Lecture 2: A Relational Logic and Analysis with Alloy
declarative modeling of software
the atlantic divide
the atlantic divide

- SMV, SPIN, Murphi
- Emphasis on verification algorithms
- American school of formal methods
the atlantic divide

American school of formal methods

European school

emphasizes on modelling

eg, Z, VDM, B

emphasizes on verification algorithms

eg, SMV, SPIN, Murphi

eg, SMV, SPIN, Murphi
the atlantic divide

American school of formal methods

> emphasis on verification algorithms (like SMV, SPIN, Murphi)
>
> European school

> emphasis on modelling (like Z, VDM, B)
>
> logical notation (like Z)

> automatic analysis (like SMV)

Alloy brings together
The Atlantic Divide

American school of formal methods

- Emphasis on verification algorithms
  - e.g., SMV, SPIN, Murphi

European school

- Emphasis on modelling
  - e.g., Z, VDM, B

Alloy brings together

- Logical notation (like Z)
- Automatic analysis (like SMV)

Pittsburgh, home of SMV
Oxford, home of Z
Pittsburgh, home of SMV

The Atlantic divide

- Logica, notation (like Z)
- Automatic analysis (like SMV)

Alloy brings together
- Eg, Z, VDM, B
- Emphasis on modelling
- European school

- Eg, SMV, SPIN, Murphi
- Emphasis on verification algorithms
- American school of formal methods
First order effects
First order effects
first order effects

Alloy is first order

- no need to distinguish scalars from singleton sets
- no constructors: composites by projection
- design implications
to allow exhaustive search

First order effects
Alloy is first order to allow exhaustive search

to allow exhaustive search

Alloy is first order

First order effects
atoms
structures are built from atoms & relations
Atoms are indivisible. They can’t be broken into smaller parts. They are immutable. They don’t change over time. They are uninterpreted. They have no built-in properties. Structures are built from atoms & relations.
What's atomic in the real world?

- Very little -- a modelling abstraction
- No built-in properties
- Uninterpreted
- Immutable
- Can't be broken into smaller parts

Atoms are indivisible

Atoms & relations

Structures are built from atoms
atoms & relations

- indivisible: can’t be broken into smaller parts
- immutable: don’t change over time
- uninterpreted: no built-in properties

What’s atomic in the real world?

Very little -- a modelling abstraction

Atoms are built from structures & relations
Atoms and structures are built from atoms.

Atoms are indivisible:
- Can't be broken into smaller parts
- Immutable: Don't change over time
- Uninterpreted: No built-in properties

Very little -- a modelling abstraction.

What's atomic in the real world?

State 1: Contents
State 0: Next
Atoms & relations

Atoms are indivisible
- Can’t be broken into smaller parts
- Immutable
- No built-in properties
- Uninterpreted

What’s atomic in the real world?
- Very little -- a modelling abstraction

Structures are built from atoms & relations
types

universe

\{\text{FILESYSTEM} = \{\text{FILESYSTEM0, FILESYSTEM2}\},\ \text{DATE} = \{\text{JAN1, JAN2, ... DECE1}\},\ \text{PERSON} = \{\text{ALICE, BOB, CAROL}\},\ \text{STATE} = \{\text{STATE0, STATE1, STATE2}\}\} 

\text{partitioned into basic types, each a set } 
\text{a finite (but perhaps big) set } 
\text{contains all atoms types}
types

universe
contains all atoms
a finite (but perhaps big) set
contains all atoms

no subtyping, so

atoms that share properties share a type

Employer = \{Alice\}
Employee = \{Bob, Carol\}

\{Employee in PERSON\}
\{Employer in PERSON\}
\{Date = JAN1, JAN2, ..., DEC31\}
\{State = STATE0, STATE1, STATE2\}
\{FileSystem = FILESYSTEM0, FILESYSTEM2\}

partitioned into basic types, each a set

\{Employer = \{Alice\}\, Employee = \{Bob, Carol\}\, State = \{State0, State1, State2\}\, FileSystem = \{FileSystem0, FileSystem2\}\}
relations

a tuple is a list of atoms

a relation is a set of tuples

\[
\begin{align*}
\text{likes} & = \{(\text{ALICE}, \text{BOB}), (\text{BOB}, \text{CAROL}), (\text{CAROL}, \text{BOB})\} \\
\text{birthday} & = \{(\text{ALICE}, \text{MAY1}), (\text{BOB}, \text{JAN4}), (\text{CAROL}, \text{DEC9})\}
\end{align*}
\]
Relations

A relation is a non-empty list of basic types.

Typing

<table>
<thead>
<tr>
<th>Relation Type</th>
<th>Corresponding Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>likes</td>
<td>(PERSON, PERSON)</td>
</tr>
<tr>
<td>birthday</td>
<td>(PERSON, DATE)</td>
</tr>
</tbody>
</table>

Types:

- If the type is $T_i$, then the $i$-th atom in each tuple is in $T_i$.

Example:

likes = \{(Alice, Bob), (Bob, Carol), (Carol, Bob)\}

birthday = \{(Alice, May 5), (Bob, January 4), (Carol, December 9)\}
relations as tables
relations as tables

- can view relation as table
- atoms as entries, tuples as rows
- order of columns matters, but not order of rows
- can have zero rows, but not zero columns
- no blank entries

Relations as tables
relations as tables

-can view relation as table
- atoms as entries, tuples as rows
- order of columns matters, but not order of rows
- can have zero rows, but not zero columns
- no blank entries

Example

birthday = \{(\text{ALICE}, \\text{MAY1}), (\text{BOB}, \text{JAN4}), (\text{CAROL}, \text{DEC9})\}
dimensions

- finite, \( > 0 \)
- unary, binary, ternary for \( k = 1, 2, 3 \)
- relation of arity \( k \) is a \( k \)-relation
- number of columns
- arity
dimensions

# $p$ is an integer expression giving the size of $p$

\[ \# \]

- finite, $\geq 0$
- number of rows
- size

\[ \]

- finite, $>0$
- unary, binary, ternary for $k=1, 2, 3$
- relation of arity $k$ is a $k$-relation
- number of columns
- arity


dimensions

dimensions

arity

number of columns

relation of arity \( k \) is homogeneous

homogeneity

\#p

is an integer expression giving the size of \( p \)

finite \( \geq 0 \)

number of rows

size

finite \( > 0 \)

unary, binary, ternary for \( k = 1, 2, 3 \)

relation of arity \( k \) is a \( k \)-relation

number of columns

arity

else heterogeneous

relation of type \( (T, T, \ldots, T) \) is homogeneous
relations as graphs
can view 2-relation as graph

relations as graphs

< tuples as arcs
< atoms as nodes
relations as graphs

can view 2-relation as graph

likes = \{ (Alice, Bob), (Bob, Carol), (Carol, Bob) \}

example

\langle \text{tuples as arcs} \rangle
\langle \text{atoms as nodes} \rangle

can view 2-relation as graph

Relations as graphs
can view 2-relation as graphs

Example

- tuples as arcs
- atoms as nodes

\[ \text{likes} = \{(\text{Alice}, \text{Bob}), (\text{Bob}, \text{Carol}), (\text{Carol}, \text{Bob})\} \]
sets and scalars
sets and scalars

sets and scalars

represented as relations

set: a unary relation

-- note ()'s!

PERSON = \{(ALICE), (BOB), (CAROL)\}

Employee = \{(BOB), (CAROL)\}

Employee = \{(ALICE)\}

Alice = \{(ALICE)\}

sets and scalars

scalar: a unary, singleton relation

sets and scalars
sets and scalars

represented as relations

- set: a unary relation
- scalar: a unary, singleton relation

\{a\}, \{(a)\}, \{a\} \neq \{(a)\}

no distinction between

unlike standard set theory

\{a\} \neq \{(a)\}

\{a\} \neq \{(a)\}

Alice = \{(ALICE)\}

Employer = \{(ALICE)\}

Employee = \{(BOB), (CAROL)\}

Person = \{(ALICE), (BOB), (CAROL)\}

-- note ()'s!

sets and scalars

sets and scalars
ternary relations
ternary relations for relationships involving 3 atoms

\[
\text{salary} = \{(\text{ALICE, APPLE, $60K}), (\text{BOB, BIOGEN, $70K})\}
\]

salary: [PERSON, COMPANY, SALARY]
ternary relations for relationships involving 3 atoms

salary: \{ \text{PERSON, COMPANY, SALARY} \}
\{ (ALICE, APPLE,$60k), (BOB, BIOGEN,$70k) \}

birthdayRecords: \{ \text{BIRTHDAYBOOK, PERSON, DATE} \}
\{ (BB0, ALICE, MAY1), (BB0, BOB, JAN4), (BB1, CAROL, DEC9) \}

for associating binary relations with atoms

salary: \{ \text{PERSON, COMPANY, SALARY} \}
\{ (ALICE, APPLE,$60k), (BOB, BIOGEN,$70k) \}

for relationships involving 3 atoms

ternary relations
left and right sets of $p$ is set of atoms in left-(right-)most column
left and right sets

\[ \text{left (right) set of } p \text{ is set of atoms in left-(right-)most column} \]

left and right types

\[ \text{left (right) type of } p \text{ is the first (last) basic type of } p \text{'s type} \]
left-type(likes) = right-type(likes) = PERSON
right-set(likes) = \{ (BOB, CAROL) \}
left-set(likes) = \{ (ALICE, BOB, CAROL) \}
likes = \{ (ALICE, BOB), (BOB, CAROL), (CAROL, BOB) \}

eamples

\begin{itemize}
  \item Left (right) type of p is the first (last) basic type of p's type
  \item Left and right types
  \item Left (right) set of p is set of atoms in left-(right-)most column
  \item Left and right sets
\end{itemize}
set operators
standard set operators

set operators
set operators

Standard set operators

\( p + q \)

contains tuples of \( p \) and tuples of \( q \)
set operators

standard set operators

intersection $p \cap q$
union $p + q$

contains all tuples in both $p$ and $q$
contains tuples of $p$ and tuples of $q$
set operators

- standard set operators
  - union $p + q$
  - intersection $p \cap q$
  - difference $p - q$

contains tuples in $p$ but not in $q$
contains all tuples in both $p$ and $q$
contains tuples of $p$ and tuples of $q$
set operators

standard set operators

interpretation of +
contains tuples in p but not in q
interpretation of -
contains all tuples in both p and q
interpretation of \cap
contains tuples of p and tuples of q
interpretation of \cup

set operators

standard set operators

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>For scalars, makes a set</td>
</tr>
<tr>
<td>-</td>
<td>Difference</td>
</tr>
<tr>
<td>&amp;</td>
<td>Intersection</td>
</tr>
<tr>
<td></td>
<td>Union</td>
</tr>
</tbody>
</table>

Alice + Bob

contains tuples in p but not in q

contains all tuples in both p and q

contains tuples of p and tuples of q
set operators

standard set operators

union $p + q$ contains tuples of $p$ and tuples of $q$
intersection $p \& q$ contains all tuples in both $p$ and $q$
difference $p - q$ contains tuples in $p$ but not in $q$

For scalars, makes a set

For sets, makes a new set

Employer + Employee

Alice + Bob

contains all tuples in both $p$ and $q$
contains tuples in $p$ but not in $q$
difference $p - q$ contains tuples in $p$ but not in $q$
intersection $p \& q$ contains all tuples in both $p$ and $q$
union $p + q$ contains tuples of $p$ and tuples of $q$
set operators

standard set operators

union $p + q$ contains tuples of $p$ and tuples of $q$
intersection $p \& q$ contains all tuples in both $p$ and $q$$p - q$ difference contains tuples in $p$ but not in $q$

interpretation of +

for scalars, makes a set
for relations, combines maps
for sets, makes a new set

likes + Alice $\rightarrow$ Bob
Employer + Employee
Alice + Bob
set operators

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p + q</code></td>
<td>contains tuples of <code>p</code> and <code>q</code></td>
</tr>
<tr>
<td><code>p &amp; q</code></td>
<td>contains tuples in both <code>p</code> and <code>q</code></td>
</tr>
<tr>
<td><code>p - q</code></td>
<td>contains tuples in <code>p</code> but not in <code>q</code></td>
</tr>
</tbody>
</table>

- **Interpretation of `+`**
  - For scalars, makes a scalar set.
  - For sets, makes a new set.
  - For relations, combines maps.

- **Subset and Equality**
  - `$p \subseteq q$` for relations, combines maps.
  - `$p = q$` for sets, makes a new set.
  - `$p + q$` makes a set that contains all tuples in both `p` and `q`.

- **Standard Set Operators**
  - `likes + Alice -> Bob`:
    - Employer + Employee
    - Alice + Bob

- **Example**
  - `$likes + Alice -> Bob$`
  - `$Employer + Employee$`
  - `$Alice + Bob$`
Standard set operators

- Union: \( p + q \) contains tuples of \( p \) and tuples of \( q \)
- Intersection: \( p \times q \) contains all tuples in both \( p \) and \( q \)
- Difference: \( p - q \) contains tuples in \( p \) but not in \( q \)
- Subset and equality: \( p \subseteq q \) contains every tuple of \( p \)

Interpretation of +
- For scalars, makes a set
- For sets, makes a new set
- For relations, combines maps

Alice + Alice = Bob
Likes + Alice -> Bob
Employer + Employee
Alice + Bob
Standard set operators

\[ p \cup q \text{ contains tuples of } p \text{ and tuples of } q \]

\[ p \cap q \text{ contains all tuples in both } p \text{ and } q \]

\[ p - q \text{ contains tuples in } p \text{ but not in } q \]

\[ p \subseteq q \text{ if every tuple } p \text{ contains } q \]

\[ p = q \text{ if } p \text{ and } q \text{ contain the same set of tuples} \]

**Set Operators**

- **Union** 
  
  - For scalars, makes a set
  
  - For sets, makes a new set
  
  - For relations, combines maps

- **Intersection**
  
  - For sets, makes a new set
  
  - For scalars, makes a set

- **Difference**
  
  - For sets, makes a new set

**Set Interpretation of +**

- **For scalars**, makes a set

**Subset and Equality**

- **Subset**
  
  - If every tuple in \( p \) contains every tuple in \( q \)

- **Equality**
  
  - If \( p \) and \( q \) contain the same set of tuples
If \( p \) contains \( (p_1, \ldots, p_n) \) and \( q \) contains \( (q_1, \ldots, q_m) \) then \( b \prec d \) contains \( (p_1, \ldots, p_n, q_1, \ldots, q_m) \).
For scalars $a$ and $b$, $q \leftarrow e$ is tuple.
For sets $s$ and $t$, $s \rightarrow t$ is Cartesian product.

If $p$ contains $(p_1, \ldots, p_n)$ and $q$ contains $(q_1, \ldots, q_m)$, then $b \leftarrow d$ contains $(d_1, \ldots, d_{p+n-q+m})$.

For $s$ and $t$, $s \rightarrow t$ is Cartesian product.
definition

if \( p \) contains \((p_1, \ldots, p_n)\) and \( q \) contains \((q_1, \ldots, q_m)\)
then \( p \rightarrow q \) contains \((p_1, \ldots, p_n, q_1, \ldots, q_m)\)

puns

for sets \( s \) and \( t \), \( s \rightarrow t \) is cartesian product
for scalars \( a \) and \( b \), \( a \rightarrow b \) is tuple

examples

birthday = Alice \( \rightarrow \) May1 + Bob \( \rightarrow \) Jan4 + Carol \( \rightarrow \) Dec9

Employee \( \rightarrow \) Employee in likes
if \( p \) contains \((p_1, \ldots, p_{n-1}, p_n)\) and \( q \) contains \((q_1, \ldots, q_m)\) and \( p_n = q_1 \) then \( b \cdot d \) contains \((p_1, \ldots, p_{n-1}, q_2, \ldots, q_m)\).

and \( p_n = q_1 \)

and \( q \) contains \((q_1, \ldots, q_m)\)

If \( p \) contains \((p_1, \ldots, p_{n-1}, p_n)\)

definition

\textbf{Join}
$\text{right-type}(p) = \text{left-type}(q)$

$\text{arity}(p) + \text{arity}(q) < 2$

then $b \cdot d$ contains $(p_1, \ldots, p_n, q_2, \ldots, q_m)$

and $p_n = q_1$

and $q$ contains $(q_1, \ldots, q_m)$

if $p$ contains $(p_1, \ldots, p_n)$

definition

join
Join, examples

Given

\{(\text{Alice,May 1}), (\text{Bob,Jan 4}), (\text{Carol,Dec 9})\}

\{(\text{Bob, Alice}), (\text{Bob, Bob}), (\text{Carol, Bob})\}

likes = \{(\text{Alice, Bob}), (\text{Bob, Carol}), (\text{Carol, Bob})\}

birthday = \{(\text{Alice, May 1}), (\text{Bob, Jan 4}), (\text{Carol, Dec 9})\}

birthdayRecords = \{(\text{Bob, Alice, May 1}), (\text{Bob, Bob, Jan 4}), (\text{Carol, Bob, Dec 9})\}
Given examples, we have

\[
\begin{align*}
\text{Alice}.\text{birthdayRecords} &= \{(\text{Alice}, \text{MAY1}), (\text{Bob}, \text{JAN4})\} \\
\text{Bob}.\text{birthdayRecords} &= \{(\text{Alice}, \text{MAY1}), (\text{Bob}, \text{JAN4})\} \\
\text{likes.birthday} &= \{(\text{Alice}, \text{JAN4}), (\text{Bob}, \text{DEC9}), (\text{Carol}, \text{JAN4})\} \\
\text{Alice}.\text{likes} &= \{(\text{Bob})\} \\
\text{Bob}.\text{likes} &= \{\} \\
\text{likes.Alice} &= \{\} \\
\text{likes.birthday} &= \{(\text{Alice}, \text{JAN4}), (\text{Bob}, \text{DEC9}), (\text{Carol}, \text{JAN4})\} \\
\text{bb0}.\text{birthdayRecords} &= \{(\text{Alice}, \text{MAY1}), (\text{Bob}, \text{JAN4})\} \\
\text{Alice}.(\text{bb0}.\text{birthdayRecords}) &= \{(\text{MAY1})\}
\end{align*}
\]
join, puns
for binary relation \( r \) of type \((S,T)\),

\( r.S \) is right-set of \( r \)

\( r.T \) is left-set of \( r \)

\[ r.s = \text{standard join of } r \text{ and } s \]

\[ r.p = \text{image of } s \text{ under } r \]
join variants
Join variants for non-binary relations; join is not associative.
Join variants

For non-binary relations, join is not associative.
Join variants

for non-binary relations, join is not associative

3 syntactic variants of join

binding power: :: most, then ., then []

equivalent expressions

\[ p.q = p::q = q[p] \]

\[ p.q::r = p.(q.r) \]

\[ p.q[r] = r.(p.q) \]

\[ (b\cdot d)\cdot r = [r]b\cdot d \]

\[ (r\cdot b)\cdot d = r::b\cdot d \]

\[ [d]b = b::d = b\cdot d \]

Alice.(bb0.birthdayRecords)

Alice.bb0::birthdayRecords

Alice.(bb0.birthdayRecords)
transpose
$\sim r$ has type $(T,S)$

$\sim r$ contains $(b,a)$ whenever $r$ contains $(a,b)$

For relation $r: (S,T)$

transpose
\( r \sim s = s \sim r \)

for set \( s \) and binary relation \( r \),

is a theorem

\( r \sim \) has type \( (T, S) \)

\( r \sim \) contains \( (b, a) \) whenever \( r \) contains \( (a, b) \)

for relation \( r : (S, T) \)

transpose
For relations \( p, q \) of \( (\Sigma, \Gamma) \):

\[ p \text{ contains } (a, b) \text{ and } q \text{ does not map } a \]

\[ q \text{ contains } (a, b) \text{ or } \]

\[ q \text{ contains } (a, b) \text{ whenever } \]

\[ b+b+d \]

overrides
Given

p contains (a, b) and q does not map a
q contains (a, b)’ or

b + d

for relations p, b: (S, T)

or

p contains (a, b)’ whenever

b + d

for relations p, b: (S, T)
We have

\[
\text{birthday} = \{(\text{alice, mar3}), (\text{bob, jan4}), (\text{carol, dec9})\}
\]

\[
\text{Alice} \{\} \cdot \text{March3} = \{(\text{mar3})\}
\]

Given

p contains \((a, q)\) and \(q\) does not map a

q contains \((a, q)\), or

for relations \((a, b)\) whenever \(b + d\)

override
is smallest reflexive & transitive relation \( p \) containing \( r \)

\[
\cdots + 1 \cdot 1 + 1 \cdot 1 \cdot 1 + 1 \cdot 1 \cdot 1 \cdot 1 + \cdots = 1^*_r
\]

is smallest transitive relation \( p \) containing \( r \)

\[
\cdots + 1 \cdot 1 + 1 \cdot 1 \cdot 1 + 1 \cdot 1 \cdot 1 \cdot 1 + \cdots = 1^*_r
\]

For relation \( r \): \((1, 1)\)

closure
Closure for relation \( r: (T,T) \):

\[ \text{next} \sim \doteq \text{precedes} \]

Reachs = connected

\( \text{ancestor} \equiv \text{parent} \)

Examples

\( \text{ancestor} = \text{parentreaches} \)

\( \text{precedes} = \text{connects} \)

\( \text{parent} = \text{ancestor} \)

\( \text{idem}[\tau] = [\tau] \)

Is smallest reflexive and transitive relation \( p \) containing \( r \)

\[ \ldots + 1 \times 1 \times 1 + 1 \times 1 \times 1 \times 1 + \ldots = \tau \)

Is smallest transitive relation \( p \) containing \( r \)

\[ \ldots + 1 \times 1 \times 1 + 1 \times 1 \times 1 \times 1 + \ldots = \tau \]

For relation \( \tau \): \((\tau,\tau)\)

closure
<table>
<thead>
<tr>
<th>Operator Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p+q, p-q, p&amp;q: (T1,…,Tn), p in q</td>
<td>tuple of types</td>
</tr>
<tr>
<td>p,q: (T1,…,Tn)</td>
<td>tuple of types</td>
</tr>
<tr>
<td>none[p], univ[p]: (T1,…,Tn)</td>
<td>tuple of types</td>
</tr>
<tr>
<td>iden[p]: (T,T)</td>
<td>tuple of types</td>
</tr>
<tr>
<td>*p, ^p: (T,T)</td>
<td>tuple of types</td>
</tr>
<tr>
<td>~p: (T,S)</td>
<td>tuple of types</td>
</tr>
<tr>
<td>p.q: (S1,…,Sn-1,T2,…,Tm)</td>
<td>tuple of types</td>
</tr>
</tbody>
</table>

**Example**

If $p$ is true, then $q$ is true.
navigation expressions
navigation expressions from 2-relations and the operators

\sim \times v + \cdot

navigation expressions
follow p backwards \( d \sim \)
follow p zero or more times \( d _ * \)
follow p once or more \( d _ v \)
follow p or q \( b + d \)
follow p then q \( b \cdot d \)
interpret as path-sets

\[ \sim \ 
\star \ 
\ast \ 
\cdot \ 
\] from 2-relations and the operators

navigation expressions
Navigation expressions from 2-relations and the operators + * ~
interpret as path-sets

cousin = parent.sibling.~parent

Example

follow p backwards  d~
follow p zero or more times  d*
follow p once or more  dv
follow p or q  b+d
follow p then q  b·d

from 2-relations and the operators

navigation expressions
Navigation expressions from 2-relations and the operators +, ^, *, ~

- p.q: follow p then q
- p+q: follow p or q
- ^p: follow p once or more
- *p: follow p zero or more times
- ~p: follow p backwards

Example:

cousin = parent.sibling.~parent

daniel, spouse, sibling =

Tim

Daniel

Spouse, sibling, sibling

Claudia

Interpret as path-sets

\[ d^\sim \]
\[ d^*_v \]
\[ b+d \]
\[ b \cdot d \]

From 2-relations and the operators +, *, ^, ~

Navigation expressions
a navigation example
a navigation example
a navigation example

all messages queued on links emanating from a node have a 'from' field of that node to say

<table>
<thead>
<tr>
<th>source</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>node</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>link</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>link</th>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>to</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>to</th>
<th>from</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>node</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node</th>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>msg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>msg</th>
<th>node</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td>to</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>to</th>
<th>from</th>
</tr>
</thead>
<tbody>
<tr>
<td>node</td>
<td>node</td>
</tr>
</tbody>
</table>
a navigation example

all \( n \colon \text{Node} \mid \text{all messages queued on links emanating from a node have a } \text{from field of that node} \)

to say

we can write

all \( n \colon \text{Node} \mid \text{all messages queued on links} \)

\( \equiv \text{all messages queued on links} \)

\( \text{to say} \)
all \text{n} \in \text{Node} \mid \text{n} \cdot \text{~source.queue.els.from} = \text{n}

or equivalently

\text{\text{source.queue.els.from} in iden[Node]}

We can write

\text{all messages queued on links emanating from a node have a}

\text{from field of that node}

\text{to say}

a navigation example
logical operators
Logical operators

standard connectives

F \iff G
F \implies G \text{ else } H
F \lor G
\{ F \land G \}
\text{ not } F
F \rightarrow G \text{ } H
F \Rightarrow G
F \Rightarrow G \iff G
Logical operators

**if-then-else expressions**

\[ \text{if } F \text{ then } e \text{ else } e' \]

\[ F \text{ implies } G \text{ else } H \]

\[ F \text{ or } G \]

\[ \{ \text{if } F \text{ then } e \text{ else } e' \} \]

\[ F \text{ and } G \]

\[ \text{not } F \]

Standard connectives
negated operators

if \( P \) then \( e \) else \( e' \)

if-then-else expressions

\( e \) in \( e' \)

\( e \neq e' \)

logical operators

Standard connectives

\( F \) implies \( G \) else \( H \)

\( F \) or \( G \)

\( \{ F \land G \} \)

\( F \) not \( G \)

Logical operators
set declarations
set declarations

form

set declarations
set declarations

form

set declarations

meaning

\[
\forall v \in e \land \# v \leq 1 \land \forall v : \text{set} \in e \\
\forall v : \text{option} \in e \\
\forall v \in e \land \# v = 1
\]
set declarations

form

var : \texttt{option \mid set\texttt{expr}}

set declarations

meaning

\texttt{var : \texttt{Person \rightarrow Date}}

\texttt{Employee : set Person}

\texttt{Employee is a subset of Person}

\texttt{p : Person}

\texttt{p is a scalar in Person}

\texttt{bb : Person \rightarrow Date}

\texttt{not unary, so no scalar constraint}

\texttt{\texttt{\exists v \in e \ and \ \#v \leq 1}}

\texttt{\texttt{\exists v \in e \ and \ \#v = 1}}

\texttt{\texttt{\exists v \in e \ and \ \#v > 1}}
set declarations
form

var : Person

meaning

same meaning as

set declarations
form

but first order

in a other languages'

Employee : Person

p is a scalar in Person

Employee is a subset of Person

bb : Person -> Date

not unary, so no scalar constraint

"examples"
Relation declarations
relation declarations

form

\[ \text{var : expr \, [mul] \rightarrow \, [mul] \, expr} \]
relation declarations

\[ \text{form} \]

\[ \text{var} : \text{expr} \rightarrow [\text{mult}] \text{expr} \]

multiplicity symbols

- one or more \( + \)
- exactly one \( ! \)
- zero or one \( ? \)

Relation declarations
relation declarations
form
var : expr [mult] \rightarrow [mult] expr

multiplicity symbols
? zero or one
\| exactly one
+ one or more

meaning

m e0's for each e1
and n e1's for each e0,

r \in e0 \rightarrow e1

meaning

r e0 m \rightarrow n e1

Relation declarations
relation declarations
form
var : expr [mult] -> [mult] expr

multiplicity symbols
? zero or one
+ one or more
* exactly one
_ zero or one

meaning

examples

r is a bijection
r: A ->! B

r is an injective
r: A ->? B

r is a total function
r: A =! B

r is a partial function
r: A =? B

r: e0 -> e1 and n e1's for each e0, m e0's for each e1

r: in e0 -> e1

r in e0 -> e1

r: e0 m -> e1

Relational declaratives
object models
What is an object model?

- Parentless box has implicit type
- Boxes denote sets, arcs relations
- Set of declarations drawn as graph

Object models
What is an object model?

- Set of declarations drawn as graph
- Boxes denote sets, arcs relations
- Parentless box has implicit type

Object models
comprehensions
comprehensions

{ var : setexpr | ... formula }

General form
comprehensions

general form

\{ \text{var} : \text{setexpr} , \ldots \} \text{ | } \text{formula}

\text{meaning}

\{ v_0 : e_0, v_1 : e_1, \ldots \} \text{ | } F

\text{and} \{(a_0) \in e_0, \{a_1\} \in e_1, \ldots \}

\text{such that } F \text{ holds when } v_0 = \{(a_0)\}, v_1 = \{(a_1)\}, \ldots \}

\{(a_0, a_1, \ldots) \in F \}

\text{where containing tuples (a_0, a_1, \ldots)}
comprehensions

general form
\{ \text{var} : \text{setexpr}, \ldots \mid \text{formula} \}

meaning
\{ \text{v0: e0, v1: e1, \ldots \mid F} \}

example
siblings = \{ a, b : \text{Person} \mid a.\text{parents} = b.\text{parents} \land a \neq b \}

and \{(a0, \{\}) \in e0, \{\} \in e1, \ldots \}

is the relation containing tuples \((a0, a1, \ldots)\) such that \(F\) holds when \(v0 = (a0), v1 = (a1), \ldots\)
quantification
universal quantification

\[ \forall \text{ var} : \text{ setexpr} \quad \ldots \quad \text{ formula} \]
quantification

all \var \in \setexpr, \ldots \text{ | formula}
Example

\[
\forall \text{ Person } a, \forall \text{ in parents } a
\]

meaning

universal quantification
other quantifiers
F holds for exactly one \( x \in e \)
F holds for at most one \( x \in e \)
F holds for no \( x \in e \)
F holds for some \( x \in e \)
F holds for all \( x \in e \)
\( \text{one} \ x : e \mid F \)
\( \text{sole} \ x : e \mid F \)
\( \text{no} \ x : e \mid F \)
\( \text{some} \ x : e \mid F \)
\( \text{all} \ x : e \mid F \)
F | … | one \[\forall_0 \cdot e_0 \land \forall_1 \cdot e_1 \land \ldots \land \forall_n \cdot e_n \land \top \land \neg F \land \neg \top\]

one \[\forall_0 \cdot e_0 \land \forall_1 \cdot e_1 \land \ldots \land \forall_n \cdot e_n \land \top \land \neg F \land \neg \top\]
is not equivalent to

all \[\forall_0 \cdot e_0 \land \forall_1 \cdot e_1 \land \ldots \land \forall_n \cdot e_n \land \top \land \neg F \land \neg \top\]
is equivalent to

Note

F holds for exactly one \(x\) in \(e\)
F holds for at most one \(x\) in \(e\)
F holds for no \(x\) in \(e\)
F holds for some \(x\) in \(e\)
F holds for all \(x\) in \(e\)

Other quantifiers
quantified expressions
quantified expressions

for quantifier $\alpha$ and expression $e$, make formula $\alpha e$
For quantifier $Q$ and expression $e$, make formula

$Q.e$

meaning:

- $\#e = 1$ e has one tuple
- $\#e \geq 1$ e has at most one tuple
- $\#e = 0$ e is empty
- $\#e \leq 1$ e is non-empty
- $\#e > 0$ e is non-empty

Quantified expressions
For quantifier $Q$ and expression $e$, make formula $Q(e)$ meaning:

- some $e$ has at most one tuple $e \leq 1$
- one $e$ has one tuple $e = 1$
- no $e$ has non-empty $e > 0$
- sole $e$ has empty $e = 0$

Example:

- no Man & Woman
- no person is both a man and a woman

quantified expressions
sample quantifications
biological constraints

sample quantifications

no p: Person | p in p.parents
all p: Person | one p.mother
sample quantifications

biological constraints

no p: Person | some p.spouse & p.siblings

all p: Person | sole p.spouse

no p: Person | in p.parents

all p: Person | one p.mother

-cultural constraints

biological constraints

sample quantifications
sample quantifications

biological constraints

one eve: Person | Person in eve. ~mother

biblical constraints

no p: Person | p: spouse & p: siblings

all p: Person | sole p: spouse

-cultural constraints

no p: Person | p: in p: parents

all p: Person | one p: mother

-biological constraints


summary: doing more with less
everything's a relation

summary: doing more with less
summary: doing more with less

everything's a relation

\[ a \rightarrow b \] for \((a, b)\) and \(a \in A, b \in B\)

First-order operators

\[ r : A \rightarrow B \] means \(r \in \mathcal{P}(A \times B)\)

everything's a relation

summary: doing more with less
summary: doing more with less

dot operator

plays many roles

First-order operators

everything's a relation

\[ \forall x \in A \implies \text{for } (a, b) \in r \text{ and } a \in A \implies b \in B \]

\[ r : A \rightarrow B \]

\[ \forall x \in A \implies A \backslash B \text{ replaces } \mathcal{P}(A) \backslash \mathcal{P}(B) \]

\[ r \rightarrow \mathcal{P}(A) \backslash \mathcal{P}(B) \text{ means } i : A \rightarrow B \]

\[ \forall x \in A \implies A \backslash B \text{ replaces } \mathcal{P}(A) \backslash \mathcal{P}(B) \]
Summary: doing more with less

Doing more with less

Everything is a relation

\( a \rightarrow b \) in \( r \)

\( a \in r \) and \( b \in r \)

First-order operators

\[ r : A \rightarrow B \]

means \( r \) replaces \( r \)

\( r \) replaces \( r \)

\( A \neq B \)

\( \text{dot operator} \)

Plays many roles

Intractable

Tractable
summary: doing more with less

Everything's a relation

For $(a, b)$ and $a \text{ rel } b$ in $R$, 

emph{dot operator} plays many roles

first-order operators

$\text{expressive} \rightarrow \text{tractable}$

expressive inexpressive

tractable intractable

intergenic
summary: doing more with less

everything's a relation

\[ a \rightarrow b \text{ in } r \]

for \((a, b) \in r \text{ and } a \neq b\)

First-order operators

\[ r : A \rightarrow B \text{ means } i \in A \text{ and } i \in P(A, B) \]

\[ r \text{ dot operator} \]

\[ a \rightarrow b \] plays many roles

dot operator

inexpressive

tractable

inexpressive

expressive

expressive

inexpressive

a challenge
a challenge

write a constraint that says it is acyclic on an undirected graph

a challenge
a challenge

that says it is acyclic

on an undirected graph

write a constraint
Write a constraint on an undirected graph that says it is acyclic.
a solution
a solution
sample graph
higher-order quantifiers
higher-order quantifiers

General form

quantifier decl \ldots formula
higher-order quantifiers

general form

quantifier decl: \( \ldots \mid \text{formula} \)

modified set expressions

\( \text{all} \ : \ \text{option} \ S \mid F \)

\( \text{all} \ : \ \text{set} \ S \mid F \)

\( F \) holds for all \( o = S' \) where \( S' \) in \( S \)

\( F \) holds for all \( s = S' \) where \( S' \) in \( S \)
higher-order quantifiers

general form

quantifier decl, ... formula

modified set expressions

all \ s :: set S in S

relation set expressions

all \ o :: option S in S

all \ x :: R in R

F holds for all x = R' where R' in R

F holds for all o = S' where S' in S

F holds for all s = S' where S' in S

higher-order quantifiers

Examples

\[ \forall p : S \left( \text{\texttt{set}} \ S \right) \ L \ L p \] \[ \forall r : \text{\texttt{set}} \ R \left( \text{\texttt{relational\ expressions}} \right) \ F \] \[ \forall r : \text{\texttt{option}} \ S \left( \text{\texttt{set\ expressions}} \right) \ F \] \[ \forall s : \text{\texttt{set\ expressions}} \left( \text{\texttt{modified\ set\ expressions}} \right) \ F \] \[ \forall \text{\texttt{quantifier\ decl, \···\ formula}} \] 

General form
model checking

no transitive closure, etc
must encode in records, arrays
only low-level datatypes

39
model checking

- only low-level datatypes
- must encode in records, arrays
- built-in communications
- not suited for abstract schemes
- fixed topology of processes
- no transitive closure, etc
model checking

- missing at operation level
- modularity
- fixed topology of processes
- not suited for abstract schemes
- built-in communications
- no transitive closure, etc
- must encode in records, arrays
- only low-level datatypes

model checking
model checking

- keep counters, discard model or vice versa?
- but in software, essence is incremental modelling
- emphasizes finding showstopper flaws
- culture of model checking

- missing at operation level
- modularity
  - fixed topology of processes
  - not suited for abstract schemes
  - built-in communications
  - no transitive closure, etc
  - must encode in records, arrays
  - only low-level datatypes

model checking