Chapter 3

LTL model checking

Course “Model checking”
Martin Steffen
Autumn 2021

Section

Introduction

Chapter 3 “LTL model checking”
Course “Model checking”
Martin Steffen
Autumn 2021

Temporal logic?

- Temporal logic: is the/a logic of “time”
- *modal* logic.
- different ways of modeling time.
  - linear vs. branching time
  - time instances vs. time intervals
  - discrete time vs. continuous time
  - past and future vs. future only
  - ...

LTL

- linear time temporal logic
- one central temporal logic in CS
- supported by Spin and other model checkers
- many variations

Section

LTL

Chapter 3 “LTL model checking”
Course “Model checking”
Martin Steffen
Autumn 2021

LTL: speaking about "time"

- linear time temporal logic
- one central temporal logic in CS
- supported by Spin and other model checkers
- many variations

we can describe properties like, for instance, the following:
assume time is a *sequence* of discrete points i in time, then:
if i is *now*,

- p holds in i and every following point (the future)
- p holds in i and every preceding point (the past)

$$\ldots \longrightarrow \bullet^{p}_{i-2} \longrightarrow \bullet^{p}_{i-1} \longrightarrow \bullet^{p}_{i} \longrightarrow \bullet^{p}_{i+1} \longrightarrow \bullet^{p}_{i+2} \longrightarrow \ldots$$

Syntax

ψ			propositional/first-order formula
φ	$::=$	ψ	formulas of the "core" logics
	$\mid$	$\neg\varphi \mid \varphi \wedge \varphi \mid \varphi \rightarrow \varphi \mid \ldots$	boolean combinations
	$\mid$	$\bigcirc\varphi$	next φ
	$\mid$	$\Box\varphi$	always φ
	$\mid$	$\Diamond\varphi$	eventually φ
	$\mid$	$\varphi\ U\ \varphi$	"until"
	$\mid$	$\varphi\ R\ \varphi$	"release"
	$\mid$	$\varphi\ W\ \varphi$	"waiting for", "weak until"

Semantics:s Paths and computations

Definition (Path)

- A path is an infinite sequence

$$\pi = s_0, s_1, s_2, \ldots$$

of states.

Semantics:s Paths and computations

Definition (Path)

- A path is an infinite sequence

$$\pi = s_0, s_1, s_2, \ldots$$

of states.
- π^k denotes the *path* $s_k, s_{k+1}, s_{k+2}, \ldots$

Semantics:s Paths and computations

Definition (Path)

- A path is an infinite sequence

$$\pi = s_0, s_1, s_2, \ldots$$

  of states.
- π^k denotes the *path* $s_k, s_{k+1}, s_{k+2}, \ldots$
- π_k denotes the *state* s_k.

Semantics

$\pi \models \psi$	iff	$\pi_0 \models_{\mathsf{ul}} \psi$ with ψ from the underlying core language
$\pi \models \neg\varphi$	iff	$\pi \not\models \varphi$
$\pi \models \varphi_1 \wedge \varphi_2$	iff	$\pi \models \varphi_1$ and $\pi \models \varphi_2$
$\pi \models \bigcirc\varphi$	iff	$\pi^1 \models \varphi$
$\pi \models \varphi_1 \ U \ \varphi_2$	iff	$\pi^k \models \varphi_2$ for some $k \geq 0$, and
		$\pi^i \models \varphi_1$ for every i such that $0 \leq i < k$

Semantics: derived operators

$\pi \models \Box\varphi$ iff $\pi^k \models \varphi$ for all $k \geq 0$

$\pi \models \Diamond\varphi$ iff $\pi^k \models \varphi$ for some $k \geq 0$

$\pi \models \varphi_1 \; R \; \varphi_2$ iff for every $j \geq 0$,
if $\pi^i \not\models \varphi_1$ for every $i < j$ then $\pi^j \models \varphi_2$

$\pi \models \varphi_1 \; W \; \varphi_2$ iff $\pi \models \varphi_1 \; U \; \varphi_2$ or $\pi \models \Box\varphi_1$

Validity and semantic equivalence

Definition (Validity and equivalence)

- φ is (temporally) valid, written $\models \varphi$, if

$$\pi \models \varphi \text{ for all paths } \pi.$$

Validity and semantic equivalence

Definition (Validity and equivalence)

- φ is (temporally) valid, written $\models \varphi$, if

$$\pi \models \varphi \text{ for all paths } \pi.$$

- φ_1 and φ_2 are equivalent, written $\varphi_1 \sim \varphi_2$, if

$$\models \varphi_1 \leftrightarrow \varphi_2 \text{ (i.e. } \pi \models \varphi_1 \text{ iff } \pi \models \varphi_2\text{, for all } \pi\text{).}$$

Validity and semantic equivalence

Definition (Validity and equivalence)

- φ is (temporally) valid, written $\models \varphi$, if
$$\pi \models \varphi \text{ for all paths } \pi.$$
- φ_1 and φ_2 are equivalent, written $\varphi_1 \sim \varphi_2$, if
$$\models \varphi_1 \leftrightarrow \varphi_2 \text{ (i.e. } \pi \models \varphi_1 \text{ iff } \pi \models \varphi_2\text{, for all } \pi).$$

Example

$\Box$ distributes over $\wedge$, while $\Diamond$ distributes over $\vee$.

$$\Box(\varphi \wedge \psi) \sim (\Box\varphi \wedge \Box\psi)$$
$$\Diamond(\varphi \vee \psi) \sim (\Diamond\varphi \vee \Diamond\psi)$$

Illustrations

$$\pi \models \Box p$$

$$\bullet_0^p \longrightarrow \bullet_1^p \longrightarrow \bullet_2^p \longrightarrow \bullet_3^p \longrightarrow \bullet_4^p \longrightarrow \dots$$

$$\pi \models \Diamond p$$

$$\bullet_0 \longrightarrow \bullet_1 \longrightarrow \bullet_2 \longrightarrow \bullet_3^p \longrightarrow \bullet_4 \longrightarrow \dots$$

$$\pi \models \bigcirc p$$

$$\bullet_0 \longrightarrow \bullet_1^p \longrightarrow \bullet_2 \longrightarrow \bullet_3 \longrightarrow \bullet_4 \longrightarrow \dots$$

Some more illustrations

$\pi \models p \; U \; q$ (sequence of p's is *finite*)

$$\bullet_0^{p} \longrightarrow \bullet_1^{p} \longrightarrow \bullet_2^{p} \longrightarrow \bullet_3^{q,(p)} \longrightarrow \bullet_4 \longrightarrow \dots$$

$\pi \models p \; R \; q$ (sequence of qs may be infinite)

$$\bullet_0^{q} \longrightarrow \bullet_1^{q} \longrightarrow \bullet_2^{q} \longrightarrow \bullet_3^{p,q} \longrightarrow \bullet_4 \longrightarrow \dots$$

$\pi \models p \; W \; q$

The sequence of ps may be infinite. ($p \; W \; q \sim p \; U \; q \vee \Box p$).

$$\bullet_0^{p} \longrightarrow \bullet_1^{p} \longrightarrow \bullet_2^{p} \longrightarrow \bullet_3^{p} \longrightarrow \bullet_4^{p} \longrightarrow \dots$$

The past

Observation

- Manna and Pnueli [4] uses pairs (π, j) of paths and positions instead of just the path π because they have past-formulas: formulas without future operators (the ones we use) but possibly with past operators, like $\Box^{-1}$ and $\Diamond^{-1}$.

$$(\pi, j) \models \Box^{-1}\varphi \quad \text{iff} \quad (\pi, k) \models \varphi \text{ for all } k,\ 0 \leq k \leq j$$

$$(\pi, j) \models \Diamond^{-1}\varphi \quad \text{iff} \quad (\pi, k) \models \varphi \text{ for some } k,\ 0 \leq k \leq j$$

The past

Observation

- Manna and Pnueli [4] uses pairs (π, j) of paths and positions instead of just the path π because they have past-formulas: formulas without future operators (the ones we use) but possibly with past operators, like $\Box^{-1}$ and $\Diamond^{-1}$.

$$(\pi, j) \models \Box^{-1}\varphi \quad \text{iff} \quad (\pi, k) \models \varphi \text{ for all } k,\ 0 \leq k \leq j$$
$$(\pi, j) \models \Diamond^{-1}\varphi \quad \text{iff} \quad (\pi, k) \models \varphi \text{ for some } k,\ 0 \leq k \leq j$$

- However, it can be shown that for any formula φ, there is a future-formula (formulae without past operators) ψ such that

$$(\pi, 0) \models \varphi \quad \text{iff} \quad (\pi, 0) \models \psi$$

The past: example

Example ($\Box(p \to \Diamond^{-1} q)$?)

$(\pi, 0) \models \Box(p \to \Diamond^{-1} q)$

$$\bullet^{p \to \Diamond^{-1} q} \longrightarrow \bullet^{p \to \Diamond^{-1} q} \longrightarrow \bullet^{p \to \Diamond^{-1} q} \longrightarrow \bullet^{p \to \Diamond^{-1} q} \longrightarrow \bullet \longrightarrow \ldots$$

$(\pi, 0) \models q \; R \; (p \to q)$

$$\bullet^{p \to q} \longrightarrow \bullet^{p \to q} \longrightarrow \bullet^{p \to q, q} \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \ldots$$

Formalization of “informal” properties

Example (Informal statement: “when φ then ψ”)

Formalization of “informal” properties

Example (Informal statement: “when φ then ψ”)

- $\varphi \to \psi$?

Formalization of “informal” properties

Example (Informal statement: “when φ then ψ”)

- $\varphi \to \psi$? $\varphi \to \psi$ holds in the initial state.

Formalization of “informal” properties

Example (Informal statement: “when φ then ψ”)

- $\varphi \rightarrow \psi$? $\varphi \rightarrow \psi$ holds in the initial state.
- $\Box(\varphi \rightarrow \psi)$?

Formalization of “informal” properties

Example (Informal statement: “when φ then ψ”)

- $\varphi \rightarrow \psi$? $\varphi \rightarrow \psi$ holds in the initial state.
- $\Box(\varphi \rightarrow \psi)$? $\varphi \rightarrow \psi$ holds in every state.

Formalization of "informal" properties

Example (Informal statement: "when φ then ψ")

- $\varphi \to \psi$? $\varphi \to \psi$ holds in the initial state.
- $\Box(\varphi \to \psi)$? $\varphi \to \psi$ holds in every state.
- $\varphi \to \Diamond\psi$?

Formalization of “informal” properties

Example (Informal statement: “when φ then ψ”)

- $\varphi \rightarrow \psi$? $\varphi \rightarrow \psi$ holds in the initial state.
- $\Box(\varphi \rightarrow \psi)$? $\varphi \rightarrow \psi$ holds in every state.
- $\varphi \rightarrow \Diamond\psi$? φ holds in the initial state, ψ will hold in some state.

Formalization of "informal" properties

Example (Informal statement: "when φ then ψ")

- $\varphi \to \psi$? $\varphi \to \psi$ holds in the initial state.
- $\Box(\varphi \to \psi)$? $\varphi \to \psi$ holds in every state.
- $\varphi \to \Diamond\psi$? φ holds in the initial state, ψ will hold in some state.
- $\Box(\varphi \to \Diamond\psi)$?

Formalization of "informal" properties

Example (Informal statement: "when φ then ψ")

- $\varphi \rightarrow \psi$? $\varphi \rightarrow \psi$ holds in the initial state.
- $\Box(\varphi \rightarrow \psi)$? $\varphi \rightarrow \psi$ holds in every state.
- $\varphi \rightarrow \Diamond\psi$? φ holds in the initial state, ψ will hold in some state.
- $\Box(\varphi \rightarrow \Diamond\psi)$? "response"

Some examples

Example (Temporal properties)

1. If φ holds initially, then ψ holds eventually.
2. Every φ-position is responded by a later ψ-position (response)
3. There are infinitely many ψ-positions.
4. Sooner or later, φ will hold *permanently* (permanence, stabilization).
5. The first φ-position must coincide or be preceded by a ψ-position.
6. Every φ-position initiates a sequence of ψ-positions, and if terminated, by a χ-position.

Example

Example

$\varphi \rightarrow \Diamond\psi$: If φ holds initially, then ψ holds eventually.

$$\bullet^{\varphi} \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet^{\psi} \longrightarrow \bullet \longrightarrow \ldots$$

This formula will also hold in every path where φ does not hold initially.

$$\bullet^{\neg\varphi} \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \ldots$$

Response

Example (Response)

$\Box(\varphi \rightarrow \Diamond\psi)$

Every φ-position coincides with or is followed by a ψ-position.

$$\bullet \longrightarrow \bullet^{\varphi} \longrightarrow \bullet \longrightarrow \bullet^{\psi} \longrightarrow \bullet \longrightarrow \bullet^{\varphi,\psi} \longrightarrow \dots$$

This formula will also hold in every path where φ never holds.

$$\bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \dots$$

Example: ∞

Example (∞)

$\Box\Diamond\psi$

There are infinitely many ψ-positions.

$$\bullet^{\psi} \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet^{\psi} \longrightarrow \bullet \longrightarrow \bullet^{\psi} \longrightarrow \bullet \longrightarrow \ldots$$

- model-checking?
- run-time verification?

Permanence

Example (Permanence: $\Diamond\Box\varphi$)

Eventually φ will hold permanently.

$$\bullet \longrightarrow \bullet^{\varphi} \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet^{\varphi} \longrightarrow \bullet^{\varphi} \longrightarrow \bullet^{\varphi} \longrightarrow \ldots$$

Equivalently: there are *finitely* many $\neg\varphi$-positions.

And another one

Example

$(\neg\varphi)\ W\ \psi$

The first φ-position must coincide or be preceded by a ψ-position.

$$\bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\psi} \longrightarrow \bullet^{\varphi} \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \dots$$

φ may never hold

$$\bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \bullet^{\neg\varphi} \longrightarrow \dots$$

LTL example

Example

$\Box(\varphi \to \psi \; W \; \chi)$

Every φ-position initiates a sequence of ψ-positions, and if terminated, by a χ-position.

$$\bullet \longrightarrow \bullet^{\varphi,\psi} \longrightarrow \bullet^{\psi} \longrightarrow \bullet^{\psi} \longrightarrow \bullet^{\chi} \longrightarrow \bullet \longrightarrow \bullet^{\varphi,\psi} \longrightarrow$$

The sequence of ψ-positions need not terminate.

$$\bullet \longrightarrow \bullet^{\varphi,\psi} \longrightarrow \bullet^{\psi} \longrightarrow \bullet^{\psi} \longrightarrow \bullet^{\psi} \longrightarrow \bullet^{\psi} \longrightarrow \bullet^{\psi} \longrightarrow$$

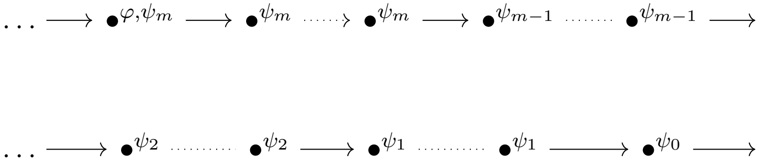
Nested waiting-for

A nested waiting-for formula is of the form

$$\Box(\varphi \rightarrow (\psi_m \; W \; (\psi_{m-1} \; W \cdots (\psi_1 \; W \; \psi_0) \cdots))),$$

where $\varphi, \psi_0, \ldots, \psi_m$ in the underlying logic. For convenience, we write

$$\Box(\varphi \rightarrow \psi_m \, W \, \psi_{m-1} \, W \; \cdots \; W \, \psi_1 \, W \, \psi_0).$$



Duality

Definition (Duals)

For binary boolean connectives $\circ$ and $\bullet$, we say that $\bullet$ is the dual of $\circ$ if

$$\neg(\varphi \circ \psi) \sim (\neg\varphi \bullet \neg\psi).$$

Similarly for unary connectives: $\bullet$ is the dual of $\circ$ if $\neg \circ \varphi \sim \bullet\neg\varphi$.

Duality is *symmetric*:

- If $\bullet$ is the dual of $\circ$ then
- $\circ$ is the dual of $\bullet$, thus
- we may refer to two connectives as dual (of each other).

Dual connectives

Dual connectives

- $\wedge$ and $\vee$ are duals:

$$\neg(\varphi \wedge \psi) \sim (\neg\varphi \vee \neg\psi).$$

Dual connectives

- $\wedge$ and $\vee$ are duals:

$$\neg(\varphi \wedge \psi) \sim (\neg\varphi \vee \neg\psi).$$

- $\neg$ is its own dual:

$$\neg\neg\varphi \sim \neg\neg\varphi.$$

Dual connectives

- $\wedge$ and $\vee$ are duals:

$$\neg(\varphi \wedge \psi) \sim (\neg\varphi \vee \neg\psi).$$

- $\neg$ is its own dual:

$$\neg\neg\varphi \sim \neg\neg\varphi.$$

- What is the dual of $\rightarrow$?

Dual connectives

- $\wedge$ and $\vee$ are duals:

$$\neg(\varphi \wedge \psi) \sim (\neg\varphi \vee \neg\psi).$$

- $\neg$ is its own dual:

$$\neg\neg\varphi \sim \neg\neg\varphi.$$

- What is the dual of $\rightarrow$? It's $\not\leftarrow$:

$$\neg(\varphi \not\leftarrow \psi)$$

Dual connectives

- $\wedge$ and $\vee$ are duals:

$$\neg(\varphi \wedge \psi) \sim (\neg\varphi \vee \neg\psi).$$

- $\neg$ is its own dual:

$$\neg\neg\varphi \sim \neg\neg\varphi.$$

- What is the dual of $\rightarrow$? It's $\not\leftarrow$:

$$\neg(\varphi \not\leftarrow \psi) \sim \varphi \leftarrow \psi$$

Dual connectives

- $\wedge$ and $\vee$ are duals:

$$\neg(\varphi \wedge \psi) \sim (\neg\varphi \vee \neg\psi).$$

- $\neg$ is its own dual:

$$\neg\neg\varphi \sim \neg\neg\varphi.$$

- What is the dual of $\rightarrow$? It's $\not\leftarrow$:

$$\begin{aligned} \neg(\varphi \not\leftarrow \psi) &\sim \varphi \leftarrow \psi \\ &\sim \psi \rightarrow \varphi \end{aligned}$$

Dual connectives

- $\wedge$ and $\vee$ are duals:

$$\neg(\varphi \wedge \psi) \sim (\neg\varphi \vee \neg\psi).$$

- $\neg$ is its own dual:

$$\neg\neg\varphi \sim \neg\neg\varphi.$$

- What is the dual of $\rightarrow$? It's $\not\leftarrow$:

$$\begin{aligned}\neg(\varphi \not\leftarrow \psi) &\sim \varphi \leftarrow \psi \\ &\sim \psi \rightarrow \varphi \\ &\sim \neg\varphi \rightarrow \neg\psi\end{aligned}$$

Complete sets of connectives

- A set of connectives is /complete/ (for boolean formulae) if every other connective can be defined in terms of them.

Example

$\{\vee, \neg\}$ is complete.

Complete sets of connectives

- A set of connectives is /complete/ (for boolean formulae) if every other connective can be defined in terms of them.
- Our set of connectives is complete (e.g., $\not\leftarrow$ can be defined), but also subsets of it, so we don't actually need all the connectives.

Example

$\{\vee, \neg\}$ is complete.

Complete sets of connectives

- A set of connectives is /complete/ (for boolean formulae) if every other connective can be defined in terms of them.
- Our set of connectives is complete (e.g., $\nleftarrow$ can be defined), but also subsets of it, so we don't actually need all the connectives.

Example

$\{\vee, \neg\}$ is complete.

Complete sets of connectives

- A set of connectives is /complete/ (for boolean formulae) if every other connective can be defined in terms of them.
- Our set of connectives is complete (e.g., $\nleftarrow$ can be defined), but also subsets of it, so we don't actually need all the connectives.

Example

$\{\vee, \neg\}$ is complete.

- $\wedge$ is the dual of $\vee$.

Complete sets of connectives

- A set of connectives is /complete/ (for boolean formulae) if every other connective can be defined in terms of them.
- Our set of connectives is complete (e.g., $\not\leftarrow$ can be defined), but also subsets of it, so we don't actually need all the connectives.

Example

$\{\vee, \neg\}$ is complete.

- $\wedge$ is the dual of $\vee$.
- $\varphi \rightarrow \psi$ is equivalent to $\neg\varphi \vee \psi$.

Complete sets of connectives

- A set of connectives is /complete/ (for boolean formulae) if every other connective can be defined in terms of them.
- Our set of connectives is complete (e.g., $\not\leftarrow$ can be defined), but also subsets of it, so we don't actually need all the connectives.

Example

$\{\vee, \neg\}$ is complete.

- $\wedge$ is the dual of $\vee$.
- $\varphi \rightarrow \psi$ is equivalent to $\neg\varphi \vee \psi$.
- $\varphi \leftrightarrow \psi$ is equivalent to $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$.

Complete sets of connectives

- A set of connectives is /complete/ (for boolean formulae) if every other connective can be defined in terms of them.
- Our set of connectives is complete (e.g., $\not\leftarrow$ can be defined), but also subsets of it, so we don't actually need all the connectives.

Example

$\{\vee, \neg\}$ is complete.

- $\wedge$ is the dual of $\vee$.
- $\varphi \rightarrow \psi$ is equivalent to $\neg\varphi \vee \psi$.
- $\varphi \leftrightarrow \psi$ is equivalent to $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$.
- $\top$ is equivalent to $p \vee \neg p$

Complete sets of connectives

- A set of connectives is /complete/ (for boolean formulae) if every other connective can be defined in terms of them.
- Our set of connectives is complete (e.g., $\not\leftarrow$ can be defined), but also subsets of it, so we don't actually need all the connectives.

Example

$\{\vee, \neg\}$ is complete.

- $\wedge$ is the dual of $\vee$.
- $\varphi \rightarrow \psi$ is equivalent to $\neg\varphi \vee \psi$.
- $\varphi \leftrightarrow \psi$ is equivalent to $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$.
- $\top$ is equivalent to $p \vee \neg p$
- $\bot$ is equivalent to $p \wedge \neg p$

Duals in LTL

- What is the dual of $\Box$? And of $\Diamond$?
- $\Box$ and $\Diamond$ are duals.

$$\neg\Box\varphi \sim \Diamond\neg\varphi$$
$$\neg\Diamond\varphi \sim \Box\neg\varphi$$

- Any other?
- U and R are duals.

$$\neg(\varphi\ U\ \psi) \sim (\neg\varphi)\ R\ (\neg\psi)$$
$$\neg(\varphi\ R\ \psi) \sim (\neg\varphi)\ U\ (\neg\psi)$$

Complete set of LTL operators

Proposition

The set of operators $\vee, \neg, U, \bigcirc$ *is complete for LTL.*

Classification of properties

We can classify properties expressible in LTL. *Examples*:

Classification

invariant $\Box\varphi$

"liveness" $\Diamond\varphi$

obligation $\Box\varphi \vee \Diamond\psi$

recurrence $\Box\Diamond\varphi$

persistence $\Diamond\Box\varphi$

reactivity $\Box\Diamond\varphi \vee \Diamond\Box\psi$

- φ, ψ: non-temporal formulas

Safety (slightly simplified)

- important basic class of properties
- relation to testing and run-time verification
- informally "nothing bad ever happens"

Definition (Safety/invariant)

- A invariant formula is of the form

$$\Box\varphi$$

for some prop. formula φ.

Safety formulae express *invariance* of some state property φ: that φ holds in every state of the computation.

Safety (slightly simplified)

- important basic class of properties
- relation to testing and run-time verification
- informally "nothing bad ever happens"

Definition (Safety/invariant)

- A invariant formula is of the form

$$\Box\varphi$$

  for some prop. formula φ.
- A conditional safety formula is of the form

$$\varphi \rightarrow \Box\psi$$

  for some prop. formulas φ and ψ.

Safety formulae express *invariance* of some state property φ: that φ holds in every state of the computation.

Safety property example

Example (Mutex)

Mutual exclusion is a safety property. Let c_i denote that process P_i is executing in the critical section. Then

$$\Box\neg(c_1 \wedge c_2)$$

expresses that it should always be the case that not both P_1 and P_2 are executing in the critical section.

Observe: the negation of a safety formula is a liveness formula; the negation of the formula above is the liveness formula

$$\Diamond(c_1 \wedge c_2)$$

which expresses that eventually it *is* the case that both P_1 and P_2 are executing in the critical section.

Liveness properties (simplified)

Definition (Liveness)

- A liveness formula is of the form

$$\Diamond\varphi$$

for some prop. formula φ.

Liveness formulae *guarantee* that some event φ eventually happens: that φ holds in at least one state of the computation.

Liveness properties (simplified)

Definition (Liveness)

- A liveness formula is of the form

$$\Diamond\varphi$$

for some prop. formula φ.

- A conditional liveness formula is of the form

$$\varphi \rightarrow \Diamond\psi$$

for prop. formulas φ and ψ.

Liveness formulae *guarantee* that some event φ eventually happens: that φ holds in at least one state of the computation.

Connection to Hoare logic

Observation

- Partial correctness is a safety property. Let P be a program and ψ the post condition.

$$\Box(terminated(P) \rightarrow \psi)$$

Connection to Hoare logic

Observation

- Partial correctness is a safety property. Let P be a program and ψ the post condition.

$$\Box(terminated(P) \rightarrow \psi)$$

- In the case of full partial correctness, where there is a precondition φ, we get a *conditional safety* formula,

$$\varphi \rightarrow \Box(terminated(P) \rightarrow \psi),$$

which we can express as $\{\ \varphi\ \}\ P\ \{\ \psi\ \}$ in Hoare Logic.

Total correctness and liveness

Observation

- Total correctness is a liveness property. Let P be a program and ψ the post condition.

$$\Diamond(terminated(P) \wedge \psi)$$

Total correctness and liveness

Observation

- Total correctness is a liveness property. Let P be a program and ψ the post condition.

$$\Diamond(terminated(P) \wedge \psi)$$

- In the case of full total correctness, where there is a precondition φ, we get a *conditional liveness* formula,

$$\varphi \rightarrow \Diamond(terminated(P) \wedge \psi).$$

Duality of partial and total correctness

Observation

Partial and total correctness are dual.

Let

$$PC(\psi) \triangleq \Box(terminated \to \psi)$$
$$TC(\psi) \triangleq \Diamond(terminated \land \psi)$$

Then

$$\neg PC(\psi) \leftrightarrow TC(\neg\psi)$$
$$\neg TC(\psi) \leftrightarrow PC(\neg\psi)$$

Obligation

Definition (Obligation)

A simple obligation formula is of the form

$$\Box\varphi \vee \Diamond\psi$$

for propositional formulas φ and ψ.

- equivalently

$$\Diamond\varphi \rightarrow \Diamond\psi$$

Obligation (2)

Proposition

Every safety and liveness formula is also an obligation formula.

Proof.

It's a consequence of the following equivalences.

$$\Box\varphi \leftrightarrow \Box\varphi \vee \Diamond\bot$$
$$\Diamond\varphi \leftrightarrow \Box\bot \vee \Diamond\varphi$$

and the facts that $\models \neg\Box\bot$ and $\models \neg\Diamond\bot$. □

Recurrence

Definition (Recurrence)

A recurrence formula is of the form

$$\Box\Diamond\varphi$$

for some propositional formula φ.

- *infinitely many* positions satisfies φ.

Observation

A response formula, of the form $\Box(\varphi \rightarrow \Diamond\psi)$, is equivalent to a recurrence formula, of the form $\Box\Diamond\chi$, if we allow χ to be a past-formula.

$$\Box(\varphi \rightarrow \Diamond\psi) \leftrightarrow \Box\Diamond(\neg\varphi)\ W^{-1}\ \psi$$

Recurrence

Proposition

Weak fairness can be specified as the following recurrence formula.

$$\Box\Diamond(enabled(\tau) \rightarrow taken(\tau))$$

Observation

An equivalent form is

$$\Box(\Box enabled(\tau) \rightarrow \Diamond taken(\tau)),$$

Persistence

Definition (Persistence)

A persistence formula is of the form

$$\Diamond\Box\varphi$$

for some propositional formula φ.

- dual to "infinitely often"
- aka: stabilization

Recurrence and Persistence

Observation

Recurrence and persistence are duals.

$$\neg(\Box\Diamond\varphi) \leftrightarrow (\Diamond\Box\neg\varphi)$$

$$\neg(\Diamond\Box\varphi) \leftrightarrow (\Box\Diamond\neg\varphi)$$

Reactivity

Definition (Reactivity)

- A simple reactivity formula is of the form

$$\Box\Diamond\varphi \vee \Diamond\Box\psi$$

for prop. formulas φ and ψ.

Reactivity

Definition (Reactivity)

- A simple reactivity formula is of the form

$$\Box\Diamond\varphi \vee \Diamond\Box\psi$$

  for prop. formulas φ and ψ.
- A very general class of formulae are conjunctions of reactivity formulae.

Reactivity

Definition (Reactivity)

- A simple reactivity formula is of the form

$$\Box\Diamond\varphi \lor \Diamond\Box\psi$$

  for prop. formulas φ and ψ.
- A very general class of formulae are conjunctions of reactivity formulae.
- equivalent:

$$\Box\Diamond\psi' \rightarrow \Box\Diamond\varphi$$

Reactivity

Proposition

Strong fairness can be specified as the following reactivity formula.

$$\Box\Diamond enabled(\tau) \rightarrow \Box\Diamond taken(\tau)$$

GCD code

Program: GCD

$$
\begin{array}{l}
P :: [\ \text{in } a, b : \text{ integer where } a > 0, b > 0\,; \\
\quad \text{local } x, y : \text{ integer where } x = a, y = b\,; \\
\quad \text{out } g : \text{ integer}\,; \\
P_1 :: [\ l_0 : [\ l_1 : \text{ while } x \neq y \text{ do } l_2 : [\\
\qquad [\ l_3 : \text{ await } x > y\,;\ l_4 : x := x - y\,;\ ] \\
\qquad \text{or} \\
\qquad [\ l_5 : \text{ await } y > x\,;\ l_6 : y := y - x\,;\ ]\ ] \\
\quad l_7 : g := x\,;\ l_8 : \ ]\ ]\ ]
\end{array}
$$

GCD Example

Below is a computation π of our recurring GCD program.

P-computation

States are of the form $\langle l_n, x, y, g \rangle$.

$$
\begin{aligned}
\pi : \quad & \langle l_1, 21, 49, 0\rangle \rightarrow \langle l_2^b, 21, 49, 0\rangle \rightarrow \langle l_6, 21, 49, 0\rangle \rightarrow \\
& \langle l_1, 21, 28, 0\rangle \rightarrow \langle l_2^b, 21, 28, 0\rangle \rightarrow \langle l_6, 21, 28, 0\rangle \rightarrow \\
& \langle l_1, 21, 7, 0\rangle \rightarrow \langle l_2^a, 21, 7, 0\rangle \rightarrow \langle l_4, 21, 7, 0\rangle \rightarrow \\
& \langle l_1, 14, 7, 0\rangle \rightarrow \langle l_2^a, 14, 7, 0\rangle \rightarrow \langle l_4, 14, 7, 0\rangle \rightarrow \\
& \langle l_1, 7, 7, 0\rangle \rightarrow \langle l_7, 7, 7, 0\rangle \rightarrow \langle l_8, 7, 7, 7\rangle \rightarrow \cdots
\end{aligned}
$$

GCD Example

Below is a computation π of our recurring GCD program.

- a and b are fixed: $\pi \models \Box(a \doteq 21 \wedge b \doteq 49)$.

P-computation

States are of the form $\langle l_n, x, y, g \rangle$.

$$\begin{aligned}
\pi: \quad & \langle l_1, 21, 49, 0\rangle \to \langle l_2^b, 21, 49, 0\rangle \to \langle l_6, 21, 49, 0\rangle \to \\
& \langle l_1, 21, 28, 0\rangle \to \langle l_2^b, 21, 28, 0\rangle \to \langle l_6, 21, 28, 0\rangle \to \\
& \langle l_1, 21, 7, 0\rangle \to \langle l_2^a, 21, 7, 0\rangle \to \langle l_4, 21, 7, 0\rangle \to \\
& \langle l_1, 14, 7, 0\rangle \to \langle l_2^a, 14, 7, 0\rangle \to \langle l_4, 14, 7, 0\rangle \to \\
& \langle l_1, 7, 7, 0\rangle \to \langle l_7, 7, 7, 0\rangle \to \langle l_8, 7, 7, 7\rangle \to \cdots
\end{aligned}$$

GCD Example

Below is a computation π of our recurring GCD program.

- a and b are fixed: $\pi \models \Box(a \doteq 21 \land b \doteq 49)$.
- $terminated$ denotes the formula $at(l_8)$.

P-computation

States are of the form $\langle l_n, x, y, g\rangle$.

$$\begin{aligned}
\pi: \quad & \langle l_1, 21, 49, 0\rangle \rightarrow \langle l_2^b, 21, 49, 0\rangle \rightarrow \langle l_6, 21, 49, 0\rangle \rightarrow \\
& \langle l_1, 21, 28, 0\rangle \rightarrow \langle l_2^b, 21, 28, 0\rangle \rightarrow \langle l_6, 21, 28, 0\rangle \rightarrow \\
& \langle l_1, 21, 7, 0\rangle \rightarrow \langle l_2^a, 21, 7, 0\rangle \rightarrow \langle l_4, 21, 7, 0\rangle \rightarrow \\
& \langle l_1, 14, 7, 0\rangle \rightarrow \langle l_2^a, 14, 7, 0\rangle \rightarrow \langle l_4, 14, 7, 0\rangle \rightarrow \\
& \langle l_1, 7, 7, 0\rangle \rightarrow \langle l_7, 7, 7, 0\rangle \rightarrow \langle l_8, 7, 7, 7\rangle \rightarrow \cdots
\end{aligned}$$

GCD Example

Do the following properties hold for π? And why?

GCD Example

Do the following properties hold for π? And why?

1. $\Box terminated$ (safety)

GCD Example

Do the following properties hold for π? And why?

1. $\Box terminated$ (safety)
2. $at(l_1) \rightarrow terminated$

GCD Example

Do the following properties hold for π? And why?

1. $\Box terminated$ (safety)
2. $at(l_1) \rightarrow terminated$
3. $at(l_8) \rightarrow terminated$

GCD Example

Do the following properties hold for π? And why?

1. $\Box terminated$ (safety)
2. $at(l_1) \rightarrow terminated$
3. $at(l_8) \rightarrow terminated$
4. $at(l_7) \rightarrow \Diamond terminated$ (conditional liveness)

GCD Example

Do the following properties hold for π? And why?

1. $\Box terminated$ (safety)
2. $at(l_1) \rightarrow terminated$
3. $at(l_8) \rightarrow terminated$
4. $at(l_7) \rightarrow \Diamond terminated$ (conditional liveness)
5. $\Diamond at(l_7) \rightarrow \Diamond terminated$ (obligation)

GCD Example

Do the following properties hold for π? And why?

1. $\Box terminated$ (safety)
2. $at(l_1) \rightarrow terminated$
3. $at(l_8) \rightarrow terminated$
4. $at(l_7) \rightarrow \Diamond terminated$ (conditional liveness)
5. $\Diamond at(l_7) \rightarrow \Diamond terminated$ (obligation)
6. $\Box(\mathrm{gcd}(x, y) \doteq \mathrm{gcd}(a, b))$ (safety)

GCD Example

Do the following properties hold for π? And why?

1. $\Box terminated$ (safety)
2. $at(l_1) \to terminated$
3. $at(l_8) \to terminated$
4. $at(l_7) \to \Diamond terminated$ (conditional liveness)
5. $\Diamond at(l_7) \to \Diamond terminated$ (obligation)
6. $\Box(\mathrm{gcd}(x, y) \doteq \mathrm{gcd}(a, b))$ (safety)
7. $\Diamond terminated$ (liveness)

GCD Example

Do the following properties hold for π? And why?

1. $\Box terminated$ (safety)
2. $at(l_1) \rightarrow terminated$
3. $at(l_8) \rightarrow terminated$
4. $at(l_7) \rightarrow \Diamond terminated$ (conditional liveness)
5. $\Diamond at(l_7) \rightarrow \Diamond terminated$ (obligation)
6. $\Box(\mathrm{gcd}(x, y) \doteq \mathrm{gcd}(a, b))$ (safety)
7. $\Diamond terminated$ (liveness)
8. $\Diamond\Box(y \doteq \mathrm{gcd}(a, b))$ (persistence)

GCD Example

Do the following properties hold for π? And why?

1. $\Box terminated$ (safety)
2. $at(l_1) \rightarrow terminated$
3. $at(l_8) \rightarrow terminated$
4. $at(l_7) \rightarrow \Diamond terminated$ (conditional liveness)
5. $\Diamond at(l_7) \rightarrow \Diamond terminated$ (obligation)
6. $\Box(\mathrm{gcd}(x, y) \doteq \mathrm{gcd}(a, b))$ (safety)
7. $\Diamond terminated$ (liveness)
8. $\Diamond\Box(y \doteq \mathrm{gcd}(a, b))$ (persistence)
9. $\Box\Diamond terminated$ (recurrence)

Exercises

Exercises

1. Show that the following formulae are (not) LTL-valid.
   1.1 $\Box\varphi \leftrightarrow \Box\Box\varphi$
   1.2 $\Diamond\varphi \leftrightarrow \Diamond\Diamond\varphi$
   1.3 $\neg\Box\varphi \rightarrow \Box\neg\Box\varphi$
   1.4 $\Box(\Box\varphi \rightarrow \psi) \rightarrow \Box(\Box\psi \rightarrow \varphi)$
   1.5 $\Box(\Box\varphi \rightarrow \psi) \vee \Box(\Box\psi \rightarrow \varphi)$
   1.6 $\Box\Diamond\Box\varphi \rightarrow \Diamond\Box\varphi$
   1.7 $\Box\Diamond\varphi \leftrightarrow \Box\Diamond\Box\Diamond\varphi$
2. A *modality* is a sequence of $\neg$, $\Box$ and $\Diamond$, including the empty sequence ϵ. Two modalities π and τ are *equivalent* if $\pi\varphi \leftrightarrow \tau\varphi$ is valid.
   2.1 Which are the non-equivalent modalities in LTL, and
   2.2 what are their relationship (ie. implication-wise)?

Section

Logic model checking: What is it about?

Chapter 3 "LTL model checking"
Course "Model checking"
Martin Steffen
Autumn 2021

Logic model checking (1)

- a technique for verifying *finite-state* (concurrent) systems

Often involves steps as follows

1. Modeling the system
   - It may require the use of *abstraction*
   - Often using some kind of *automaton*
2. Specifying the properties the design must satisfy
   - It is impossible to determine all the properties the systems should satisfy
   - Often using some kind of temporal logic
3. Verifying that the system satisfies its specification
   - In case of a negative result: error trace
   - An error trace may be product of a specification error

Logic model checking (2)

The *application* of model checking at the design stage of a system typically consists of the following steps:

1. Choose the properties (correctness requirements) critical to the sytem you want to build (software, hardware, protocols)
2. Build a model of the system (will use for verification) guided by the above correctness requirements
   - The model should be as small as possible (for efficiency)
   - It should, however, capture everything which is relevant to the properties to be verified
3. Select the appropriate verification method based on the model and the properties (LTL-, CTL*-based, probabilistic, timed, weighted . . .)
4. Refine the verification model and correctness requirements until all correctness concerns are adequately satisfied

State-space explosion

Main causes of combinatorial complexity in SPIN/Promela (and in other model checkers.)

- The number of and size of buffered channels
- The number of asynchronous processes

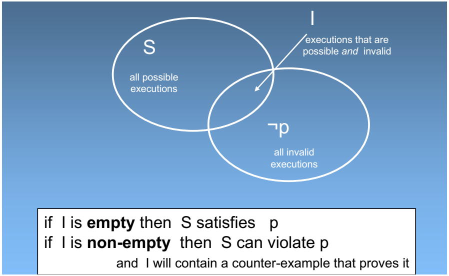
The basic method

- System: $\mathcal{L}(S)$ (set of possible behaviors/traces/words of S)
- Property: $\mathcal{L}(P)$ (the set of valid/desirable behaviors)
- Prove that $\mathcal{L}(S) \subseteq \mathcal{L}(P)$ (everything possible is valid)
  - Proving language inclusion is complicated

The basic method

- System: $\mathcal{L}(S)$ (set of possible behaviors/traces/words of S)
- Property: $\mathcal{L}(P)$ (the set of valid/desirable behaviors)
- Prove that $\mathcal{L}(S) \subseteq \mathcal{L}(P)$ (everything possible is valid)
  - Proving language inclusion is complicated
- Method
  - Let $\overline{\mathcal{L}(P)}$ be the language $\Sigma^\omega \setminus \mathcal{L}(P)$ of words not accepted by P
  - Prove $\mathcal{L}(S) \cap \overline{\mathcal{L}(P)} = \emptyset$
    - there is no accepted word by S disallowed by P

The basic method



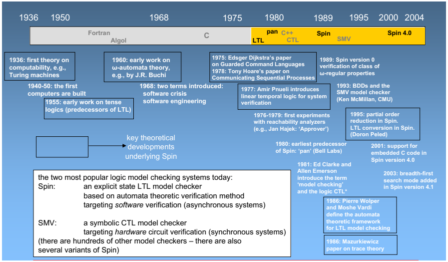
Scope of the method

Logic model checkers (LMC) are suitable for *concurrent* and *multi-threading finite-state* systems.

Some of the errors LMC may catch:

- Deadlocks {(two or more competing processes are waiting for the other to finish, and thus neither ever does)}
- Livelocks {(two or more processes continually change their state in response to changes in the other processes)}
- Starvation {(a process is perpetually denied access to necessary resources)}
- Priority and locking problems
- Race conditions {(attempting to perform two or more operations at the same time, which must be done in the proper sequence in order to be done correctly)}
- Resource allocation problems
- Incompleteness of specification
- Dead code {(unreachable code)}
- Violation of certain system bounds
- Logic problems: e.g, temporal relations
- ...

A bit of history



On correctness (reminder)

- A system is correct if it meets its design requirements.
- There is no notion of "absolute" correctness: It is always wrt. a given specification
- Getting the properties (requirements) right is as important as getting the model of the system right

Examples of correctness requirements

- A system should not *deadlock*
- No process should *starve* another
- *Fairness* assumptions
  - E.g., an infinite often enabled process should be executed infinitely often
- *Causal relations*
  - E.g., each time a request is send, and acknowledgment must be received (*response* property)

On models and abstraction

- The use of abstraction is needed for building models (systems may be extremely big)
  - A model is always an abstraction of the reality
- The choice of the model/abstractions depends on the requirements to be checked
- A good model keeps only relevant information
  - A trade-off must be found: too much detail may complicate the model; too much abstraction may oversimplify the reality
- Time and probability are usually abstracted away in LMC

Building verification models

- Statements about system design and system requirement must be separated
  - One formalism for specifying behavior (system design)
  - Another formalism for specifying system requirements (correctness properties)
- The two types of statements define a verification model
- A model checker can now
  - Check that the behavior specification (the design) is logically consistent with the requirement specification (the desired properties)

Distributed algorithms

Two asynchronous processes may easily get blocked when competing for a shared resource

in real-life conflicts ultimately get resolved by *human judgment.*
computers, though, must be able to resolve it with fixed algorithms

A small multi-threaded program

```
int   x,  y,  r;
int  *p, *q, *z;
int **a;

thread_1(void)      /* initialize p, q, and r */
{
   p = &x;
   q = &y;
   z = &r;
}
thread_2(void)      /* swap contents of x and y */
{
   r = *p;
   *p = *q;
   *q = r;
}
thread_3(void)      /* access z via a and p */
{
   a = &p;
   *a = z;
   **a = 12;
}
```

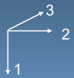
3 asynchronous threads
accessing shared data
3 statements each
how many test runs are needed to
check that no data corruption can occur?

Thread interleaving

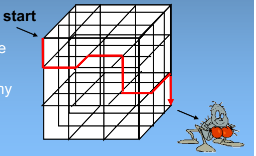
- the number of possible thread interleavings is...

$$\frac{9!}{6!.3!} \cdot \frac{6!}{3!.3!} \cdot \frac{3!}{3!} = \text{1,680 possible executions}$$

placing 3 sets of 3 tokens in 9 slots

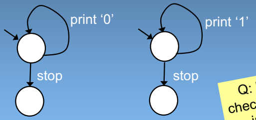


- are all these executions okay?
- can we check them all? should we check them all?
- in classic system testing, how many would normally be checked?



A simpler example

- consider two 2-state automata
  - representing two asynchronous processes
- one can print an arbitrary number of ‘0’ digits, or stop
- the other can print an arbitrary number of ‘1’ digits, or stop



Q: how could a model checker deal with possibly infinite executions?

how many different combined executions are there?
i.e., how many different binary numbers can be printed?
how would one test that this system does what we think it does?

Section

Automata and logic

Chapter 3 “LTL model checking”
Course “Model checking”
Martin Steffen
Autumn 2021

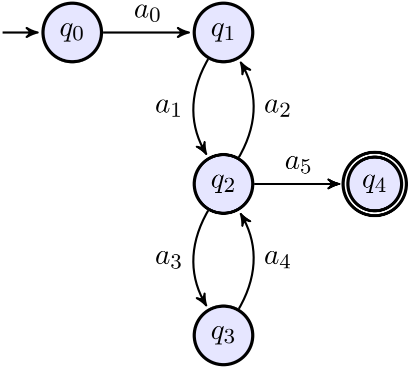
FSA

Definition (Finite-state automaton)

A *finite-state automaton* is a quintuple $(Q, q_0, , \Sigma, F, \rightarrow)$, where

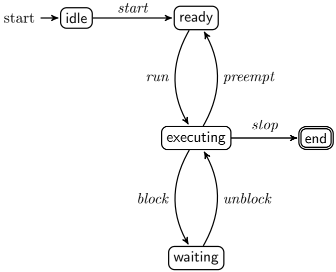
- Q is a finite set of states
- $q_0 \in Q$ is a distinguished initial state
- the "alphabet" Σ is a finite set of labels (symbols)
- $F \subseteq Q$ is the (possibly empty) set of final states
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation, connecting states in Q.

Example FSA



Example: An interpretation

The above automaton may be interpreted as a *process scheduler*:



Determinism vs. non-determinism

Definition (Determinism)

A finite state automaton $\mathcal{A} = (Q, q_0, \Sigma, F, \rightarrow)$ is deterministic iff

$$q_0 \xrightarrow{a} q_1 \ \wedge q_0 \xrightarrow{a} q_2 \implies q_1 = q_2$$

Runs

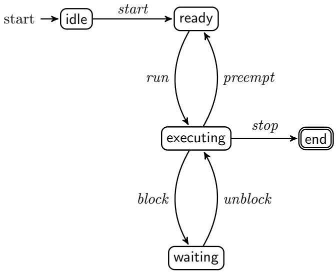
Definition (Run)

A *run* of a finite state automaton $\mathcal{A} = (Q, q_0, \Sigma, F, \rightarrow)$ is a (possibly infinite) sequence

$$\sigma = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \ldots$$

- $q \xrightarrow{a} q'$ is meant as $(q, a, q') \in \rightarrow$
- each run corresponds to a state sequence (a word) over Q and a word over Σ

Example run



- state sequences from runs: idle ready (execute waiting)*
- corresponding words in Σ: $start\ run(block, unblock)^*$
- A single state sequence may correspond to more than one word
- non-determinism: the same Σ-word may correspond to different state sequence

"Traditional" acceptance

Definition (Acceptance)

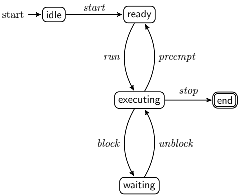
An *accepting* run of a finite state automaton $\mathcal{A} = (Q, q_0, \Sigma, F, \rightarrow)$ is a finite run $\sigma = q_0 \stackrel{a_0}{\rightarrow} q_1 \stackrel{a_1}{\rightarrow} \ldots \stackrel{a_{n-1}}{\rightarrow} q_n$, with $q_n \in F$.

Accepted language

Definition (Language)

The *language* $[\![\mathcal{A}]\!]$ (sometimes also written $\mathcal{L}(\mathcal{A})$ of automaton $\mathcal{A} = (Q, q_0, \Sigma, F, \rightarrow)$ is the set of words over Σ that correspond to the set of all the accepting runs of $\mathcal{A}$.

- generally: *infinitely* many words in a language
- remember: regular expressions etc.



Reasoning about runs

Sample correctness claim (positive formulation)

If first p becomes true and afterwards q becomes true, then afterwards, r can no longer become true

Seen negatively

It's an error if in a run, one sees first p, then q, and then r.

- reaching accepting state $\Rightarrow$ correctness property violation
- accepting state represents error

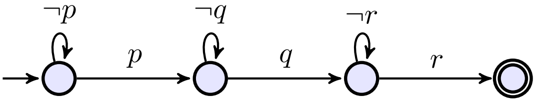
Reasoning about runs

Sample correctness claim (positive formulation)

If first p becomes true and afterwards q becomes true, then afterwards, r can no longer become true

Seen negatively

It's an error if in a run, one sees first p, then q, and then r.



- reaching accepting state ⇒ correctness property violation
- accepting state represents error

Comparison to FSA in "standard" language theory

- remember classical FSA (and regular expressions)
- for instance: *scanner* or *lexer*
- (typically infinite) languages of finite words
- remember: accepting runs are finite
- in "classical" language theory: infinite words completely out of the picture

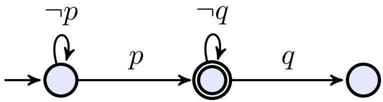
Reasoning about infinite runs

Some liveness property

"if p then eventually q."

Seen negatively

It's an error if one sees p and afterwards never q (i.e., forever $\neg q$)



- violation: only possible in an infinite run
- not expressible by *conventional* notion of acceptance

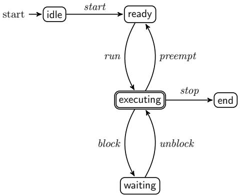
Büchi acceptance

- infinite run: often called ω-*run* ("omega run")
- corresponding acceptance properties: ω-acceptance
- different versions: Büchi, Muller, Rabin, Streett, parity etc., acceptance conditions
  - Here, for now: Büchi acceptance condition [3] [2]

Definition (Büchi acceptance)

An accepting ω-run of the finite state automaton $\mathcal{A} = (Q, q_0, \Sigma, F, \rightarrow)$ is an infinite run σ such that some $q_i \in F$ occurs infinitely often in σ.

Example: "process scheduler"



- accepting ω-runs
- ω-language

infinite state sequence	**ω-word**
idle (ready executing)$^\omega$	*start* (*run preempt*)$^\omega$

Generalized Büchi automata

Definition (Generalized Büchi automaton)

A *generalized Büchi automaton* is an automaton $\mathcal{A} = (Q, q_0, \Sigma, F, \rightarrow)$, where $F \subseteq 2^Q$.
Let $F = \{f_1, \ldots, f_n\}$ and $f_i \subseteq Q$. A run σ of $\mathcal{A}$ is *accepting* if

$$\text{for each } f_i \in F,\ inf(\sigma) \cap f_i \neq \emptyset.$$

- $inf(\sigma)$: states visited infinitely often in σ
- generalized Büchi automaton: multiple accepting sets instead of only one ($\neq$ "original" Büchi Automata)
- generalized Büchi automata: *equally* expressive

Stuttering

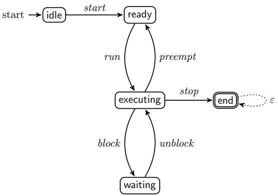
- treat finite and infinite acceptance uniformely
- finite runs as inifite ones, where, at some point, infinitely often “nothing” happens (stuttering)
  - Let ε be a predefined nil symbol
  - alphabet/label set extended to $\Sigma + \{\varepsilon\}$
  - extend a finite run to an equivalent infinite run: keep on stuttering after the end of run. The run must end in a final state.

Definition (Stutter extension)

The *stutter extension* of a finite run σ with last state s_n, is the ω-run

$$\sigma\ (s_n, \varepsilon, s_n)^\omega\ . \tag{1}$$

Stuttering example



From Kripke structures to Büchi automata

- LTL formulas can be interpreted on sets of infinite runs of Kripke structures
- Kripke structure/model:
  - "automaton" or "transition system"
  - transitions unlabelled (typically)
  - states (or worlds): "labelled", in the most basic situation: sets of propositional variables

Kripke structure (reminder)

Definition (Kripke structure)

A Kripke structure M is a four-tuple (S, R, S_0, V) where

- S is a finite non-empty set of states (also "worlds")
- $R \subseteq S \times S$ is a total relation between states (transition relation, aka accessibility relation)
- $S_0 \subseteq S$ is the set of starting states
- $V : S \to 2^P$ is a map labeling each state with a set of propositional variables

Notation: $\to$ for accessibility relation
A path in M is an infinite sequence $\sigma = s_0, s_1, s_2, \ldots$ of states such that $s_i \to s_{i+1}$ (for all $i \geq 0$).

BAs vs. KSs

- "subtle" differences
- labelled transitions vs. labelled states
- easy to transform one representation into the other
- here: from KS to BA.
  - states: basically the same
  - initial state: just make a unique initial one
  - transition labels: all possible combinations of atomic props
  - states and transitions: transitions in $\mathcal{A}$ allowed if
    - covered by accesssibility in the KS (+ initial transition added)
    - transition labelled by the "post-state-labelling" from KS

KS to BA

Given $M = (W, R, W_0, V)$. An automaton $\mathcal{A} = (Q, q_0, \Sigma, F, \rightarrow)$ can be obtained from a Kripke structure as follows

transition labels: $\Sigma = 2^P$

states:

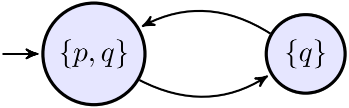
- $Q = W + \{i\}$
- $q_0 = i$
- $F = W + \{i\}$

transitions:

- $s \xrightarrow{a} s'$ iff $s \rightarrow_M s'$ and $a = V(s')$ $s, s' \in W$
- $i \xrightarrow{a} s \in T$ iff $s \in W_0$ and $a = V(s)$

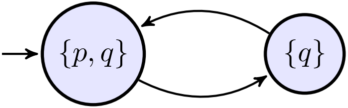
Example: KS to BA

A Kripke structure (whose only infinite run satisfies (for instance) $\Box q$ and $\Box\Diamond p$):

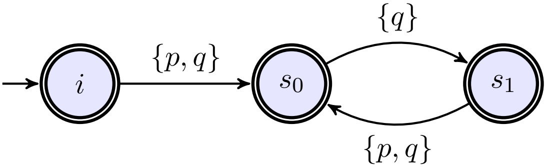


Example: KS to BA

A Kripke structure (whose only infinite run satisfies (for instance) $\Box q$ and $\Box\Diamond p$):



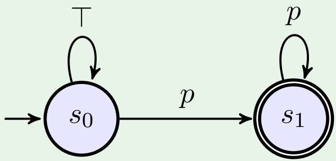
The corresponding Büchi automaton:



From logic to automata

- cf. regular expressions and FSAs
- for any LTL formula φ, there exists a Büchi automaton that accepts precisely those runs for which the formula φ is satisfied

Example (stabilization: "eventually always p", $\Diamond\Box p$:)



(Lack of?) expressiveness of LTL

- note: analogy with regular expressions and FSAs: not 100%
- in the finite situation: "logical" specification language (regexp) correspond fully to machine model (FSA)
- here: LTL is weaker! than BAs
- ω-regular expressions + ω-regular languages
- generalization of regular languages
- allowed to use r^ω (not just r^*)

Generalization of RE / FSA to infinite words

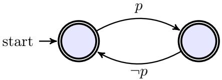
ω-regular language correspond to NBAs

ω-regular properties strictly more expressive than LTL

Temporal property

p is always false after an *odd* number of steps

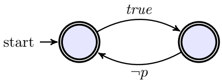
$$p \land \Box(p \to \bigcirc \neg p) \land \Box(\neg p \to \bigcirc p)$$



ω-regular properties strictly more expressive than LTL

Temporal property

p is always false after an *odd* number of steps

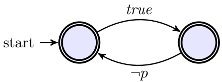


ω-regular properties strictly more expressive than LTL

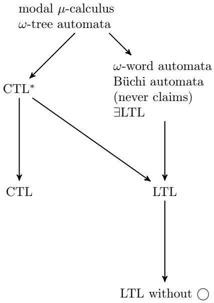
Temporal property

p is always false after an *odd* number of steps

$$\exists t.\ t \land \Box(t \to \bigcirc \neg t) \land \Box(\neg t \to \bigcirc t) \land \Box(\neg t \to p)$$



Expressiveness



Core part of automata-based MC

- remember: MC checks "system against formula" $S \models \varphi$

Linear time approach

- ω-language of the behavior of S is *contained* in the language allowed by φ

- core idea then: instead of

$$\mathcal{L}(S) \subseteq \mathcal{L}(P_\varphi)$$

do

$$\mathcal{L}(S) \cap \overline{\mathcal{L}(P_\varphi)} = \emptyset$$

where S is a model of the system P_φ represents the property φ

What's needed for automatic MC?

$$\mathcal{L}(S) \cap \overline{\mathcal{L}(P_\varphi)} = \emptyset$$

Algorithms needed for

1. translation LTL to Büchi
2. language emptiness: are there any accepting runs?
3. language intersection: are there any runs accepted by two or more automata?
4. language complementation

- thankfully: all that is *decidable*

How could one do it, then?

- *system* represented as Büchi automaton A
  - The automaton corresponds to the *asynchronous* product of automata $A_1, \ldots, A_n$ (representing the asynchronous processes)

$$A = \prod_{i=1}^{n} A_i$$

- *property* originally given as an LTL formula φ
- translate φ into a Büchi automaton B_φ
- check

$$\mathcal{L}(A) \cap \overline{\mathcal{L}(B)} = \emptyset$$

Better avoid complementation

In practice (e.g., in SPIN): avoid automata complementation:

- Assume A as before
- The negation of the property φ is automatically translated into a Büchi automaton $\overline{B}$ (since $\overline{\mathcal{L}(B)} \equiv \mathcal{L}(\overline{B})$)
- By making the synchronous product of A and $\overline{B}$ ($\overline{B} \otimes A$) we can check:

$$\mathcal{L}(A) \cap \mathcal{L}(\overline{B}) = \emptyset$$

- If intersection is empty: $A \models \varphi$, i.e., "property φ holds for A" or "A satisfies property φ"
- else:
  - $A \not\models \varphi$
  - bonus: accepted word in the intersection counter example

Two kinds of products

- conceptually standard (but see terminating condition = definition of final states)

asynchronous

- prog's running in parallel
- interleaving
- no synchronization!
- one automaton does something, the others not

synchronous

- together with (the automaton representing) the formula
- lock-step
- however: *stuttering*.

Asychronous product

Definition (Asynchronous product)

The *asynchronous product* of two automata A_1 and A_1, (written $A_1 \times A_2$, or $A_1 \parallel A_2$) is given as $(Q, q_0, \Sigma, F, \rightarrow)$ where

- $Q = Q_1 \times Q_2$,
- $q_0 = q_0^1 \times q_0^2$,
- $\Sigma = \Sigma_1 \cup \Sigma_2$, and
- $F = \{(q_1, q_2) \mid q_1 \in F_1 \text{ or } q_2 \in F_2\}$.

$$\frac{q_1 \rightarrow_1 q_1'}{(q_1, q_2) \rightarrow (q_1', q_2)} \text{ PAR}_1 \qquad \frac{q_2 \rightarrow_2 q_2'}{(q_1, q_2) \rightarrow (q_1, q_2')} \text{ PAR}_1$$

3n+1 inspired example

- 3n+1 problem
- Assume 2 non-terminating asynchronous processes A_1 and A_2:
- A_1 tests whether the value of a variable x is odd, in which case updates it to $3 * x + 1$
- A_2 tests whether the value of a variable x is even, in which case updates it to $x/2$

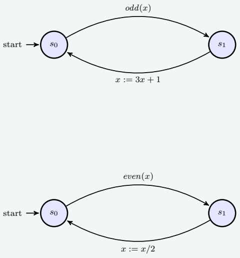
Question

Does the corresponding function *terminate* for all inputs x?

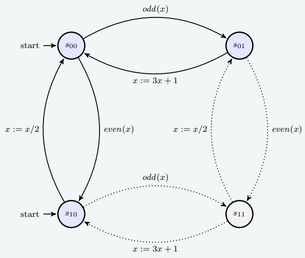
- Let φ the following property: $\Box\Diamond(x \geq 4)$ (negated $\Diamond\Box(x < 4)$)

Example: async product

A_1 and A_2



$A_1 \times A_2$



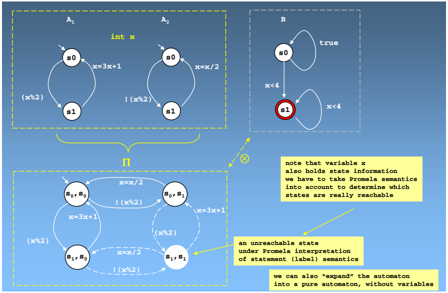
Tests or guards on transitions

- guarded commands (thanks to Dijsktra)
- conditional transitions, predicated on a guard
- Promela semantics, an expression statement has to evaluate to non-zero to be executable (*enabled*). So to test whether a variable `x` is even, we write `!(x%2)` and `x%2` for checking whether `x` is odd.

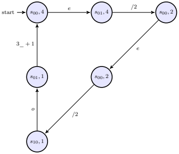
E.g.: given `x=4`, `!(4%2)` evaluates to `!(0)` or written more clearly as `!(false)` which is `(true)`.

Example: Async. product

- ignore B on the right-hand side first
- final states not really important



Example: Pure automaton



Synchronous product

Definition (Synchonous product (special case))

The synchronous product of two finite automata $\mathcal{A}_1$ and $\mathcal{A}_2$ (written $\mathcal{A}_1 \otimes \mathcal{A}_2$), for the special case where $F_1 = Q_1$, is defined as finite state automaton $\mathcal{A} = (Q, q_0, \Sigma, F, \rightarrow)$ where:

- $Q = Q_1 \times Q_1$
- $q_0 = (q_{01}, q_{02})$
- $\Sigma = \Sigma_1 \times \Sigma_2$.
- $\rightarrow = \rightarrow_1 \times \rightarrow_2$
- $(q_1, q_2) \in F$ if $q_2 \in F_2$

Let the system automaton stutter

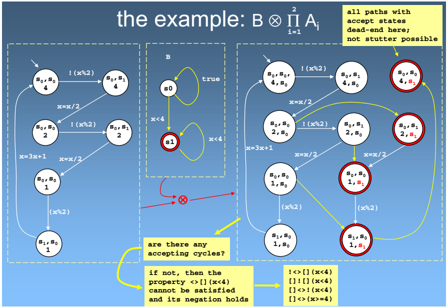
- asymmetric situation
- one automaton: "system"
- second one:
  - "recognizer"
  - automaton that represents the logical LTL formula
- for system automating: add stuttering
- stutter: a self-loop labeled with ε at every every state in without outgoing transitions

Definition (Stuttering synchonous product)

The synchronous product of two finite automata P and B (written $P \otimes B$ is defined as finite state automaton $\mathcal{A} = (Q, q_0, \Sigma, F, \rightarrow)$ where:

- $Q = Q'_1 \times Q_1$, where P' is the *stutter closure* of P
  - A self-loop labeled with ε is attached to every state in P without outgoing transitions in $P.T$)
- $A.s_0$ is the pair $(P.s_0, B.s_0)$
- $A.L$ is the set of pairs (l_1, l_2) such that $l_1 \in P'.L$ and

Example: synch. product for $3n + 1$ system and property



Section

Model checking algorithm

Chapter 3 "LTL model checking"
Course "Model checking"
Martin Steffen
Autumn 2021

Algorithmic checking for emptyness

- for FSA: emptyness checking is easy: reachability
- For Büchi:
  - more complex acceptence (namely ω-often)
  - simple, one time reachability not enough

⇒ "repeated" reachability

⇒ from initial state, reach an accepting state, and then again, and then again . . .

- cf. "lasso" picture
- technically done with the help of SCCs.

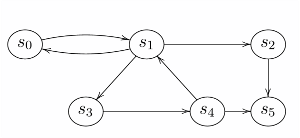
Strongly-connected components

Definition (SCC)

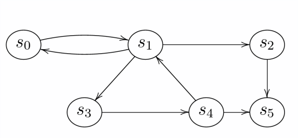
A subset $S' \subseteq S$ in a directed graph is strongly connected if there is a path between any pair of nodes in S', passing only through nodes in S'.

A strongly-connected component (SCC) is a *maximal* set of such nodes, i.e. it is not possible to add any node to that set and still maintain strong connectivity.

SCC example

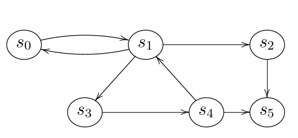


SCC example



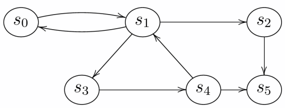
- Strongly-connected subsets:
- Strongly-connected components:

SCC example



- Strongly-connected subsets:
  $S = \{s_0, s_1\}, \quad S' = \{s_1, s_3, s_4\}, \quad S'' = \{s_0, s_1, s_3, s_4\}$
- Strongly-connected components:

SCC example



- Strongly-connected subsets:
  $S = \{s_0, s_1\}, \quad S' = \{s_1, s_3, s_4\}, \quad S'' = \{s_0, s_1, s_3, s_4\}$
- Strongly-connected components: Only
  $S'' = \{s_0, s_1, s_3, s_4\}$

Checking emptiness

Büchi automaton $A = (Q, s_0, \Sigma, \rightarrow, F)$ with accepting run σ

Core observation

As Q is finite, there is some suffix σ' of σ s.t. every state on σ' is reachable from any other state on σ'

- I.a.w: those set of states is strongly connected.
- This set is reachable from an initial state and contains an accepting state

Emptyness check

Checking non-emptiness of $[\![A]\!]$ is equivalent to finding a SCC in the graph of A that is reachable from an initial state and contains an accepting state

Emptyness checking and counter example

- different algos for SCC. E.g.:
  - Tarjan's version of the *depth-first search* (DFS) algorithm
  - SPIN *nested depth-first search* algorithm

Emptyness checking and counter example

- different algos for SCC. E.g.:
  - Tarjan's version of the *depth-first search* (DFS) algorithm
  - SPIN *nested depth-first search* algorithm
- If the language $[\![A]\!]$ is non-empty, then there is a counterexample which can be represented in a finite way
  - It is *ultimately periodic*, i.e., it is of the form $\sigma_1\sigma_2^\omega$, where σ_1 and σ_2 are finite sequences

Model checking algorithm

- Let A be the automaton specifying the system and $\overline{B}$ the automaton corresponding to the negation of the property φ

Model checking algorithm

- Let A be the automaton specifying the system and $\overline{B}$ the automaton corresponding to the negation of the property φ

1. Construct the intersection automaton $C = A \cap \overline{B}$

Model checking algorithm

- Let A be the automaton specifying the system and $\overline{B}$ the automaton corresponding to the negation of the property φ

1. Construct the intersection automaton $C = A \cap \overline{B}$
2. Apply an algorithm to find SCCs reachable from the initial states of C

Model checking algorithm

- Let A be the automaton specifying the system and $\overline{B}$ the automaton corresponding to the negation of the property φ

1. Construct the intersection automaton $C = A \cap \overline{B}$
2. Apply an algorithm to find SCCs reachable from the initial states of C
3. If none of the SCCs found contains an accepting state
   - The model A satisfies the property/specification φ

Model checking algorithm

- Let A be the automaton specifying the system and $\overline{B}$ the automaton corresponding to the negation of the property φ

1. Construct the intersection automaton $C = A \cap \overline{B}$
2. Apply an algorithm to find SCCs reachable from the initial states of C
3. If none of the SCCs found contains an accepting state
   - The model A satisfies the property/specification φ
4. Otherwise,
   4.1 Take one strongly-connected component *SC* of C
   4.2 Construct a path σ_1 from an initial state of C to some accepting state s of *SC*
   4.3 Construct a cycle from s and back to itself (such cycle exists since *SC* is a strongly-connected component)
   4.4 Let σ_2 be such cycle, excluding its first state s
   4.5 Announce that $\sigma_1\sigma_2^\omega$ is a counterexample that is accepted by A, but it is not allowed by the property/specification φ

LTL to Büchi

- translation to Generalized Büchi GBA
- cf. Thompson's construction
- *structural* translation
- crucial idea: connect semantics to the syntax.
- compare Hintikka-sets or similar constructions for FOL

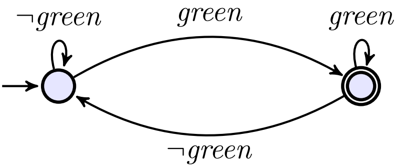
Source and terminology: Baier and Katoen [1]

- transition systems TS:
  - corresponds to Kripke systems
  - state-labelled (transition labels irrelevant)
  - labelled by sets of atomic props: $\Sigma = 2^P$
  - "language" or behavior of the TS: (traces): infinite sequences over Σ

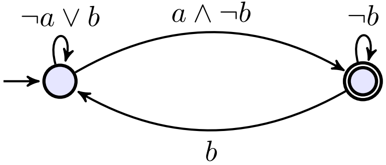
Illustrative examples (5.32)

1. $\Box\Diamond green$
2. $\Box(request \rightarrow \Diamond response)$
3. $\Diamond\Box a$

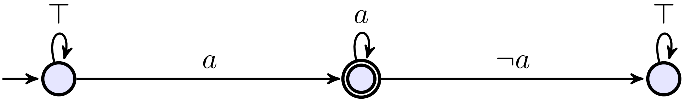
$\Box\Diamond green$



$\square(request \rightarrow \lozenge response)$



$\Diamond\Box a$



Reminder: Generalized NBA

- equi-expressive than NBA
- used in the construction
- different way of defining acceptance
  - acceptance: set of acceptance sets = set of sets of elements of Q.
  - acceptance: each acceptance set F_i must be "hit" infinitely often

Basic idea for $\mathcal{G}_\varphi$

- not the construction yet, but: "insightful" property
- find a mental picture:
  - what are the states of the automaton
  - (and how are they connected by transitions)
- $A_i \in \Sigma$, sets of atomic props
- B_i : "extended" (by sub-formulas of φ), i.e., $B_i \supseteq A_i$.

States as sets of formulas

Namely those that are intended to be in the "language of that state". I.e., the B_i's form the states of $\mathcal{G}_\varphi$.

Given $\sigma = A_0 A_1 A_2 \ldots \in [\![\varphi]\!]$.
Extension to $\hat{\sigma} = B_0 B_1 B_2 \ldots$

$$\psi \in B_i \qquad \text{iff} \qquad \underbrace{A_i, A_{i+1} A_{i+2} \ldots}_{\sigma^i} \models \psi$$

$\hat{\sigma}$ = run (ultimately: state-sequence) in $\mathcal{G}_\varphi$

IN5110 – Verification and specification of parallel systems

Cf. FSAs

- states as "sets" of "words" (language resp. set of ltl formulas)
- cf. Myhill-Nerode
- a bit different, (equivalence on languages of finite words)
- represent states by equivence classes of words

Closure of φ

- related to Fisher-Ladner closure
- See page 276
- "states" A_i from the mental picture
- what's a "closure" in general?
- extending A_i to B_i not by *all* true formulas, but only those that could conceivably play a role in an automaton checking φ
- $\Rightarrow$ achieving "finiteness" of the construction

How to extend A_i's

- not by irrelevant stuff (closure of φ).
- two other conditions:
  - avoid contradictions (consistency)
  - for every ψ: either ψ or $\neg\psi$ included
- maximally consistent sets! (here called *elementary*)
- in one state: local perspective only (but don't forget U)
- Cf: KS has an interpretation for each P, here now (in the intended BA),

"semantics" (states) by "syntax"

"interpretation" for *all relevant formulas* "in" each state (subformulas of φ and their negation)

Avoid contradictions

Definition (Propositional consistency)

A set B is consistent wrt. propositional logic (and relative the closure of φ) if

1. $\psi \in B$ implies $\neg\psi \notin B$.
2. $\bot \notin B$.

Logical consequences

Definition (Closed wrt. propositional entailment)

A set B is closed under propositional entailment if

1. $\varphi_1 \wedge \varphi_2 \in B$ iff $\varphi_1 \in B$ and $\varphi_2 \in B$.
2. if $\top \in closure(\varphi)$ then $\top \in B$

Definition (Local consequences of until)

A set B is closed under local entailments wrt. the until operator (and relative the closure of φ) if

1. $\varphi_2 \in B$ implies $\varphi_1 \; U \; \varphi_2 \in B$
2. $\varphi_1 \; U \; \varphi_2 \in B$ and $\varphi_2 \notin B$ implies $\varphi_1 \in B$.

Cover all formulas or their negations

Definition (Maximality)

A set is maximal (relative to the closure of φ), if for all $\psi \in closure(\varphi)$

$$\psi \notin B \quad \text{implies} \quad \neg\psi \in B \ .$$

Definition

Given a LTL-formula φ. A set B is *maximally consistent* (or elementary) wrt. φ if it is propositionally consistent, closed under propositonal entailment and locally entailed formulas wrt. until, and if it is maximal.

Example: $\varphi = a \; U \; (\neg a \wedge b)$

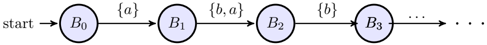
$$
\begin{array}{lclrrrrl}
B_0 & = & \{ & a, & b, & \neg(\neg a \wedge b), & \varphi & \} \\
B_1 & = & \{ & a, & b, & \neg(\neg a \wedge b), & \neg\varphi & \} \\
B_2 & = & \{ & a, & \neg b, & \neg(\neg a \wedge b), & \varphi & \} \\
B_3 & = & \{ & a, & \neg b, & \neg(\neg a \wedge b), & \neg\varphi & \} \\
B_4 & = & \{ & \neg a, & \neg b, & \neg(\neg a \wedge b), & \neg\varphi & \} \\
B_5 & = & \{ & \neg a, & b, & \neg a \wedge b, & \varphi & \}
\end{array}
$$

$$\{a, b\} \subseteq closure(\varphi)$$

Example: $\varphi = a\ U\ (\neg a \wedge b)$

$$\sigma = \{a\}\{a,b\}\{b\}\ldots \ = \ A_0 A_1 A_2 \ldots$$

- Extending (for example): A_0 to B_0
- extending σ to $\hat{\sigma}$



Construction of GNBA: general

- given P and φ
- given φ, construct an GNBA such that

$$\mathcal{L}(B) = words(\varphi)$$

- 3 core ingredients
  1. states = sets of formulas which (are suppsed to) "hold" in that state
  2. transition relation: connect the states appropriately,
  3. transitions labelled by sets of P.
- labeled transition connected states to match the semantics: for $\bigcirc\varphi$:

simplified for $\bigcirc$

go from a state containing $\bigcirc\varphi$ to a state containing φ. Label the transition with the APs from the start state.

Transition relation

$$\delta : Q \times 2^P \to 2^Q$$

- if $A \neq B \cap P$: $\delta(B, A) = \emptyset$
- if $A = B \cap P$, then $\delta(B, A)$ is the set B' such that
  - for every $\bigcirc\psi \in closure(\varphi)$:

    $$\bigcirc\psi \in B \quad \text{iff} \quad \psi \in B'$$

  - for every $\varphi_1 \ U \ \varphi_2 \in closure(\varphi)$:

    $$\varphi_1 \ U \ \varphi_2 \in B \quad \text{iff} \quad \varphi_2 \in B \quad \text{or} \quad (\varphi_1 \in B \quad \text{and} \quad \varphi_1 \ U \ \varphi_2 \in B')$$

Accepting states

$$F_{\varphi_1 U \varphi_2} = \{B \in Q \mid \varphi_2 \in B \text{ or } \varphi_1 \ U \ \varphi_2 \notin B\} \ .$$

References I

Bibliography

[1] Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.

[2] Büchi, J. R. (1960). Weak second-order arithmentic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92.

[3] Büchi, J. R. (1962). On a decision method in restricted second-order logic. In *Proceedings of the 1960 Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press.

[4] Manna, Z. and Pnueli, A. (1992). *The temporal logic of reactive and concurrent systems—Specification*. Springer Verlag, New York.