

Reasoning about Business Processes

Banking: rules for meeting anti-fraud legislation

Example: rules for granting a request to open a bank account

Rule 1: A risk assessment must eventually be carried out for each request to open a bank account

Rule 2: A request to open a bank account is granted only if the risk is assessed as low

Rule 3: A due diligence assessment must eventually be carried out for each request to open a bank account

Rule 4: If a person fails due diligence then he or she must be blacklisted

English: prose used so that bank staff can understand rules

Question: how to check these rules for consistency and sanity?



Reasoning About Compilers

Original Code

```
1: y := 1
2: if z = x*x*x
3:   then y := x*x + y
4: endif
```

Compiled Code

```
1: y := 1
2: R1 := x*x
3: R2 := R1*x
4: jmpNE(z,R2,6)
5: y := R1+1
```

Problem:

prove equivalence of source and target program



Reasoning about Web Applications

Suppose we want to monitor web-based applications.

How to specify things like:

Rule 1: allow a program to connect to a remote site if and only if it has neither tried to open a local file that it has not created, nor tried to modify a file it has created, nor tried to create a sub-process:

Rule 2: allow a program to open local files in user-specified directories for modifications if and only if it has created them, and it has neither tried to connect to a remote site nor tried to create a sub-process.



Classical Propositional Logic (CPL)

Syntax: captures "not", "and", "or", "if then", "iff" via

$atom ::= p \mid q \mid r \mid \dots$

$\varphi ::= atom \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi$

Model: for every atom, an assignment to **t** or else to **f**

Example: $(p \wedge (p \rightarrow q)) \rightarrow q$

A1: $p := \mathbf{t}$ and $q := \mathbf{t}$

$(\mathbf{t} \rightarrow \mathbf{t})$ evaluates to **t**

$(\mathbf{t} \wedge \mathbf{t})$ evaluates to **t**

$(\mathbf{t} \rightarrow \mathbf{t})$ evaluates to **t**

$(\mathbf{t} \wedge (\mathbf{t} \rightarrow \mathbf{t})) \rightarrow \mathbf{t}$

$(\mathbf{t} \wedge \mathbf{t}) \rightarrow \mathbf{t}$

$\mathbf{t} \rightarrow \mathbf{t}$

t

So: $(p \wedge (p \rightarrow q)) \rightarrow q$ is **t** in model $p := \mathbf{t}$ and $q := \mathbf{t}$



SATisfiability, Validity and Automated Reasoning

Semantics: two important inter-related notions

Satisfiable: if φ is true under some assignment

Valid: if φ is true under all assignments

Lemma: φ is valid if and only if $\neg\varphi$ is unsatisfiable

Example: $(p \wedge (p \rightarrow q)) \rightarrow q$ is satisfiable ($p := \mathbf{t}$ and $q := \mathbf{t}$)

Is it valid?: check all 2^n assignments for n atoms ($n = 2$)

Question: is there a purely syntactic way?

Automated Reasoning: decide satisfiability or validity efficiently on a computer using purely syntactic methods

Goal: input φ and get answer "satisfiable" or "unsatisfiable"

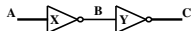
Ideally: output an/all assignment(s) that makes it satisfiable

Ideally: output proof of why it is unsatisfiable

◀ ▶ ↻ 🔍

Reasoning about Digital Circuits

Knowledge Base



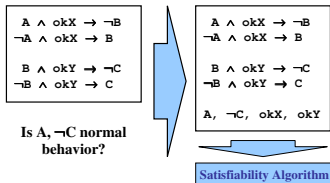
$$\text{KB} = \left\{ \begin{array}{l} A \wedge \text{ok}X \rightarrow \neg B \\ \neg A \wedge \text{ok}X \rightarrow B \\ B \wedge \text{ok}Y \rightarrow \neg C \\ \neg B \wedge \text{ok}Y \rightarrow C \end{array} \right\}$$

Is $A, \neg C$ normal behavior?

◀ ▶ ↻ 🔍

Reasoning about Digital Circuits

Answering Single Query

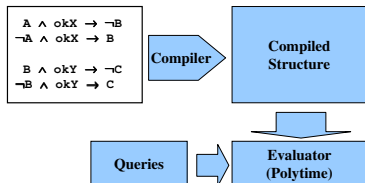


◀ ▶ ↻ 🔍

Reasoning about Digital Circuits

Jinbo Huang works in knowledge compilation

Multiple Queries



◀ ▶ ↻ 🔍

Reasoning About Business Processes

Banking: rules for meeting anti-fraud legislation

Example: rules for granting a request to open a bank account

Rule 1: A risk assessment (*ra*) must eventually be carried out for each request to open a bank account (*ro*)

Rule 2: A request to open a bank account (*ro*) is granted (*rog*) only if the risk is assessed as low (*ral*)

Rule 3: A due diligence assessment (*dd*) must eventually be carried out for each request to open a bank account (*ro*)

Rule 4: If a person fails due diligence (*ddf*) then he or she must be blacklisted (*bl*)

Question: how to check these rules for consistency and sanity?

Need: a logic which captures temporal notions like "eventually", "after", "before" as well as "if then", "only if" etc.



Reasoning About Business Processes Using LTL

Business Process: finite linear discrete sequence

$s_0, s_1, s_2, \dots, s_n$ of states from start state s_0 to end state s_n

Rule 1: A risk assessment (*ra*) must be carried out for each request to open a bank account (*ro*)

$$\mathbf{G}(ro \rightarrow \mathbf{F}ra)$$

Rule 2: A request to open a bank account (*ro*) is granted (*rog*) only if the risk is low (*ral*)

$$\mathbf{G}(rog \rightarrow ral)$$

Rule 3: A due diligence assessment (*dd*) must be carried out for each request to open a bank account (*ro*)

$$\mathbf{G}(ro \rightarrow \mathbf{F}dd)$$

Rule 4: If a person fails due diligence (*ddf*) then he or she must be blacklisted (*bl*)

$$\mathbf{G}((ddf \wedge ddf) \rightarrow \mathbf{F}bl)$$



Linear Temporal Logic (LTL)

Syntax: represents linguistic notions using logical connectives

CPL: captures "and", "or", "not", "if then", "iff"

LTL: adds "next", "until", "before", "always", "eventually"

$$atm ::= p \mid q \mid r \mid \dots$$

$$\varphi ::= atm \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \varphi \leftrightarrow \varphi$$

$$\mid \mathbf{X}\varphi \mid \varphi \mathbf{U}\varphi \mid \varphi \mathbf{B}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi$$

Models: a truth value **t** **exor** **f** to each atomic formula p at each state s_i of a **linear discrete infinite sequence** s_0, s_1, s_2, \dots

Evaluate CPL formula: at state s_i using truth tables

Evaluate "temporal" formula: using relative order of states

Satisfiable: if φ is true at some state in some sequence

Valid: if φ is true in all states of every sequence

Lemma: φ is valid if and only if $\neg\varphi$ is unsatisfiable



Compiler Validation Using First-Order Logic

Peter Baumgartner

Problem: prove equivalence of source and target program

1: $y := 1$	1: $y := 1$
2: $\text{if } z = x*x*x$	2: $R1 := x*x$
3: $\text{then } y := x*x + y$	3: $R2 := R1*x$
4: endif	4: $\text{jmpNE}(z, R2, 6)$
	5: $y := R1+1$

Refer to values at line numbers where index 0 = initial values

Prove:

If $y_1 = 1 \wedge z_0 = x_0 * x_0 * x_0 \wedge y_3 = x_0 * x_0 + y_1$
 $\wedge y'_1 = 1 \wedge R1_2 = x'_0 * x'_0 \wedge R2_3 = R1_2 * x'_0 \wedge z'_0 = R2_3$
 $\wedge y'_5 = R1_2 + 1 \wedge x_0 = x'_0 \wedge y_0 = y'_0 \wedge z_0 = z'_0$
 then $y_3 = y'_5$

Variables: must be able to take arbitrary values, not just **t** or **f**



First Order Logic and Applications

Syntax: extend CPL with quantifiers, relations, functions etc
Forall: $\forall x.p(x)$ every value of x makes $p(\cdot)$ true
Exists: $\exists x.p(x)$ some value of x makes $p(\cdot)$ true
Relations: $R(x, y)$ relation R holds between x and y
Functions: $f(x, y)$ function f has a value for x and y

Models: assign elements of a domain D to variables and assign specific relations and functions over that domain to relation and function symbols

Satisfiability: some model makes the formula true

Validity: all models make the formula true

Lemma: φ satisfiable iff $\neg\varphi$ unsatisfiable

Theorem: validity/satisfiability problem is undecidable!



Compiler Validation Using First-Order Logic

Peter Baumgartner

Problem: prove equivalence of source and target program

1: $y := 1$	1: $y := 1$
2: if $z = x*x*x$	2: $R1 := x*x$
3: then $y := x*x + y$	3: $R2 := R1*x$
4: endif	4: jmpNE ($z, R2, 6$)
	5: $y := R1+1$

Refer to values at line numbers where index 0 = initial values

Prove:

If $eq(y_1, 1) \wedge eq(z_0, mult(mult(x_0, x_0), x_0))$
 $\wedge eq(y_3, plus(mult(x_0, x_0), y_1))$
 ...
then $eq(y_3, y'_5)$



Research: Logic and Computation

Logic: design a new logic L for reasoning about ...

Calculi: find a new or better calculus for logic L and prove that it is correct and complete

Automated Deduction: find new procedures for deciding satisfiability or validity in logic L

Interactive Proof: verify properties of existing calculi/systems

Complexity: analyse computational complexity of new procedure

Extensions: can the calculus be extended with specific theories like equality and arithmetic?

Prototypes: implement a prototype of the new method

Optimisations: find and implement new optimisations or faster data structures and prove them correct

Experiments: compare prototype with existing ones

\$\$\$: commercialise your technology



Applications of Logics in Other Areas

Artificial Intelligence: logics for reasoning about "belief", "knowledge", "desires", "intentions"

Diagnosis: fault diagnosis in electricity networks

Constraints: methods for solving large sets of constraints

Software Verification: verification of the L4 micro-kernel

Ontologies: logics for knowledge representation e.g. Galen

Web Service Composition: synthesis of new services from given set of basic ones

Discrete Event Systems: how to capture the continuous dynamics of an aeroplane for example

Software Engineering: program verification and synthesis

OCaml: functional programming language based on logic



People in L&C: Name and Place of PhD

ANU Staff

Clem Baker-Finch UTasmania
Ranald Clouston Cambridge
Jeremy Dawson Sheffield
Rajeev Goré Cambridge
John Lloyd ANU
John Slaney ANU
Alwen Tiu Pittsburgh

NICTA Staff

Andreas Bauer Munich
Peter Baumgartner Koblenz
Jinbo Huang UCLA
Michael Norrish Cambridge

Current PhD students: Hok Lie, Jimmy Thomson, Peter Gammie, Zhe Hou

Past PhD Students: working at Oracle Research Canberra, Google Zurich, Imperial College London, INRIA Paris, DSTO Adelaide, Quintessence Labs ANU

Future PhD students: always welcome!



Want to find out more ...

COMP2600: Formal Methods for Software Engineering

COMP2620: Logic

MATH3343: Foundations of Mathematics (Honours)

COMP4630: Overview of Logic

Logic Summer School: <http://lss.cecs.anu.edu.au/>

Our Home Page:

<http://cs.anu.edu.au/research/groups/lc>

Independent Study: can be designed for you with appropriate permissions just ask



Do we time for a demo?

Business Process: finite linear discrete sequence

$s_0, s_1, s_2, \dots, s_n$ of states from start state s_0 to end state s_n

Rule 1: A risk assessment (ra) must be carried out for each request to open a bank account (ro)

$$\mathbf{G}(ro \rightarrow \mathbf{F}ra)$$

Rule 2: A request to open a bank account (ro) is granted (rog) only if the risk is low (ral)

$$\mathbf{G}(rog \rightarrow ral)$$

Rule 3: A due diligence assessment (dd) must be carried out for each request to open a bank account (ro)

$$\mathbf{G}(ro \rightarrow \mathbf{F}dd)$$

Rule 4: If a person fails due diligence (ddf) then he or she must be blacklisted (bl)

$$\mathbf{G}((dd \wedge ddf) \rightarrow \mathbf{F}bl)$$



For the curious ...

Digital Circuits: No, $A \wedge \neg C$ is not normal since $KB \wedge A \wedge \neg C$ is unsatisfiable

Banking Rules: are flawed because we forgot to say that blacklisted people should not be given bank accounts

Security Protocols: the protocol is flawed, see Needham Schroeder protocol on wikipedia (the bug was found using logical methods)

