Systeme Hoher Qualität und Sicherheit
Vorlesung 5 vom 18.11.2013: High-Level Specification and Modelling

Christoph Lüth & Christian Liguda

Universität Bremen

Wintersemester 2013/14

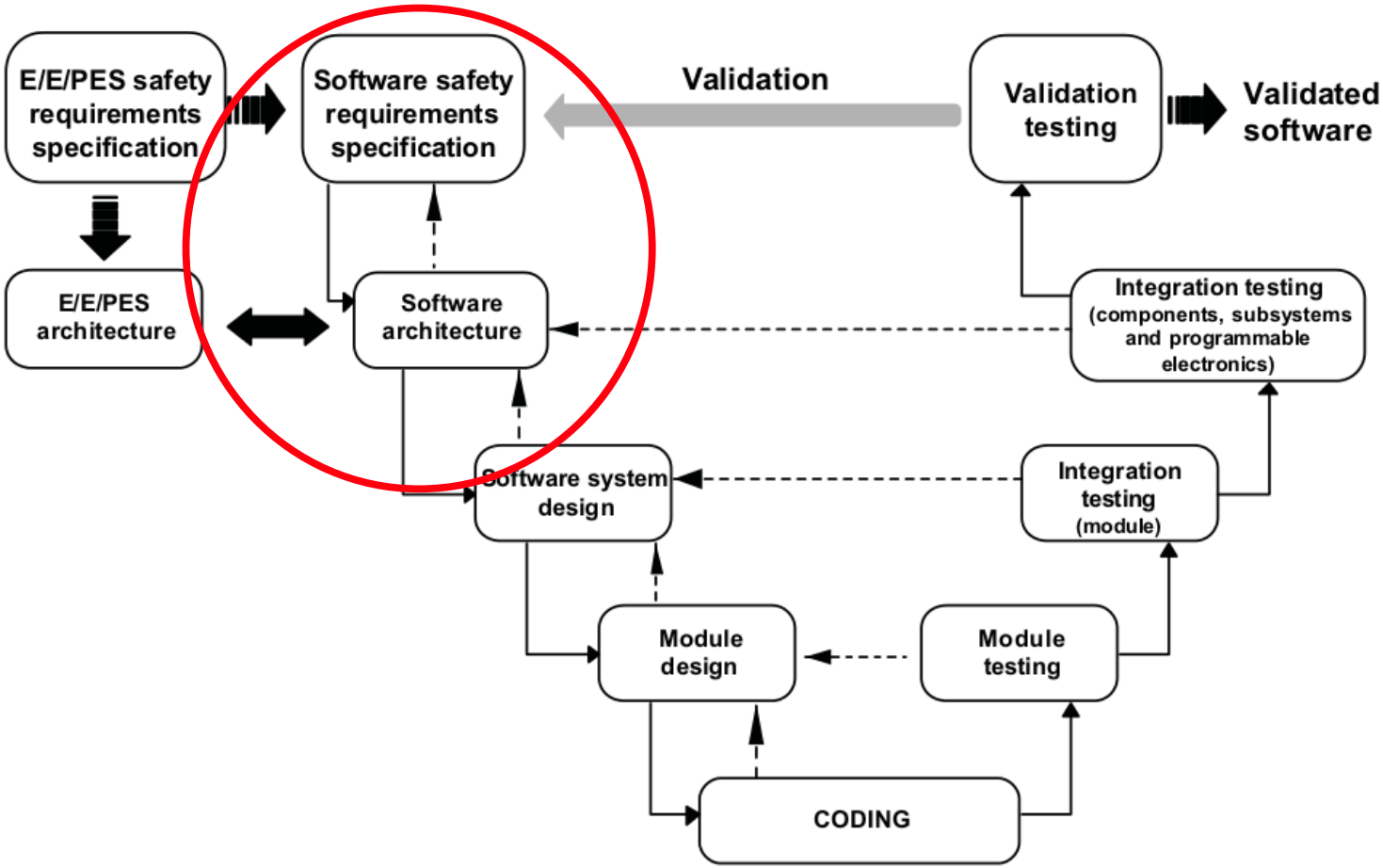
Where are we?

- Lecture 1: Concepts of Quality
- Lecture 2: Concepts of Safety and Security, Norms and Standards
- Lecture 3: Quality of the Software Development Process
- Lecture 4: Requirements Analysis
- **Lecture 5: High-Level Design & Formal Modelling**
- Lecture 6: Detailed Specification, Refinement & Implementation
- Lecture 7: Testing
- Lecture 8: Program Analysis
- Lecture 9: Verification with Floyd-Hoare Logic
- Lecture 10: Verification Condition Generation
- Lecture 11: Model-Checking with LTL and CTL
- Lecture 12: NuSMV and Spin
- Lecture 13: Conclusions

Your Daily Menu

- High-Level Specification and Modelling
- The Z Notation as an example of a modelling language
  - Basics, Schema Calculusm, Mathematical Library
  - Canonical Example: the Birthday Book
- Running Example *OmniProtect*
  - Modelling the safe robot

High-Level Specification and Modelling



- Here, we want to be able to express **high-level** requirements **abstractly**, **precisely** and **without** regards for the implementation.

Why look at Z?

- Z is a good example of a **modelling language**.
- It allows us to model high-level specifications in a **mathematically precise** fashion, unambigious and exact.
- Z is **easy to grasp**, as opposed to other mechanisms — we quickly get off the ground.
- Alternatives would be UML (in particular, class diagrams plus OCL), but that is on the one hand already geared towards implementation, and on the other hand there is less tool support for OCL. UML support is more geared towards code generation, not so much abstract modelling as appropriate in this phase of the design process.

The Z Notation

- Z is a **notation** based on **typed set theory**.
  - That means everything is described in terms of **set** (sets are types)
  - There is a lot of syntactic convention ("syntactic sugar")
- It is geared towards the specification of **imperative** programs
  - State and state change built-in
- Developed late 80s (Jean-Claude Abrial, Oxford PRG; IBM UK)
- Used industrially (IBM, Altran Praxis ex. Praxis Critical Systems))
- LaTeX-Notation and tool support (Community Z Tools, ProofPower)

Introducing the Birthday Book

- The birthday book is a well-known example introducing the main concepts of the Z language. It can be found e.g. in the Z reference manual (freely available, see course home page).
- It models a **birthday calendar**, where one can keep track of birthdays (of family, acquaintances, business contacts . . .)
- Thus, we have names and dates as types, and operations to
  - add a birthday,
  - find a birthday,
  - and get reminded of birthdays.

Birthday Book: Types

- We start by **declaring** the types for names and date. We do not say what they are:

  $[NAME, DATE]$

- In Z, we define operations in terms of state transitions.
- We start with defining the **state space** of our birthday book system. This is our **abstract** view of the system state.
- The system state should contain **names** and **birthdays**, and they should be related such that we can map names to birthdays.

Birthday Book: The System State

- The system state is specified in form of a **Z schema**. A schema consists of two parts: **variable declarations** and **axioms**.

BirthdayBook

$$known : \mathbb{P}\, NAME$$
$$birthday : NAME \pfun DATE$$

$$known = \mathrm{dom}\, birthday$$

- This says that there is a set *known* of names, and a **partial map** from names to dates. (Z has a library, called the Mathematical Toolkit, of predefined types and operations, such as the power set and partial map used here.)
- The axiom is an **invariant** (meaning it has to be preserved by all operations). It says that the set of known names is the domain of the *birthday* map.

Schema Operations: Pre- and Poststate

- Operations are defined as schemas as well.
- Operations have a prestate (before the operation is applied) and a poststate (after the operation has been applied). The poststate is denoted by dashed variables.
- Here is an operation which just adds my name, *cxl*, to a set of known names:

AddMe

$$known : \mathbb{P}\, NAME$$
$$known' : \mathbb{P}\, NAME$$

$$known' = known \cup \{cxl\}$$

- In order to minimise repetition, schemas can comprise other schemas. We can also dash whole schemas. Further, the **schema operator** ΔS is shorthand for $\Delta S \stackrel{def}{=} S \wedge S'$, or "include S and S'".

Birthday Book: First Operation

- As a first operation, we want to add a birthday.
- This requires a name and a birthday as **input variables**.

AddBirthday

$$\Delta BirthdayBook$$
$$name? : NAME$$
$$date? : DATE$$

$$name? \notin known$$
$$birthday' = birthday \cup \{name? \mapsto date?\}$$

- Input variables are only defined in the prestate. (Similary, output variables, written as *name*!, are only defined in the poststate.)
- The *Birthday* invariant holds both in pre- and poststate. From this, we can show that the following sensible property holds:

$$known' = known \cup \{name?\}$$

Birthday Book: Finding a birthday

- Finding a birthday gives the name as input, and a date as output:

FindBirthday

$$\Xi BirthdayBook$$
$$name? : NAME$$
$$date! : DATE$$

$$name? \in known$$
$$date! = birthday(name?)$$

- This introduces the Ξ operator. It is shorthand for $\Xi S \stackrel{def}{=} (\Delta S \land S = S')$ (or, "for schema S, nothing changes".)
- The *FindBirthday* operation has a precondition (the *name* must be in the set of known names); only if that holds, the postcondition is guaranteed to hold as well.

Birthday Book: Reminders

- A reminder takes a date as input, and returns the names of entries with this birthday.

Remind

$$\Xi BirthdayBook$$
$$today? : DATE$$
$$cards! : \mathbb{P}\, NAME$$

$$cards! = \{n : known \mid birthday(n) = today?\}$$

- A variation of this schema just selects one name. It is an example of a **non-deterministic** operation.

Birthday Book: Reminders

- A reminder takes a date as input, and returns the names of entries with this birthday.

RemindOne

$\Xi BirthdayBook$

$today? : DATE$

$card! : NAME$

$card! \in known$

$birthday\ card! = today?$

- A variation of this schema just selects one name. It is an example of a **non-deterministic** operation.

Birthday Book: Putting it all together

- We need an **initial state**. It does not say explicitly that $birthday'$ is empty, but that is implicit, because its domain is empty.

$$\begin{array}{|l}
\textit{InitBirthdayBook} \\
\hline
\textit{BirthdayBook}' \\
\hline
known' = \{\}
\end{array}$$

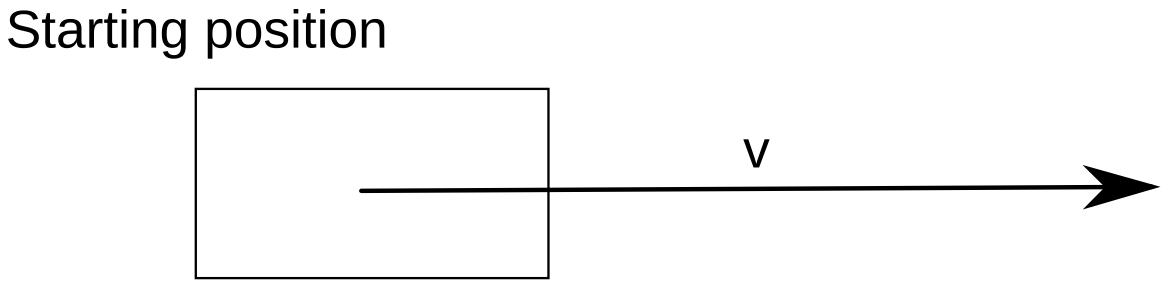
- And we put it all together by conjoining the schemas:

$$System == InitBirthdayBook \land (AddBirthday \land FindBirthday \land Remin$$

Case Study: the *OmniProtect* Project

- The objective of the *OmniProtect* project is to develop a safety module for omnimobile **robots**.
- These robots have a behaviour which is easily describable: they move with a velocity which is given by a vector $\vec{v}$ (per time t).
- The velocity can be changed **instanteneously**, but we assume that braking is **linear**.
- The shape of the robot is described by a **convex polygon**.
- We move the robot by moving the polygon by the given velocity $\vec{v}$.
- We will first describe this movement, and the area covered by this movement (modelling). We will then (next lecture) describe the actual operations (high-level specification), and investigate how to implement them (low-level specification).

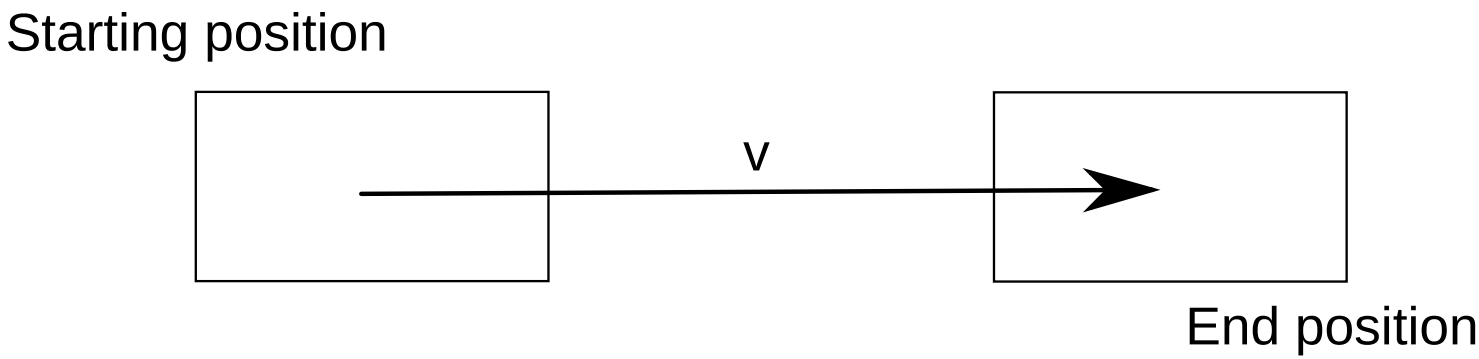
Modelling the Safe Robot: Planar Movement



- Braking time and braking distance:

$$v(t) = v_0 - a_{brk}\,t \qquad s(t) = v_0\,t - \frac{a_{brk}}{2}\,t^2 \qquad T = \frac{v_0}{a_{brk}} \qquad S = \frac{v_0^2}{2a_{brk}}$$

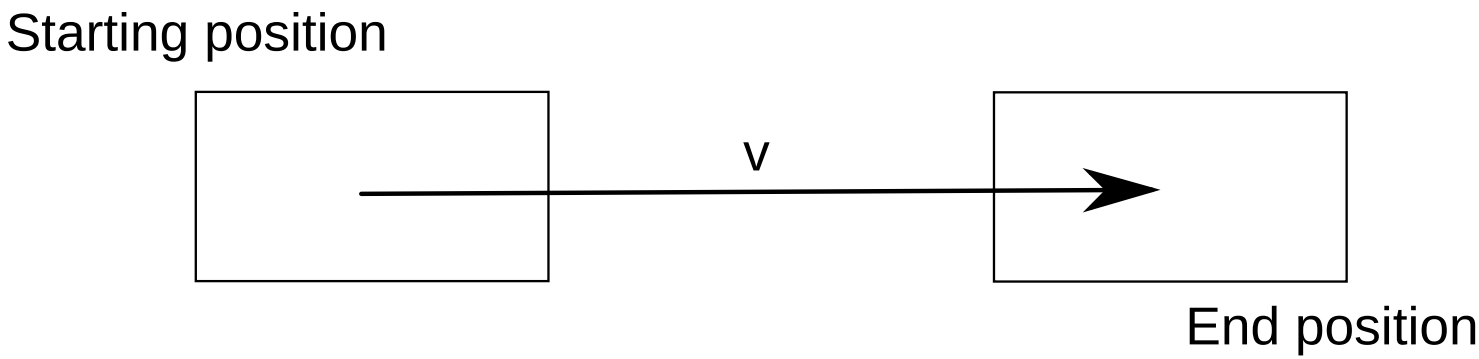
Modelling the Safe Robot: Planar Movement



- Braking time and braking distance:

$$v(t) = v_0 - a_{brk}\,t \qquad s(t) = v_0\,t - \frac{a_{brk}}{2}\,t^2 \qquad T = \frac{v_0}{a_{brk}} \qquad S = \frac{v_0^2}{2a_{brk}}$$

Modelling the Safe Robot: Planar Movement



- Braking time and braking distance:

$$v(t) = v_0 - a_{brk}\,t \qquad s(t) = v_0\,t - \frac{a_{brk}}{2}\,t^2 \qquad T = \frac{v_0}{a_{brk}} \qquad S = \frac{v_0^2}{2a_{brk}}$$

- Modelling in **Z**: Calculating the braking distance

$$brk : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$\forall\, v, a : \mathbb{N} \bullet brk\,(v, a) = (v * v)\ \mathrm{div}\ (2 * a)$$

Mathematical Modelling: Points and Vectors

- Schema for points (vectors):

$$
\begin{array}{|l}
\hline
VEC \\
x : \mathbb{Z} \\
y : \mathbb{Z} \\
\hline
\end{array}
$$

- Type für Polygons und segments:

$$POLY == \{s : \operatorname{seq} VEC \mid \#s > 3 \land head\, s = last\, s\}$$
$$SEG == VEC \times VEC$$

- This introduces the type of **sequents**, seq, or finite lists, from the Mathematical Toolkit. There are a number of useful predefined functions on lists.

Mathematical Modelling: Vector Operations

- Addition and scalar multiplication of vectors

$$
\begin{array}{|l}
add : VEC \times VEC \to VEC \\
smult : R \times VEC \to VEC \\
\hline
\forall\, p, q : VEC \bullet add(p, q) = (p.x + q.x, p.y + q.y) \\
\forall\, n : R;\ p : VEC \bullet smult\,(n, p) = (n * p.x, n * p.y)
\end{array}
$$

- We have slightly cheated here — Z does not really know real numbers.

More abouts Points and Vectors

- A segment defines a **left half-plane** (as a set of points)

$$left : SEG \to \mathbb{P}\, VEC$$

$$\forall\, a, b : VEC \bullet left\,(a, b) = \{p : VEC \mid (b.y - a.y) * (p.x - b.x) - (p.y - b.y) * (b.x - a.x) < 0\}$$

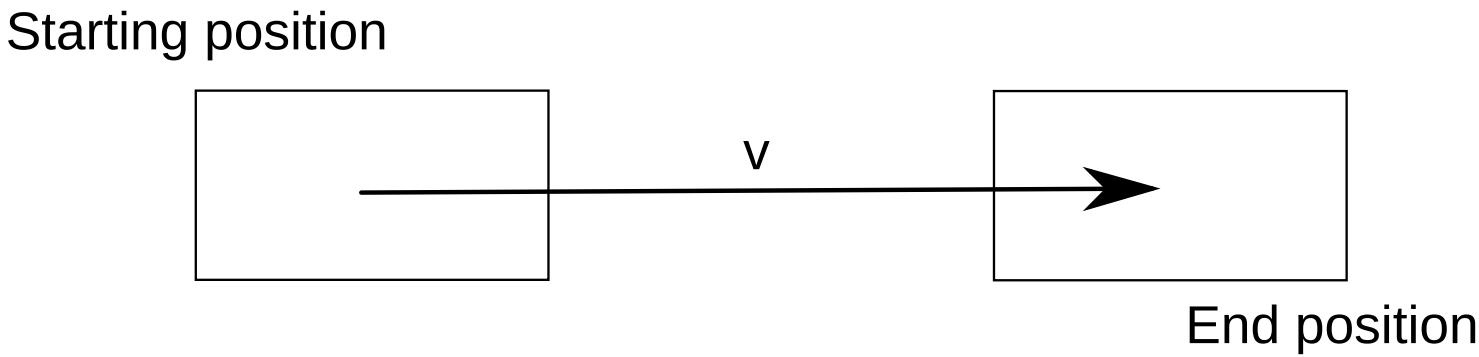
- The **area** of a (convex!) polygon is the intersection of the left half-planes given by its sides.

$$sides : POLY \to \mathbb{P}\, SEG$$
$$area : POLY \to \mathbb{P}\, VEC$$

$$\forall\, p : POLY \bullet sides\, p = \{s : SEG \mid \langle s.1, s.2 \rangle \text{ in } p\}$$
$$\forall\, p : POLY \bullet area\, p = \bigcap \{s : SEG \mid s \in sides\, p \bullet left\, s\}$$

- We should make the restriction on convex explicit (next lecture).

Moving Polygons



- Moving a polygon by a vector:

$$move : POLY \times VEC \to POLY$$

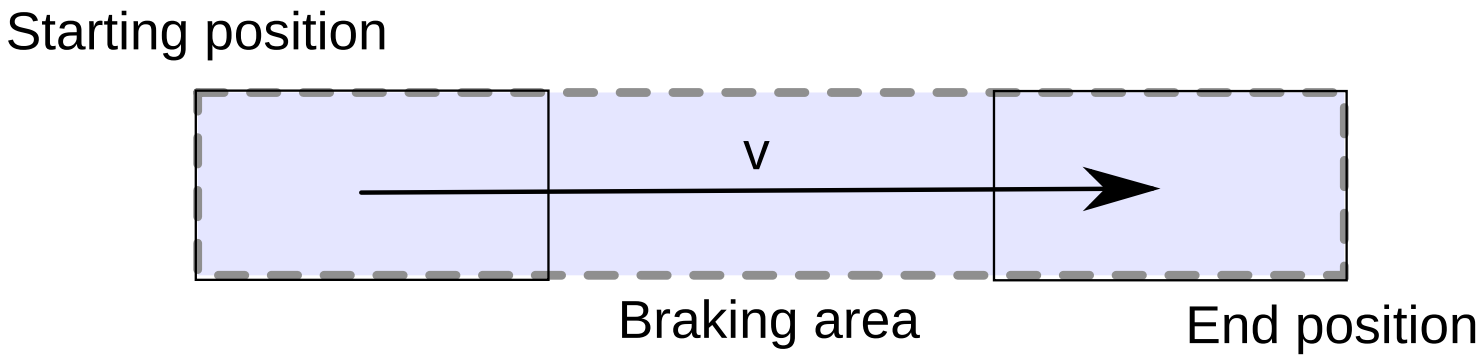
$$\forall\, p : POLY;\ v : VEC \bullet move(p, v) = (\lambda\, x : VEC \bullet add(x, v)) \circ p$$

- Area covered by this movement

$$cov : POLY \times VEC \to \mathbb{P}\, VEC$$

$$\forall\, p : POLY;\ v : VEC \bullet$$
$$cov\,(p, v) = \bigcup \{\tau : R \mid 0 \leq \tau \leq 1 \bullet area(move(p, smult(\tau, v)))\}$$

Moving Polygons



- Moving a polygon by a vector:

$$move : POLY \times VEC \to POLY$$

$$\forall\, p : POLY;\ v : VEC \bullet move(p, v) = (\lambda\, x : VEC \bullet add(x, v)) \circ p$$

- Area covered by this movement

$$cov : POLY \times VEC \to \mathbb{P}\, VEC$$

$$\forall\, p : POLY;\ v : VEC \bullet$$
$$cov\,(p, v) = \bigcup \{\tau : R \mid 0 \leq \tau \leq 1 \bullet area(move(p, smult(\tau, v)))\}$$

Summary

- Z is a modelling language based on **typed set theory**
- Its **elements** are
  - axiomatic definitions
  - schema and the schema calculus
  - the Mathematical Toolkit (standard library)
- In Z, we start with modelling the **system state(s)**, followed by the **operations** (which are state transitions)
- The birthday book example can be found in the Z reference manual.
- We have started with modelling the robot.
- Next lecture: the **safe** robot, and its operations.