Everything You Always Wanted to Know About Dynamic Logic* (*But Were Afraid to Ask)

Wolfgang Ahrendt

Chalmers University of Technology, Gothenburg, Sweden

KeY Workshop, Bergen, 2023

Outline

- Modal Logic
- Temporal Logic
- Propositional DL
- First-order DL
- Applications
- Reflections

Part I

Base Logics

Modal Logic

- Pre-history: Aristotle, ..., W. of Ockham, ...
- Modern modal logic: C.I. Lewis (1912)
- Semantics: A. Tarski (1930); A. Prior, J. Hintikka, S. Kripke (all 1950s)
- Modal logics come in many flavours: K, T, S4, S5, D, ...
  (vary in properties of accessibility relation)
- Application areas:
  philosophy of language, epistemology, metaphysics, computation

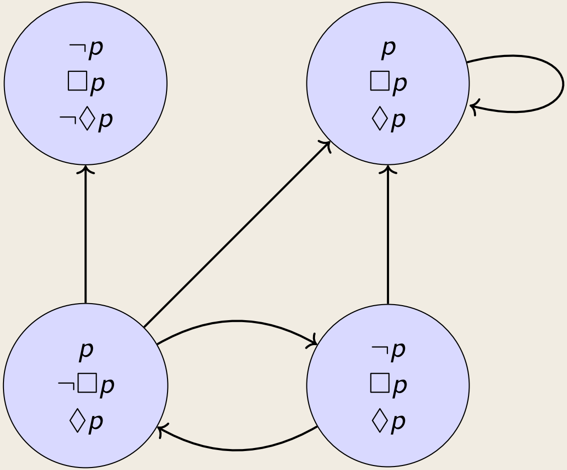
Modal Logic

Modal Logic represents statements about necessity and possibility

- $\Box\varphi$ "φ in all states we can *access from here*"
- $\Diamond\varphi$ "φ in some state we can *access from here*"
- "*access*" is **one single step** in *accessibility* relation

Modal Logic

Kripke Structure: Possible Worlds with (one-step) Accessibility Relation



Temporal Logic

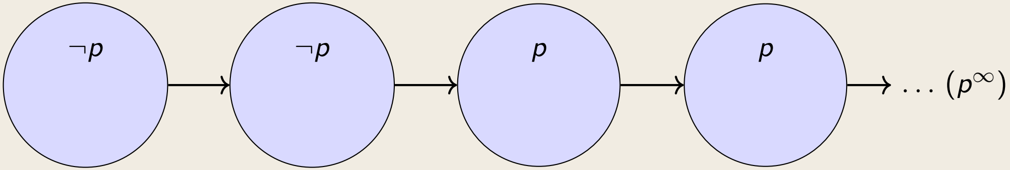
Here: Linear Temporal Logic (LTL) with only $\Box$ and $\Diamond$

- Conceived by:
  A.N. Prior 1957, N. Rescher and A. Urquhart 1971, A. Pnueli 1977
  (Pnueli writes G and F instead of $\Box$ and $\Diamond$)
- $\Box\varphi$ "φ in all future states"
- $\Diamond\varphi$ "φ in some future state"

Question: Is LTL, with only $\Box$ and $\Diamond$, a Modal Logic?

Temporal Logic

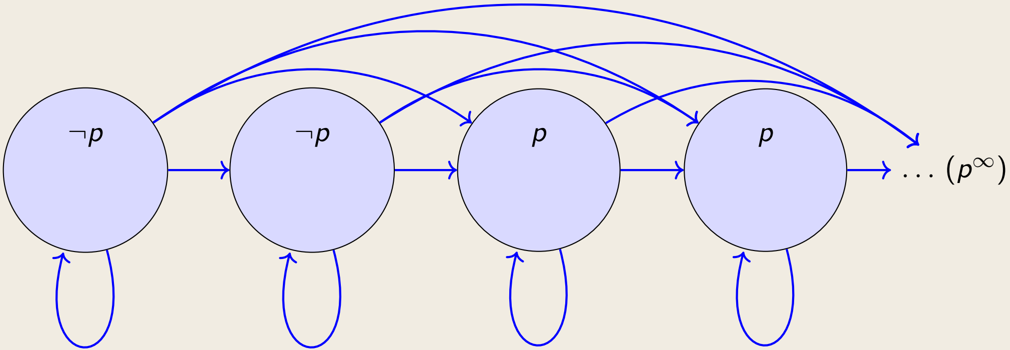
Linear Chain of Worlds with Next-World Relation $\rightarrow$



- Modal accessibility is one step in reflexive transitive closure of next-world relation

Temporal Logic

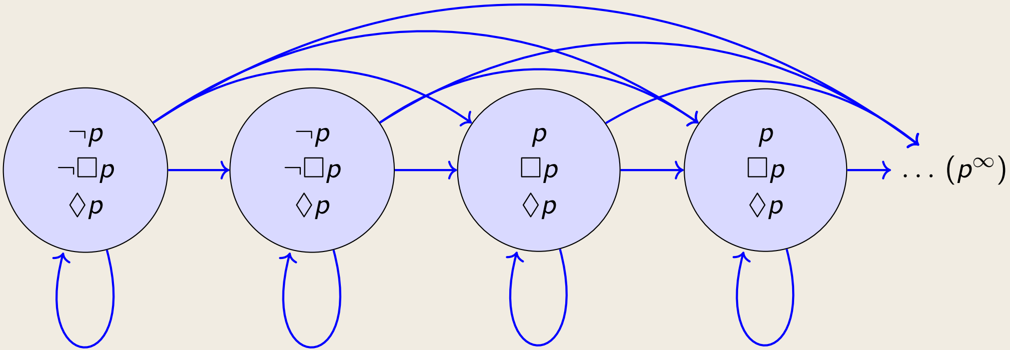
Linear Chain of Worlds with Next-World Relation $\to$



- Modal accessibility is one step in reflexive transitive closure of next-world relation

Temporal Logic

Linear Chain of Worlds with Next-World Relation $\rightarrow$



- Modal accessibility is one step in reflexive transitive closure of next-world relation

Pnueli'77 on LTL as Modal Logic

these developments which are consistent with the transition mechanism of the system. These will be discussed later.

The keen observer would have realized by now that the system presented is completely isomorphic to the modal logic system S4[27,23]. Indeed one way of arriving at it is to give a temporal interpretation to the basic notion of modality, regarding "possible worlds" as "worlds developable in the future starting from the present world". In this isomorphism G stands for $\Box$ and F for $\Diamond$. We resist full identification of the two not only because of typographic reasons but because we believe that the full K_b and even more powerful tense systems will have to be used for proving properties stronger than eventualities. Once one introduces possible worlds both in the past and in the future the correspondence between G and $\Box$ fails. On the other hand in our discussion we will fully utilize this isomorphism as exemplified in the following:

$\underline{p(s) \wedge R(s,}$
$p \supset Fq$

This enables us to
eventualities, those ho
of the system.

Inevitability Axiom:

If we intend to pr
we must give expression
scheduling, which assur
every processor will ul
step. In order to capt
system framework we par

finite number of action

definition of execution
tion:

For no $A \epsilon \mathcal{A}$ is the
$\forall j \; (j \geqslant i) \supset \neg A$

Dynamic Logic: A Multi Modal Logic

Conceived by:

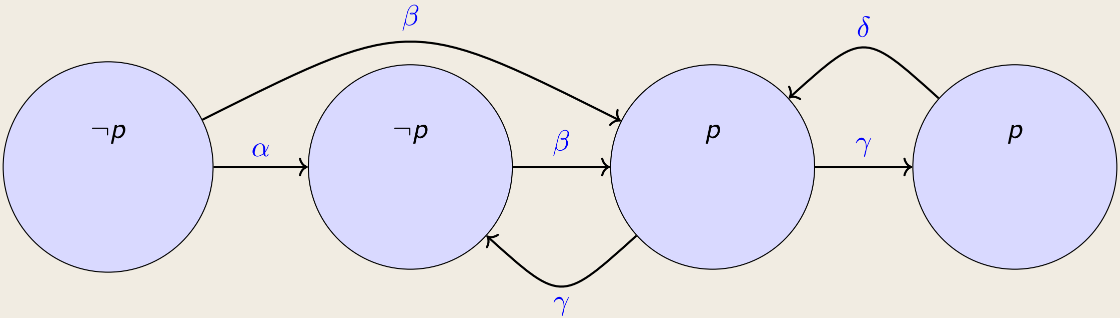
- V.R. Pratt 1976: *"Semantical considerations on Floyd-Hoare Logic"*
- D. Harel 1979: *"First-Order Dynamic Logic"*

Dynamic logic has modalities "*parameterised*" by actions.

- $[\alpha]\varphi$ "φ in all states we can **access by** α"
- $\langle\alpha\rangle\varphi$ "φ in some state we can **access by** α"
- "access by α" refers to **one step** in α-accessibility relation

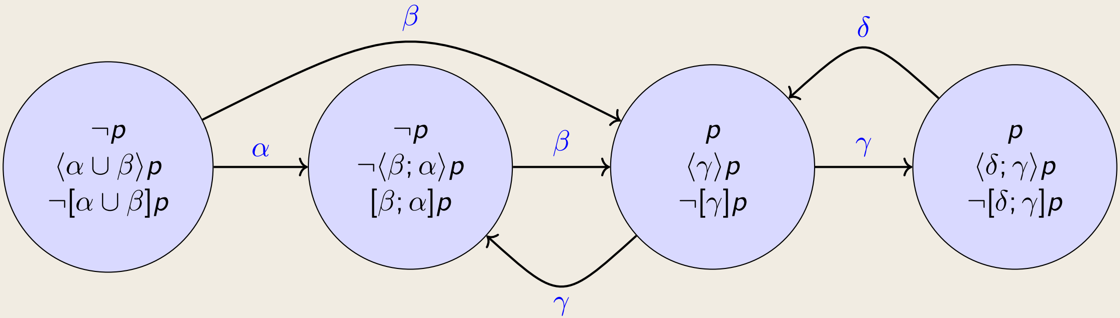
Dynamic Logic as Multi Modal Logic

Worlds with Multiple Next World Relations: Actions



Dynamic Logic as Multi Modal Logic

Worlds with Multiple Next World Relations: Actions



Part II

Propositional Dynamic Logic

Propositional Dynamic Logic (PDL)

- Normally defined for non-deterministic programs
- Non-determinism serves different purposes:
  - a means of abstraction
  - modelling an uncontrollable environment

Propositional Dynamic Logic (PDL)

Propositional DL Formulas

(Assume sets of atomic formulas and programs.)

If φ, ψ are formulas, and α, β are programs, then

- $\neg\varphi$
- $\varphi \vee \psi$
- $\langle\alpha\rangle\varphi$ (some execution of α leads to a state where φ)

are also formulas, and

- $\alpha;\beta$ (sequence)
- $\alpha \cup \beta$ (non-deterministic choice)
- α^* (execute α a **finite**, non-deterministic number of times)
- $?\varphi$ (test φ, proceed if true, fail if false)

are also programs.

Semantics of PDL

Assume:

- atomic formulas: AF
- atomic programs: AP

Semantics of PDL Formulas

Kripke model $\mathcal{M} = (S, \mathcal{I})$ where

- Set of states $S = \{u, v, \ldots\}$
- Interpretation of atomic formulas $\mathcal{I}\colon AF \to 2^S$
- Interpretation of atomic programs $\mathcal{I}\colon AP \to 2^{S \times S}$

Semantics of PDL

Let p be any atomic formula, a be any atomic program

Semantics of PDL Formulas

Meaning of formula $\varphi^{\mathcal{M}} \subseteq S$:

- $p^{\mathcal{M}} = \mathcal{I}(p)$
- $a^{\mathcal{M}} = \mathcal{I}(a)$
- $(\varphi \vee \psi)^{\mathcal{M}} = \varphi^{\mathcal{M}} \cup \psi^{\mathcal{M}}$
- $(\neg\varphi)^{\mathcal{M}} = S - \varphi^{\mathcal{M}}$

- Note: Definition avoids truth values. Instead: Formulas evaluate to sets of states.
- $\wedge, \rightarrow, \leftrightarrow, \mathit{true}, \mathit{false}$ are defined from $\neg, \vee$

Semantics of PDL

Semantics of PDL Formulas

Meaning of formula $\varphi^{\mathcal{M}} \subseteq S$, meaning of program $\alpha^{\mathcal{M}} \subseteq S \times S$:

- $(\alpha; \beta)^{\mathcal{M}} = \{(u, v) \mid \exists w.\ (u, w) \in \alpha^{\mathcal{M}}$ and $(w, v) \in \beta^{\mathcal{M}}\}$
- $(\alpha \cup \beta)^{\mathcal{M}} = \alpha^{\mathcal{M}} \cup \beta^{\mathcal{M}}$
- $(\alpha^*)^{\mathcal{M}} =$ "reflexive transitive closure of $\alpha^{\mathcal{M}}$"
- $(?\varphi)^{\mathcal{M}} = \{(u, u) \mid u \in \varphi^{\mathcal{M}}\}$
- $(\langle\alpha\rangle\varphi)^{\mathcal{M}} = \{u \mid \exists v.\ (u, v) \in \alpha^{\mathcal{M}}$ and $v \in \varphi^{\mathcal{M}}\}$

- Whenever φ holds, $?\varphi$ is "`skip`".
- Whenever φ does not hold, $?\varphi$ does *not result in any state* ('*fails*').
- I.p., $(?\mathit{false})^{\mathcal{M}} = \{(u, u) \mid u \in \mathit{false}^{\mathcal{M}}\} = \{(u, u) \mid u \in \emptyset\} = \emptyset$

Semantics of PDL

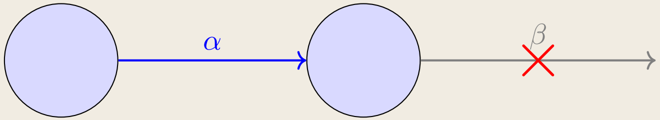
Semantics of PDL Formulas

Meaning of formula $\varphi^{\mathcal{M}} \subseteq S$, meaning of program $\alpha^{\mathcal{M}} \subseteq S \times S$:

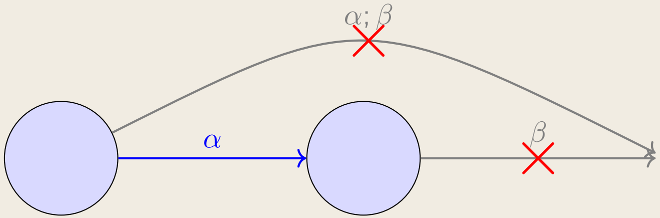
- $(\alpha; \beta)^{\mathcal{M}} = \{(u, v) \mid \exists w.\ (u, w) \in \alpha^{\mathcal{M}} \text{ and } (w, v) \in \beta^{\mathcal{M}}\}$
- $(\alpha \cup \beta)^{\mathcal{M}} = \alpha^{\mathcal{M}} \cup \beta^{\mathcal{M}}$
- $(\alpha^*)^{\mathcal{M}} =$ "reflexive transitive closure of $\alpha^{\mathcal{M}}$"
- $(?\varphi)^{\mathcal{M}} = \{(u, u) \mid u \in \varphi^{\mathcal{M}}\}$
- $(\langle\alpha\rangle\varphi)^{\mathcal{M}} = \{u \mid \exists v.\ (u, v) \in \alpha^{\mathcal{M}} \text{ and } v \in \varphi^{\mathcal{M}}\}$

- If α or β *fail*, then $\alpha; \beta$ *fails*.

Sequential Composition and Failure



Sequential Composition and Failure



Semantics of PDL

Semantics of PDL Formulas

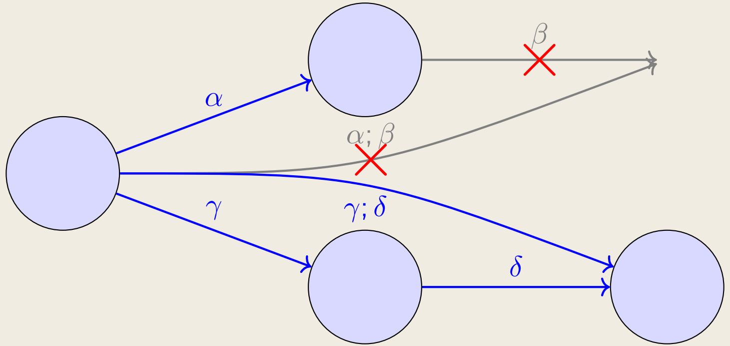
Meaning of formula $\varphi^{\mathcal{M}} \subseteq S$, meaning of program $\alpha^{\mathcal{M}} \subseteq S \times S$:

- $(\alpha; \beta)^{\mathcal{M}} = \{(u, v) \mid \exists w.\ (u, w) \in \alpha^{\mathcal{M}} \text{ and } (w, v) \in \beta^{\mathcal{M}}\}$
- $(\alpha \cup \beta)^{\mathcal{M}} = \alpha^{\mathcal{M}} \cup \beta^{\mathcal{M}}$
- $(\alpha^*)^{\mathcal{M}} =$ "reflexive transitive closure of $\alpha^{\mathcal{M}}$"
- $(?\varphi)^{\mathcal{M}} = \{(u, u) \mid u \in \varphi^{\mathcal{M}}\}$
- $(\langle\alpha\rangle\varphi)^{\mathcal{M}} = \{u \mid \exists v.\ (u, v) \in \alpha^{\mathcal{M}} \text{ and } v \in \varphi^{\mathcal{M}}\}$

- If α *fails*, then $\alpha \cup \beta \ \equiv \ \beta$ (and vice versa)

Non-determinism and Failure

- Example: $\alpha; \beta \cup \gamma; \delta$ where β fails after α



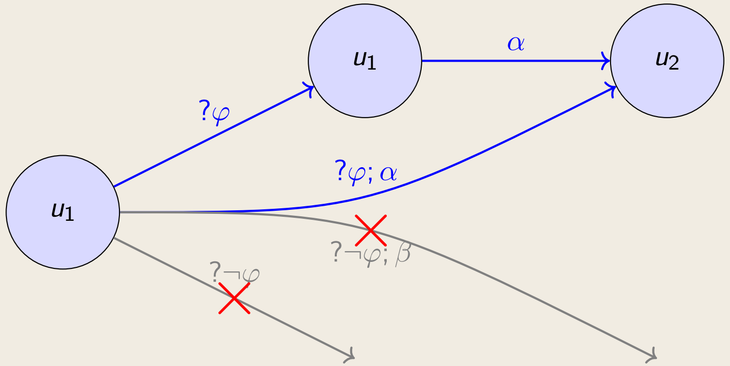
- In this case: $\alpha; \beta \cup \gamma; \delta \equiv \gamma; \delta$
- **transaction mechanism**: if an alternative fails *anywhere*, it "never happened"

Derived Formulas and Programs

- $[\alpha]\varphi \;\equiv\; \neg\langle\alpha\rangle\neg\varphi$ (all executions of α lead to a state where φ)
- **skip** $\equiv$?*true*
- **fail** $\equiv$?*false*
- **if** φ **then** α **else** β **fi** $\equiv$ $(?\varphi;\alpha) \cup (?\neg\varphi;\beta)$
- **while** φ **do** α **od** $\equiv$ $(?\varphi;\alpha)^*\,;\,?\neg\varphi$

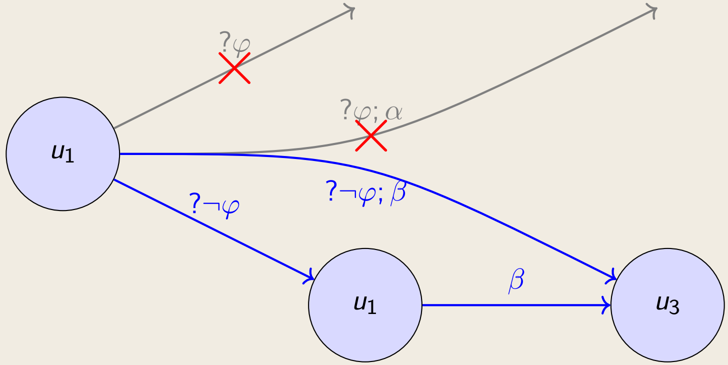
Derived Programs

- **if** φ **then** α **else** β **fi** $\equiv (?\varphi;\alpha) \cup (?\neg\varphi;\beta) \equiv ?\varphi;\alpha \equiv \alpha$



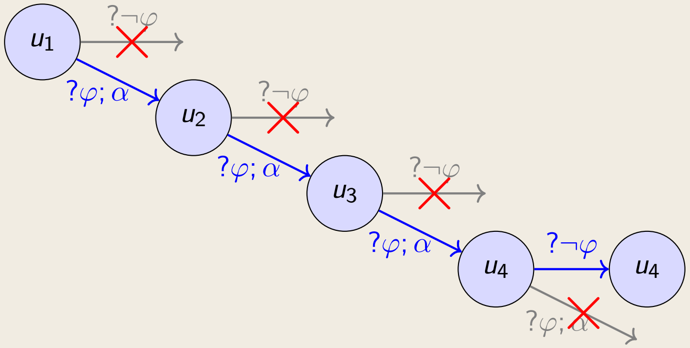
Derived Programs

- **if** φ **then** α **else** β **fi** $\equiv (?\varphi;\alpha) \cup (?\neg\varphi;\beta) \equiv ?\neg\varphi;\beta \equiv \beta$



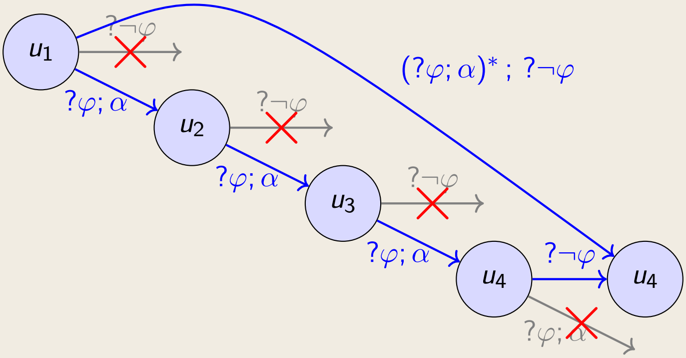
Derived Programs

- **while** φ **do** α **od** $\equiv (?\varphi; \alpha)^* \; ; \; ?\neg\varphi$



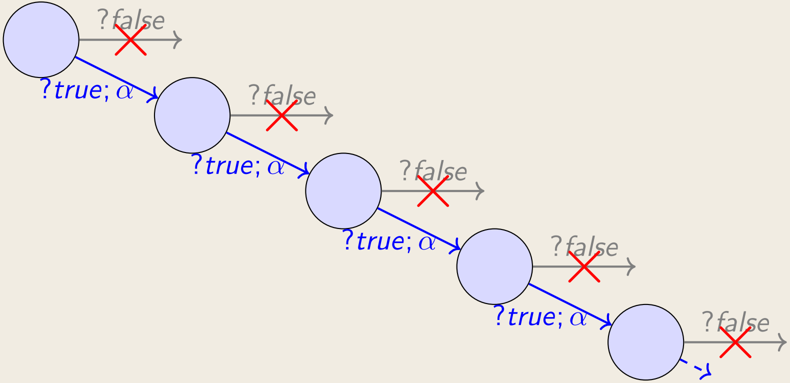
Derived Programs

- **while** φ **do** α **od** $\equiv (?\varphi;\alpha)^*\,;\,?\neg\varphi$



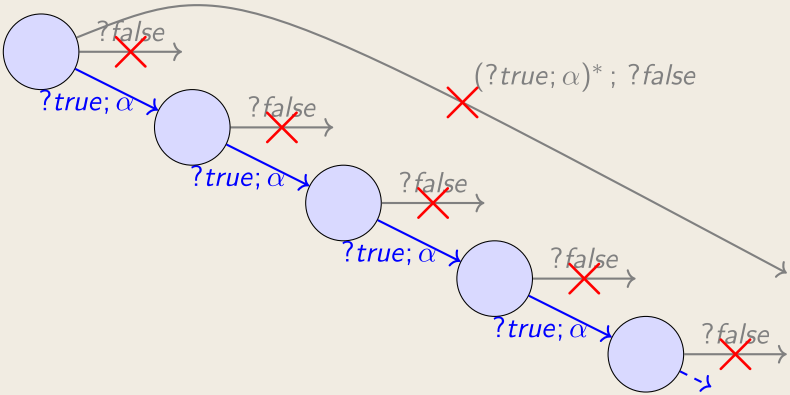
Derived Programs

- **while** *true* **do** α **od** $\equiv (?true;\alpha)^*\,;\,?\neg true \equiv (?true;\alpha)^*\,;\,?false$



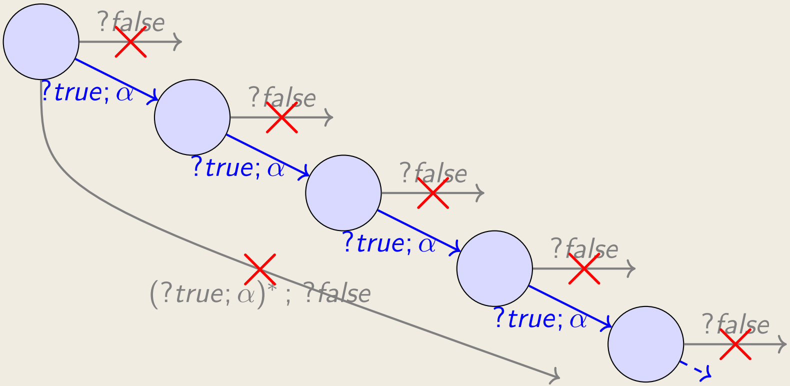
Derived Programs

- **while** *true* **do** α **od** $\equiv (?true; \alpha)^* ; ?\neg true \equiv (?true; \alpha)^* ; ?false$



Derived Programs

- **while** *true* **do** α **od** $\equiv$ $(?true; \alpha)^*$; $?\neg true$ $\equiv$ $(?true; \alpha)^*$; $?false$



Derived Formulas

- Hoare triples: $\{\varphi\}\,\alpha\,\{\psi\} \;\equiv\; \varphi \rightarrow [\alpha]\psi$
- Weakest precondition: $\mathit{wp}\,.\,\alpha\,.\,\varphi \;\equiv\; \langle\alpha\rangle\varphi$
- Weakest liberal precondition: $\mathit{wlp}\,.\,\alpha\,.\,\varphi \;\equiv\; [\alpha]\varphi$

I recommend: *"Dijkstra's Legacy on Program Verification"* by Reiner Hähnle

Some Valid PDL Formulas

- $\langle \alpha \cup \beta \rangle \varphi \;\leftrightarrow\; \langle \alpha \rangle \varphi \lor \langle \beta \rangle \varphi$
- $[\alpha \cup \beta]\varphi \;\leftrightarrow\; [\alpha]\varphi \land [\beta]\varphi$
- $\langle \alpha ; \beta \rangle \varphi \;\leftrightarrow\; \langle \alpha \rangle \langle \beta \rangle \varphi$
- $[\alpha ; \beta]\varphi \;\leftrightarrow\; [\alpha][\beta]\varphi$
- $\langle ?\psi \rangle \varphi \;\leftrightarrow\; \psi \land \varphi$
- $[?\psi]\varphi \;\leftrightarrow\; \psi \rightarrow \varphi$
- $(\varphi \rightarrow [\alpha]\varphi) \;\rightarrow\; (\varphi \rightarrow [\alpha^*]\varphi)$

Meta-properties of PDL

PDL is not compact

- $\{\neg\varphi, \neg\langle\alpha\rangle\varphi, \neg\langle\alpha;\alpha\rangle\varphi, \neg\langle\alpha;\alpha;\alpha\rangle\varphi, \ldots\} \cup \{\langle\alpha^*\rangle\varphi\}$

  is finitely satisfiable, but not satisfiable.

PDL is complete

- There exists a proof system $\vdash$ such that: if $\models \varphi$ then $\vdash \varphi$.

PDL Complexity

- PDL satisfiability is deterministic exponential time complete.
  (Regardless of allowing $\langle\ \rangle$, $[\ ]$ inside ?-tests.)

Deterministic PDL

A program α is deterministic if it describes a partial function:

$$\alpha^{\mathcal{M}} \in S \rightharpoonup S$$

Deterministic while programs

- $\cup$, * appear *only* to abbreviate **if** and **while**

In deterministic PDL:

- $[\alpha]\varphi$ is partial correctness
- $\langle\alpha\rangle\varphi$ is total correctness
- $\langle\alpha\rangle\varphi \to [\alpha]\varphi$ is valid

Part III

First-order Dynamic Logic

First-order Dynamic Logic (DL)

Changes to PDL:

- Atomic programs have forms:
  - $v := t$ (deterministic assignment)
  - $v := *$ (non-deterministic assignment)
- Atomic formulas are of the forms:
  - $p(t_1, \ldots, t_n)$
  - $t_1 = t_2$
- If φ is a DL formula, then so are $\exists x.\varphi$, $\forall x.\varphi$
- φ appearing in $?\varphi$ must be a quantifier-free first-order formula

DL Formula Examples

Note: Definition is fully recursive. It allows, e.g.:

- $\forall x. (\langle t := a; a := b; b := t \rangle b = x \ \leftrightarrow \ \langle a := a + b; b := a - b; a := a - b \rangle b = x)$
- $\langle \alpha \rangle \exists x. \varphi(x)$
- $\exists x. \langle \alpha \rangle \varphi(x)$

Some Valid DL Formulas

- $[v := *]\varphi(v) \;\leftrightarrow\; \forall x.\varphi(x)$
- $\langle v := *\rangle\varphi(v) \;\leftrightarrow\; \exists x.\varphi(x)$
- $\langle v := t\rangle\varphi \;\leftrightarrow\; \varphi[v/t]$
  ($\varphi[v/t]$ result of substituting v by t)
  weakest precondition reasoning
- $[v := t]\varphi \;\leftrightarrow\; \varphi[v/t]$

Meta-properties of (first-order) DL

DL is in-complete

- There exists no proof system $\vdash$ such that:
  if $\models \varphi$ then $\vdash \varphi$.

DL is relatively complete

- Let $\mathcal{A}$ be an arithmetical structure.
- Assume $T_{\mathcal{A}}$ to be all theorems of $\mathcal{A}$.
- There exists a proof system $\vdash$ such that:
  if $\mathcal{A} \models \varphi$ then $T_{\mathcal{A}} \vdash \varphi$.

Part IV

Smart Contract Verification

Solidity Smart Contract: Auction (snippet)

```
...

function withdraw() public {
  // A bidder can withdraw all her money

  withdrawCounter = withdrawCounter + 1;

  require(bidded[msg.sender]);

  msg.sender.transfer(bid[msg.sender]);
  bid[msg.sender] = 0;
}

...
```

- Solidity's `require`(φ) is **exactly** $?\varphi$ from (theoretical) DL
- If `bidded[msg.sender]` is `false`, execution fails, and `withdrawCounter` is not incremented!

Solidity Dynamic Logic

Solidity DL:

- $[\mathrm{p}]\varphi$: If p executes *successfully* then φ holds afterwards
- $\langle\mathrm{p}\rangle\varphi$: p executes *successfully* and φ holds afterwards

Successful execution: does not fail, no state reverted.

(What about non-termination?)

Calculus Rules: require

Rules for `require`

$$\frac{\Gamma, \mathcal{U}(\mathtt{b} = \mathit{true}) \Longrightarrow \mathcal{U}[\omega]\varphi, \Delta}{\Gamma \Longrightarrow \mathcal{U}[\mathtt{require(b);}\ \omega]\varphi, \Delta} \quad \text{b simple}$$

assume $\mathcal{U}(\mathtt{b} = \mathit{true})$ when verifying remaining code

Part V

Abstract Object Creation

Approach Taken

- a logic that can only ‘talk about’ created objects
- problem:
  calculus cannot ‘substitute’ new objects into pre-conditions
- solution:
  non-standard substitution using meta-knowledge about ‘newness’

Semantics

informal

- $[\![u := \text{new}]\!]_\sigma$: create new object and assign it to u

- $[\![e]\!]_\sigma \in$ set of objects existing in σ
- $[\![\forall o.\varphi]\!]_\sigma$: φ holds for all objects existing in σ
- $[\![\exists o.\varphi]\!]_\sigma$: φ holds for some object existing in σ

examples:

$\forall l.\langle u := \text{new}\rangle\neg(u = l)$	true in all states
$\langle u := \text{new}\rangle\forall l.\neg(u = l)$	false in all states

References

- W. Ahrendt, F. de Boer, I. Grabe
  Abstract Object Creation in Dynamic Logic
  – *To Be or Not To Be Created*
  FM'09

Part VI

Reflections

Approaches to Logics of Programs

Endogenous Logics Program fixed outside the formulas
e.g.: LTL

Exogenous Logics Formulas include program fragments
e.g.: Dynamic Logic, Hoare Logic

Pnueli'77 on Endogenous and Exogenous Logics

These suggest a uniform formalism which deals in formulas whose constituents are both logical assertions and program segments, and can express very rich relations between programs and assertions. We will be the first to admit the many advantages of Exogenous systems over Endogenous systems. These include among others:

a. The uniform formalism is more elegant and universal, richer in expressibility, no need for the two phase process of Endogenous systems.

b. Endogenous systems live within a single program. There is no way to compare two programs such as proving equivalence or inclusion.

c. Endogenous systems assume the program to be rigidly given, Exogenous systems provide tools and guidance for constructing a correct system rather than just analyse an existent one.

Against these advantages endogenous system can

Science, Rehovot,

11\. - Francez, N. and
For Cyclic Progra
Conference on Par

12\. - Francez, N. and
perties of Paral
Invariants," to a

13\. - Keller, R.M.: "
Programs," CACM

14\. - Kröger, F.: "Log
ing About Program
on Automata, Lang
Edinburgh, Edinbu
87-98.

15\. - Kröger, F: "A Un
Description, Spec
Proof Techniques

Pnueli'77 on Endogenous and Exogenous Logics

system rather than just analyse an existent one.

Against these advantages endogenous system can offer the following single line of defense: When the going is tough, and we are interested in proving a single intricate and difficult program, we do not care about generality, uniformity or equivalence. It is then advantageous to work with a fixed context rather than carry a varying context with each statement. Under these conditions, endogenous systems attempt to equip the prover with the strongest possible tools to formalize his intuitive thinking and ease his way to a rigorous proof.

References:

1. - Aschroft E.A. and Manna Z (1970): "Formalization of Properties of Parallel Programs," Machine Intelligence 6, Edinburgh University Press.

15. - Kröger, F: "A Ur
Description, Spec
Proof Techniques
Informatik der Te

16. - Lamport, L (1976)
Multiprocess Prog
Associates, Inc.

17. - Manna Z: "Mathem
McGraw-Hill (197

18. - Manna Z. and Pnue
Total Correctness
263.

19. - Manna Z. and Wald
sometimes better
assertions in pro
Proc. 2nd Interna
Engineering, San
39.

Remarks

- I did not cover applications and tooling for PDL
- I did not do justice to rich theory of (P)DL
  but see:
  David Harel, Dexter Kozen, Jerzy Tiuryn
  Dynamic Logic
  MIT Press 2000

Thanks!