collection of predicates $\{r_w \mid w \in W\}$:

Proof

By specializing q to false in loop rule, we get the following three conditions:

- $p \subseteq r$,
- $(\forall\, w \in W \bullet \mathsf{true} \cap r_w \;\{\!|\, S \,|\!\}\; r_{<w})$, and
- $\neg\, \mathsf{true} \cap r \subseteq \mathsf{false}$.

Winning strategy for Nim

For the angel to be assured of winning this game, it is necessary that he always makes the number x of matches that remain satisfy the condition

$$x \ \mathsf{mod}\ 3 = 1$$

by removing either one or two matches.

If $x\ \mathsf{mod}\ 3 = 1$ before the angels move, then he would need to remove three matches to re-establish this condition, which is not allowed. Otherwise, removing one or two muches is sufficient to establish the condition.

We will show that

$$x\ \mathsf{mod}\ 3 \neq 1\ \{\!|\ Nim\ |\!\}\ \mathsf{false}$$

i.e, the angel has a winning strategy whenever $x\ \mathsf{mod}\ 3 \neq 1$ holds initially.

Correctness formulation

We need to show that

- $x \ \mathsf{mod}\ 3 \neq 1 \subseteq I$, and
- $(\forall n \in \mathsf{Nat} \bullet I \wedge t = n \ \{\!|\ S\ |\!\}\ I \wedge t < n)$

We choose

- $I = (x \ \mathsf{mod}\ 3 \neq 1)$, and
- $t = x$

First condition is trivially true. Second condition amounts to proving that

$$x \ \mathsf{mod}\ 3 \neq 1 \wedge x = n \ \{\!|\ S\ |\!\}\ x \ \mathsf{mod}\ 3 \neq 1 \wedge x < n$$

holds for an arbitrary natural number n, where

$$S = [x > 0]; \{\mathsf{Move}\}; \{x > 0\}; [\mathsf{Move}]$$

The proof

First, we find the intermediate condition:

$$\begin{array}{cl}
 & (\{x > 0\}; [\mathsf{Move}]).\, (x \;\mathsf{mod}\; 3 \neq 1 \wedge x < n) \\
= & \{\text{definitions}\} \\
 & x > 0 \wedge (\forall x' \bullet x - 2 \leq x' < x \;\Rightarrow\; x' \;\mathsf{mod}\; 3 \neq 1 \wedge x' < n) \\
= & \{\text{reduction, arithmetic}\} \\
 & x \;\mathsf{mod}\; 3 = 1 \wedge x \leq n
\end{array}$$

This is the condition that angel should establish on every move.
Continuing, we find the precondition

$$\begin{array}{cl}
 & ([x > 0]; \{\mathsf{Move}\}).\, (x \;\mathsf{mod}\; 3 = 1 \wedge x \leq n) \\
= & \{\text{definitions}\} \\
 & x = 0 \;\vee\; (\exists x' \bullet x - 2 \leq x' < x \wedge x' \;\mathsf{mod}\; 3 = 1 \wedge\; x' \leq n) \\
\supseteq & \{\text{reduction, arithmetic}\} \\
 & x \;\mathsf{mod}\; 3 \neq 1 \wedge x = n
\end{array}$$

Thus we have shown that the required condition holds, i.e., that the game has a winning strategy.