Refinement Calculus Overview

R. J. R. Back
(joint work with Joakim von Wright)

Turku Centre for Computer Science and
Åbo Akademi University

Introduction

New book

Presentation is based on a recent 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.

Contents of book

Foundations Lattice theoretic basis for refinement calculus. Higher order logic basis. Notion of program variables. Reasoning about simple programs.

Statements Predicate transformer semantics for statements. Refinement calculus hierarchy. Game interpretation of statements. Correctness and refinement.

Recursion and Iteration Foundations for fixpoint theory. Recursion and iteration. Continuity. Reasoning about arrays. Derivations of recursive programs. Analyzing games.

Statement Subclasses Statement subclasses based on homomorphism properties of statements. Specification statements. Refinement in context. Iteration of conjunctive statements.

States, agents and contracts

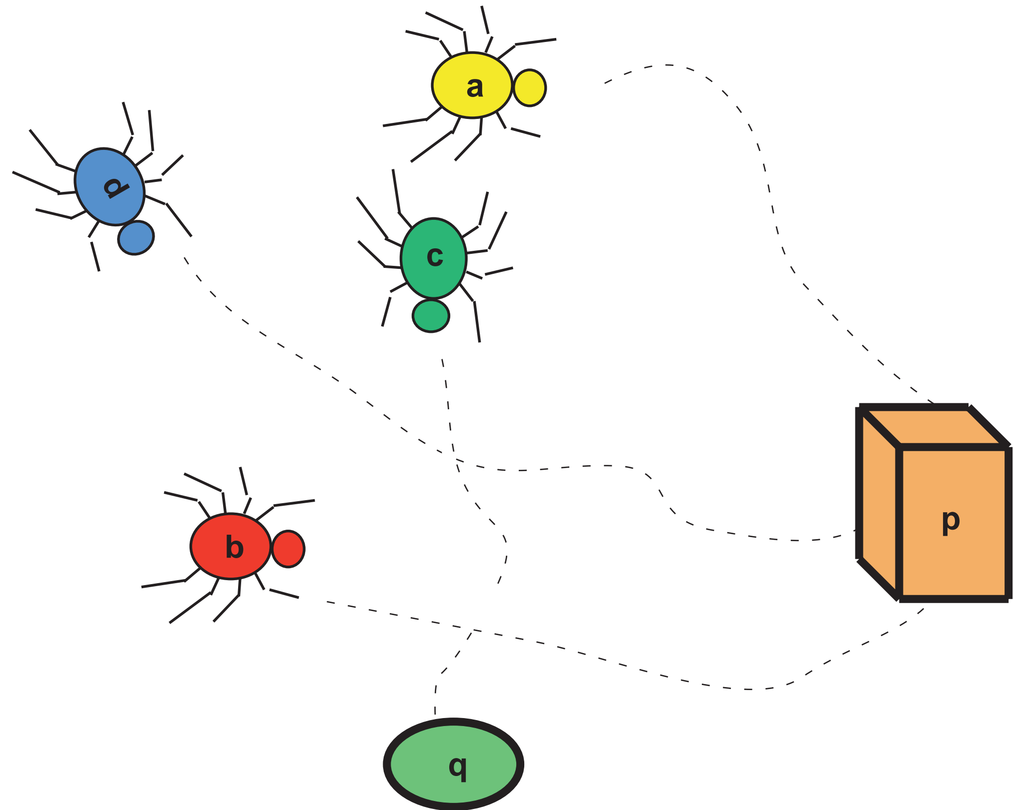
Consider

- a collection of **agents**
- that operate in a **world**
- in order to achieve their **goals**.

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.



Questions

- What is a state?
- How do you manipulate the state?
- What is a contract?
- What are the agents, and what can they do?
- What kind of goals can the agents have?
- How does an agent achieve a goal?
- How do the agents interact with each other?
- What has this all to do with program refinement?

Manipulating the state

Observing and changing the state

The world is a **state** σ in a **state space** Σ. The state is observed with **state functions**

$$f : \Sigma \to \Gamma$$

A **predicate** p is a state function of type

$$p : \Sigma \to \mathsf{Bool}$$

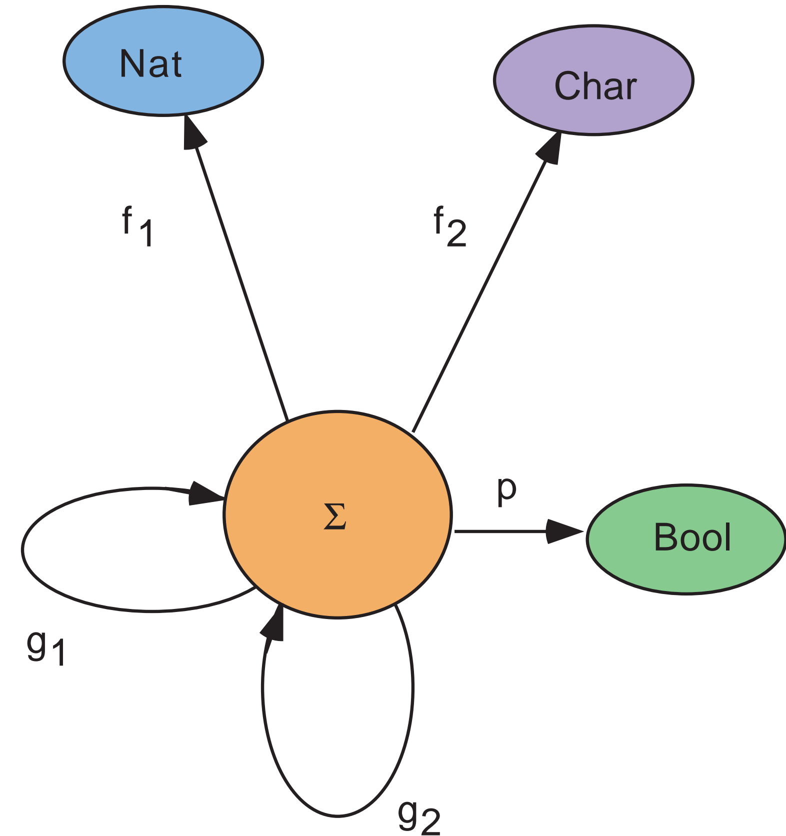
It describes a property that a state may or may not have.

We often identify a predicate p with the set of states $\{\sigma \mid p.\sigma\}$ that satisfy it.

A **state transformer** is a function

$$g : \Sigma \to \Sigma$$

An agent can **change** the present state σ to a new state $g.\sigma$ by applying a **state transformer** g.



Composing state transformers

- **Sequential composition** of state functions (and state transformers):

$$(f\,;g).\,\sigma \quad \triangleq \quad g.\,(f.\,\sigma)$$

- **Conditional composition** of state functions (and state transformers):

$$(p \longrightarrow f|g).\,\sigma \quad \triangleq \quad \begin{cases} f.\,\sigma & \text{if } p.\,\sigma \\ g.\,\sigma & \text{if } \neg p.\,\sigma \end{cases}$$

States and attributes

The state is observed and changed using a collection of **attributes** (or **program variables**)

$$x_1, \ldots, x_n$$

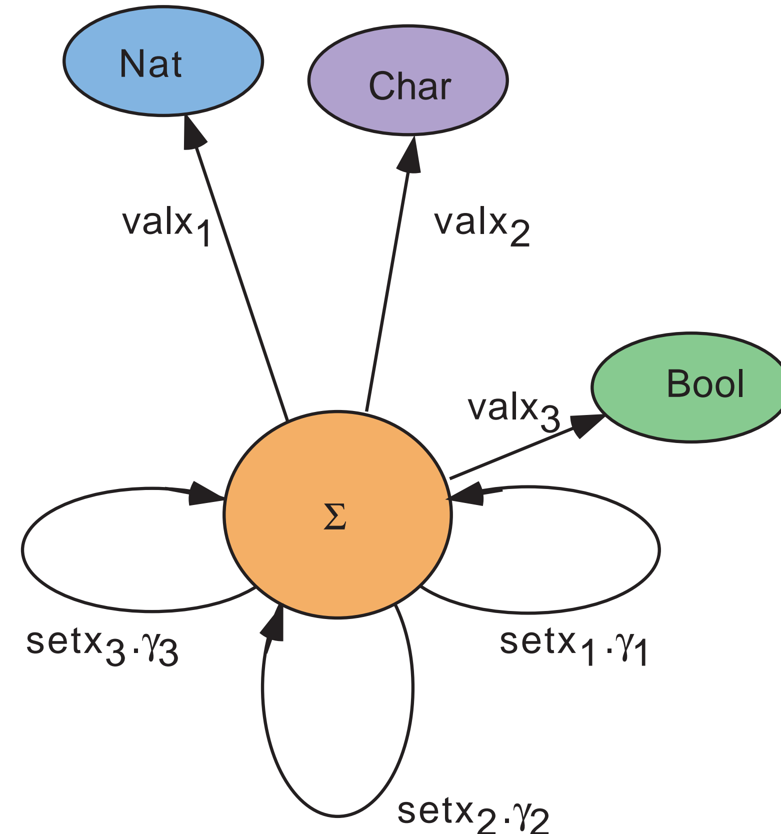
These can be **observed** and **changed** independently of each other.

An attribute x ranging over Γ is a pair of functions, $x = (valx, setx)$, where

$$valx : \Sigma \to \Gamma \quad \text{and} \quad setx : \Gamma \to \Sigma \to \Sigma$$

($valx$ is a state function and $setx.\gamma$ is a state transformer)

- $valx.\sigma$ is the **value** of attribute x in state σ, and
- $\sigma' = setx.\gamma.\sigma$ is the **updated** state where attribute x is set to value γ



Properties of program variables

1. You get what you set:

$$val x.(set x.a.\sigma) \quad = \quad a$$

2. An attribute can only record one value:

$$set x.a; set x.b \quad = \quad set x.b$$

3. Setting an attribute to the value it already has does not change the state:

$$set x.(val x.\sigma).\sigma \quad = \quad \sigma$$

4. Attributes can be set independently of each other (x, y different):

$$val y.(set x.a.\sigma) \quad = \quad val y.\sigma$$

5. The order in which two different attributes x, y are set should not matter:

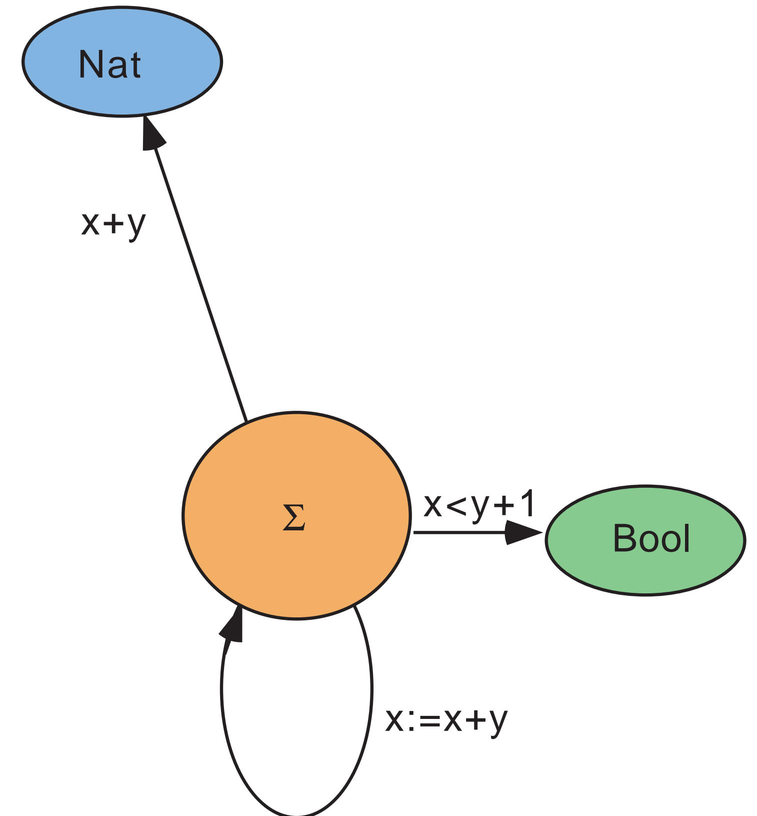
$$set x.a; set y.b \quad = \quad set y.b; set x.a$$

Example

We can use these properties to determine the value of a program variable in a given state. Example:

$$
\begin{array}{cl}
 & val\,y.\,((set\,x.\,3; set\,y.\,5; set\,x.\,0).\,\sigma) \\
= & \{\text{associativity of functional composition}\} \\
 & val\,y.\,((set\,x.\,3; (set\,y.\,5; set\,x.\,0)).\,\sigma) \\
= & \{\text{prop. (5)}\} \\
 & val\,y.\,((set\,x.\,3; set\,x.\,0; set\,y.\,5).\,\sigma) \\
= & \{\text{definition of forward composition}\} \\
 & val\,y.\,(set\,y.\,5.\,((set\,x.\,3; set\,x.\,0).\,\sigma)) \\
= & \{\text{ property (1)}\} \\
 & 5
\end{array}
$$

Expressions and assignments



Expressions are state functions expressed in terms of attributes. Expression $x+y$ is a function on states:

$$(x + y).\sigma \quad = \quad valx.\sigma + valy.\sigma$$

A **boolean expression** is a predicate:

$$(x \leq y + 1).\sigma \quad \equiv \quad valx.\sigma \leq valy.\sigma + 1$$

Expressions are used in **assignments**. The function $x := x + y$ changes the state, by updating the value of x to the value of the expression $x + y$:

$$(x := x + y).\sigma \quad = \quad setx.(valx.\sigma + valy.\sigma).\sigma$$

Assignments in general

In general, define

$$(x := e).\sigma \quad \triangleq \quad setx.(e.\sigma).\sigma$$

Define **multiple assignment** by

$$(x_1, \ldots, x_m := e_1, \ldots, e_m).\sigma \quad \triangleq$$
$$(setx_1.(e_1.\sigma); setx_2.(e_2.\sigma); \ \ldots \ ; setx_m.(e_m.\sigma)).\sigma$$

where $x_1, \ldots, x_m$ are distinct attributes.

Proving properties involving attributes

Let $\mathsf{var}\ x_1, \ldots, x_m$ state that the attribute properties (1)-(5) hold for attributes $x_1, \ldots, x_m$ (we then say that the attributes are **program variables**). Then

$$\mathsf{var}\ x_1, \ldots, x_m \ \vdash \ t = t'$$

states that expressions t and t' have the same value whenever $x_1, \ldots, x_m$ are program variables.

Proofs with program variables are **monotonic**:

$$\frac{\mathsf{var}\ x_1, \ldots, x_m \ \vdash \ t = t'}{\mathsf{var}\ x_1, \ldots, x_m, x_{m+1}, \ldots, x_n \ \vdash \ t = t'}$$

The naming of the attributes does also not matter:

$$\frac{\mathsf{var}\ x_1, \ldots, x_m \ \vdash \ t}{\mathsf{var}\ x'_1, \ldots, x'_m \ \vdash \ t[x_1, \ldots, x_m := x'_1, \ldots, x'_m]}$$

where $x'_1, \ldots, x'_m$ are all distinct and are free for $x_1, \ldots, x_m$ in t.

Working with attributes

$$
\begin{array}{ll}
 & \mathsf{var}\ x, y, \\
 & val x.\,\sigma = 3, val y.\,\sigma = 2 \\
\vdash & (x * y).\,((x := x + y).\,\sigma) \\
= & \{\text{definition of assignment}\} \\
 & (x * y).\,(set x.\,(val x.\,\sigma + val y.\,\sigma).\,\sigma) \\
= & \{\text{assumption}\} \\
 & (x * y).\,(set x.\,(3 + 2).\,\sigma) \\
= & \{\text{arithmetic}\} \\
 & (x * y).\,(set x.\,5.\,\sigma) \\
= & \{\text{value of expression}\} \\
 & (val x.\,(set x.\,5.\,\sigma) * val y.\,(set x.\,5.\,\sigma) \\
= & \{\text{properties (1) and (4), assumption }\} \\
 & 5 * 2 \\
= & \{\text{arthmetic}\} \\
 & 10
\end{array}
$$

Substitution property

In general, we havet the following substitution property (e and f expressions) in program variable(s) y:

$$\mathsf{var}\ x, y\ \vdash\ e.\,((x := f).\,\sigma) \quad = \quad e[x := f].\,\sigma$$

Example:

$$\begin{aligned}\mathsf{var}\ x, y\ \vdash\ (x * y).\,((x := x + y).\,\sigma) \quad &= \quad (x * y)[x := x + y].\,\sigma \\ &= \quad ((x + y) * y).\,\sigma\end{aligned}$$

Assignment properties

Examples of general properties of assignments:

$$\mathsf{var}\ x \vdash\ (x := e); (x := f) \ = \ (x := f[x := e])$$

$$\mathsf{var}\ x, y \vdash\ (x := e) \ = \ (y, x := y, e)$$

$$\mathsf{var}\ x, y \vdash\ (x := e); (y := f) \ = \ (x, y := e, f[x := e])$$

If x is not free in f and y is not free in e, then

$$\mathsf{var}\ x, y \vdash\ (x := e); (y := f) \ = \ (y := f); (x := e)$$

Proof of first property

$$
\begin{array}{ll}
 & \mathsf{var}\ x \\
\vdash & ((x := e); (x := f)).\,\sigma \\
= & \{\text{definition of function composition}\} \\
 & (x := f).\,((x := e).\,\sigma) \\
= & \{\text{definition of assignment}\} \\
 & setx.\,(f.\,((x := e).\,\sigma)).\,((x := e).\,\sigma) \\
= & \{\text{definition of assignment}\} \\
 & setx.\,(f.\,((x := e).\,\sigma)).\,(setx.\,(e.\,\sigma).\,\sigma) \\
= & \{\text{attribute property (2)}\} \\
 & setx.\,(f.\,((x := e).\,\sigma)).\,\sigma \\
= & \{\text{substitution property}\} \\
 & setx.\,(f[x := e].\,\sigma).\,\sigma \\
= & \{\text{definition of assignment}\} \\
 & (x := f[x := e]).\,\sigma
\end{array}
$$

Alternative ways of describing states

- **State is total function** $\sigma : \text{Variables} \to \text{Values}$, where Variables is set of all program variable names. Simple theoretically, but is untyped. Does not utilize typing that already exists in higher order logic. Have to build in typing into programming logic formalization. Does not allow to change collection of variables.

- **State is partial function** $\sigma : \text{Variables} \to \text{Values}$. Allows adding and deleting variables, but otherwise similar to previous.

- **State is tuple** $\sigma = (x_1, \ldots, x_m)$. Simple to use in higher order logic, but does not satisfy monotonicity property above.

- **State is record** $\sigma = \{x_1 = a_1, \ldots, x_m = a_m\}$. Requires that record types are built into higher order logic. Program variable names part of type, so naming of program variables matters.

Attribute model for program variables

The attribute model only states the minimial assumptions that we need about program variables, without fixing a specific model for program variables. All models above satisfy properties (1) - (5).

An attribute x is of type

$$x : (\Sigma \to \Gamma) \times (\Gamma \to \Sigma \to \Sigma)$$

Can define

$$val.\,(f, g) = f \qquad\qquad set.\,(f, g) = g$$

Then can write $val.\,x$ instead of $valx$ and $set.\,x$ instead of $setx$.

This allows to quantify over attributes, define functions over attributes etc.

Contracts

Contract statements

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

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

(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. Another special kind is $\mathsf{skip} = \langle \mathsf{id} \rangle$, which 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

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

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

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

- If $y < 1$ or $y > 4$, then agent a cannot avoid breaching 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:

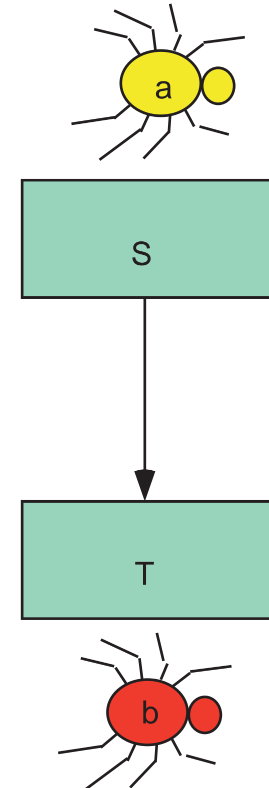
$$S \quad = \quad x := 0; (T \sqcup_a x := x + 1); \{y = x\}_a$$

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

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

The overall contract is

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

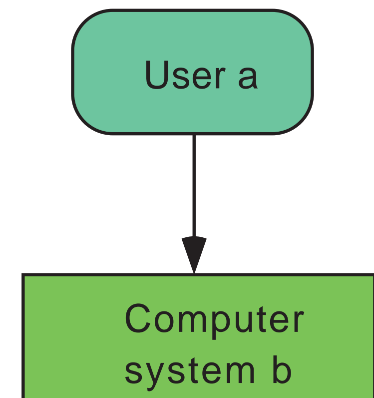


Programs

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

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

Choices are only made by the computer system. It resolves internal choices in a manner that the user cannot influence or know (**demonic nondetermism**).



Example:

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

Not permitted to do division by zero. User breaks contract if she attempts that, releasing the system from any obligations.

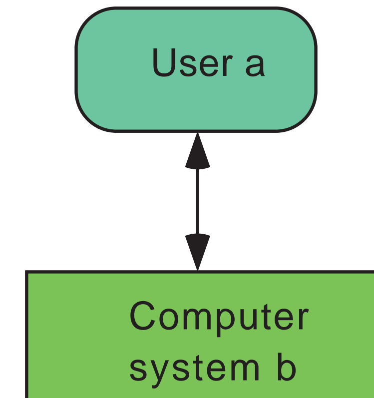
Abort statement is a total breach of contract:

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

Interactive system

Permit user to also make choices. The user a chooses between alternatives in order to influence the computation.

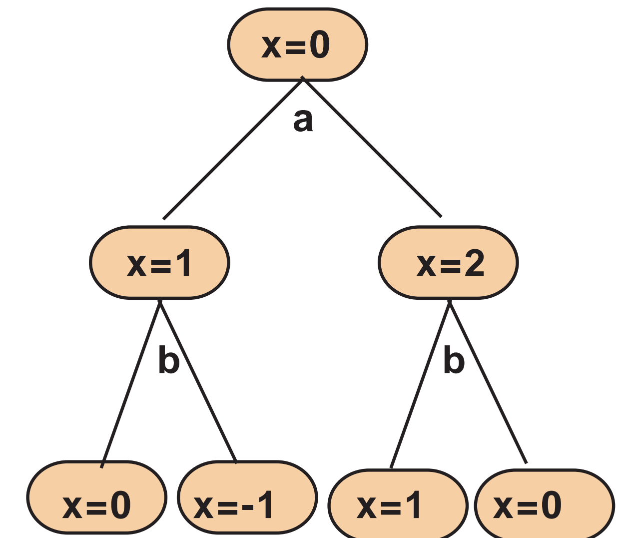
$$\begin{aligned} \text{Interaction} \quad = \quad & x := 0; \\ & (x := x + 1 \sqcup_a x := x + 2); \\ & (x := x - 1 \sqcup_b x := x - 2) \end{aligned}$$



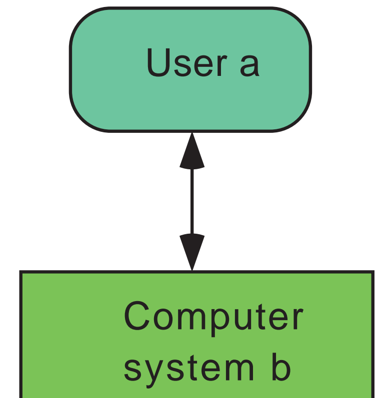
User interaction can be seen as a *menu choice.*

User should choose

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

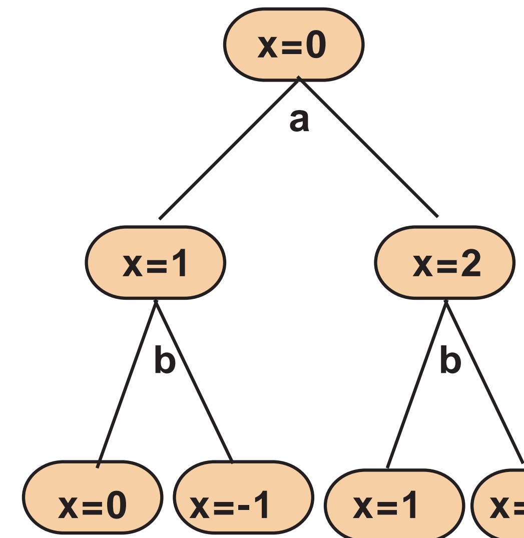


Interactive system, alternative view



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

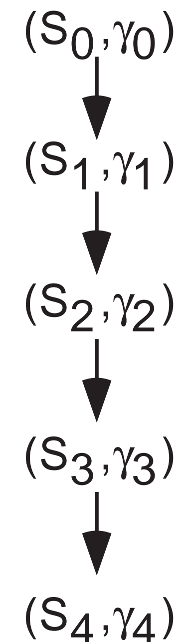


Operational semantics of contracts

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 $\bot_a$ (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

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

Assertion

$$\frac{p.\,\sigma}{(\{p\}_a, \sigma) \rightarrow (\Lambda, \sigma)} \qquad \frac{\neg p.\,\sigma}{(\{p\}_a, \sigma) \rightarrow (\Lambda, \bot_a)}$$

$$\frac{}{(\{p\}_a, \bot_b) \rightarrow (\Lambda, \bot_b)}$$

Sequential composition

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

Choice

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

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

Example derivation

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

$$\longrightarrow \quad \{\text{assertion rule}\}$$

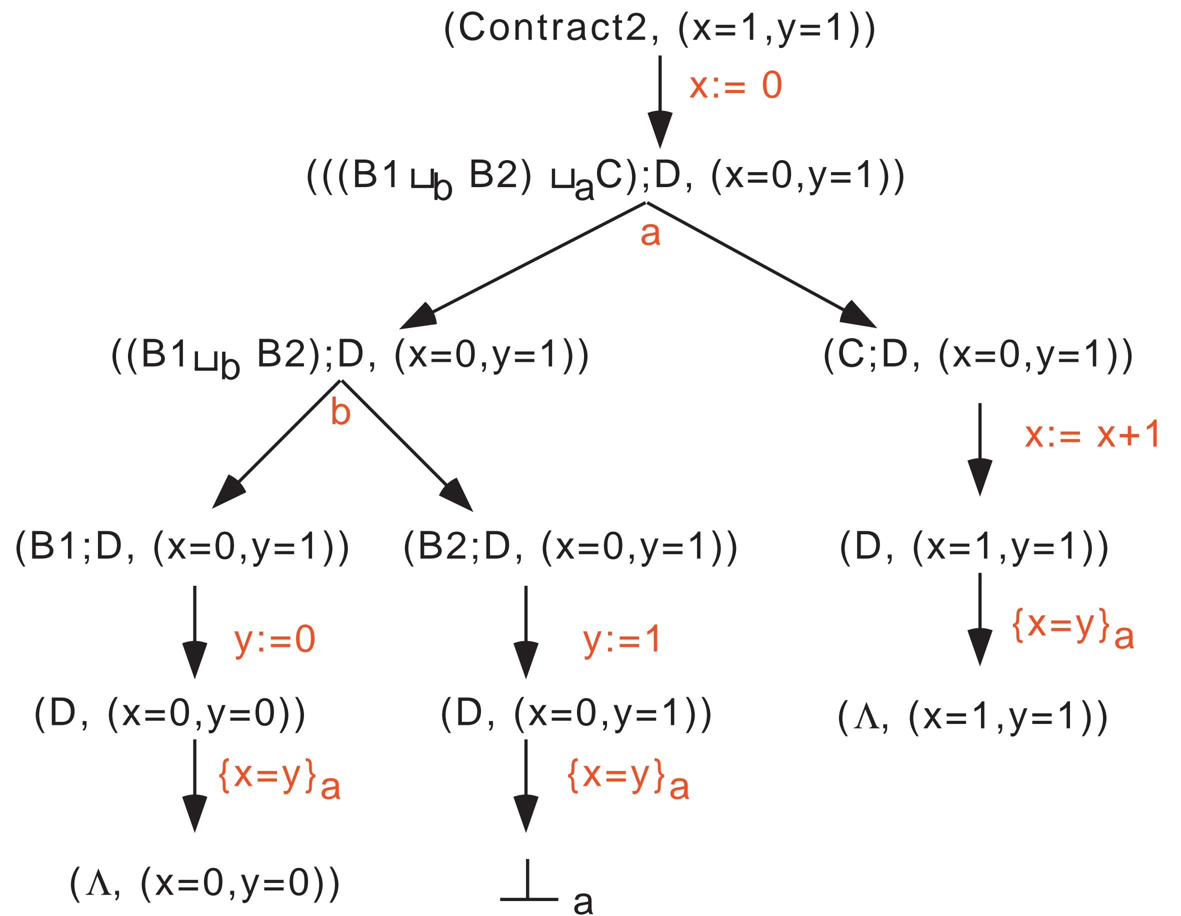
$$(\Lambda,\ (x = 1, y = 1))$$

All possible derivations

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

where

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



Operational semantics

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

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

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

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

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

where

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

All behaviors of contracts are finite.

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

Contracts vs. programs

- Contracts generalize the traditional notion of a program to allow for any number of **agents**. The choices made by the agents determine how the computation proceeds.

- Batch oriented programs and interactive programs 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

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

By looking at the state component of the final configurations, we can see what outcomes (final states) are possible, if all agents cooperate.

In reality 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.

Question?: What can one agent (or a coalition of agents) achieve with a contract.

Establishing goals

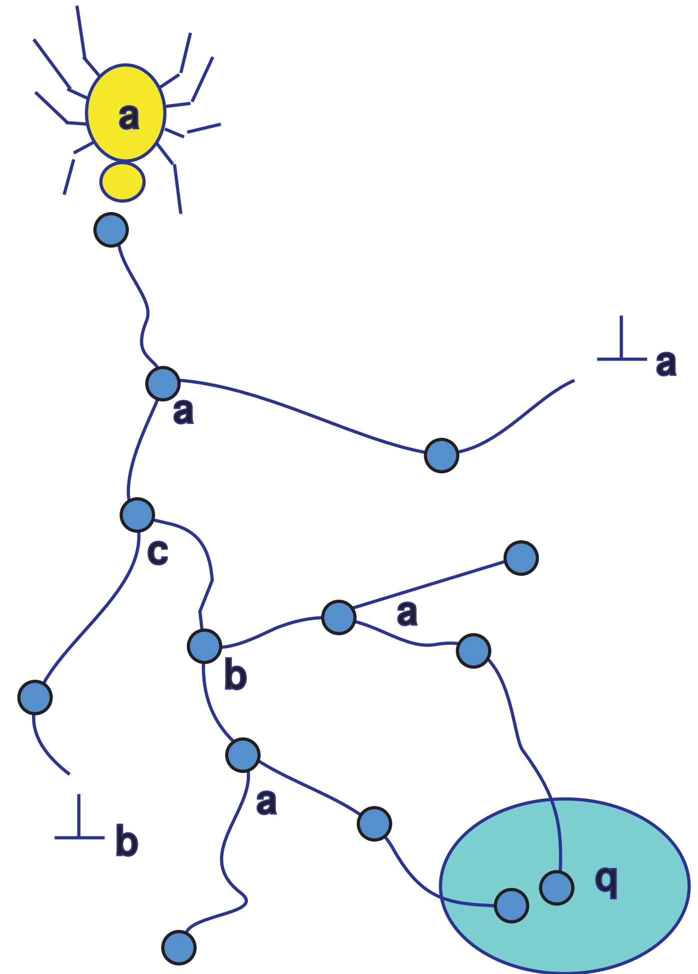
Agent a can **establish** condition q with contract S in initial state σ, denoted

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

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 must breach the contract.



Example: Contract1

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

Establishing goals:

$$(x = 3, y = 1) \ \{\!|\, \text{Contract1} \,|\!\}_a \ x = 1$$

$$(x = 3, y = 1) \ \{\!|\, \text{Contract1} \,|\!\}_a \ x = y$$

$$(x = 3, y = 2) \ \{\!|\, \text{Contract1} \,|\!\}_a \ x = y$$

but not

$$(x = 3, y = 2) \ \{\!|\, \text{Contract1} \,|\!\}_a \ x = 1$$

Example: Contract2

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

Establishing goals:

$$(x = 1, y = 1) \ \{\!|\, \text{Contract2} \,|\!\}_a \ x = y$$

$$(x = 1, y = 1) \ \{\!|\, \text{Contract2} \,|\!\}_b \ x = y$$

Example: Interaction

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

$$(x = 7) \;\{\!|\, \text{Interaction} \,|\!\}_a \; x \leq 0$$

$$(x = 7) \;\{\!|\, \text{Interaction} \,|\!\}_b \; x = 0$$

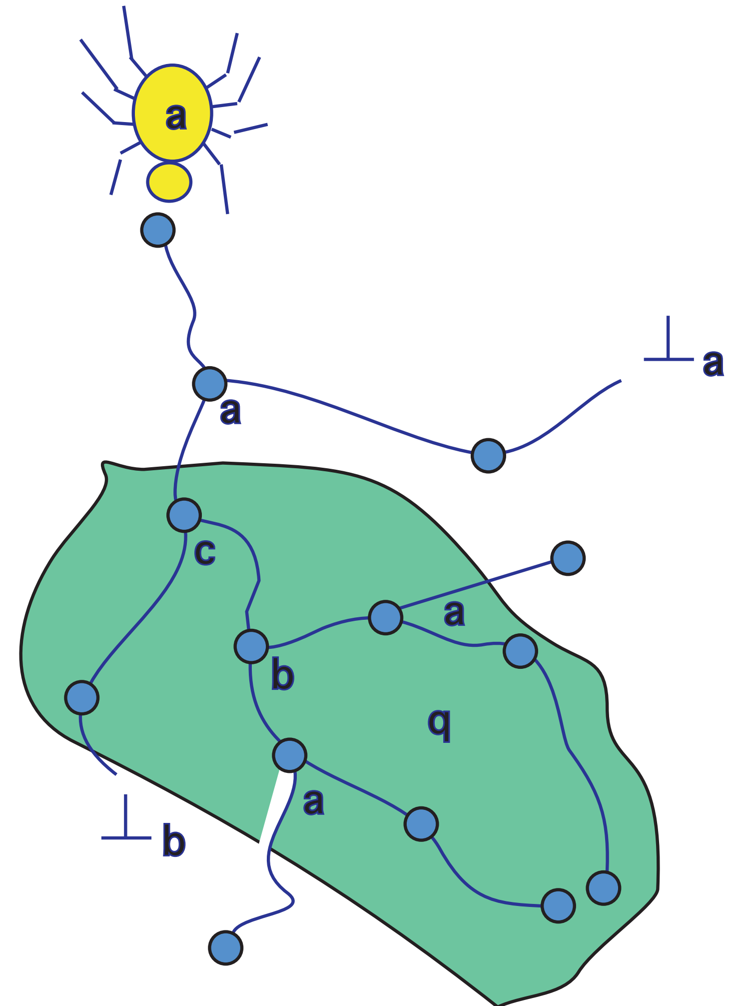
Enforcing correct behavior

We could also consider more general notions of what an agent can achieve, like enforcing that, as long as none of the other agents breach the contract,

- each state reached satisfies some condition q (i.e., $\sigma \; \{\!|\, S \,|\!\}_a \; \Box q$ holds), or
- eventually some state will satisfy condition q (i.e., that $\sigma \; \{\!|\, S \,|\!\}_a \; \Diamond q$ holds), or
- q holds until r holds ($\sigma \; \{\!|\, S \,|\!\}_a \; q \, \mathcal{U} \, r$), or
- q always leads to r ($\sigma \; \{\!|\, S \,|\!\}_a \; \Box(q \Rightarrow \Diamond r)$).

In figure, a can ensure that $\Diamond\Box q$.

We will concentrate on establishing goals (post-conditions) in the sequel, and consider enforcement of temporal properties if time permits.



Correctness

Agent a can **establish** condition q with contract S whenever initially p:

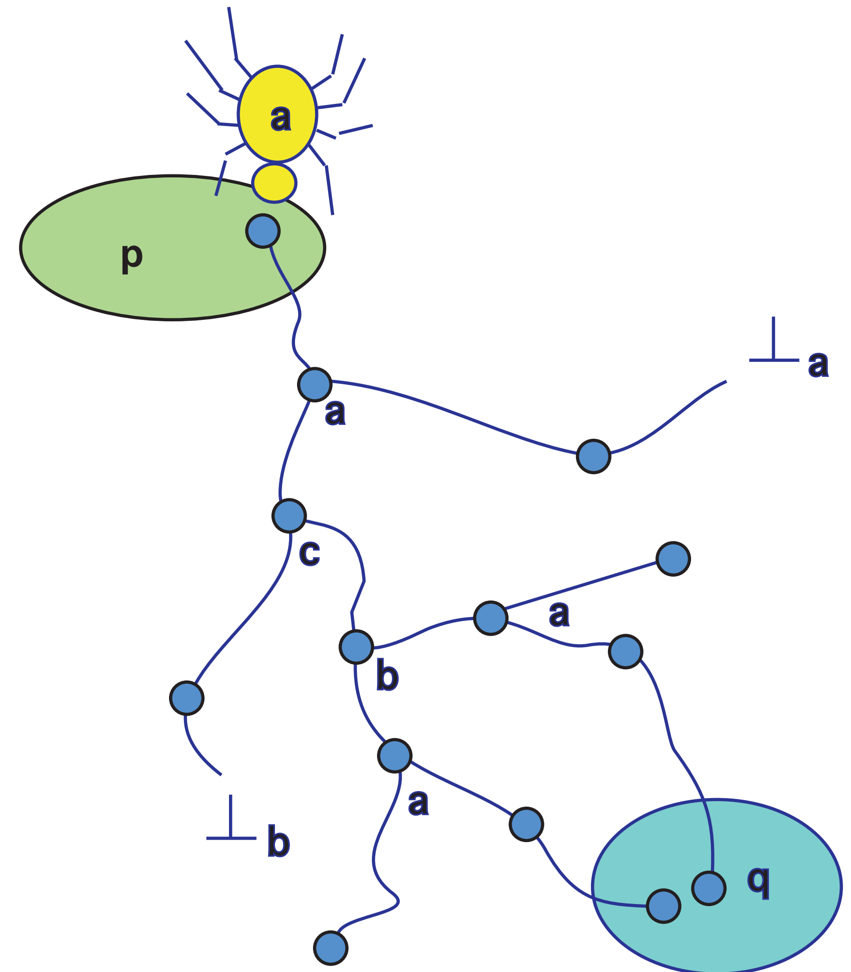
$$
\begin{array}{ll}
& p \;\{\!|\; S \;|\!\}_a \; q \\
\triangleq & (\forall \sigma \in p \bullet \sigma \;\{\!|\; S \;|\!\}_a \; q)
\end{array}
$$

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 agent a, when $p \;\{\!|\; S \;|\!\}_a \; q$. (Might be better to say that S is **appropriate** for agents to reach the goal)

Example:

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



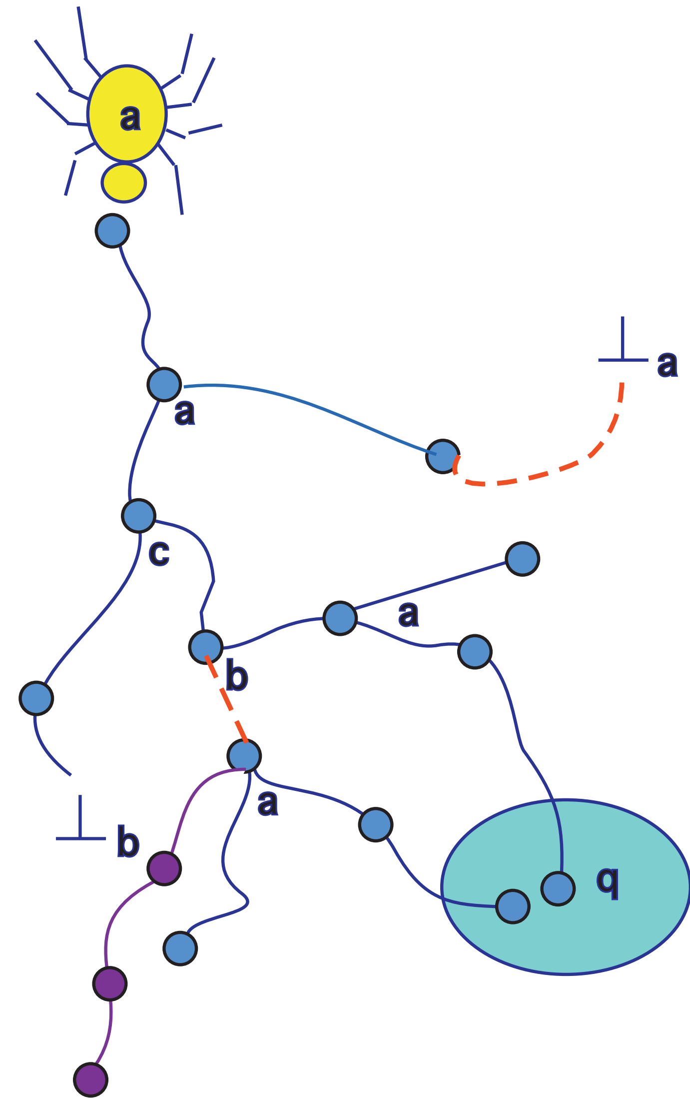
Improving contracts

Contract S' is an **improvement** of contract S for agent a if any condition that a can establish with S can also be established with S':

$$\begin{array}{rl} & S \sqsubseteq_a S' \\ \triangleq & (\forall \sigma \forall q \bullet \sigma \ \{\!|\ S \ |\!\}_a \ q \Rightarrow \sigma \ \{\!|\ S' \ |\!\}_a \ q) \end{array}$$

We will say that S is **refined by** S' (for agent a) when $S \sqsubseteq_a S'$ holds.

Contracts S and S' are equivalent from the point of view of agent a, $S =_a S'$, if $S \sqsubseteq_a S'$ and $S' \sqsubseteq_a S$.



Refinement example

Refinement for agent a means making it easier for a to achieve whatever goal it desires, by

- adding 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.

Example refinement

We have that

$$\text{Contract2} \sqsubseteq_a \text{Contract3}$$

where

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

and

$$\begin{aligned}\text{Contract3} \quad = \quad & x := 0; \\ & \{y > 0\}_b; \\ & (y := 1 \sqcup_a x := x + 1 \sqcup_a x := x + 2); \\ & \{y \leq x\}_a.\end{aligned}$$

Taking sides

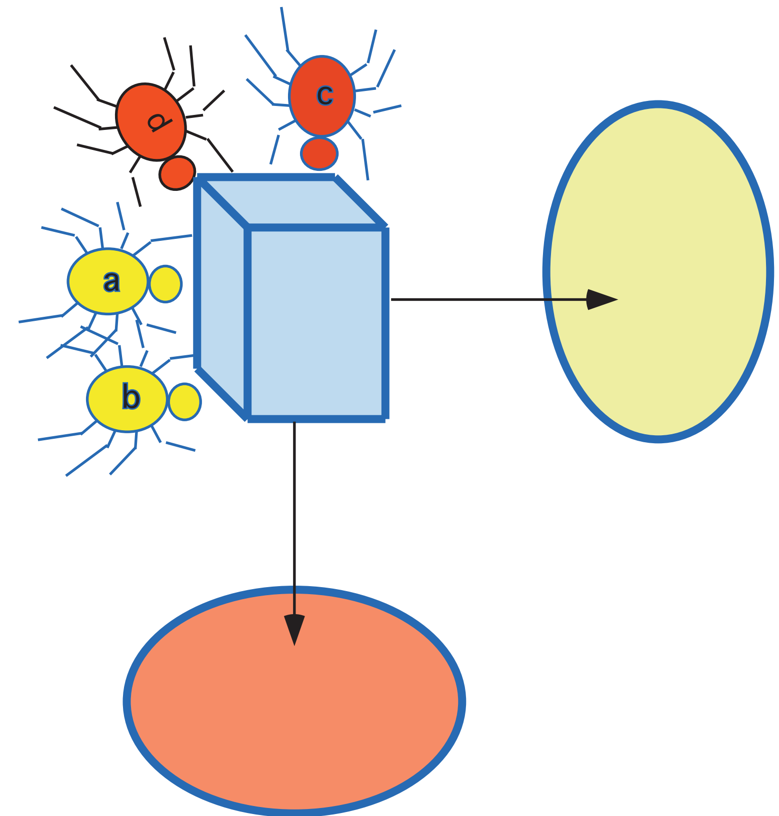
We pick out one or more agents whose side we are taking, and assume that these agents will co-operate in order to achieve a common goal.

The other agents may have other goals.

To prepare for the worst, we assume that the other agents are hostile to the goals that our agents have and try to prevent them from reaching this goal.

We call our agents collectively the **angel** and the other agents collectively the **demon**.

An **angelic choice** is made by our agents, and a **demonic choice** by the other agents.



Game interpretation

We 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 gives the **rules** of the game.
- The goal of the game is to reach a **goal**, some final state in q.

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

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

Winning strategies

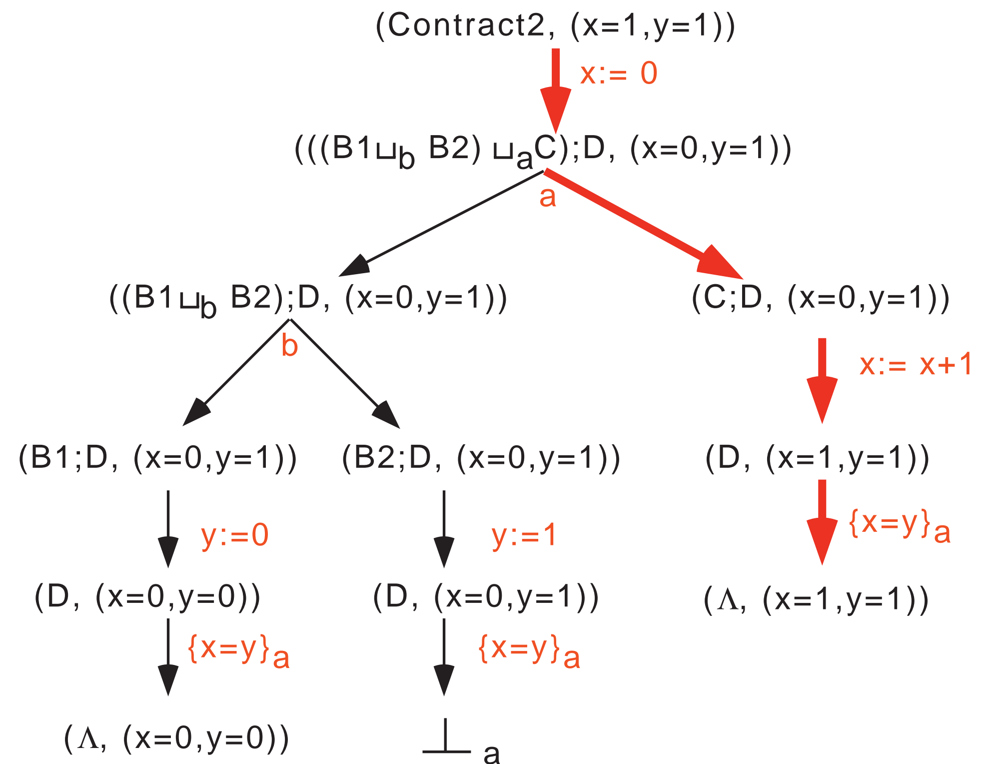
Agent a makes its choices according to a **strategy**: a function that for every configuration of the form $(S_1 \sqcup_a S_2, \gamma)$ returns either (S_1, γ) or (S_2, γ).

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

Thus $\sigma\ \{\!\mid S \mid\!\}_a\ q$ 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 op. $S.\sigma$ and q.

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

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

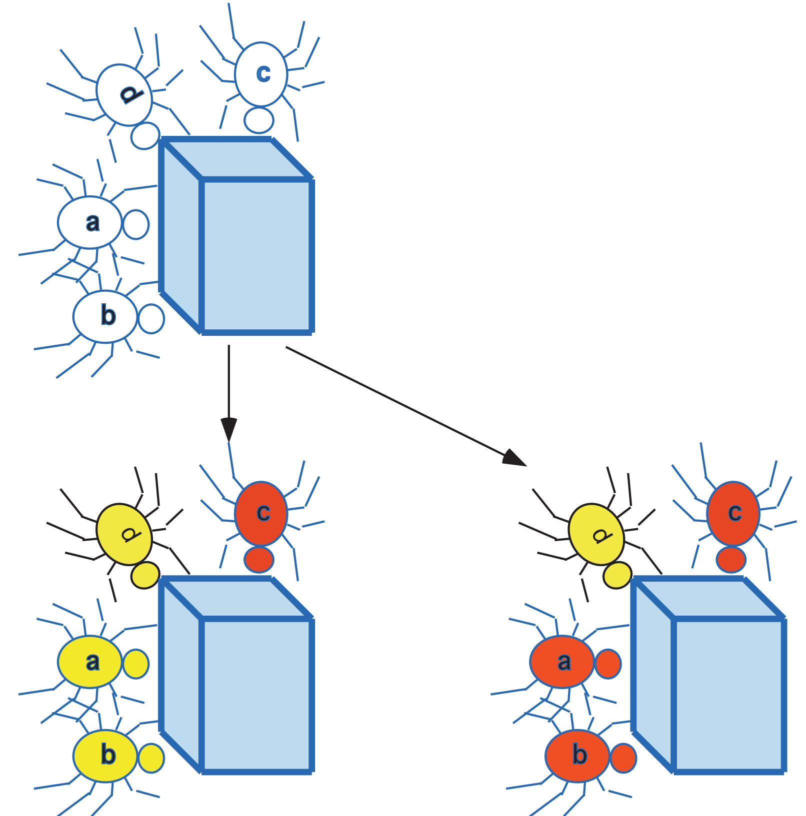


Symmetric and asymmetric system description

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

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

After we group the agents into angels and demons, we have an **asymmetric** (**game**) view of the system.



Simplified syntax

After grouping agents into angels and demons, we only have two agents left. The notation can then be simplified. We write

$$a \sqcup b \quad \triangleq \quad a \sqcup_{angel} b \qquad \textbf{angelic choice}$$

$$a \sqcap b \quad \triangleq \quad a \sqcup_{demon} b \qquad \textbf{demonic choice}$$

$$\{p\} \quad \triangleq \quad \{p\}_{angel} \qquad \textbf{assertion}$$

$$[p] \quad \triangleq \quad \{p\}_{demon} \qquad \textbf{assumption}$$

We also take the angels side when analyzing a contract:

$$\sigma \; \{\!|\; S \;|\!\}\; q \quad \triangleq \quad \sigma \; \{\!|\; S \;|\!\}_{angel}\; q \qquad \textbf{correctness}$$

$$S \sqsubseteq S' \quad \triangleq \quad S \sqsubseteq_{angel} S' \qquad \textbf{refinement}$$

Simpler syntax for contract statements with angels and demons:

$$S \quad ::= \quad \langle f \rangle \mid \{p\} \mid [p] \mid S_1 ; S_2 \mid S_1 \sqcup S_2 \mid S_1 \sqcap S_2$$

Notice that this is an **asymmetric** description of the system.

Assumptions

The **assumption** $[p]$ is a condition that the angel expects to hold in a given state. E.g.,

$$[x + y = 0]$$

states that the sum of x and y in the state is assumed to be zero.

- If the assumption holds at the specified place then the state is unchanged, and the angel carries on with the rest of the contract.
- If the assumption does not hold, then the angel is **released** from the contract (because the demon has breached the contract).

The assumption $[\mathsf{true}]$ is always satisfied.

The assumption $[\mathsf{false}]$ is an **impossible assumption**. It is never satisfied, and always releases the agent from the contract. Often refered to as magic, the so-called **miraculous statement**.

Weakest preconditions

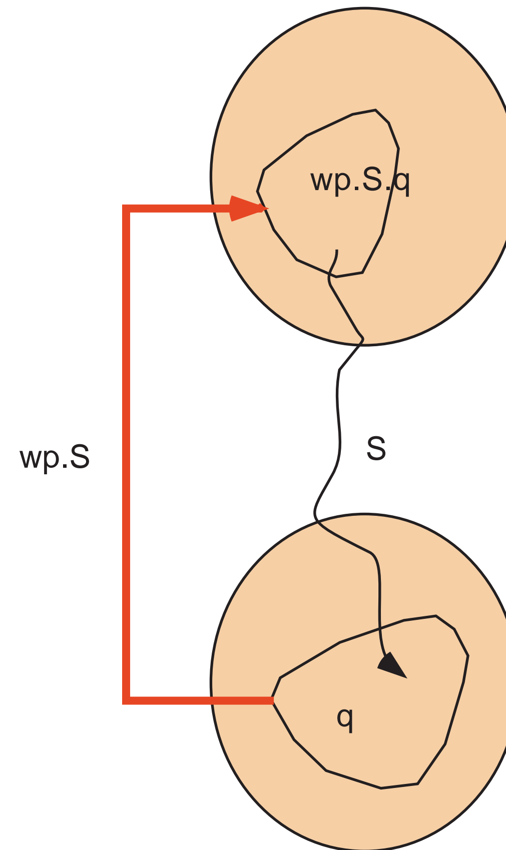
When can angel reach its goal

Let S be a contract statement. We want to **compute** the set of initial states from which the angel has a winning strategy to reach goal q with contract S.

We do this by defining a function $wp.\,S$ (by induction over the structure of S) such that it satisfies

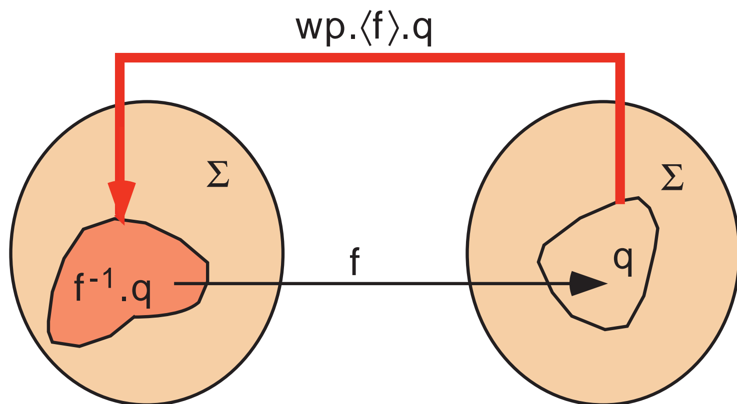
$$wp.\,S.\,q \quad = \quad \{\sigma \mid \sigma \; \{\!| \, S \, |\!\} \; q\}$$

This is called the **weakest precondition** that guarantees that the angel can achieve postcondition q.



Computing $wp.S.q$ for update statements

$$
\begin{array}{ll}
& \sigma \in wp.\langle f \rangle.q \\
\equiv & \{\text{operational semantics}\} \\
& f.\sigma \in q \\
\equiv & \{\text{inverse image: } f^{-1}.q = \{\sigma \mid f.\sigma \in q\}\} \\
& \sigma \in f^{-1}.q
\end{array}
$$



Computing $wp.\,(x := e).\,q$

We compute the weakest precondition for an assignment $x := e$ to establish a postcondition q, assuming that q is a boolean expression and e an expressions.

$$
\begin{aligned}
& wp.\,\langle x := e\rangle.\,q.\,\sigma \\
= \quad & \{\text{weakest precondition for update statements}\} \\
& (x := e)^{-1}.\,q.\,\sigma \\
= \quad & \{\text{definition of function preimage}\} \\
& q.\,((x := e).\,\sigma) \\
= \quad & \{\text{substitution lemma}\} \\
& q[x := e].\,\sigma
\end{aligned}
$$

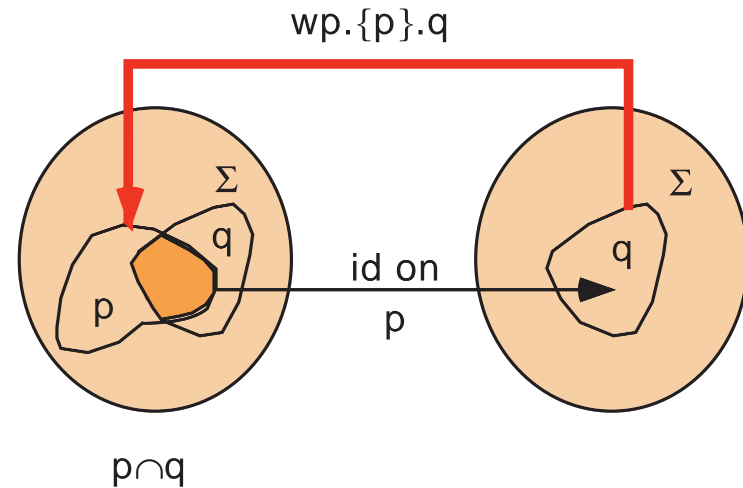
Thus, we have the *assignment axiom* (proved here as a theorem)

$$
wp.\,\langle x := e\rangle.\,q \quad = \quad q[x := e]
$$

Computing $wp.\, S.\, q$ for assertion and assumption

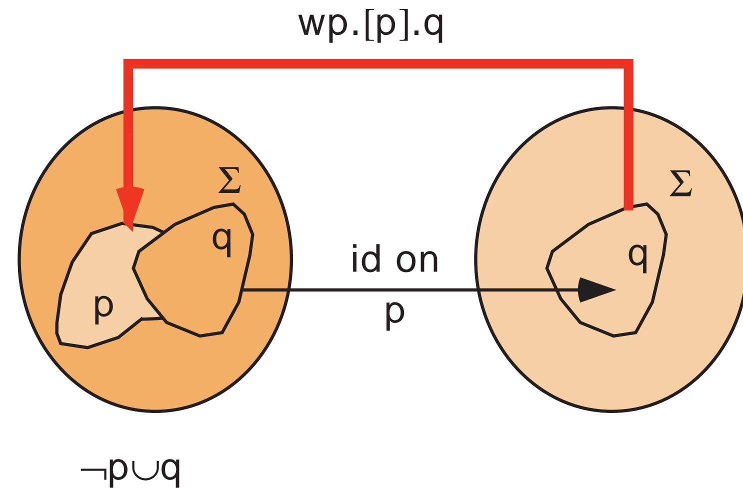
Assertion:

$$
\begin{aligned}
& \sigma \in wp.\, \{p\}.\, q \\
\equiv \quad & \{\text{Operational semantics}\} \\
& \sigma \in p \wedge \sigma \in q \\
\equiv \quad & \{\text{set theory}\} \\
& \sigma \in p \cap q
\end{aligned}
$$



Assumption:

$$
\begin{aligned}
& \sigma \in wp.\, [p].\, q \\
\equiv \quad & \{\text{Operational semantics}\} \\
& \sigma \notin p \vee \sigma \in q \\
\equiv \quad & \{\text{set theory}\} \\
& \sigma \in \neg p \cup q
\end{aligned}
$$



Computing $wp.\,S$, sequential composition.

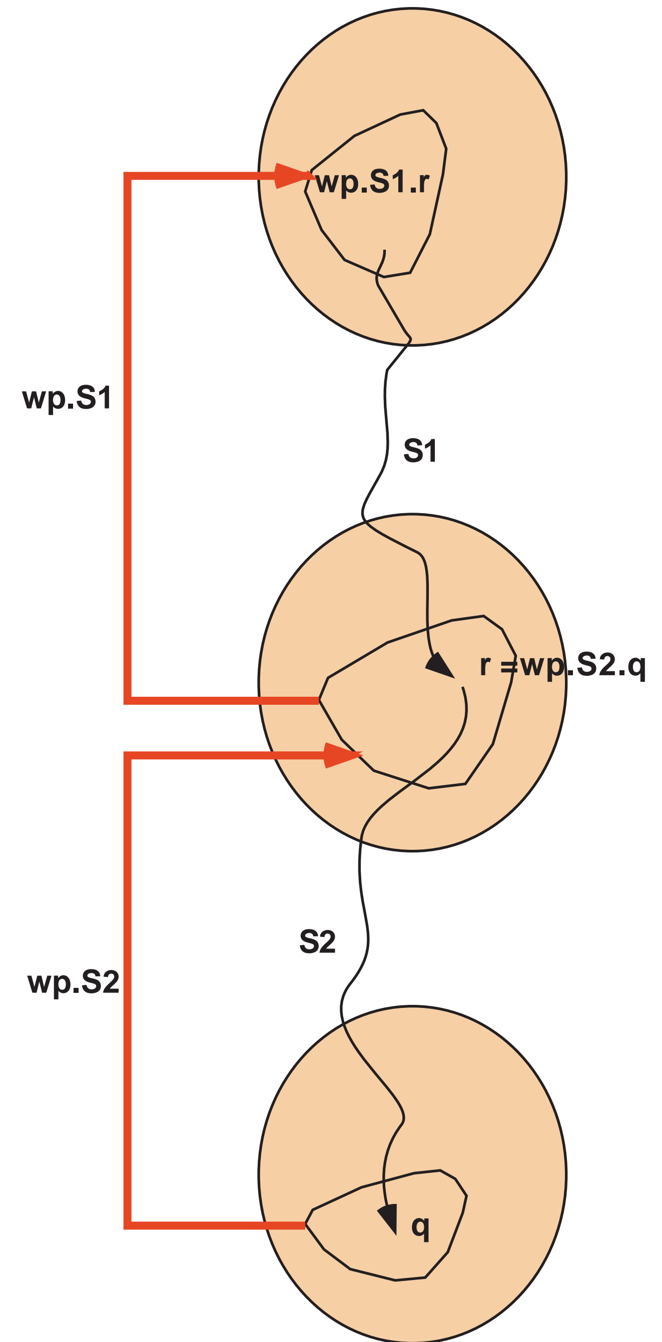
Sequential composition:

$$
\begin{array}{ll}
 & \sigma \in wp.\,(S_1; S_2).\,q \\
\equiv & \{\text{Operational semantics}\} \\
 & (\exists\, r \bullet \sigma \in wp.\,S_1.\,r \wedge (\forall\, \sigma' \in r \bullet \sigma' \in wp.\,S_2.\,q)) \\
\equiv & \{\text{motivation below}\} \\
 & \sigma \in wp.\,S_1.\,(wp.\,S_2.\,q)
\end{array}
$$

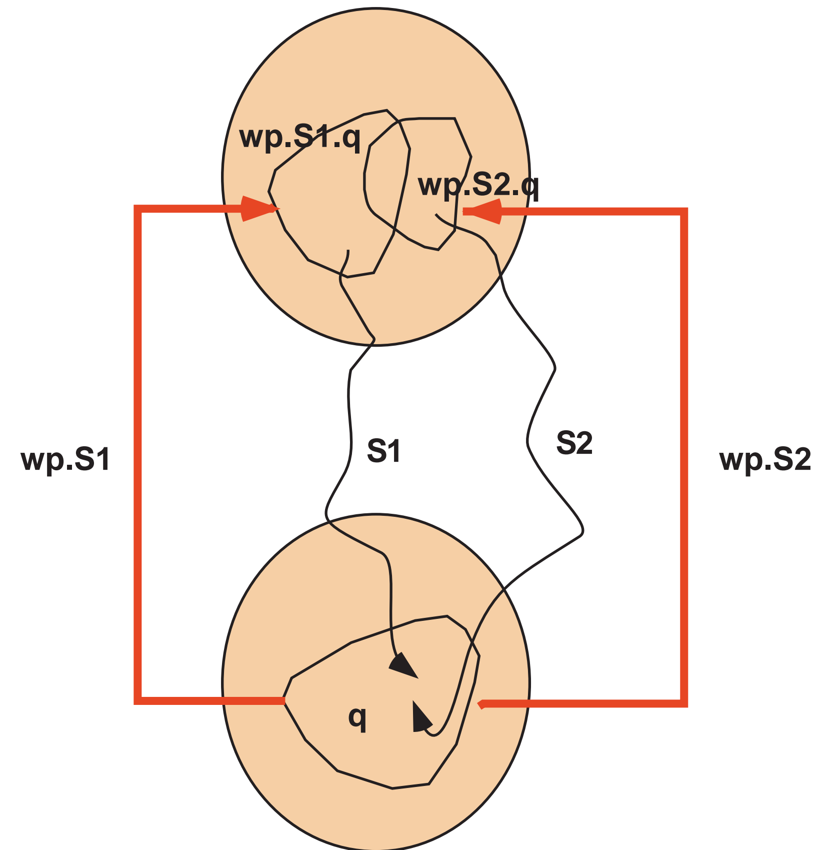
Motivation:

($\Rightarrow$) Assuming that S_1 is monotonic: $p \subseteq q \Rightarrow S_1.\,p \subseteq S_2.\,q$.

($\Leftarrow$) Witness $r = wp.\,S_2.\,q$.



Computing $wp.\,S$, choice



$S_1 \sqcup S_2$ is guaranteed to lead to q, if either S_1 or S_2 are guaranteed to lead to q.

$$
\begin{array}{ll}
 & \sigma \in wp.\,(S_1 \sqcup S_2).\,q \\
\equiv & \{\text{Operational semantics}\} \\
 & \sigma \in wp.\,S_1.\,q \vee \sigma \in wp.\,S_2.\,q \\
\equiv & \{\text{set theory}\} \\
 & \sigma \in wp.\,S_1.\,q \cup wp.\,S_2.\,q
\end{array}
$$

For $S_1 \sqcap S_2$ if both S_1 and S_2 must lead to q.

$$
\begin{array}{ll}
 & \sigma \in wp.\,(S_1 \sqcap S_2).\,q \\
\equiv & \{\text{Operational semantics}\} \\
 & \sigma \in wp.\,S_1.\,q \wedge \sigma \in wp.\,S_2.\,q \\
\equiv & \{\text{set theory}\} \\
 & \sigma \in wp.\,S_1.\,q \cap wp.\,S_2.\,q
\end{array}
$$

Define wp

We then define wp by induction on the structure of the contract:

$$
\begin{array}{rcl}
wp.\,\langle f \rangle & = & (\lambda\, q \bullet\ f^{-1}.\, q) \\
wp.\,\{p\} & = & (\lambda\, q \bullet\ p \cap q) \\
wp.\,[p] & = & (\lambda\, q \bullet\ \neg p \cup q) \\
wp.\,(S_1; S_2) & = & (\lambda\, q \bullet\ wp.\, S_1.\,(wp.\, S_2.\, q)) \\
wp.\,(S_1 \sqcup S_2) & = & (\lambda\, q \bullet\ wp.\, S_1.\, q \cup wp.\, S_2.\, q) \\
wp.\,(S_1 \sqcap S_2) & = & (\lambda\, q \bullet\ wpr.\, S_1.\, q \cap wp.\, S_2.\, q)
\end{array}
$$

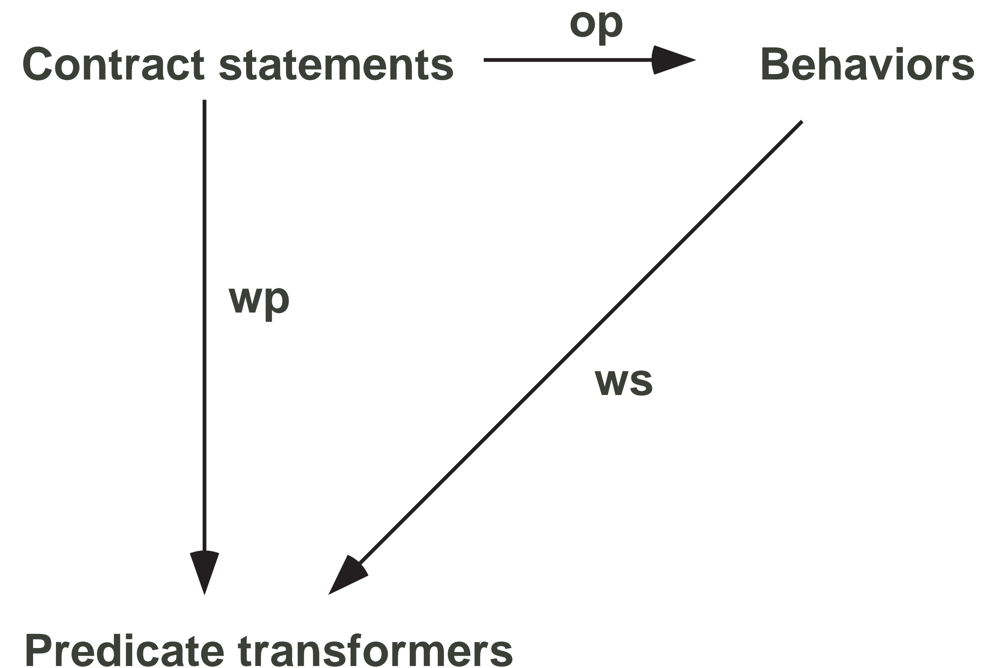
Winning strategy theorem

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

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

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

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



Example

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

$$\begin{array}{rcl} S & = & (x := x + 1 \sqcup x := x + 2) \\ & & (x := x - 1 \sqcap x := x - 2) \end{array}$$

We have that

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

Similarly, we compute that

$$wp.\,(x := x + 1 \sqcup x := x + 2).\,(x \geq 2) \quad = \quad x \geq 0$$

Hence,

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

Correctness and refinement

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

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

The following corollary holds for **refinement**:

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

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 $wp.\,S$.

Determining achievability

- 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 introduced here generalizes traditional correctness, where only system can make choices (demonic nondeterminism). Total correctness then means achieving a postcondition no matter how the system resolves its choices.
- The notion used here also defines correctness for interactive systems, where 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 when both user and system can make choices as the existence of a winning strategy for the user to reach its goal.
- The general case can, e.g., be used to define correctness of an interactive system with concurrency (e.g., background processes)

Generalizing contracts

Conditional statements

Can define **conditional statement** by

$$
\begin{aligned}
& \textsf{if} \;\; x \geq 0 \;\textsf{then}\; x := x + 1 \;\;\textsf{else}\;\; x := x - 1 \;\;\textsf{fi} \\
= \quad & \{x \geq 0\}_a ; x := x + 1 \sqcup_a \{x < 0\}_a ; x := x - 1
\end{aligned}
$$

Agent a chooses the alternatives for which the guarding assertion is true; choosing the other alternative would breach the contract. (Choice of agent a does not matter here)

Using angels and demons, we have that

$$
\begin{aligned}
& \{x \geq 0\} ; x := x + 1 \sqcup \{x < 0\} ; x := x - 1 \\
= \quad & [x \geq 0] ; x := x + 1 \sqcap [x < 0] ; x := x - 1
\end{aligned}
$$

Conditional with **demonic choice**:

$$
\begin{aligned}
& \textsf{if}\; b_1 \rightarrow S_1 \,[\!]\; \ldots \,[\!]\; b_n \rightarrow S_n \;\textsf{fi} \\
= \quad & \{b_1 \cup \ldots \cup b_n\} ; ([b_1] ; S_1 \sqcap \ldots \sqcap [b_n] ; S_n)
\end{aligned}
$$

Angelic conditional statements

$$
\begin{array}{rl}
& \mathsf{if}\ b_1 :: S_1 [\!]\ \ldots [\!]\ b_n :: S_n\ \mathsf{fi} \\
= & [b_1 \cup \ldots \cup b_n]; (\{b_1\}; S_1 \sqcup \ldots \sqcup \{b_n\}; S_n)
\end{array}
$$

Models user choosing an item from a **menu**.

- b_i is conditions for menu item i to be enabled.
- S_i is action taken when item i is chosen.
- System guarantees that some menu item is enabled
- User can freely choose any enabled menu item.
- User breaches contract if she chooses an item that is not enabled.

Relational assignment

We generalize ordinary (functional) assignment to **relational assignment**:

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

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:

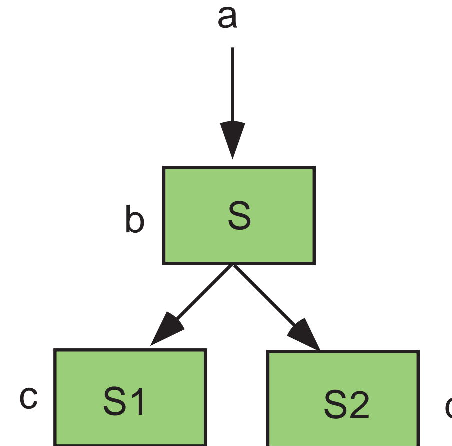
$$\{x := x' \mid 0 \leq x' < x\}_a =$$

"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	$x := 0 \sqcup_a x := 1$

Example contract

- a is a user of a program
- b is the main program module
- c and d are submodules of the program.



Contract:

- a chooses some input which must be between 0 and 100
- Then b chooses whether to pass on the value to
  - c (which is permitted if the value is below 50), or
  - d (which is always permitted).

$$S = \{x := x' \mid 0 \leq x' \leq 100\}_a; (\{x < 50\}_b; S_1 \sqcup_b S_2)$$

Input statements and specifications

User can influence the computation by giving **input** to the program during its execution. This can be achieved by a relational assignment.

Example: Compute the square root with given approximation.

$$\{x, e := x', e' | x' \geq 0 \wedge e' > 0\}_{user};$$
$$\{x := x' | -e < x'^2 - x < e\}_{system}$$

- The first statement specifies the **user's responsibility** (to give an input value that satisfies the given conditions)
- the second statement specifies the **system's responsibility** (to compute a new value for x that satisfies the given condition).

This contract thus **specifies** the interaction between the user and the computing system.

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:

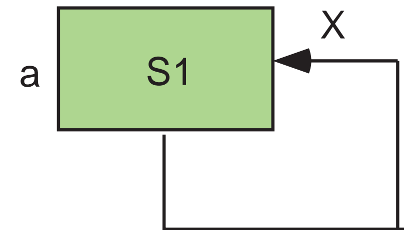
$$(\sqcup_a i \in \mathsf{Nat} \bullet x := x + i) \quad = \quad \{x := x' \mid x' \geq x\}_a$$

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



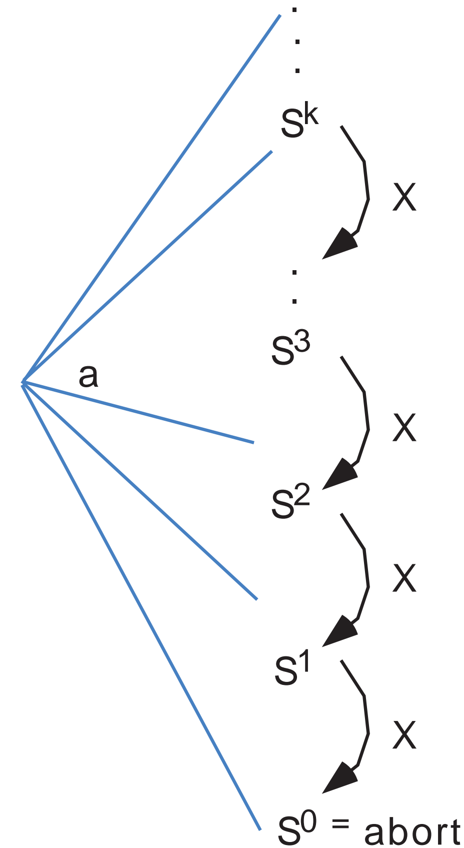
Defining recursion

Can define recursion in terms of arbitrary choice. Let

$$S^0 = \mathsf{abort}_a$$
$$S^{i+1} = S[X := S^i], \quad i = 0, 1, 2, \ldots$$

Intuitively, S^i unfolds recursion at most $i-1$ times, $i = 1, 2, \ldots$, before aborting. Then

$$(\mathsf{rec}_a X \bullet S) \quad =_a \quad (\sqcup_a \; i \in \mathsf{Nat} \bullet S^i)$$



Motivation

- Agent will try to reach goal by unfolding recursion as many times as possible
- If in some initial state n unfoldings are sufficient to reach goal, then agent can as well choose S^{n+1} directly, the effect is the same.
- If recursion does not terminate, then no number of unfoldings is sufficient. Agent can then as well choose any S^i directly, each S^i will eventually abort, as will the whole choice statement.

Unbounded nondeterminacy

Above characterization assume that there are no situations where some agent can choose between an infinite number of different states (**unbounded nondeterminacy**). This is, however, a very common situation in specifications, e.g.,

$$\{x := x' \mid x' > x\}_a$$

is unbounded.

For unbounded nondeterminacy, we need to use the ordinals. Define

$$\begin{array}{rcl} S^0 & = & \mathsf{abort}_a \\ S^{\gamma+1} & = & S[X := S^\gamma], \\ S^\beta & = & (\sqcup_a\ \gamma < \beta \bullet S^\gamma), \quad \beta \text{ is limit ordinal} \end{array}$$

Then there exists a least ordinal α such that $S^{\alpha+1} = S^\alpha$. Define

$$(\mathsf{rec}_a X \bullet S) \quad =_a \quad S^\alpha$$

Angels and demons

Can again simplify notation when there are only two agents involved:

$$
\begin{array}{rcll}
\{R\} & \triangleq & \{R\}_{angel} & \textbf{angelic update} \\
[R] & \triangleq & \{R\}_{demon} & \textbf{demonic update} \\
(\sqcup i \in I \bullet S_i) & \triangleq & (\sqcup_{angel} i \in I \bullet S_i) & \textbf{arbitrary angelic choice} \\
(\sqcap i \in I \bullet S_i) & \triangleq & (\sqcup_{demon} i \in I \bullet S_i) & \textbf{arbitrary demonic choice} \\
(\mu X \bullet S) & \triangleq & (\mathsf{rec}_{angel} X \bullet S) & \textbf{least fixpoint} \\
(\nu X \bullet S) & \triangleq & (\mathsf{rec}_{demon} X \bullet S) & \textbf{greatest fixpoint}
\end{array}
$$

General syntax for contract statements with angels and demons:

$$
\begin{array}{rcl}
S & ::= & \langle f \rangle \mid \{p\} \mid [p] \mid \\
 & & \{R\} \mid [R] \mid \\
 & & S_1 ; S_2 \mid S_1 \sqcup S_2 \mid S_1 \sqcap S_2 \mid \\
 & & (\sqcup i \in I \bullet S_i) \mid (\sqcap i \in I \bullet S_i) \mid \\
 & & X \mid (\mu X \bullet S) \mid (\nu X \bullet S)
\end{array}
$$

Weakest preconditions for generalized constructs

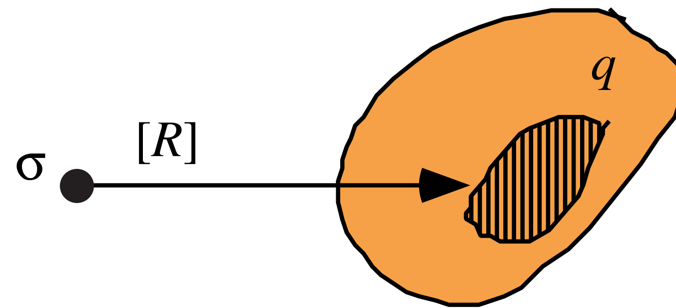
We determine the weakest preconditions for the generalized contracts.

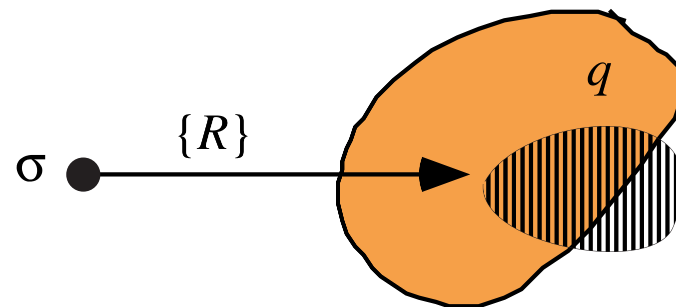
$$
\begin{aligned}
wp.\{R\}.q.\sigma &= (\exists \gamma \in \Gamma \bullet R.\sigma.\gamma \wedge q.\gamma) \\
wp.[R].q.\sigma &= (\forall \gamma \in \Gamma \bullet R.\sigma.\gamma \Rightarrow q.\gamma) \\
wp.(\sqcup i \in I \bullet S_i).q.\sigma &= (\exists i \in I \mid wp.S_i.q.\sigma) \\
wp.(\sqcap i \in I \bullet S_i).q.\sigma &= (\forall i \in I \mid wp.S_i.q.\sigma) \\
wp.(\mu X \bullet S).q.\sigma &= (\mu.(\lambda X \bullet wp.S_i)).q.\sigma \\
wp.(\nu X \bullet S).q.\sigma &= (\nu.(\lambda X \bullet wp.S_i)).q.\sigma
\end{aligned}
$$

The last two definitions assume that $wp.X = X$.

$F = (\lambda X \bullet wp.S_i)$ is a function from predicate transformers to predicate transformers, and $\mu.F$ is the **least fixpoint** of F, and $\nu.F$ is the **greatest fixpoint** of F.

Weakest precondition for update statement





Predicate transformer statements

Let us define the following predicate transformers, called **basic statements**:

$$
\begin{aligned}
\langle f \rangle &\;\triangleq\; (\lambda q \bullet f^{-1}.\,q) \\
\{p\} &\;\triangleq\; (\lambda q \bullet p \cap q) \\
[p] &\;\triangleq\; (\lambda q \bullet \neg p \cup q) \\
\{R\} &\;\triangleq\; (\lambda q\, \sigma \bullet (\exists \gamma \in \Gamma \bullet R.\,\sigma.\,\gamma \wedge q.\,\gamma)) \\
[R] &\;\triangleq\; (\lambda q\, \sigma \bullet (\forall \gamma \in \Gamma \bullet R.\,\sigma.\,\gamma \Rightarrow q.\,\gamma))
\end{aligned}
$$

Define the following operations on predicate transformers:

$$
\begin{aligned}
(F \sqcap G).\,q &\;\triangleq\; F.\,q \cap G.\,q \\
(F \sqcup G).\,q &\;\triangleq\; F.\,q \cup G.\,q \\
(F; G).\,q &\;\triangleq\; F.\,(G.\,q)
\end{aligned}
$$

A **statement** is a predicate transformer term that is built out of basic statements using these operations.

Contracts as statements

We have a direct interpretation of contracts as statements:

$$\begin{array}{rcl}
wp.\langle f \rangle & = & \langle f \rangle \\
wp.\{p\} & = & \{p\} \\
wp.[p] & = & [p] \\
wp.\{R\} & = & \{R\} \\
wp.[R] & = & [R] \\
wp.(S;T) & = & wp.S; wp.T \\
wp.(S \sqcup T) & = & wp.S \sqcup wp.T \\
wp.(S \sqcap T) & = & wp.S \sqcap wp.T \\
wp.(\sqcup\, i \in I \bullet S_i) & = & (\sqcup\, i \in I \bullet wp.S_i) \\
wp.(\sqcap\, i \in I \bullet S_i) & = & (\sqcap\, i \in I \bullet wp.S_i)
\end{array}$$

Hence, we often identify a contract statement with only two agents, an angel and a demon, with a predicate transformer statement, denoting both by $S, T, \ldots$.

Iteration

Iteration by a:

$$\begin{array}{ll} & \mathsf{while}\ _a\ g\ \mathsf{do}\ \ S\ \ \mathsf{od} \\ = & (\mathsf{rec}_a X \bullet\ \ \mathsf{if}\ \ g\ \mathsf{then}\ S; X\ \ \mathsf{else}\ \ \mathsf{skip}\ \ \mathsf{fi}) \end{array}$$

Demonic iteration

$$\begin{array}{ll} & \mathsf{do}\ \ g_1 \rightarrow S \| \ \ldots \| \ g_m \rightarrow S_m\ \ \mathsf{od} \\ = & (\mu X \bullet [g_1]; S; X \sqcap \ldots \sqcap [g_m]; S_m; X \sqcap [\neg g_1 \cap \ldots \cap \neg g_m]) \end{array}$$

Angelic iteration (User can always terminate)

$$\begin{array}{ll} & \mathsf{do}\ \ g_1 :: S \| \ \ldots \| \ g_m :: S_m\ \ \mathsf{od} \\ = & (\mu X \bullet \{g_1\}; S_1; X \sqcup \ldots \sqcup \{g_m\}; S_m; X \sqcup \mathsf{skip}) \end{array}$$

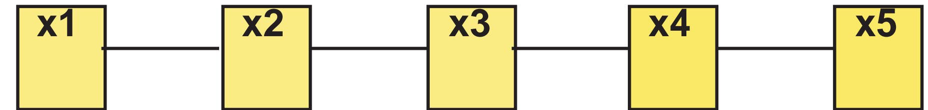
Angelic iteration (User can terminate when there is no other choice)

$$= \quad (\mu X \bullet \{g_1\}; S_1; X \sqcup \ldots \sqcup \{g_m\}; S_m; X \sqcup \{\neg g_1 \cap \ldots \cap \neg g_m\})$$

Parallel sort

Sort items $x_1, \ldots, x_5$.

- User initializes state.
- Then concurrent execution of exchange actions starts.
- If $x_i > x_{i+1}$ holds, then system can choose swap $x_i, x_{i+1} := x_{i+1}, x_i$ without breaching contract, $i = 1, 2, 3, 4$.
- The system terminates when none of the swap actions is enabled.



$\{x_1, x_2, x_3, x_4, x_5 := a_1, a_2, a_3, a_4, a_5 \mid \mathsf{true}\};$

$\mathsf{do}\ x_1 > x_2 \rightarrow x_1, x_2 := x_2, x_1$

$\parallel \quad x_2 > x_3 \rightarrow x_2, x_3 := x_3, x_2$

$\parallel \quad x_3 > x_4 \rightarrow x_3, x_4 := x_4, x_3$

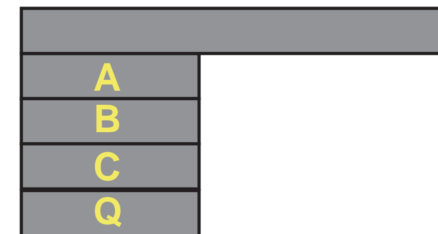
$\parallel \quad x_4 > x_5 \rightarrow x_4, x_5 := x_5, x_4$

od

Event loop

User can choose from menu items A, B, or C, or she can choose to terminate Q.

- System initializes state.
- Then event loop starts.
- If gA holds, then user can choose alternative A without breaching contract. System then executes SA (guards gA, gB, gC may then change). The loop is then repeated.
- If gB holds, then user can choose alternative B ...
- If gC holds, then user can choose alternative C...
- The user can always choose last alternative, which leads to termination after SQ.



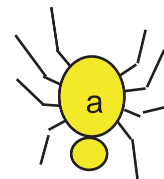
$$
\begin{aligned}
& \mathit{Init}; \\
& (\mu X \bullet \\
& \quad \{gA\}; SA; X \\
& \sqcup \ \{gB\}; SB; X \\
& \sqcup \ \{g\}; SC; X \\
& \sqcup \ SQ)
\end{aligned}
$$

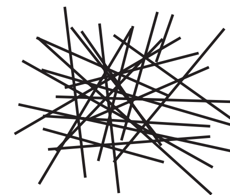
Playing games: Nim

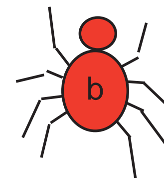
Players a and b take turns to remove either one or two sticks from a pile. The player who takes the last stick has lost. Player a starts.

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.

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







Proving properties of contracts

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
- $r \cap \neg\, g \subseteq q$.

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:

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

The loop guard true shows that the game never ends in a draw in finite time. The body shows the sequence of moves that is repeated until the game ends (or indefinitely).

The moves

- First, the guard statement $[x \neq 0]$ is a check to see whether Player has already won (Opponent has lost if the pile of matches is empty when it is Player's move).

- If there are matches left ($x \neq 0$), then Player 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 Player has lost.

- If there are matches left, then Opponent moves according to the relation Move, i.e., removes one or two matches.

Proving existence of winning strategy

How do we prove the existence of a winning strategy?

Assume that the loop $\mathsf{while}\ g\ \mathsf{do}\ S\ \mathsf{od}$ represents a game and that the initial board is described by the predicate p.

If the loop establishes postcondition false from precondition p, then from the semantics of statements we know that Player can make choices in the angelic updates in such a way as to be guaranteed to win, regardless of what choices Opponent makes in the demonic updates.

Thus there exists a winning strategy for Player under precondition p if the following total correctness assertion holds:

$$p\ \{\!|\ \mathsf{while}\ g\ \mathsf{do}\ S\ \mathsf{od}\ |\!\}\ \mathsf{false}$$

Rule for existence of winning strategy

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

Theorem Player has a winning strategy for game $\mathsf{while}\ g\ \mathsf{do}\ S\ \mathsf{od}$ under precondition p, provided that the following three conditions hold for some ranked collection of predicates $\{r_w \mid w \in W\}$:

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

The ranked predicate r_w is usually described as a conjunction of an invariant I and a variant t, so that r_w is $I\ \wedge\ t = w$. Note that in the case that $g = \mathsf{true}$, the third condition in loop rule is trivially satisfied.

Proof

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

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

Straightforward simplification shows that the third of these conditions is equivalent to $r \subseteq g$ and also to $(\forall\, w \in W \bullet r_w \subseteq g)$. This can then be used to simplify the second condition.

Winning strategy for Nim

For Player to be assured of winning this game, it is necessary that he always make the number x of matches remaining satisfy the condition $x \mathsf{\ mod\ } 3 = 1$. The strategy consists in always removing a number of matches such that $x \mathsf{\ mod\ } 3 = 1$ holds.

Correctness formulation

Assume that the precondition p is

$$x \text{ mod } 3 \neq 1$$

(since Player moves first, this is necessary to guarantee that Player can actually establish $x \text{ mod } 3 = 1$ in the first move).

The invariant I is simply p, and the variant t is x.

Since the guard of the iteration that represents Nim was true, the third condition is trivially satisfied. Furthermore, the choice of invariant is such that the first condition is also trivially satisfied. Thus, it remains only to prove the second condition, i.e., that

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

holds for an arbitrary natural number n, where

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

The proof

We prove $(*)$ by calculating $S.\,(x \text{ mod } 3 \neq 1 \land x < n)$ in two parts. First, we find the intermediate condition:

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

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

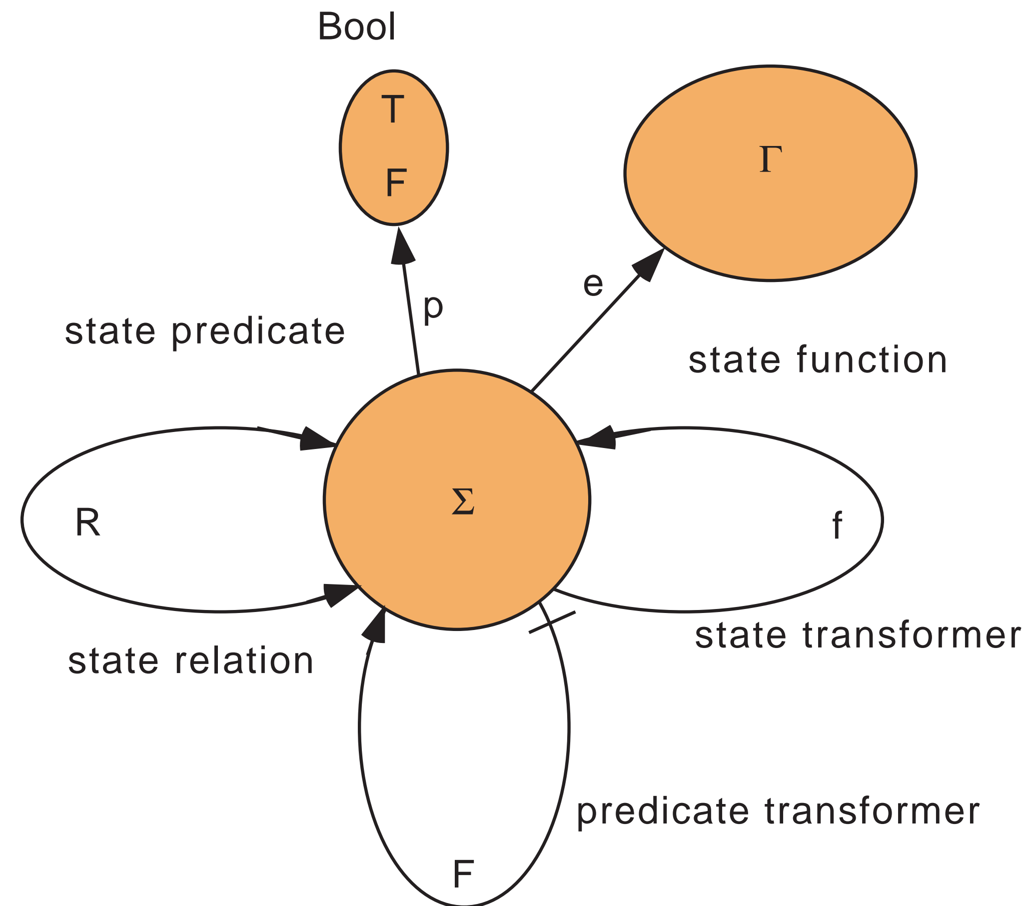
$$
\begin{array}{ll}
 & ([x > 0];\{\mathsf{Move}\}).\,(x \text{ mod } 3 = 1 \land x \leq n) \\
= & \{\text{definitions}\} \\
 & x = 0 \;\lor\; (\exists x' \bullet x - 2 \leq x' < x \land x' \text{ mod } 3 = 1 \land\; x' \leq n) \\
\supseteq & \{\text{reduction, arithmetic}\} \\
 & x \text{ mod } 3 \neq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

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

Lattice properties of contracts

Basic mathematical entities

- **State function** $f : \Sigma \to \Gamma$ is an observation of the state. Expressions are state functions.
- **State predicate** $p : \Sigma \to \mathsf{Bool}$ is a property of the state. A boolean expression describes a state predicate.
- **State transformer** $f : \Sigma \to \Sigma$ maps states to states . An assignment is a state transformer.
- **State relation** $R : \Sigma \to \Sigma \to \mathsf{Bool}$ relates a state σ to a state σ' whenever $R.\sigma.\sigma'$ holds. Relational assignment is a state relation.
- **Predicate transformers** $F : (\Sigma \to \mathsf{Bool}) \to (\Sigma \to \mathsf{Bool})$. Contracts are interpreted as predicate transformers.



Ordering

A relation $\sqsubseteq$ is a **partial ordering**, if it is **reflexive**, **transitive** and **antisymmetric**.

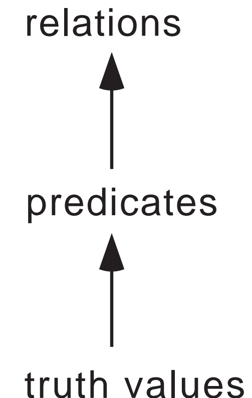
- Truth values Bool are partially ordered by implication: $b \sqsubseteq c \quad \text{iff} \quad b \Rightarrow c$

- Predicates $\mathsf{Pred} = \Sigma \to \mathsf{Bool}$ are partially ordered by inclusion, $p \sqsubseteq q \quad \text{iff} \quad p \subseteq q$. This is an **extension** of the implication ordering:

$$p \subseteq q \quad \triangleq \quad (\forall \sigma \bullet p.\sigma \Rightarrow q.\sigma)$$

- Relations $\mathsf{Rel} = \Sigma \to \mathsf{Pred} = \Sigma \to (\Sigma \to \mathsf{Bool})$ are also partially ordered by inclusion: $R \sqsubseteq R' \quad \text{iff} \quad R \subseteq R'$. This is an **extension** of the inclusion ordering:

$$\begin{aligned} R \subseteq Q \quad & \triangleq \quad (\forall \sigma \bullet R.\sigma \subseteq Q.\sigma) \\ & \equiv \quad (\forall \sigma \forall \sigma' \bullet R.\sigma.\sigma' \Rightarrow Q.\sigma.\sigma') \end{aligned}$$



Refinement ordering

A **predicate transformer** is a function that maps predicates to predicates

$$F : \mathsf{Pred} \to \mathsf{Pred}$$

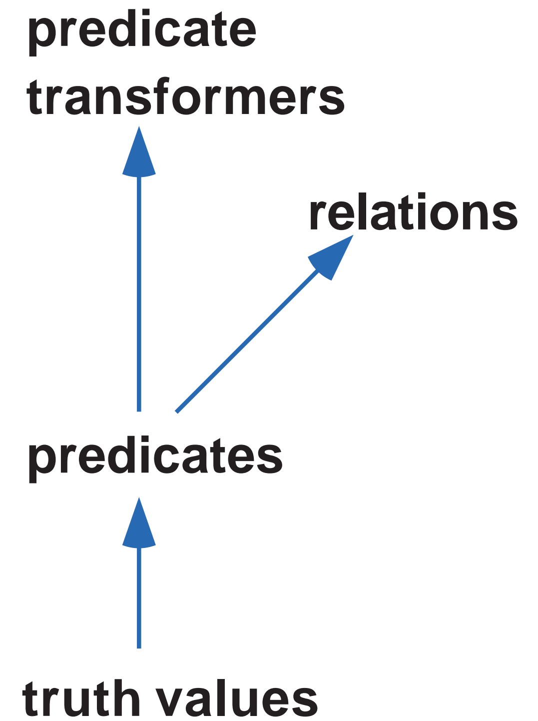
The predicate transformer ordering **extends** the inclusion ordering on predicates:

$$F \sqsubseteq F' \quad \triangleq \quad (\forall q \bullet F.q \subseteq F'.q)$$

The ordering $\sqsubseteq$ on predicate transformers is a partial ordering.

We have defined for contracts that $S \sqsubseteq S'$ if and only if $wp.S \sqsubseteq wp.S'$. Hence $\sqsubseteq$ on predicate transformers is called the **refinement ordering**.

We write $F : \Sigma \mapsto \Gamma$ for $F : \mathcal{P}(\Gamma) \to \mathcal{P}(\Sigma)$. This allows for different initial and final state spaces.



Stepwise refinement method

Transitivity of the refinement ordering justifies the stepwise refinement method for program derivation:

$$S_0 \sqsubseteq S_1 \sqsubseteq S_2 \sqsubseteq \ldots \sqsubseteq S_n \quad \Rightarrow \quad S_0 \sqsubseteq S_n$$

Here S_0 is initial high level statement (**specification**) that describes what we want to achieve, and S_n is final **program** that implements the specification.

Example:

$$\begin{array}{ll} & [x := x' \mid x' \geq x \wedge x' \geq y] \\ \sqsubseteq & \{\text{reduce nondeterminism}\} \\ & x := x \text{ max } y \\ = & \{\text{properties of conditional statement}\} \\ & \mathsf{if}\ x < y\ \mathsf{then}\ x := y\ \ \mathsf{else}\ \ \mathsf{skip}\ \mathsf{fi} \\ \sqsubseteq & \{\text{add choices}\} \\ & \mathsf{if}\ x < y\ \mathsf{then}\ x := y\ \ \mathsf{else}\ \ \mathsf{skip}\ \mathsf{fi} \sqcup x := 0 \end{array}$$

Central lattices

Truth values, predicates, relations and predicate transformers on a give state space are all **complete boolean lattices**.

Lattice	Truth values	Predicates	Relations	Predicate transformers
ordering $\sqsubseteq$	$\Rightarrow$	$\subseteq$	$\subseteq$	$\sqsubseteq$
bottom $\bot$	F	false	False	abort
top $\bot$	T	true	True	magic
meet $\sqcap$	$b \wedge c$	$p \cap q$	$P \cap Q$	$F \sqcap G$
join $\sqcup$	$b \vee c$	$p \cup q$	$P \cup Q$	$F \sqcup G$
negation $\neg$	$\neg b$	$\neg p$	$\neg Q$	$\neg F$

Operations defined by **pointwise extension**:

$$\begin{aligned} \mathsf{false}.\,\sigma &= \mathsf{F} \\ \mathsf{true}.\,\sigma &= \mathsf{T} \\ (p \cap q).\,\sigma &= p.\,\sigma \wedge q.\,\sigma \\ (p \cup q).\,\sigma &= p.\,\sigma \vee q.\,\sigma \\ (\neg p).\,\sigma &= \neg p.\,\sigma \end{aligned}$$

$$\begin{aligned} \mathsf{abort}.\,q &= \mathsf{false} \\ \mathsf{magic}.\,q &= \mathsf{true} \\ (F \sqcap F').\,q. &= F.\,q \cap F'.\,q \\ (F \sqcup F').\,q. &= F.\,q \cup F'.\,q \\ (\neg F).\,q &= \neg F.\,q \end{aligned}$$

Interpretation contracts as predicate transformers

- Contract refinement is interpreted as predicate transformer refinement
- Angelic choice is join on predicate transformers
- Demonic choice is meet on predicate transformers
- Impossible contract (abort) is bottom of predicate transformer lattice
- Miraculous contract (magic) is top of predicate transformer lattice.

Lattices

A poset A is a lattice if any two elements b_1 and b_2 have a **meet** $b_1 \sqcap b_2$ and a **join** $b_1 \sqcup b_2$ in A such that :

- **Lower bound:**

$$b_1 \sqcap b_2 \sqsubseteq b_1 \quad \text{and} \quad b_1 \sqcap b_2 \sqsubseteq b_2$$

- **Greatest lower bound:**

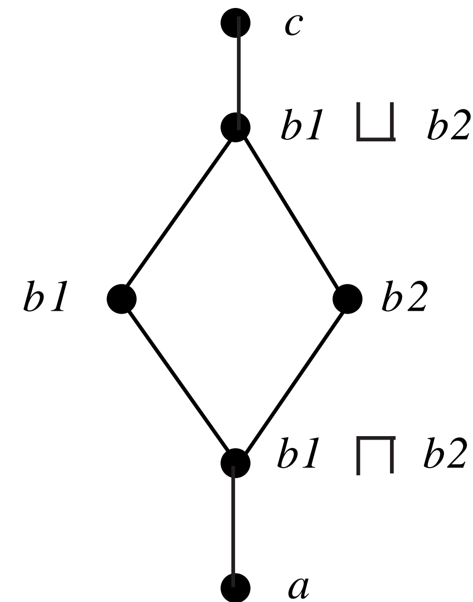
$$a \sqsubseteq b_1 \;\wedge\; a \sqsubseteq b_2 \;\Rightarrow\; a \sqsubseteq b_1 \sqcap b_2$$

- **Upper bound:**

$$b_1 \sqsubseteq b_1 \sqcup b_2 \quad \text{and} \quad b_2 \sqsubseteq b_1 \sqcup b_2$$

- **Least uppper bound:**

$$b_1 \sqsubseteq c \;\wedge\; b_2 \sqsubseteq c \;\Rightarrow\; b_1 \sqcup b_2 \sqsubseteq c$$



Lattice properties

- **Idempotence:** $a \sqcap a = a$ and $a \sqcup a = a$
- **Commutativity:** $a \sqcap b = b \sqcap a$ and $a \sqcup b = b \sqcup a$
- **Associativity** :

$$\begin{aligned} a \sqcap (b \sqcap c) &= (a \sqcap b) \sqcap c \text{ and} \\ a \sqcup (b \sqcup c) &= (a \sqcup b) \sqcup c \end{aligned}$$

- **Absorption**

$$a \sqcap (a \sqcup b) = a \quad \text{and} \quad a \sqcup (a \sqcap b) = a$$

- **Correspodence**

$$\begin{aligned} a \sqcap b = a &\equiv a \sqsubseteq b \text{ and} \\ a \sqcup b = b &\equiv a \sqsubseteq b \end{aligned}$$

Contract interpretation

- **Idempotence:** $S \sqcap S = S$. Two identical alternatives for the demon does not give anything new.
- **Commutativity:** $S \sqcap T = T \sqcap S$. The order of alternatives does not matter for demon.
- **Associativity:** $S \sqcap (T \sqcap U) = (S \sqcap T) \sqcap U$. The order of choices does not matter.
- **Absorption:** $S \sqcap (S \sqcup T) = S$. Demon does not give the angel a choice if possible
- **Correspodence:** $S \sqcap T = S \quad \equiv \quad S \sqsubseteq T$
  - If $S \sqcap T = S$, then T does not give any advantage for the demon, i.e., T can only be an improvement for the angel
  - If $S \sqsubseteq T$, then demon should not choose T over S, i.e, $S \sqcap T = S$.

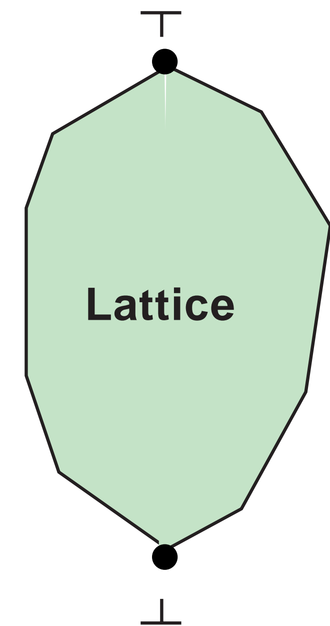
Complete lattices

Poset A is **complete**, if each set of elements $B \subseteq A$ has a meet $\sqcap B$ and a join $\sqcup B$ in A, where

- **Lower bound:** $b \in B \Rightarrow \sqcap B \sqsubseteq b$
- **Greatest lower bound:** $(\forall b \in B \bullet a \sqsubseteq b) \Rightarrow a \sqsubseteq \sqcap B$
- **Upper bound:** $b \in B \Rightarrow b \sqsubseteq \sqcup B$
- **Least uppper bound:** $(\forall b \in B \bullet b \sqsubseteq a) \Rightarrow \sqcup B \sqsubseteq a$

A complete lattice A is **bounded**, i.e., it has a least element $\bot$ and a greatest element $\top$:

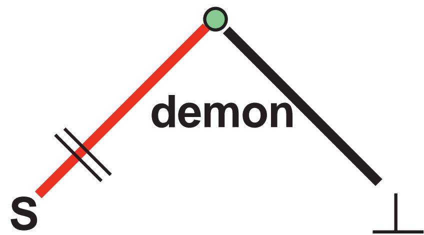
- **Least element:** $\bot = \sqcap A$ and $\bot = \sqcup \emptyset$
- **Greatest element:** $\top = \sqcup A$ and $\top = \sqcap \emptyset$

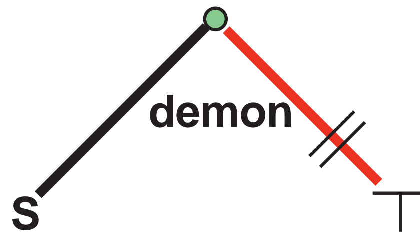


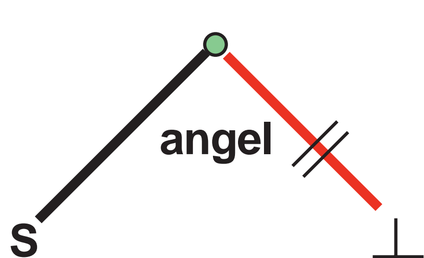
Contract interpretation of abort and magic

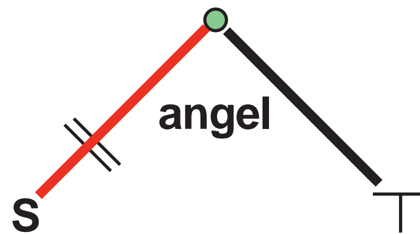
Consider following examples of properties for contracts:

- abort **is zero for demonic choice:** $S \sqcap \text{abort} = \text{abort}$
- magic **is unit for demonic choice:** $S \sqcap \text{magic} = S$
- abort **is unit for angelic choice:** $S \sqcup \text{abort} = S$
- magic **is zero for angelic choice:** $S \sqcup \text{magic} = \text{magic}$









Distributive and boolean lattices

A lattice is **distributive** if for any elements a, b and c,

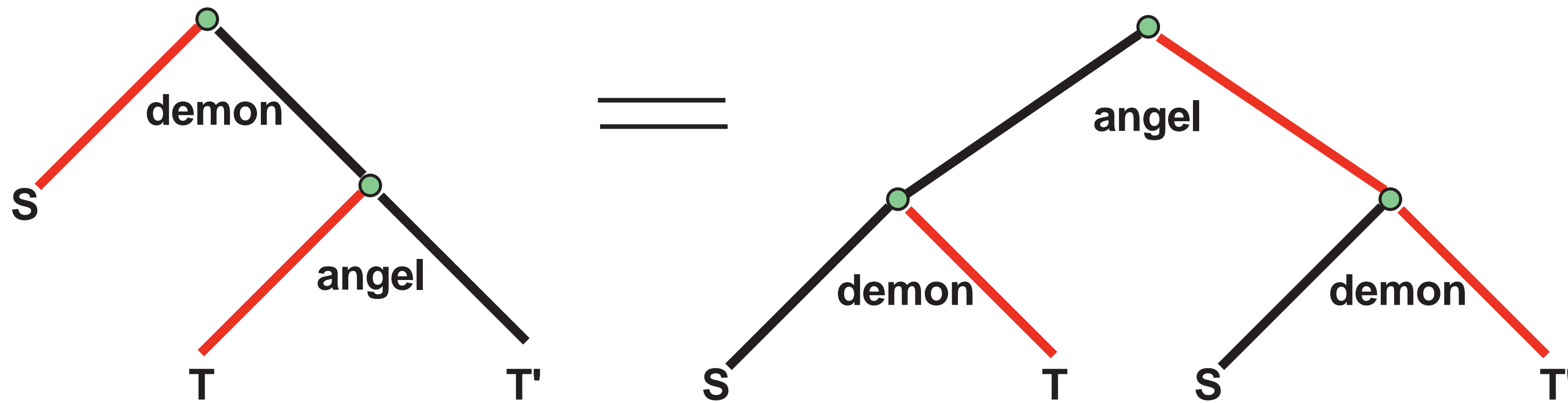
- **Meet distributive:** $a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$
- **Join distributive:** $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$

A complete distributive lattice is **complete boolean lattice**, if every element a has a unique **complement** $\neg a$ in A satisfying the conditions

- **Contradiction:** $a \sqcap \neg a = \bot$
- **Exhaustiveness:** $a \sqcup \neg a = \top$

Distributivity for contracts

- **Distributivity:** $S \sqcap (T \sqcup T') = (S \sqcap T) \sqcup (S \sqcap T')$



Duality

The **dual** $F^\circ : \Sigma \mapsto \Gamma$ of a predicate transformer $F : \Sigma \mapsto \Gamma$ is defined by

$$F^\circ .\, q \quad \triangleq \quad \neg\, F.\, (\neg\, q)$$

A consequence of the definition is that $(F^\circ)^\circ = F$, so dualization is an involution.

The following dualities hold between predicate transformers:

$$\begin{array}{rcl} \langle f \rangle^\circ & = & \langle f \rangle \\ \{p\}^\circ & = & [p] \\ \{R\}^\circ & = & [R] \end{array} \qquad \begin{array}{rcl} (S_1 ; S_2)^\circ & = & S_1^\circ ; S_2^\circ \\ (\sqcup\, i \in I \bullet S_i)^\circ & = & (\sqcap\, i \in I \bullet S_i^\circ) \end{array}$$

Thus, the operations for constructing statements come in pairs, each one with its dual. Functional update and sequential composition are their own duals. In particular, we have that

$$\mathsf{magic}^\circ = \mathsf{abort} \qquad\qquad \mathsf{skip}^\circ = \mathsf{skip}$$

Monotonic predicate transformers

Categories

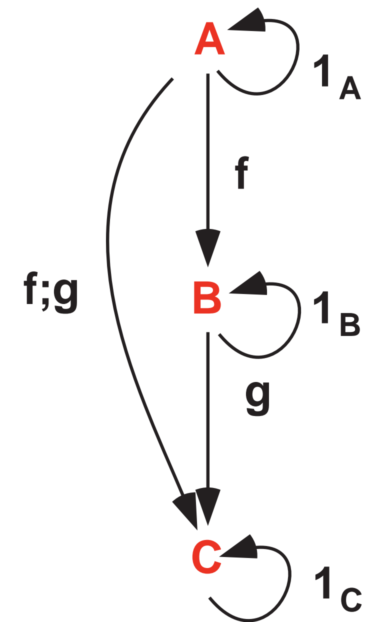
A **category** C consists of a collection of **objects** and a collection of **morphisms**. Each morphism has a **source** object and a **target** object. Write $A \xrightarrow{f} B$ for morphism f with source A and target B.

A **composition operator** takes a morphism $A \xrightarrow{f} B$ and a morphism $B \xrightarrow{g} C$ to the morphism $A \xrightarrow{f;g} B$.

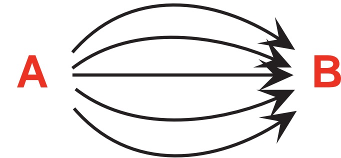
For every object A there is a special morphism 1_A, the *identity morphism* on A.

A category satisfies the following two properties:

- **Composition is associative:** $f;(g;h) = (f;g);h$
- **Identity is unit:** $1;f = f$ and $f;1 = f$.



Order enriched categories



Consider **category** C where for any objects A and B, the morphisms $C(A, B)$ between these objects are ordered by $\sqsubseteq_{A,B}$. We say that C is an **order-enriched category**, if composition is **monotonic** in both arguments:

$$f \sqsubseteq_{A,B} f' \wedge g \sqsubseteq_{B,C} g' \quad \Rightarrow \quad f; g \sqsubseteq_{A,C} f'; g'$$

The category is **left (right)order-enriched**, if composition is only monotonic in the left (right) argument:

$$f \sqsubseteq_{A,B} f' \quad \Rightarrow \quad f; g \sqsubseteq_{A,C} f'; g \qquad \text{(left)}$$
$$g \sqsubseteq_{B,C} g' \quad \Rightarrow \quad f; g \sqsubseteq_{A,C} f; g' \qquad \text{(right)}$$

State categories

State predicates, state transformations, state relations and predicate transformers all form categories, where objects are state spaces and morphisms are predicates, functions, relations and predicate transformers:

$$
\begin{array}{lll}
\Sigma \xrightarrow{p} \Sigma & \qquad & p : \mathcal{P}(\Sigma) \\
\Sigma \xrightarrow{f} \Gamma & & f : \Sigma \to \Gamma \\
\Sigma \xrightarrow{R} \Gamma & & R : \Sigma \leftrightarrow \Gamma \\
\Sigma \xrightarrow{F} \Gamma & & S : \Sigma \mapsto \Gamma
\end{array}
$$

State relations form a **complete boolean lattice-enriched category**.

Predicate transformers form a **left complete boolean lattice-enriched category** with refiement ordering and operations

$$
F ; G \triangleq F \circ G \qquad\qquad 1_\Sigma \triangleq (\lambda\, q \bullet q) : \Sigma \mapsto \Sigma
$$

State predicates and state transformers also form (trivial) order-enriched categories.

Monotonic predicate transformers

Predicate transformers only form a left-orderd category, because seqeuential composition is only montonic in the left argument:

$$F \sqsubseteq F' \quad \Rightarrow \quad F; G \sqsubseteq F' : G$$

Lattice meet and join are monotonic in both arguments:

$$F \sqsubseteq F' \wedge G \sqsubseteq G' \quad \Rightarrow \quad F \sqcap G \sqsubseteq F' \sqcap G'$$
$$F \sqsubseteq F' \wedge G \sqsubseteq G' \quad \Rightarrow \quad F \sqcup G \sqsubseteq F' \sqcup G'$$

A predicate transformer S is **monotonic**, if

$$q \subseteq q' \quad \Rightarrow \quad F.\, q \subseteq F.\, q', \qquad \text{for each } q \subseteq \Sigma$$

The collection of **monotonic predicate transformers** form a **complete lattice-enriched category**:

$$F \sqsubseteq F' \wedge G \sqsubseteq G' \quad \Rightarrow \quad F; G \sqsubseteq F'; G'$$

Top-down replacement

Monotonicity of statements allows **top-down** construction of statements:

$$
\begin{array}{ll}
 & S[T] \\
\sqsubseteq & \{\text{monotonicity}\} \\
 & \quad \begin{array}{ll} & T \\ \sqsubseteq & \{\text{motivation}\} \\ & T' \end{array} \\
 & S[T']
\end{array}
$$

This allows us to transform large programs by making small local changes to them.

Example:

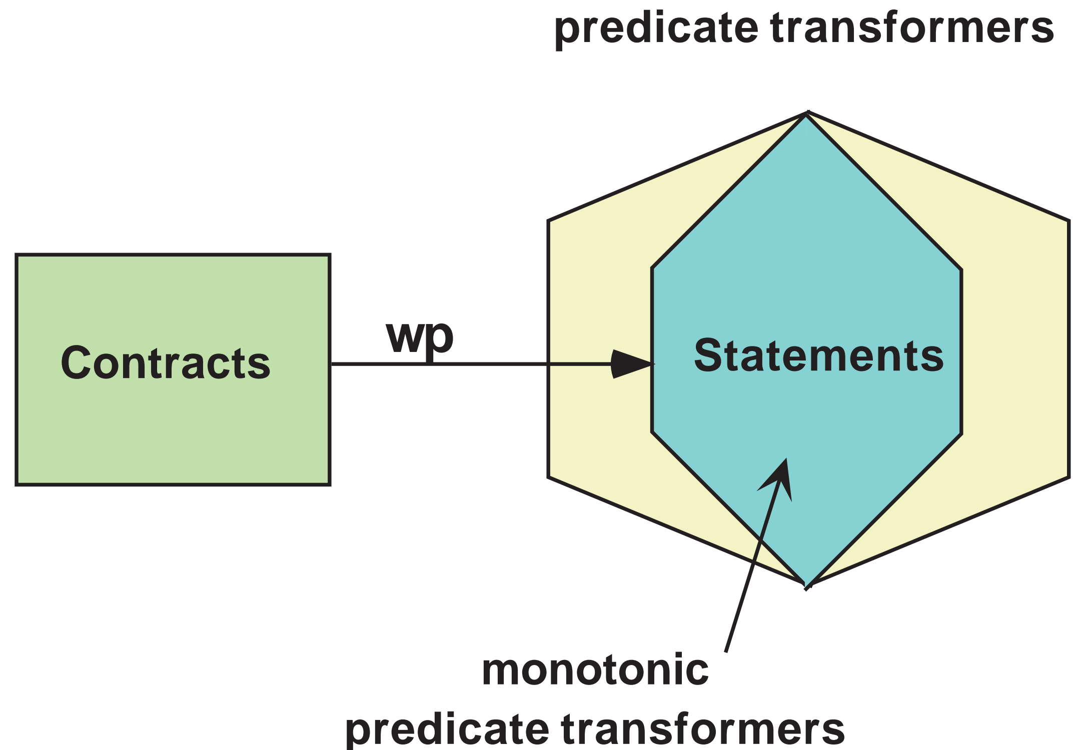
$$
\begin{array}{ll}
 & \ldots [x := x' \mid x' \geq x \wedge x' \geq y] \ldots \\
\sqsubseteq & \{\text{motivation}\} \\
 & \ldots \mathsf{if}\ x < y\ \mathsf{then}\ x := y\ \ \mathsf{else}\ \ \mathsf{skip}\ \mathsf{fi} \ldots
\end{array}
$$

Statements and monotonic predicate transformers

Theorem: All statements are monotonic.

Proof establishes that

- $\langle f \rangle$, $\{p\}$,$[p]$,$\{R\}$ and $[R]$ are all monotonic predicate transformers
- Meet, join and sequential composition of predicate transformers preserve monotonicity.



Recursive contracts are well-defined

Monotonicity of statement constructors implies that the weakest preconditions for recursive contracts is well-defined. We defined

$$\begin{aligned} wp.(\mu X \bullet S).q.\sigma &= (\mu.(\lambda X \bullet wp.S_i)).q.\sigma \\ wp.(\nu X \bullet S).q.\sigma &= (\nu.(\lambda X \bullet wp.S_i)).q.\sigma \end{aligned}$$

The function $F = (\lambda X \bullet wp.S_i)$ is a **monotonic** function from a complete lattice (the monotonic predicate transformer lattice) to itself. Hence, by the Knaster-Tarski fixpoint theorem, the function F has a **least fixpoint** $\mu.F$ and a **greatest fixpoint** $\nu.F$.

Main properties:

- $\mu.F$ and $\nu.F$ are fixpoints of F:

$$F.(\mu.F) = \mu.F \qquad\qquad F.(\nu.F) = \nu.F$$

- $\mu.F$ is the least , $\nu.F$ the greatest of all fixpoints:

$$F.x = x \Rightarrow \mu.F \sqsubseteq x \qquad\qquad F.x = x \Rightarrow x \sqsubseteq \nu.F$$

Expressibility of statements

Theorem: Any monotonic predicate transformer term can be expressed as a predicate transformer statement.

Proof constructs a **normal form** for monotonic predicate transformers

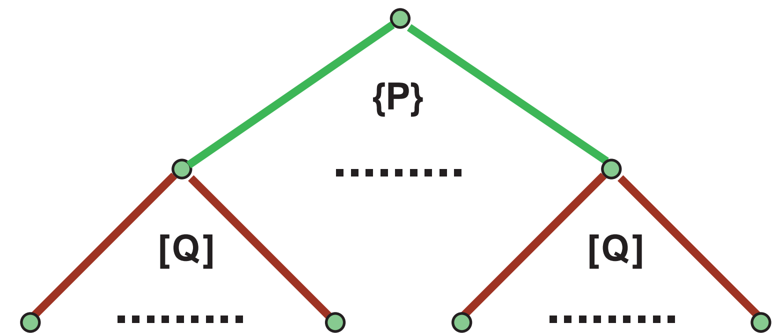
- $F = \{h.\,F\};[k.\,F]$, for any monotonic predicate transformer F.
- Characterization is implicit, because F also occurs on the right hand side.

Contracts as reduced games

Any contract can be expressed as

$$S = \{P\}; [Q]$$

This corresponds to a simple game between angel and demon, with only two turns.



Reducability of statements

Can reduce statement constructs to a few only. Define

$$|p|.\sigma.\sigma' \equiv p.\sigma \wedge \sigma = \sigma' \qquad\qquad |f|.\sigma.\sigma' \equiv f.\sigma = \sigma'$$

Then

$$\{p\} = \{\ |p|\ \} \qquad [p] = [\ |p|\ ] \qquad \langle f\rangle = \{\ |f|\ \} = [\ |f|\ ]$$

As recursion also can be expressed using arbitrary join, it is sufficient to consider statements generated by

$$F \quad ::= \quad \{R\} \mid [R] \mid F_1; F_2 \mid (\sqcup i \in I \bullet F_i) \mid (\sqcap i \in I \bullet F_i)$$

With duality, this can be further reduced to

$$F \quad ::= \quad \{R\} \mid F_1; F_2 \mid (\sqcup i \in I \bullet F_i) \mid F_1^\circ$$

Normal form implies that actually

$$F \quad ::= \quad \{R\}; [Q]$$

is sufficient to express the meaning of all contracts.

Preconditions and guards

Let $S : \Sigma \to \Gamma$ be a predicate transformer. Define

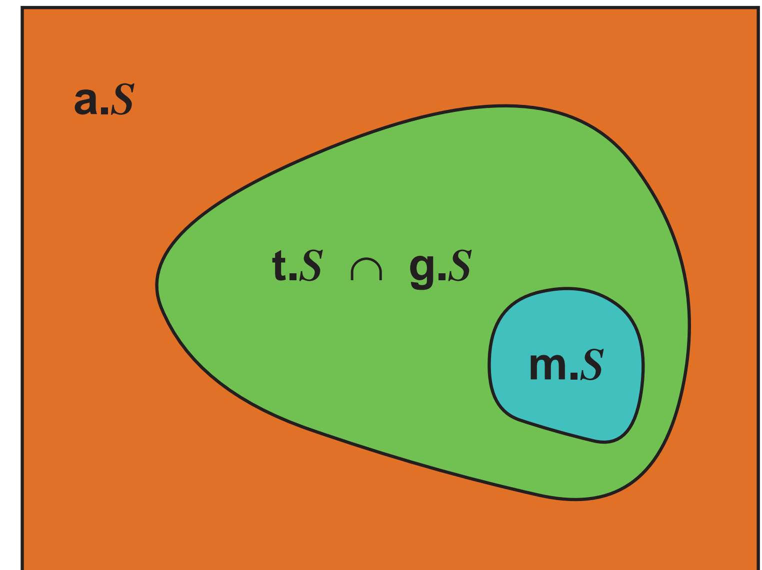
$$\begin{aligned} \mathsf{t}.\,S &\triangleq wp.\,S.\,\mathsf{true} \\ \mathsf{a}.\,S &\triangleq \neg\,\mathsf{t}.\,S \\ \mathsf{m}.\,S &\triangleq wp.\,S.\,\mathsf{false} \\ \mathsf{g}.\,S &\triangleq \neg\,\mathsf{m}.\,S \end{aligned}$$

- The **abortion guard** (or **termination precondition**) $\mathsf{t}.\,S$ characterizes those initial states in which the angel can avoid a breach of contract (it *guards against* abortion).
- The **abortion precondition** $\mathsf{a}.\,S$ characterizes those initial states in which the angel cannot avoid breaching the contract (abortion will occur).
- The **miracle precondition** $\mathsf{m}.\,S$ characterizes those initial states in which the angel can choose to be released from the contract (a **miracle** can occur)
- The **miracle guard** $\mathsf{g}.\,S$ characterizes the complement of this, i.e., the initial states in which release from the contract is not possible.

Partitioning of initial states

It is easily seen that $\mathsf{m}.\,S \subseteq \mathsf{t}.\,S$. Hence, the abortion and the miracle guards partition the initial state space into three disjoint sets:

- $\mathsf{m}.\,S$: Our agent can choose to be released from its obligations to satisfy the contract.
- $\mathsf{t}.\,S \cap \mathsf{g}.\,S$: Discharge of obligation is impossible for our agent, and a breach of contract can be avoided, so some postcondition will be established.
- $\mathsf{a}.\,S$: Our agent cannot avoid breaching the contract.

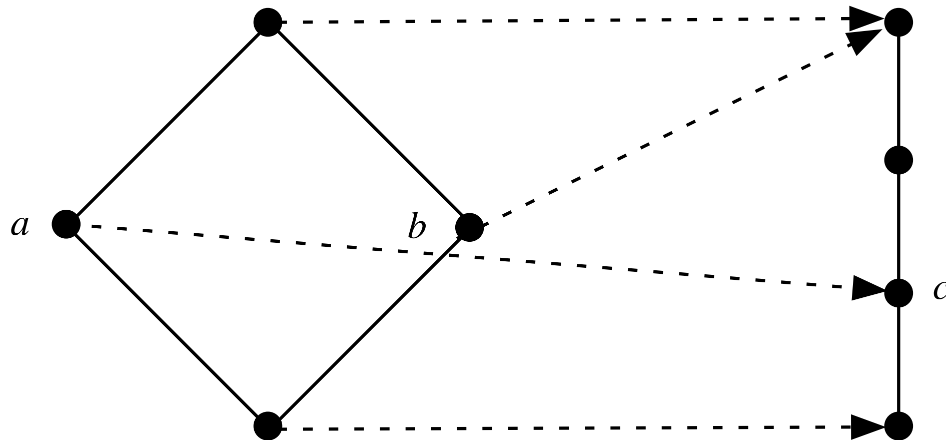


Homomorphism of statement constructors

Homomorphism properties

Consider a function $h : A \to B$, where A and B are both lattices. Then

- h is a **meet homomorphism** if $h.\,(a \sqcap_A b) = h.\,a \sqcap_B h.\,b$.
- h is a **join homomorphism** if $h.\,(a \sqcup_A b) = h.\,a \sqcup_B h.\,b$.
- h is a **negation homomorphism** if $h.\,(\neg_A.\,a) = \neg_B h.\,a$.
- h is a **bottom homomorphism** if $h.\,\bot_A = \bot_B$.
- h is a **top homomorphism** if $h.\,\top_A = \top_B$.



Homomorphism properties, cont

Consider a function $h : A \to B$, where A and B are both complete lattices. Then

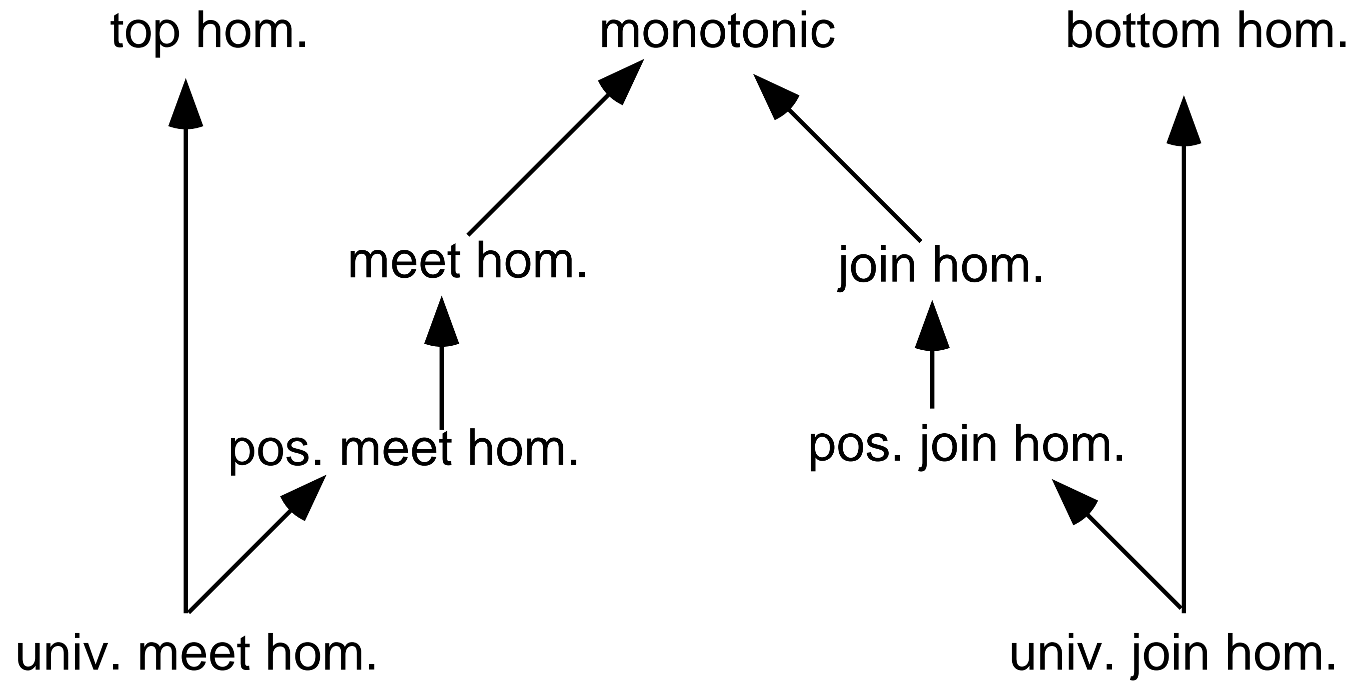
- h is a **universal meet homomorphism** if for all $\{a_i \mid i \in I\}$,

$$h.\,(\sqcap_A\ i \in I \bullet a_i) = (\sqcap_B\ i \in I \bullet h.\,a_i)$$

- h is a **positive meet homomorphism** if for all $\{a_i \mid i \in I\}$, $I \neq \emptyset$,

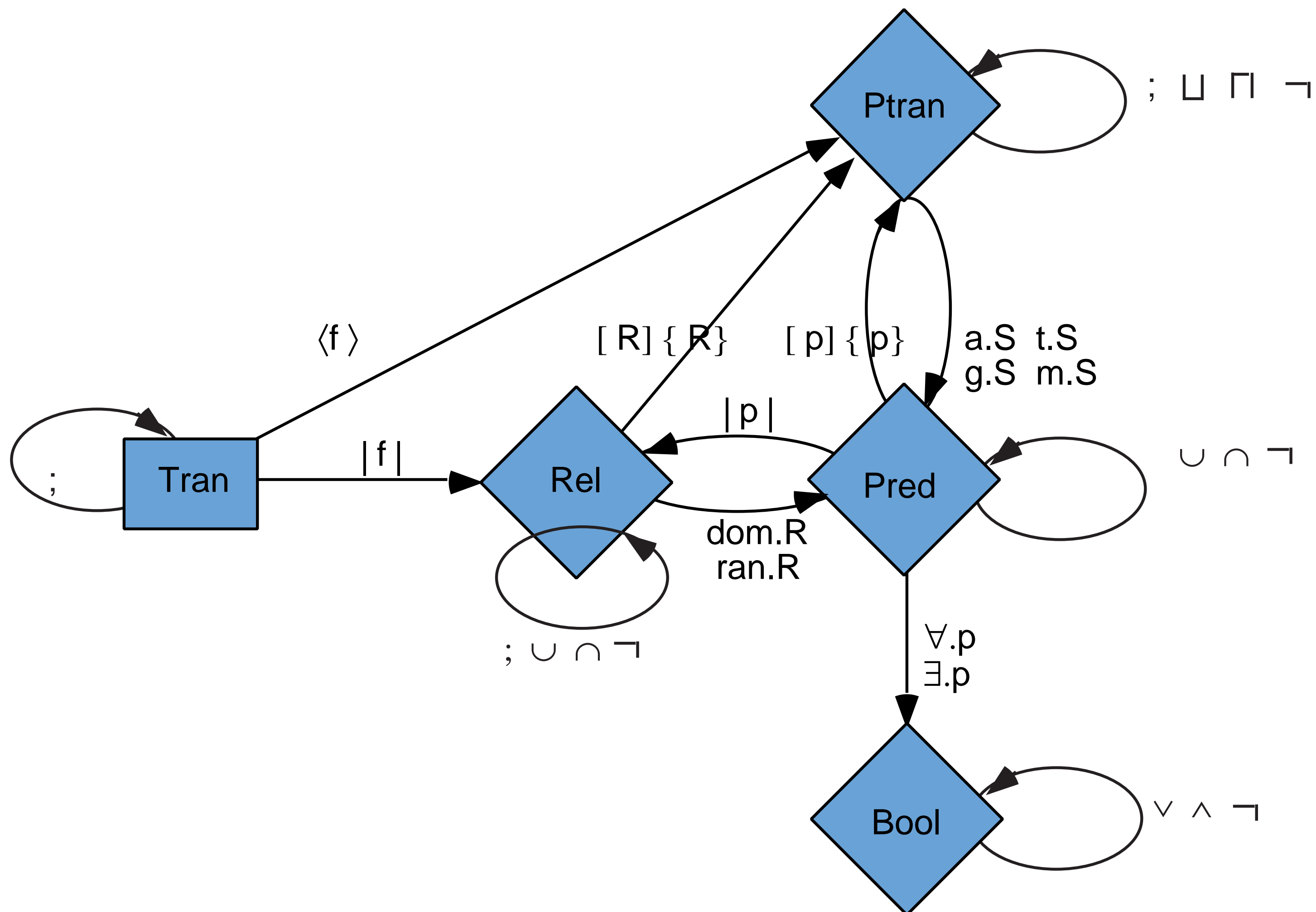
$$h.\,(\sqcap_A\ i \in I \bullet a_i) = (\sqcap_B\ i \in I \bullet h.\,a_i)$$

Implications between homomorphisms



Operations between categories

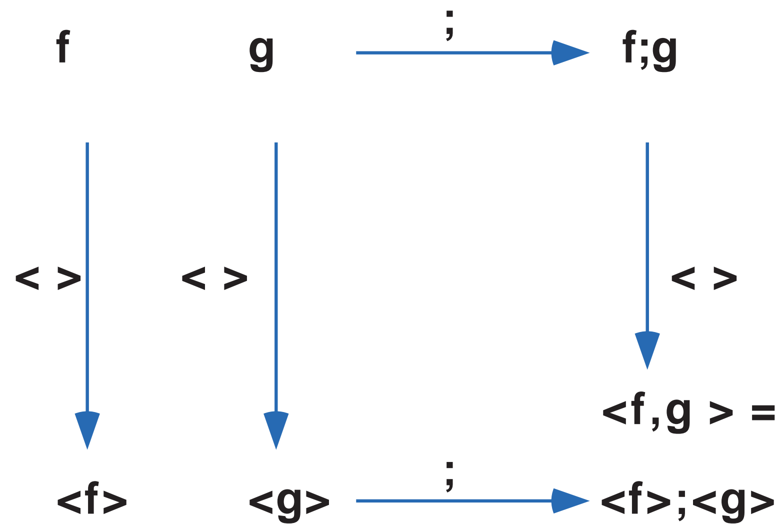
The statement constructors can be seen as operations themselves. Thus, e.g., $(\lambda R \bullet \{R\})$ maps relations to predicate transformers.



Preserving category operations

All basic statement constructors preserve identity and composition:

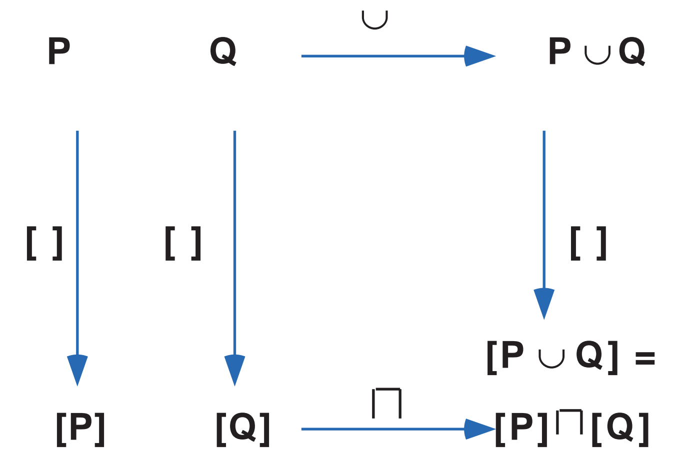
$$
\begin{array}{llcllcl}
\text{(c)} & \langle f; g \rangle & = & \langle f \rangle; \langle g \rangle & \langle \mathsf{id} \rangle & = & \mathsf{skip} \\
\text{(d)} & \{P; Q\} & = & \{P\}; \{Q\} & \{\mathsf{Id}\} & = & \mathsf{skip} \\
\text{(e)} & [P; Q] & = & [P]; [Q] & [\mathsf{Id}] & = & \mathsf{skip} \\
\text{(f)} & (S_1; S_2)^\circ & = & S_1^\circ; S_2^\circ & \mathsf{skip}^\circ & = & \mathsf{skip}
\end{array}
$$



Updates

The update constructors satisfy the following homomorphism properties:

$$
\begin{array}{lrcl}
\text{(a)} & f = g & \equiv & \langle f \rangle = \langle g \rangle \\
\text{(b)} & P \subseteq Q & \equiv & \{P\} \sqsubseteq \{Q\} \\
 & P \subseteq Q & \equiv & [P] \sqsupseteq [Q] \\
\text{(c)} & \{\mathsf{False}\} & = & \mathsf{abort} \\
 & [\mathsf{False}] & = & \mathsf{magic} \\
\text{(d)} & \{\cup i \in I \bullet R_i\} & = & (\sqcup i \in I \bullet \{R_i\}) \\
 & [\cup i \in I \bullet R_i] & = & (\sqcap i \in I \bullet [R_i])
\end{array}
$$



(demonic choice is homomorphic onto dual lattice).

Tabular description

We summarize the homomorpism properties of statement constructors in the following two tables.

is	monot.	$\bot$-hom	$\top$-hom	$\sqcap$-hom	$\sqcup$-hom	$\neg$-hom	1-hom	;-hom
$(\lambda p \bullet \{p\})$	yes	yes	no	yes [1]	yes	no	yes	yes
$(\lambda p \bullet [p])$	yes$^\circ$	yes$^\circ$	no	yes$^{\circ 1}$	yes$^\circ$	no	yes	yes
$(\lambda f \bullet \langle f \rangle)$	yes	-	-	-	-	-	yes	yes
$(\lambda R \bullet \{R\})$	yes	yes	no	no	yes	no	yes	yes
$(\lambda R \bullet [R])$	yes$^\circ$	yes$^\circ$	no	no	yes$^\circ$	no	yes	yes

[1] when the meet is taken over nonempty sets

is	monot.	$\bot$-hom	$\top$-hom	$\sqcap$-hom	$\sqcup$-hom	$\neg$-hom	1-hom	;-hom
$(\lambda S \bullet S; T)$	yes	yes	yes	yes	yes	yes	no	no
$(\lambda S \bullet S \sqcap T)$	yes	yes	no	yes	yes	no	no	no
$(\lambda S \bullet S \sqcup T)$	yes	no	yes	yes	yes	no	no	no
$(\lambda S \bullet \neg S)$	yes$^\circ$	yes$^\circ$	yes$^\circ$	yes$^\circ$	yes$^\circ$	yes	no	no
$(\lambda S \bullet S^\circ)$	yes$^\circ$	yes$^\circ$	yes$^\circ$	yes$^\circ$	yes$^\circ$	yes	yes	yes

Examples

In general:

$$
\begin{array}{rcll}
\{\mathsf{false}\} = \mathsf{abort} & (& \text{but} & \{\mathsf{true}\} \neq \mathsf{magic}) \\
[\mathsf{false}] = \mathsf{magic} & (& \text{but} & [\mathsf{true}] \neq \mathsf{abort}) \\
\mathsf{magic} \sqcup T = \mathsf{magic} & (& \text{but} & \mathsf{magic} \sqcap T \neq \mathsf{magic}) \\
\mathsf{abort}; T = \mathsf{abort} & (& \text{but} & \mathsf{skip}; T \neq \mathsf{skip})
\end{array}
$$

Example proof

As an example of using homomorphisms in proofs, we show that the property $\langle f\rangle;[f^{-1}] \sqsubseteq \mathsf{skip}$ holds for an arbitrary state transformer f:

$$
\begin{array}{ll}
 & \langle f\rangle;[f^{-1}] \\
= & \{\text{properties of mapping relation construct}\} \\
 & [\ |f|\ ];[f^{-1}] \\
= & \{\text{homomorphism}\} \\
 & [\ |f|;f^{-1}] \\
\sqsubseteq & \{|\mathsf{dom}.\,R| \subseteq R;R^{-1}\ \} \\
 & [\ |\mathsf{dom}.\,|f||\ ] \\
= & \{\text{all functions are total}\} \\
 & [\ |\mathsf{true}|\ ] \\
= & \{|\mathsf{true}| = \mathsf{Id}, \text{ demonic update is functor}\} \\
 & \mathsf{skip}
\end{array}
$$

The homomorphisms allow us to move between levels in the refinement hierarchy.

Refinement calculus hierarchy

Reasoning can be done at different levels in the hierarchy of state categories.

- Reasoning about assignment statements $\langle x_1 := e_1 \rangle; \ldots; \langle x_n := e_m \rangle)$ is reduced to reasoning about assignments $(x_1 := e_1; \ldots; x_m := e_m)$, i.e., reasoning is carried out at state transformer level.

- Reasoning about predicates (inclusion, equality) and relations is often simplest to carry out at the boolean level.

- Reasoning about predicate transformers can be reduced to reasoning about simpler notions like predicates, state transformers, or state relations.

Should reason at the level where it is easiest

- Boolean logic (with quantifiers) is very concrete, and often the simplest.

- Arguments shorter with predicate transformers using algebraic laws

- No single level is the right one, one should choose the most suitable level

- Monotonicity and homomorphism properties transfer between levels.

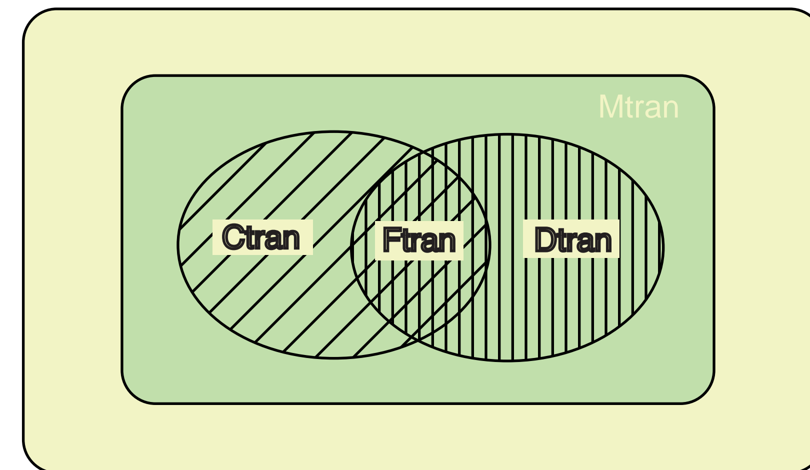
Statement subclasses

Homomorphic predicate transformers

Have previously only identified one subclass of predicate transformers, **monotonic predicate transformers**.

Can classify predicate transformers generally according to the homomorphism properties they satisfy.

- **Conjunctive predicate transformers** Ctran = positively meet homomorphic predicate tranformers.
- **Disjunctive predicate transformers** Dtran = positively join homomorphic predicate tranformers.
- The **functional predicate transformers** Ftran = positively join and positively meed homomorphic predicate tranformers.



Superscript on statement class shows whether it is $\bot$-homomorphic or $\top$-homomorphic or both.

Sublanguages and normal forms

Subclass	Language	Normal form
$\mathsf{Ctran}^{\top}$	$S ::= [g] \mid \langle f \rangle \mid [R] \mid S_1; S_2 \mid (\sqcap i \in I \bullet S_i)$	$[R]$
Ctran	$S ::= \{g\} \mid [g] \mid \langle f \rangle \mid [R] \mid S_1; S_2 \mid (\sqcap i \in I \bullet S_i)$	$\{p\}; [R]$
$\mathsf{Dtran}^{\perp}$	$S ::= \{g\} \mid \langle f \rangle \mid \{R\} \mid S_1; S_2 \mid (\sqcup i \in I \bullet S_i)$	$\{R\}$
Dtran	$S ::= \{g\} \mid [g] \mid \langle f \rangle \mid \{R\} \mid S_1; S_2 \mid (\sqcup i \in I \bullet S_i)$	$[p]; \{R\}$
$\mathsf{Ftran}^{\perp,\top}$	$S ::= \langle f \rangle \mid S_1; S_2 \mid$ (conditional, prim.rec.)	$\langle f \rangle$
$\mathsf{Ftran}^{\perp}$	$S ::= \{g\} \mid \langle f \rangle \mid S_1; S_2 \mid$ (conditional, rec.)	$\{p\}; \langle f \rangle$
Ftran	$S ::= \{g\} \mid [g] \mid \langle f \rangle \mid S_1; S_2 \mid$ (conditional, rec.)	$\{p\}; [q]; \langle f \rangle$

- $\mathsf{Ftran}^{\perp,\top}$ are the total functions.
- $\mathsf{Ftran}^{\perp}$ is usual class of deterministic programs with possibility for nontermination and run-time abortion.
- $\mathsf{Ctran}^{\top}$ is language of relations.
- Ctran is **Guarded Command's** langauge (no continuity restriction).
- $\mathsf{Dtran}^{\perp}$ is language for angelic choice (search algorithms).

Choice semantics

Forward and backward semantics

Predicate transformer semantics maps postconditions to (weakest) preconditions:

$$S : (\Gamma \to \mathsf{Bool}) \to (\Sigma \to \mathsf{Bool})$$

Define a new function $\bar{S}$ of type

$$\bar{S} : \Sigma \to ((\Gamma \to \mathsf{Bool}) \to \mathsf{Bool})$$

by

$$\bar{S}.\,\sigma.\,q \quad \triangleq \quad S.\,q.\,\sigma$$

Thus, $\bar{S}$ is a function from initial states to **sets of predicates** on the final states.

Lemma: If S is monotonic, then $\bar{S}$ is **upward closed**:

$$(\forall\, p, q \bullet p \in \bar{S}.\,\sigma \wedge p \subseteq q \quad \Rightarrow \quad q \in \bar{S}.\,\sigma)$$

Lemma: Given an upward closed family of sets of states $A_\sigma \subseteq \mathcal{P}(\Gamma)$ for every state σ, there is a unique monotonic predicate transformer S such that $\overline{S}.\sigma = A_\sigma$ for all σ.

Choice semantics

We define the **choice semantics** $\mathsf{ch}.S$ of a contract statement S inductively.

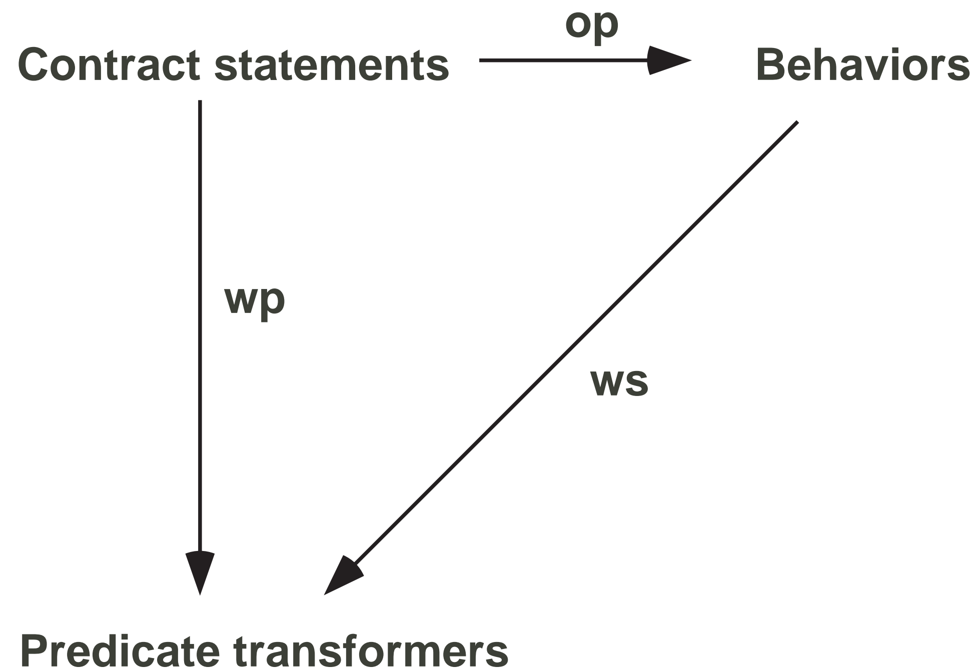
$$
\begin{aligned}
\mathsf{ch}.\{R\}.\sigma &= \{q \mid R.\sigma \cap q \neq \emptyset\} \\
\mathsf{ch}.[R].\sigma &= \{q \mid R.\sigma \subseteq q\} \\
\mathsf{ch}.(S_1; S_2).\sigma &= (\cup p \in \mathsf{ch}.S_1.\sigma \bullet (\cap \sigma' \in p \bullet \mathsf{ch}.S_2.\sigma')) \\
\mathsf{ch}.(\sqcap i \in I \bullet S_i).\sigma &= (\cap i \in I \bullet \mathsf{ch}.S_i.\sigma) \\
\mathsf{ch}.(\sqcup i \in I \bullet S_i).\sigma &= (\cup i \in I \bullet \mathsf{ch}.S_i.\sigma)
\end{aligned}
$$

$\mathsf{ch}.S$ is a function that maps every initial state to a set of predicates over the final state space. The contract statement S is seen as a simple game in which the angel chooses a predicate $q \in \mathsf{ch}.S.\sigma$, and the demon then chooses a final state $\gamma \in q$.

If $\mathsf{ch}.S.\sigma$ is empty, then the angel loses, since it cannot choose any predicate.

If the angel chooses the empty predicate false, then it wins, since the demon cannot choose any state in false.

Comparing all three semantics



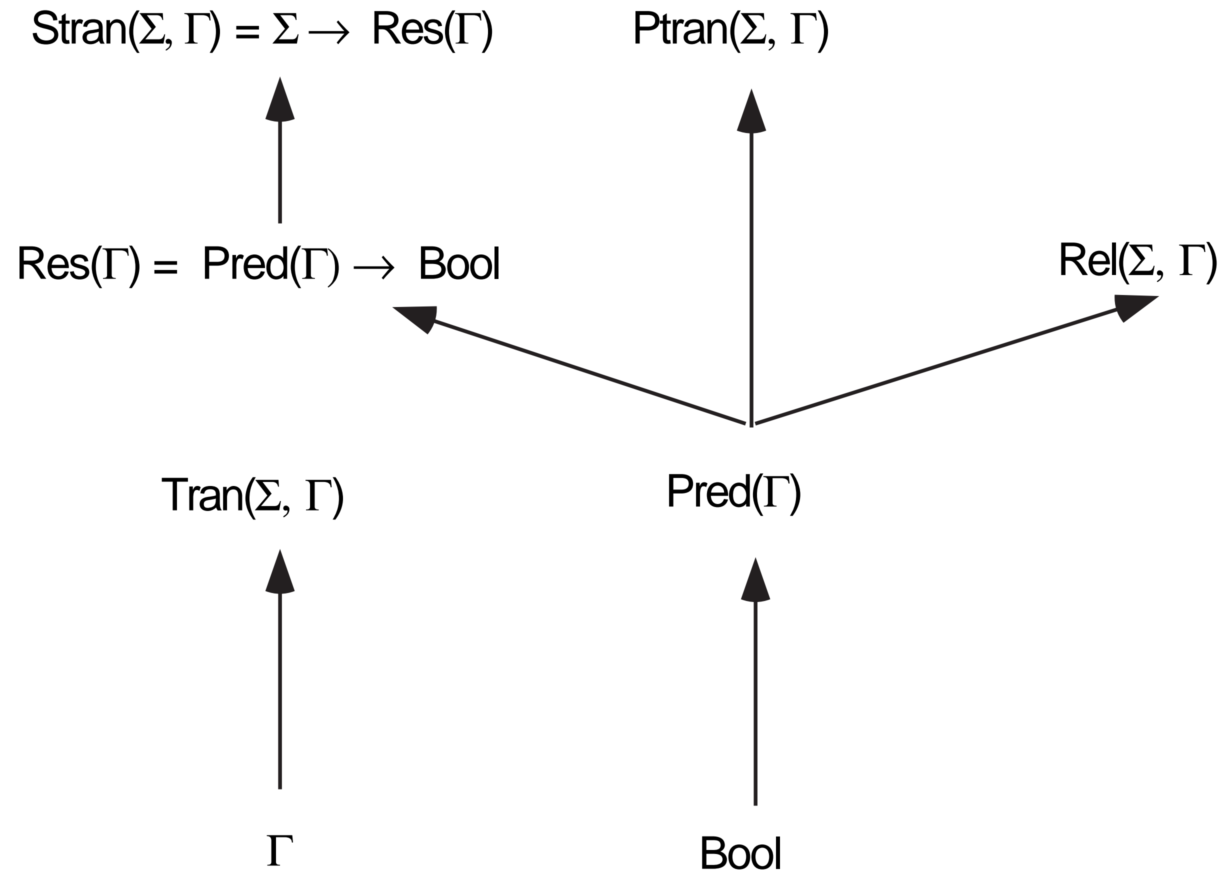
New domains

The choice semantics introduces two new domains into the refinement calculus hierarchy:

$$\begin{aligned} \mathsf{Res}(\Gamma) &\triangleq \{Q : \mathcal{P}(\mathcal{P}(\Gamma)) \mid Q \text{ upward closed}\} \\ \mathsf{Stran}(\Sigma, \Gamma) &\triangleq \Sigma \to \mathsf{Res}(\Gamma) \end{aligned}$$

- $\mathsf{Res}(\Gamma)$ is a complete lattice when ordered by subset inclusion $\subseteq$.
- $\mathsf{Stran}(\Sigma, \Gamma)$ is a complete lattice when ordered by the pointwise extension of subset inclusion.

Extended hierarchy



Conclusions

Conclusions

- We describe a computing system in terms of agents that interact with each other.
- The agents manipulate the world by changing a state, described in terms of program variables.
- We give a language of contracts that allows us to describes the options and obligations that the agents have, and an operational semantics for this language.
- We study whether a group of agents can reach their goal by co-operating in their choices, in spite of hostile actions from the other agents (existence of winning strategy)
- We show that by extending the traditional weakest precondition semantics, we get simple rules for computing when a collection of agents can reach their goal.
- The same weakest precondition rules allow us to reason about correctness of contracts, games and programs, as well as about refinement of these.

- We study the lattice theoretic properties of contracts, when interpreted as predicate transformers
- We study how to reason at different levels of the refinement calculus hierarchy.
- We identify standard subclasses of contracts, corresponding to traditional semantics
- We introduce a new forward semantics for contracts.