Event-B: Introduction and First Steps[1]

Manuel Carro
manuel.carro@upm.es

Universidad Politécnica de Madrid &
IMDEA Software Institute

[1] Many slides borrowed from J. R. Abrial

Conventions

I will sometimes use boxes with different meanings.

- Quiz to do together during the lecture.

Q: What happens in this case?

solution
solution
solution

- Material / solutions that I want to develop during the lecture.

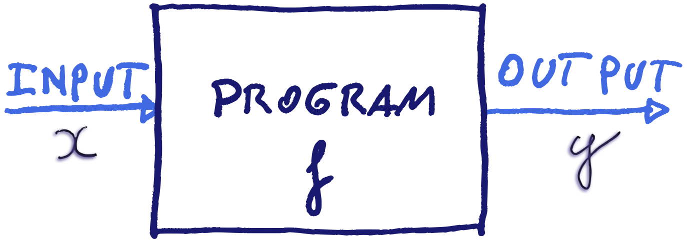
Something to complete here

aaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaa

Event B

An industry-oriented method, language, and set of supporting tools to describe systems of interacting, reactive software, hardware components, and their environment, and to reason about them.

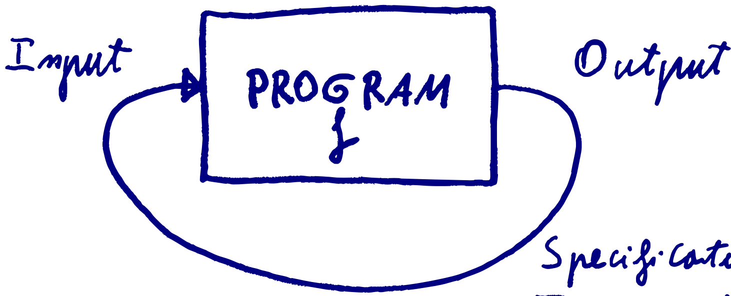
Sequential vs. reactive software



$y = f(x)$

Specification: remember sorting program.

Sequential vs. reactive software



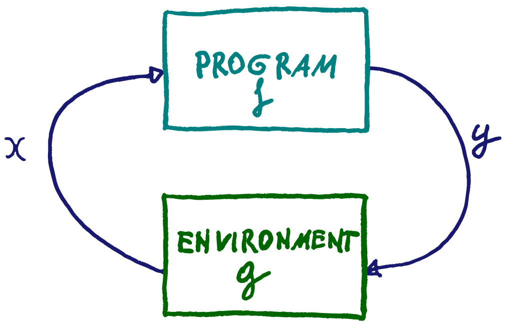
Specification?

Termination?

Correctness?

$x_{i+1} = f(x_i)$

Sequential vs. reactive software



$y_0 = f(x_0),\ x_1 = g(y_0),\ y_1 = f(x_1),\ x_2 = g(y_1), \dots$

Effects of environment?

Typical approaches and problems

Usual approach

- Choose a platform.
- Write software specifications (which often neglect or under-represent the environment).
- Design by cutting in small pieces with well-defined communication.
- Code and test / verify units.
- Integrate and test.

Pitfalls

- Often too many details / interactions / properties to take into account.
- Cutting in pieces: poor job in taming complexity.
  - Small pieces: easy to prove them right.
  - Additional relationships created!
  - Overall complexity reduced?
- Modeling environment?
  E.g., we expect a car driver to stop at a red light.
- Result: system as a whole seldom verified.

The Event B approach

Complexity: Model Refinement

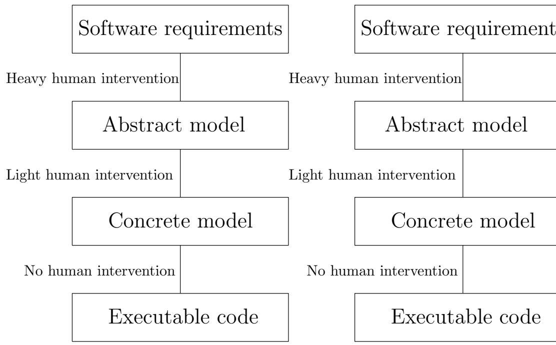
- System built incrementally, monotonically.
  - Take into account subset of requirements at each step.
  - Build model of a *partial* system.
  - Prove its correctness.
- **Add** requirements to the model, ensure correctness:
  - The requirements correctly captured by the new model.
  - New model preserves properties of previous model.

Details: Tool Support

- Tool to edit Event B models (Rodin).
- Generates *proof obligations*: theorems to be proved to ensure correctness.
- Interfaced with (interactive) theorem provers.
- Extensible.

Refinement

- Refinement allows us to build a model gradually.
- Ordered sequence of more precise partial models.
- Each model is a refinement of the one preceding it.
- Each model is proven:
  - Correct.
  - Respecting the boundaries of the previous one.

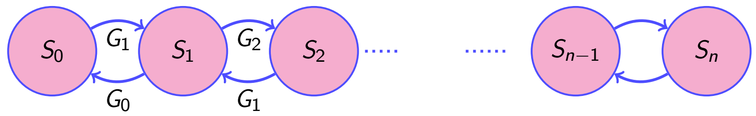


Basic ideas

- Model: formal description of a discrete system.
  - Formal: sound mechanism to decide whether some properties hold
  - Discrete: can be represented as a transition system
- Formalization contains models of:
  - The future software components
  - The future equipments surrounding these components

Models and states

A discrete model is made of states



- States are represented by constants, variables, and their relationships

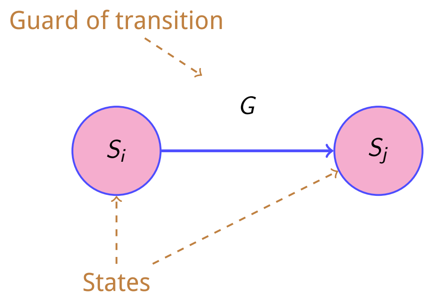
$$S_i = \langle c_1, \ldots, c_n, v_1, \ldots, v_m \rangle$$

- Relationships among constants and variables written using set-theoretic expressions

What is its relationship with a regular program?

States and transitions

- Transitions between states: triggered by events
- Events: guards and actions
  - Guard (G_i) denote enabling conditions of events
  - Actions denote how states are modified by events
- Guards and actions written with set-theoretic expressions (e.g., first-order, classical logic).



Examples:

$S_i \equiv x = 0 \wedge y = 7$
$S_j \equiv x, y \in \mathbb{N} \wedge x < 4 \wedge y < 5 \wedge x + y < 7$

Write extensional definition for the latter

A simple example – informal introduction!

Search for element `k` in array `f` of length `n`, assuming `k` is in `f`.

Constants / Axioms

```
CONST n ∈ ℕ
CONST f ∈ 1..n ⟶ ℕ
CONST k ∈ ran(f)
```

Variables / Invariants

```
VARIABLE i ∈ 1..n
```

```
Event Search
  when
    i < n ∧ f(i) ≠ k
  then
    i := i + 1
  end
```

```
Event Found
  when
    f(i) = k
  then
    skip
  end
```

(initialization of `i` not shown for brevity)

Events

```
Event EventName
  when
    guard:  G(v, c)
  then
    action:  v := E(v, c)
  end
```

- Executing an event (normally) changes the system state.
- An event may[2] fire when its guard evaluates to true.
- G(v, c) predicate that enables EventName
- v := E(v, c) is a state transformer.

[2] Not "must"!

Intuitive operational interpretation

```
Initialize;
while (some events have true guards) {
  Choose one such event;
  Modify the state accordingly;
}
```

```
Event EventName
  when
    guard:  G(v, c)
  then
    action:  v := E(v, c)
  end
```

- Now: **informal** Event B semantics.
- Actual Event B semantics based on set theory and invariants — Later!

- An event execution takes no time.
  - No two events occur simultaneously.
- If all guards false, system stops.
- Otherwise: choose one event with enabled guard, execute action, modify state.
- Repeat previous point if possible.

Fairness: what is it? What should we expect?

Comments on the operational interpretation

- Stopping is not necessary: a discrete system may run forever.
- This interpretation is just given here for informal understanding
- The meaning of such a discrete system will be given by the proofs which can be performed on it (next lectures).

On using sequential code

To help understanding, we will now write some sequential code first, translate it into Event B, and then proving correctness. This does not follow Event B workflow, which goes in the opposite direction: write Event B models and derive sequential / concurrent code from them.

Running example (sequential code)

$$a = \left\lfloor \frac{b}{c} \right\rfloor$$

- Characterize it: we want to define integer division, without using division.

Q: specification of division

$$\forall b \forall c\, [b \in \mathbb{N} \land c \in \mathbb{N} \land c > 0 \Rightarrow \exists a \exists r\, [a \in \mathbb{N} \land r \in \mathbb{N} \land r < c \land b = c \times a + r]]$$

It is useful to categorize the specification as assumptions (preconditions)

$$b \in \mathbb{N} \land c \in \mathbb{N} \land c > 0$$

and results (postconditions)

$$a \in \mathbb{N} \land r \in \mathbb{N} \land r < c \land b = c \times a + r$$

Input / output / variables / constants / types?

Two Math Notes

Zero

There is no universal agreement about whether to include zero in the set of natural numbers. Some authors begin the natural numbers with 0, corresponding to the non-negative integers 0, 1, 2, 3, ..., whereas others start with 1, corresponding to the positive integers 1, 2, 3, ... This distinction is of no fundamental concern for the natural numbers as such.

I will assume that $0 \in \mathbb{N}$. That is the convention in computer science.

If you write $\forall b \in \mathbb{N}, c \in \mathbb{N}, c > 0 \cdot \exists a \in \mathbb{N}, r \in \mathbb{N}, r < c \cdot b = c \times a + r$ remember:

- Quantifier scope sometimes implicit.
- Commas mean conjunction.
- Nesting may need disambiguation.

- $\forall x \in D \cdot P(x)$ means $\forall x[x \in D \Rightarrow P(x)]$
- $\exists x \in D \cdot P(x)$ means $\exists x[x \in D \land P(x)]$

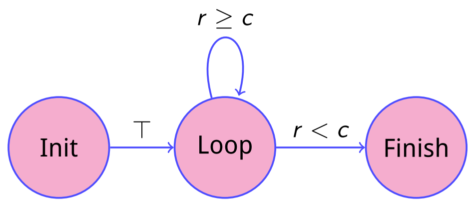
See https://twitter.com/lorisdanto/status/1354128808740327425?s=20
and https://twitter.com/lorisdanto/status/1354214767590842369?s=20

Programming integer division

- We have addition and substracion
- We have a simple procedural language
- Variables, assignment, loops, if-then-else, + & -, arith. operators, ...

Q: integer division code

```
a := 0
r := b
while r >= c
   r := r - c
   a := a + 1
```



Copy the code! We will need it!

This step is not taken in Event B. We are writing this code only for illustration purposes.

Towards events

Template

```
Event EventName
  when
    G(v, c)
  then
    v := E(v, c)
  end
```

Code

```
a := 0
r := b
while r >= c
  r := r - c
  a := a + 1
end
```

- Special initialization event (**INIT**).
- Sequential program (special case):
  - *Finish* event, *Progress* events
  - Determinism: guards exclude each other Prove!
  - Non-deadlock: some guard always true Prove!
  - Termination: a variable is always reduced Prove!

Q: integer division events

```
Event INIT              Event Progress              Event Finish
  a, r = 0, b             when                        when
end                         r >= c                      r < c
                          then                        then
                            r, a := r - c, a + 1        skip
                          end                         end
```

Categorizing elements

Constants	Axioms (Write them down separately!)
Q: constants b c	Q: axioms $b \in \mathbb{N}$ $c \in \mathbb{N}$ $c > 0$
Variables	**Invariants**
Q: variables a r	Later!

```
Event INIT         Event Progress           Event Finish
  a, r = 0, b        when r >= c              when r < c
end                  then                     then
                       r, a := r - c, a + 1     skip
                     end                      end
```

Proving correctness

How do **you** prove your programs correct?

- Correctness in sequential programs: post-condition holds.
- Easy if no (or statically bound) loops.
- Prove that this code swaps x and y:

$\{x = a, y = b\}$
x := x + y; $\{x = a + b, y = b\}$
y := x − y; $\{x = a + b, y = a\}$
x := x − y; $\{x = b, y = a\}$
$\{x = b, y = a\}$

Proving correctness: invariants in a nutshell

Loops: much more difficult

- # iterations unknown. (remember Collatz's conjecture)

$\{I(a, r)\}$
while r >= c do
  $\{I(a, r)\}$
  r := r − c
  a := a + 1
  $\{I(a, r)\}$
end
$\{I(a, r) \wedge r < c \Rightarrow a = \lfloor \frac{b}{c} \rfloor\}$

Note: we should prove termination as well!

Invariant: formula that is "always" true.

- Procedural code: beginning and end of every loop iteration.
- Event-B: after initialization, after every event (essentially same idea).

Intuitition:

- If invariant and negation of loop condition implies postcondition, the postcondition is proved.
- Nobody gives us invariants.
  - We have to find them.
  - We have to prove they are invariants.

Finding invariants

Which assertions are invariant in our model?

Q: model invariants

I_1: $a \in \mathbb{N}$ // *Type invariant*
I_2: $r \in \mathbb{N}$ // *Type invariant*
I_3: $b = a \times c + r$

One formula that is an invariant for **any** Event-B model / loop.

Q: trivial invariant

$\top$

```
Event INIT
  a, r = 0, b
end
```

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

```
Event Finish
  when r < c
  then
    skip
  end
```

Copy invariants somewhere else – we will need to have them handy

Invariant preservation in Event B

- Invariants must be true before and after event execution.
- For all event i, invariant j:

Establishment:

$$A(c) \vdash I_j(E_{\text{init}}(v, c), c)$$

Preservation:

$$A(c), G_i(v, c), I_{1\ldots n}(v, c) \vdash I_j(E_i(v, c), c)$$

- $A(c)$ axioms
- $G_i(v, c)$ guard of event i
- $I_j(v, c)$ invariant j
- $I_{1\ldots n}(v, c)$ all the invariants
- $E_i(v, c)$ result of action i

Sequent

$$\Gamma \vdash \Delta$$

Show that Δ can be proved using assumptions Γ

Invariant preservation

If an invariant holds and the guards of an event are true and we execute the event's action, the invariant should hold.

Invariant preservation proofs

- Invariant preservation proven using model and math axioms.
- Three invariants & three events: nine proofs
- Named as e.g. $E_{\text{Progress}}/I_2/\text{INV}$
  - Other proofs will be necessary later

E_{INIT} / I_1 / INV

INIT I1 invariant proof

$$\dfrac{\dfrac{}{\vdash 0 \in \mathbb{N}}\ \text{P0}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0 \vdash 0 \in \mathbb{N}}\ \text{MON}$$

E_{INIT} / I_2 / INV

INIT I2 invariant proof

$$\dfrac{\dfrac{}{b \in \mathbb{N} \vdash b \in \mathbb{N}}\ \text{HYP}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0 \vdash b \in \mathbb{N}}\ \text{MON}$$

```
Event INIT
  a, r = 0, b
end
```

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

Invariant preservation proofs

E_{INIT} / I_3 / INV

INIT I3 invariant proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash b = b}\ \text{EQL}}{\vdash b = 0 + b}\ \text{Arith}}{\vdash b = 0 \times c + b}\ \text{Arith}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0 \vdash b = 0 \times c + b}\ \text{MON}$$

E_{Progress} / I_1 / INV

Progress I1 invariant proof

$$\dfrac{\dfrac{}{a \in \mathbb{N} \vdash a + 1 \in \mathbb{N}}\ \text{P1}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, r \in \mathbb{N}, b = a \times c + r, a \in \mathbb{N} \vdash a + 1 \in \mathbb{N}}\ \text{MON}$$

```
Event INIT
  a, r = 0, b
end
```

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

Sequents

- Mechanize proofs
  - Humans “understand”; proving is tiresome and error-prone
  - Computers manipulate symbols
- How can we mechanically construct correct proofs?
  - Every step crystal clear
  - For a computer to perform
- Several approaches
- For Event B: sequent calculus
  - To read: [Pau] (available at course web page), at least Sect. 3.3 to 3.5 , 5.4, and 5.5. Note: when we use $\Gamma \vdash \Delta$, Paulson uses $\Gamma \Rightarrow \Delta$.
  - Also: [Orib, Oria], available at the course web page.
- Admissible deductions: inference rules.

Inference rules

- An inference rule is a tool to build a formal proof.
  - It not only tells you whether $\Gamma \vdash \Delta$: it tells you how.
- It is denoted by:

$$\frac{A}{C}\,R$$

- A is a (possibly empty) collection of sequents: the antecedents.
- C is a sequent: the consequent.
- R is the name of the rule.

The proofs of each sequent of A

——— together give you ———

a proof of sequent C

An example of inference rule

Note: not exactly the inference rules we will use.
Only an intuitive example.

- A(lice) and B(ob) are siblings:

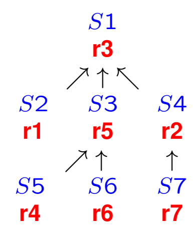
$$\frac{\text{C is mother of A} \qquad \text{C is mother of B}}{\text{A and B are siblings}}\,\text{Sibling-M}$$

$$\frac{\text{C is father of A} \qquad \text{C is father of B}}{\text{A and B are siblings}}\,\text{Sibling-F}$$

- Note: we do not consider the case that, e.g., C is a father and a mother.

Recording the Proof of Sequent $S1$

$$\frac{}{S2}\mathbf{r1} \quad \frac{S7}{S4}\mathbf{r2} \quad \frac{S2 \;\; S3 \;\; S4}{S1}\mathbf{r3} \quad \frac{}{S5}\mathbf{r4} \quad \frac{S5 \;\; S6}{S3}\mathbf{r5} \quad \frac{}{S6}\mathbf{r6} \quad \frac{}{S7}\mathbf{r7}$$



- The proof is a tree

Deduction systems

- There are many formal deduction systems [Ben12, Sect. 3.9].
- We will use a variant of the so-called *Gentzen* deduction systems.

Sequent $\Gamma \vdash \Delta$ in a Gentzen system

- Γ: (possibly empty) collection of formulas (the hypotheses)
- Δ: collection of formulas (the goal)
- Objective: show that, under hypotheses Γ, some formula(s) in Δ can be proven.

$\Gamma \equiv P_1, P_2, \ldots, P_n$ stands for $P_1 \wedge P_2 \wedge \ldots \wedge P_n$

$\Delta \equiv Q_1, Q_2, \ldots, Q_m$ s.f. $Q_1 \vee Q_2 \vee \ldots \vee Q_m$

$$\boxed{P_1, P_2, \ldots, P_n \vdash Q_1, Q_2, \ldots, Q_m}$$
is
$$\boxed{P_1 \wedge P_2 \wedge \ldots \wedge P_n \vdash Q_1 \vee Q_2 \vee \ldots \vee Q_m}$$

- We will use a proof calculus where the goal is a single formula.
- More constructive proofs — but see [Oria, Section 11.2] for interesting remarks.

Inside a sequent

- We need a language to express hypothesis and goals.
  - Not formally defined yet
  - We will assume it is first-order, classical logic
  - Recommended references: [Pau, HR04, Ben12]
- We need a way to determine if (and how) Δ can prove Γ.
  - Inference rules.

Logic and inference rules

Inference rules

Structural
- Hypothesis
- Monotony
- Cut

Depending on logic
- Propositional
- First order
- Temporal
- Higher order
- ...

For specific theories
- Sets
- Relations
- Functions
- (Linear) Arithmetic
- Reals
- Strings
- Arrays
- Bitvectors
- Records
- Difference logic
- Inductive data types
- Empty theory
- ...

Structural inference rules

- Three structural inference rules, independent of the logic used.

HYPothesis

$$\frac{\top}{H, P \vdash P}\ \mathrm{HYP}$$

If the goal is among the hypothesis, we are done.

MONotony

$$\frac{H \vdash Q}{H, P \vdash Q}\ \mathrm{MON}$$

If goal is proved without hypothesis P, then it can be proven with P.

CUT

$$\frac{H \vdash P \qquad H, P \vdash Q}{H \vdash Q}\ \mathrm{CUT}$$

A goal can be proven with an intermediate deduction P. Nobody tells us what is P or how to come up with it. It *cuts* the proof into smaller pieces. (*Cut Elimination Theorem*)

More rules

- There are many other inference rules for:
  - Logic itself (propositional / predicate logic)
    - Look at the slides / documents in the course web page
  - reasoning on arithmetic (Peano axioms),
  - reasoning on sets,
  - reasoning on functions,
  - ...
- We will not list all of them here (see online documentation).
- We may need to explain them as they appear.
- But a mechanical prover has them as "inside knowledge" (plus tactics, strategies)

Connectives

- Given predicates P and Q, we can construct:

- NEGATION: $\neg P$

- CONJUNCTION: $P \wedge Q$

- IMPLICATION: $P \Rightarrow Q$

- Precedence: $\neg, \wedge, \Rightarrow$.
  - Examples
- Parenthesis added when needed.
  - If in doubt: add parentheses!
- Can you build the truth tables?
- $\vee$, $\Leftrightarrow$ are defined based on them.
  - Define them
  - Can we use a **single** connective?

Rules for conjunction

$$\frac{H \vdash Q \qquad H \vdash P}{H \vdash P \wedge Q}\ \text{AND-R}$$

A conjunction on the RHS needs both branches of the conjunction to be proven independently of each other.

$x \in \mathbb{N}1, y \in \mathbb{N}1, x + y < 5 \vdash x < 4 \wedge y < 4$

$$\frac{H, P, Q \vdash R}{H, P \wedge Q \vdash R}\ \text{AND-L}$$

By definition of sequent.

Rules for disjunction

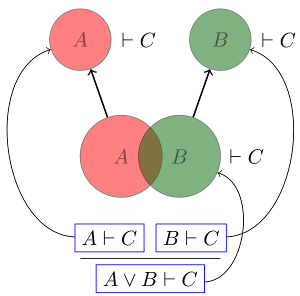
$$\frac{H, Q \vdash R \qquad H, P \vdash R}{H, P \vee Q \vdash R}\ \text{OR-L}$$

A disjunction on the LHS needs both branches of the disjunction be discharged separately.

$(x < 0 \wedge y < 0) \vee x + y > 0 \vdash x \times y > 0$

Counterexample?

LHS: **all** conditions in which RHS has to hold. Removing part of disjunction makes "condition space" smaller (removing part of conjunction makes the "condition space" larger, more general). Proofs with more general assumptions are valid for less general assumptions, not the other way around.



Rules for disjunction (cont.)

$$\frac{H \vdash P}{H \vdash P \vee Q}\ \text{OR-R1} \qquad \frac{H \vdash Q}{H \vdash P \vee Q}\ \text{OR-R2}$$

A disjunction on the RHS only needs ***one*** *of the branches to be proven. There is a rule for each branch.*

$$\frac{H, \neg P \vdash Q}{H \vdash P \vee Q}\ \text{NEG}$$

Part of a disjunctive goal can be negated, moved to the hypotheses, and used to discharge the proof. Related to $\neg P \vee Q$ *being* $P \Rightarrow Q$*.*

$x \in \mathbb{N}, y \in \mathbb{N}, x + y > 1, y > x \vdash x > 0 \vee y > 1$

Rules for negation

$$\frac{}{\perp \vdash Q}\ \text{CNTR}$$

$$\frac{}{P, \neg P \vdash Q}\ \text{NOT-L}$$

If we reach to a contradiction in the hypotheses, anything can be proven (principle of explosion). Note: not everyone accepts this – more on that later.

$$\frac{H, \neg P \vdash \neg Q \qquad H, \neg P \vdash Q}{H \vdash P}\ \text{NOT-R}$$

Reductio ad absurdum: assume the negation of what we want to prove and reach a contradiction. Similarly with $H \vdash \neg P$*.*

$P \wedge \neg P \equiv \perp$ (False) $\qquad P \vee \neg P \equiv \top$ (True) $\qquad \top = \neg\perp$

Rules for implication

$$\frac{H \vdash P \qquad H, Q \vdash R}{H, P \Rightarrow Q \vdash R}\ \text{IMP-L}$$

If we want to use $P \Rightarrow Q$*, we show that* P *is deducible from* H *and that, assuming* Q*, we can infer* R*.*

$$\frac{H, P \vdash Q}{H \vdash P \Rightarrow Q}\ \text{IMP-R}$$

We move the LHS P *to the hypotheses. Note that since* $P \Rightarrow Q$ *is* $\neg P \vee Q$*, we are applying the NEG rule in disguise.*

$x \in \mathbb{N}, y \in \mathbb{N}, x + y > k \vdash x = k \Rightarrow y > 0$

Additional rules

Equality axiom

$$\frac{}{\vdash E = E}\ \text{EQL}$$

Equality propagation

$$\frac{H(F), E = F \vdash P(F)}{H(E), E = F \vdash P(E)}\ \text{EQL-LR}$$

First Peano axiom

$$\frac{}{\vdash 0 \in \mathbb{N}}\ \text{P0}$$

Second Peano axiom

$$\frac{}{n \in \mathbb{N} \vdash n + 1 \in \mathbb{N}}\ \text{P1}$$

Forthcoming proofs and propositional rules

The following proofs feature variables. Strictly speaking, they are not propositional. We will however not use quantifiers, so we will treat formulas as propositions when applying the previous rules.
We will assume the existence of simple, well-known arithmetic rules.

Invariant preservation proofs

E_{Progress} / I_2 / INV

Progress I2 invariant proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 \in \mathbb{N}}\,\text{P0}}{\vdash c - c \in \mathbb{N}}\,\text{Arith}}{c \in \mathbb{N}, c \in \mathbb{N} \vdash c - c \in \mathbb{N}}\,\text{MON}}{c \in \mathbb{N}, r = c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\,\text{EQ-LR} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{r - c > 0 \vdash r - c \in \mathbb{N}}\,\text{Arith}^*}{c \in \mathbb{N}, r - c > 0, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\,\text{MON}}{c \in \mathbb{N}, r - c > c - c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\,\text{Simp-M-Minus}}{c \in \mathbb{N}, r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\,\text{Arith-M-M-R}}{c \in \mathbb{N}, r = c \vee r > c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\,\text{OR-L}}{c \in \mathbb{N}, r \geq c, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\,\text{Arith}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, b = a \times c + r, r \in \mathbb{N} \vdash r - c \in \mathbb{N}}\,\text{MON}$$

I_2: $r \in \mathbb{N}$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

Invariant preservation proofs

E_{Progress} / I_3 / INV

Progress I3 invariant proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{b = a \times c + r \vdash b = a \times c + r}\,\text{HYP}}{b = a \times c + r \vdash b = a \times c + c + r - c}\,\text{Arith-M-Pl-Dist}}{b = a \times c + r \vdash b = (a + 1) \times c + r - c}\,\text{Arith-M-Pl-Dist}}{b = a \times c + r \vdash b = (a + 1) \times c + (r - c)}\,\text{Arith-Pl-M}}{b \in \mathbb{N}, c \in \mathbb{N}, c > 0, r \geq c, a \in \mathbb{N}, r \in \mathbb{N}, b = a \times c + r \vdash b = (a + 1) \times c + (r - c)}\,\text{MON}$$

I_3: $b = a \times c + r$

```
Event Progress
  when r >= c
  then
    r, a := r - c, a + 1
  end
```

Invariant preservation proofs

Proofs for `Finish`

- $E_{\text{Finish}}/I_1/\text{INV}$
- $E_{\text{Finish}}/I_2/\text{INV}$
- $E_{\text{Finish}}/I_3/\text{INV}$

are trivial (Finish does not change anything)

Correctness: when Finish is executed, $I_3 \wedge G_{\texttt{Finish}} \Rightarrow a = \left\lfloor \frac{b}{c} \right\rfloor$ (with the definition given for integer division).

The first-order predicate calculus and its rules

- Handling of variables, expressions, quantifiers, instantiation.
- There is a universe of objects.
- An expression is a formal text denoting an object: *apple, adam, father(adam), 3, 8 + 3^2, {adam, apple, 3^2}*.
  - Expressions include set-theoretic and arithmetic notation.
- Predicates state properties of objects through the expressions that denote them.
- A predicate denotes nothing.
- An expression cannot be proved.
- A predicate cannot be evaluated.
- Predicates and expressions are not interchangeable.

First-order predicate calculus: informal

We have a universe of objects. We make statements about these objects. Some examples follow.

$P(a)$: property P is true for object a

$P(a) \wedge \neg Q(b)$: property P is true for object a and property Q is false for object b

$R(a, b) \Longrightarrow P(a) \vee P(b)$: if property R is true for a and b, then P is true for a, for b, or for both.

$\forall x \cdot P(x)$: For all elements x, P is true. P can be arbitrarily complex.

$\exists x \cdot P(x)$: For some element x, P is true. P can be arbitrarily complex.

Sweet Reason: A Field Guide to Modern Logic [HGTA11] is a delightful introduction to logic with many examples.

First-order predicate calculus: informal

$l(x, y)$	x loves y
$\forall x \cdot \forall y \cdot l(x, y)$	everyone loves everyone else (including themself)
$\exists x \cdot \exists y \cdot l(x, y)$	at least a person loves someone
$\forall x \cdot \exists y \cdot l(x, y)$	everybody loves someone (not necessarily the same person)
$\exists y \cdot \forall x \cdot l(x, y)$	there is someone who is loved by everybody
$\forall y \cdot \exists x \cdot l(x, y)$	everybody is loved by someone
$\exists x \cdot \forall y \cdot l(x, y)$	there is someone who loves everybody
$\forall x \cdot \neg l(x, x)$	no one loves themself

$\forall x \cdot \forall y \cdot [l(x, y) \wedge \exists z \cdot l(y, z) \Rightarrow \neg l(x, z)]$

What happens if we don't want someone loving him/herself to be taken into account?

"If there is someone who is loved by everybody, then it is not the case that no one loves themself." We usually want to prove statements true or false. We use inference rules to prove truth or falsehood.

Some deductions and (non) equivalences

$$\forall x \cdot P(x) \equiv \neg \exists x \cdot \neg P(x)$$

(definition of existential quantifier)

$$\exists x \cdot \forall y \cdot P(x, y) \Rightarrow \forall y \cdot \exists x \cdot P(x, y)$$

$$\forall y \cdot \exists x \cdot P(x, y) \not\Rightarrow \exists x \cdot \forall y \cdot P(x, y)$$

(Counterexample?)

$$P(a) \Rightarrow \exists x \cdot P(x)$$

$$\forall x \cdot (P(x) \Rightarrow B) \equiv (\exists x \cdot P(x) \Rightarrow B) \quad (x \notin vars(B))$$

$$\forall x \cdot (P(x) \wedge Q(x)) \equiv \forall x \cdot P(x) \wedge \forall x \cdot Q(x)$$

$$\exists x \cdot (P(x) \vee Q(x)) \equiv \exists x \cdot P(x) \vee \exists x \cdot Q(x)$$

$$\forall x \cdot (P(x) \vee Q(x)) \not\equiv \forall x \cdot P(x) \vee \forall x \cdot Q(x)$$

(Counterexample?)

$$\exists x \cdot (P(x) \wedge Q(x)) \not\equiv \exists x \cdot P(x) \wedge \exists x \cdot Q(x)$$

(Counterexample?)

First-order predicate calculus: inference rules

$$\frac{\mathbf{H},\ \forall \mathbf{x} \cdot \mathbf{P(x)},\ \mathbf{P(E)} \ \vdash \ \mathbf{Q}}{\mathbf{H},\ \forall \mathbf{x} \cdot \mathbf{P(x)} \ \vdash \ \mathbf{Q}} \quad \mathbf{ALL\_L}$$

where **E** is an expression

$$\frac{\mathbf{H} \ \vdash \ \mathbf{P(x)}}{\mathbf{H} \ \vdash \ \forall \mathbf{x} \cdot \mathbf{P(x)}} \quad \mathbf{ALL\_R}$$

- In rule **ALL_R**, variable **x** is not free in **H**

First-order predicate calculus: inference rules

$$\frac{\mathbf{H},\ \mathbf{P(x)}\ \vdash\ \mathbf{Q}}{\mathbf{H},\ \exists \mathbf{x}\cdot\mathbf{P(x)}\ \vdash\ \mathbf{Q}}\ \textbf{XST\_L}$$

- In rule **XST_L**, variable **x** is not free in **H** and **Q**

$$\frac{\mathbf{H}\ \vdash\ \mathbf{P(E)}}{\mathbf{H}\ \vdash\ \exists \mathbf{x}\cdot\mathbf{P(x)}}\ \textbf{XST\_R}$$

where **E** is an expression

First-order predicate calculus: inference rules

Rules for equality (some already seen):

$$\frac{\mathbf{H(F),\ E = F\ \vdash\ P(F)}}{\mathbf{H(E),\ E = F\ \vdash\ P(E)}}\ \textbf{EQ\_LR} \qquad \frac{\mathbf{H(E),\ E = F\ \vdash\ P(E)}}{\mathbf{H(F),\ E = F\ \vdash\ P(F)}}\ \textbf{EQ\_RL}$$

$$\frac{}{\vdash\ \mathbf{E = E}}\ \textbf{EQL}$$

$$\frac{\mathbf{H\ \vdash\ E\ =\ G\ \wedge\ F\ =\ I}}{\mathbf{H\ \vdash\ E \mapsto F\ =\ G \mapsto I}}\ \textbf{PAIR}$$

Note: $\mathsf{E} \mapsto \mathsf{F}$ denotes a *pair* (E, F) — we will use them later.

Inductive and non-inductive invariants

- We want to prove

$$A(c)\ \vdash I_j(E_{\text{init}}(v,c),c)$$
$$A(c), G_i(v,c), I_{1\ldots n}(v,c)\ \vdash I_j(E_i(v,c),c)$$

- I_j: *inductive invariant* (base case + inductive case)
- Invariants can be true but non-inductive if they cannot be proved from program

```
Event INIT          Event Loop
  a:  x := 1          a:  x := 2*x - 1
end                 end
```

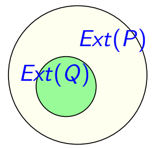
- $x \geq 0$ looks like an invariant. Prove it is preserved.
- It is not inductive (`Loop`: $x \geq 0\ \vdash 2 * x - 1 \geq 0$?)
- $x > 0$ is inductive (Prove it!)

- $x > 0$ is stronger than $x \geq 0$ (if $A \Rightarrow B$, A stronger than B.)
- Stronger invariants are preferred – as long as they are still invariants!

Proof by contradiction: why?

$$\frac{}{\perp\ \vdash P}\ \text{CNTR}$$

- Common sense: if we are in an impossible situation, just do not bother.
- Proof-based:
  - Let's assume Q and $\neg Q$.
  - Then $\neg Q$.
  - Then $\neg Q \vee P \equiv Q \Rightarrow P$.
  - But since $Q \wedge (Q \Rightarrow P)$, then P.
- Model-based:
  - If $Q \Rightarrow P$, then $Q\ \vdash P$.
  - Extension: $Ext(P) = \{x | P(x)\}$ (id. Q).
  - $Q \Rightarrow P$ iff $Ext(Q) \subseteq Ext(P)$. Why???



  - If $Q \equiv R \wedge \neg R$, $Ext(Q) = \varnothing$.
  - $\varnothing \subseteq S$, for any S.
  - Therefore, $Ext(R \wedge \neg R) \subseteq Ext(P)$ for any P.
  - Thus, $R \wedge \neg R \Rightarrow P$ and then $\perp\ \vdash P$.

Mordechai Ben-Ari.
Mathematical Logic for Computer Science, 3rd Edition.
Springer, 2012.

James M. Henle, Jay L. Garfield, Thomas Tymoczko, and Emily Altreuter.
Sweet Reason: A Field Guide to Modern Logic.
Wiley-Blackwell, 2nd edition, 211.
ISBN: 978-1-444-33715-0.

Michael Huth and Mark Ryan.
Logic in Computer Science: Modelling and Reasoning About Systems.
Cambridge University Press, New York, NY, USA, 2004.

Original Author Unclear.
Lecture 11: Refinement Logic.
Available at `https://www.cs.cornell.edu/courses/cs4860/2009sp/lec-11.pdf`, last acccessed on Jan 30, 2022.

Original Author Unclear.
Lecture 9: From Analytic Tableaux to Gentzen Systems.
Available at `https://www.cs.cornell.edu/courses/cs4860/2009sp/lec-09.pdf`, last acccessed on Jan 30, 2022.

Lawrence C. Paulson.
Logic and Proof.
Lecture notes, U. of Cambridge, available at `https://www.cl.cam.ac.uk/teaching/2122/LogicProof/logic-notes.pdf`, last acccessed on Feb 9, 2022.